РОССИЙСКАЯ АКАДЕМИЯ НАУК
СИБИРСКОЕ ОТДЕЛЕНИЕ
ВЫЧИСЛИТЕЛЬНЫЙ ЦЕНТР
(г. Красноярск)

НЕЙРОКОМПЬЮТЕР

ПРОЕКТ СТАНДАРТА

Ответственный редактор
доктор физико-математических наук В.Л.Дунин-Барковский

Новосибирск
«Наука»
Сибирская издательская фирма РАН
1998

Нейрокомпьютер. Проект стандарта / Е.М.Миркес – Новосибирск: Наука, Сибирская издательская фирма РАН, 1998.

Многолетние усилия многих исследовательских групп привели к тому, что к настоящему моменту накоплено большое число различных «правил обучения» и архитектур нейронных сетей, способов оценивать и интерпретировать их работу, приемов использования нейронных сетей для решения прикладных задач.

В книге предпринята попытка описать различные сети, алгоритмы обучения и другие компоненты идеального нейрокомпьютера на едином языке. Такой подход преследует две цели. Во-первых сделать нейросетевые программы совместимыми по способу описания нейронных сетей и сопутствующих компонент, что сильно упростит жизнь пользователям нейросетевых приложений. Во-вторых единый подход к описанию позволяет корректно сравнивать между собой различные архитектуры нейронных сетей и алгоритмов обучения. Возможность сравнения, в свою очередь, позволит приступить к построению единой теории нейронных сетей.

Для специалистов по нейроинформатике, экспертным системам, разработчиков программного обеспечения, а также для широкого круга пользователей, интересующихся нейронными сетями.

Утверждено к печати Вычислительным центром
СО РАН (г. Красноярск)

Книга издана при финансовой поддержке
Сибирского отделения РАН,
Красноярского краевого фонда науки и
ЗАО «Сибирская Аудиторская Компания»

Введение

Многолетние усилия многих исследовательских групп привели к тому, что к настоящему моменту накоплено большое число различных «правил обучения» и архитектур нейронных сетей, способов оценивать и интерпретировать их работу, приемов использования нейронных сетей для решения прикладных задач.

До сих пор эти правила, архитектуры, системы оценки и интерпретации, приемы использования и другие интеллектуальные находки существуют в виде «зоопарка» сетей. Каждая сеть из зоопарка имеет свою архитектуру, правило обучения и решает конкретный набор задач.

Мы предлагаем систематизировать «зоопарк». Для этого полезен такой подход: каждая нейронная сеть из зоопарка должна быть представлена как реализованная на идеальном нейрокомпьютере, имеющем заданную структуру. Несомненно, структура этого идеального нейрокомпьютера со временем будет эволюционировать. Однако преимущества даже от первых шагов стандартизации несомненны. В этом нас убеждает собственный опыт восьмилетней работы по использованию нейронных сетей в различных задачах: распознавания образов [64, 290, 285], медицинской диагностики [18, 49 – 52, 72, 90, 91, 160, 161, 165, 182 – 187, 190 – 208, 255, 295 – 298, 316, 317, 341 – 345, 351, 361], прогноза [299 – 301, 364] и др.

Группа НейроКомп в течение двенадцати лет отрабатывала принципы организации нейронных вычислений. Различные варианты этих принципов были реализованы в серии программ-нейроимитаторов. Возможность формирования большинства архитектур, алгоритмов и способов использования нейронных сетей на основе небольшого числа стандартных блоков существенно облегчает создание программного обеспечения.

В данной работе описана функциональная структура идеального нейрокомпьютера для реализации большинства нейронных сетей одного из крупных отделов «зоопарка». Речь идет о сетях, связанных с методом обратного распространения ошибки - это мощная и широко применяемая технология обучения нейронных сетей. К сожалению, она получила распространение в виде алгоритма, а не в виде способа построения алгоритмов. Более общая теория обучения нейронных сетей - принцип двойственности [64, 250, 290, 283] - мало известна. На данный момент в литературе встречается описание более чем двух десятков различных алгоритмов обучения нейронных сетей по методу обратного распространения ошибки. Предлагаемый в этой работе проект стандарта ориентирован в первую очередь на сети, обучаемые по методу обратного распространения ошибки, но в приведенных примерах показана применимость этого стандарта и для других типов нейронных сетей – сетей ассоциативной памяти (Хопфилд) и сетей, обучающихся без учителя (Кохонен).

После тщательного анализа описания всех доступных из литературы алгоритмов обучения нейронных сетей, опираясь на принцип двойственности в обучении нейронных сетей и на свой двенадцатилетний опыт, нам удалось сформулировать принципы структурно-функциональной организации нейрокомпьютеров.

В данной работе предлагается два уровня стандартизации. Первый уровень состоит в создании единого языка описания функциональных компонент нейрокомпьютера. При этом не важно кем и для каких компьютеров был разработан программный имитатор. Возможность иметь внешнее, по отношению к программному имитатору, описание всех основных компонент нейрокомпьютера призвана облегчить разработку и распространение архитектур нейронных сетей, правил интерпретации ответов и их оценки, алгоритмов обучения, методов контрастирования (скелетонизации) и т.д. При этом результат становится не зависящим от программы, при помощи которой он был получен, и воспроизводимым другими исследователями.

Второй уровень предлагаемого проекта стандарта предусматривает возможность взаимозамены различных компонент в пределах одной программы. Предполагается, что возможно использование компонент одного разработчика программ совместно с компонентами, разработанными другими разработчиками. Этот стандарт по своему применению существенно уже первого, поскольку возможности переноса разработок между различными вычислительными платформами сильно ограничены.

Несколько слов о структуре книги. В первой главе выделяются основные компоненты нейрокомпьютера по следующим признакам.

1. Относительная функциональная обособленность: каждый компонент имеет четкий набор функций. Его взаимодействие с другими компонентами может быть описано в виде небольшого числа запросов.
2. Возможность реализации большинства используемых алгоритмов.
3. Возможность взаимозамены различных реализаций любого компонента без изменения других компонентов.

Во второй главе описаны стандарты типов данных и общий базис языка описания различных компонентов нейрокомпьютера. В ней также приведено описание запросов, исполняемых всеми или большинством компонент. Кроме того, в этой главе приведены способы работы с нестандартными типами данных, такими как «цвет» примера в обучающей выборке и др.

Каждая из остальных глав посвящена описанию одного или нескольких тесно связанных между собой компонентов нейрокомпьютера. Главы фактически самостоятельны. Если возникает необходимость привлечения материала других глав, то дается точная ссылка на раздел, в котором приводится нужный материал. Каждая из этих глав, состоит из трех частей. В первой части приводится обсуждение рассматриваемого компонента нейрокомпьютера, приводятся примеры. Во второй части главы описывается предлагаемый стандарт языка описания компоненты, а в третьей – описание запросов, исполняемых этим компонентом.

Благодарности. Идея написания этой книги родилась на основе двенадцатилетней работы Красноярской группы НейроКомп. Так выделение функциональных компонент явилось результатом разработки ряда нейросетевых программ Гилевым С.Е., Коченовым Д.А., Россиевым Д.А, и автором. Автор благодарен Гилеву С.Е, Дорреру М.Г., Коченову Д.А., Новоходько А.Ю., Россиеву Д.А., Сиротининой Н.Ю., Царегородцеву В.Г. и Чертыкову П.В. за неоднократные и очень полезные обсуждения предлагаемых в книге стандартов. Автор благодарен директору фирмы «АЗА» И.Г.Сулькису, директору Красноярского высшего колледжа информатики Г.М.Цибульскому и директору ИВМ СО РАН В.В.Шайдурову за неоценимую организационную поддержку. Особую благодарность автор выражает своему учителю, руководителю группы НейроКомп А.Н.Горбаню.

Работа над книгой была поддержана Красноярским краевым фондом науки (грант ???)

1. Функциональные компоненты

Эта глава посвящена выделению функциональных компонентов, составляющих универсальный нейрокомпьютер. Основные компоненты нейрокомпьютера выделяются по следующим признакам:

1. Относительная функциональная обособленность: каждый компонент имеет четкий набор функций. Его взаимодействие с другими компонентами может быть описано в виде небольшого числа запросов.
2. Возможность реализации большинства используемых алгоритмов.
3. Возможность взаимозамены различных реализаций любого компонента без изменения других компонентов.

Однако, прежде чем приступать к выделению компонент, опишем рассматриваемый набор нейронных сетей и процесс их обучения.

1.1 Краткий обзор нейронных сетей

Можно по разному описывать «зоопарк» нейронных сетей. Приведем классификацию нейронных сетей по решаемым ими задачам.

1. Классификация без учителя или поиск закономерностей в данных. Наиболее известным представителем этого класса сетей является сеть Кохонена, реализующая простейший вариант решения этой задачи. Наиболее общий вариант решения этой задачи известен как метод динамических ядер [223, 261].
2. Ассоциативная память. Наиболее известный представитель – сети Хопфилда. Эта задача также позволяет строить обобщения. Наиболее общий вариант описан в [77 – 79].
3. Аппроксимация функций, заданных в конечном числе точек. К сетям, решающим эту задачу, относятся персептроны, сети обратного распространения.

В центре нашего внимания будут сети, предназначенные для решения третьей задачи, однако предлагаемый вариант стандарта позволяет описать любую сеть. Конечно, невозможно использовать учитель, предназначенный для построения ассоциативной памяти, для решения задачи классификации без учителя и наоборот.

Среди сетей, аппроксимирующих функции, необходимо выделить еще два типа сетей – с дифференцируемой и пороговой характеристической функцией. Дифференцируемой будем называть сеть, каждый элемент которой реализует дифференцируемую функцию (точнее, непрерывно дифференцируемую). Вообще говоря, альтернативой дифференцируемой сети является недифференцируемая, а не пороговая, но на практике, как правило, все недифференцируемые сети являются пороговыми. Отметим, что для того, чтобы сеть была пороговой, достаточно вставить в нее один пороговый элемент.

Основное различие между дифференцируемыми и пороговыми сетями состоит в способе обучения. Для дифференцируемых сетей есть конструктивная процедура обучения, гарантирующая результат, если он достижим – метод двойственного обучения (обратного распространения ошибки). Для обучения пороговых сетей используют правило Хебба или его модификации. Однако, для многослойных сетей с пороговыми элементами правило Хебба не гарантирует обучения. (В случае однослойных сетей – персептронов, доказана теорема о достижении результата в случае его принципиальной достижимости). С другой стороны, в работе [145] доказано, что многослойные сети с пороговыми нейронами можно заменить эквивалентными однослойными.

1.2 Выделение компонент

Первым основным компонентом нейрокомпьютера является *нейронная сеть*. Относительно архитектуры сети принцип двойственности предполагает только одно – все элементы сети реализуют при прямом функционировании характеристические функции из класса $C^1(E)$ (непрерывно дифференцируемые на области определения E, которой, как правило, является вся числовая ось).

Для обучения нейронной сети необходимо наличие *задачника*. Однако чаще всего, обучение производится не по всему задачнику, а по некоторой его части. Ту часть задачника, по которой в данный момент производится обучение, будем называть обучающей выборкой. Для многих задач обучающая выборка имеет большие размеры (от нескольких сот до нескольких десятков тысяч примеров). При обучении с использованием скоростных методов обучения (их скорость на три-четыре порядка превышает скорость обучения по классическому методу обратного распространения ошибки) приходится быстро сменять примеры. Таким образом, скорость обработки обучающей выборки может существенно влиять на скорость обучения нейрокомпьютера. К сожалению, большинство разработчиков аппаратных средств

не предусматривает средств для быстрой смены примеров. Таким образом, задачник выделен в отдельный компонент нейрокомпьютера.

При работе с обучающей выборкой удобно использовать привычный для пользователя формат данных. Однако, этот формат чаще всего непригоден для использования нейросетью. Таким образом, между обучающей выборкой и нейросетью возникает дополнительный компонент нейрокомпьютера – *предобработчик*. Из литературных источников следует, что разработка эффективных предобработчиков для нейрокомпьютеров является новой, почти совсем не исследованной областью. Большинство разработчиков программного обеспечения для нейрокомпьютеров склонно возлагать функции предобработки входных данных на обучающую выборку или вообще перекладывают ее на пользователя. Это решение технологически неверно. Дело в том, что при постановке задачи для нейрокомпьютера трудно сразу угадать правильный способ предобработки. Для его подбора проводится серия экспериментов. В каждом из экспериментов используется одна и та же обучающая выборка и разные способы предобработки входных данных сети. Таким образом, выделен третий важный компонент нейрокомпьютера – предобработчик входных данных.

Заметим, что если привычный для человека способ представления входных данных непригоден для нейронной сети, то и формат ответов нейронной сети часто малопригоден для человека. Необходимо интерпретировать ответы нейронной сети. Интерпретация зависит от вида ответа. Так, если ответом нейронной сети является действительное число, то его, как правило, приходится масштабировать и сдвигать для попадания в нужный диапазон ответов. Если сеть используется как классификатор, то выбор интерпретаторов еще шире. Большое разнообразие интерпретаторов при невозможности решить раз и навсегда вопрос о преимуществах одного из них над другими приводит к необходимости выделения *интерпретатора ответа* нейронной сети в отдельный компонент нейрокомпьютера.

С интерпретатором ответа тесно связан еще один обязательный компонент нейрокомпьютера – *оценка*. Невнимание к этому компоненту вызвано практикой рассматривать метод обратного распространения ошибки в виде алгоритма. Доминирование такой точки зрения привело к тому, что, судя по публикациям, большинство исследователей даже не подозревает о том, что «уклонение от правильного ответа», подаваемое на вход сети при обратном функционировании, есть ни что иное, как производная функции оценки по выходному сигналу сети (если функция оценки является суммой квадратов уклонений). Возможно (и иногда очень полезно) конструировать другие оценки (см. главу «Оценка и интерпретатор ответа»). Нашей группой в ходе численных экспериментов было выяснено, что для обучения сетей-классификаторов функция оценки вида суммы квадратов, пожалуй, наиболее плоха. Использование альтернативных функций оценки позволяет в несколько раз ускорить обучение нейронной сети.

Шестым необходимым компонентом нейрокомпьютера является *учитель*. Этот компонент может меть множество реализаций. Обзор наиболее часто употребляемых и наиболее эффективных учителей приводится в главе «Учитель».

Принцип относительной функциональной обособленности требует выделения еще одного компонента, названного *исполнителем запросов учителя* или просто *исполнителем*. Назначение этого компонента не так очевидно, как всех предыдущих. Заметим, что для всех учителей, обучающих сети по методу обратного распространения ошибки, и при тестировании сети характерен следующий набор операций с каждым примером обучающей выборки:

1. Тестирование решения примера
   - 1.1. Взять пример у задачника.
   - 1.2. Предъявить его сети для решения.
   - 1.3. Предъявить результат интерпретатору ответа.
2. Оценивание решения примера
   - 2.1. Взять пример у задачника.
   - 2.2. Предъявить его сети для решения.
   - 2.3. Предъявить результат оценке.
3. Оценивание решения примера с вычислением градиента.
   - 3.1. Взять пример у задачника.
   - 3.2. Предъявить его сети для решения.
   - 3.3. Предъявить результат оценке с вычислением производных.
   - 3.4. Предъявить результат работы оценки сети для вычисления градиента.
4. Оценивание и тестирование решения примера.
   - 4.1. Взять пример у задачника.
   - 4.2. Предъявить его сети для решения.
   - 4.3. Предъявить результат оценке.
   - 4.4. Предъявить результат интерпретатору ответа.

Заметим, что все четыре варианта работы с сетью, задачником, интерпретатором ответа и оценкой легко объединить в один запрос, параметры которого позволяют указать последовательность действий. Таким образом, исполнитель исполняет всего один запрос – обработать пример. Однако выделение этого компонента позволяет исключить необходимость в прямых связях таких компонентов, как контрастер и учитель, с компонентами оценка и интерпретатор ответа, а их взаимодействие с компонентом сеть свести исключительно к запросам связанным с модификацией обучаемых параметров сети.

Последним компонентом, которого необходимо выделить, является *контрастер* нейронной сети. Этот компонент является надстройкой над учителем. Его назначение – сводить число связей сети до минимально необходимого или до «разумного» минимума (степень разумности минимума определяется пользователем). Кроме того, контрастер, как правило, позволяет свести множество величин весов связей к 2-4, реже к 8 выделенным пользователем значениям. Наиболее важным следствием применения процедуры контрастирования является получение логически прозрачных сетей – сетей, работу которых легко описать и понять на языке логики [75, 82].

Для координации работы всех компонент нейрокомпьютера вводится макрокомпонента *Нейрокомпьютер*. Основная задача этого компонента – организация интерфейса с пользователем и координация действий всех остальных компонентов.

1.3 Запросы компонентов нейрокомпьютера

В этом разделе приводится основной список запросов, которые обеспечивают функционирование нейрокомпьютера. За редким исключением приводятся только запросы, которые генерируются компонентами нейрокомпьютера (некоторые из этих запросов могут поступать в нейрокомпьютер от пользователя). Здесь рассматривается только форма запроса и его смысл. Полный список запросов каждого компонента, детали их исполнения и форматы данных рассматриваются в соответствующих главах, в разделах «Стандарт второго уровня компонента ... «.

На рис. 1. приведена схема запросов в нейрокомпьютере. При построении схемы предполагается, что на каждый запрос приходит ответ. Вид ответа описан при описании запросов. Стрелки, изображающие запросы, идут от объекта, инициирующего запрос, к объекту его исполняющему.

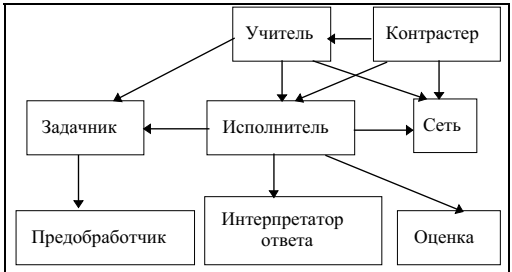


Рис 1. Схема запросов в нейрокомпьютере

1.3.1 Запросы к задачнику

Запросы к задачнику позволяют последовательно перебирать все примеры обучающей выборки, обращаться непосредственно к любому примеру задачника и изменять обучающую выборку. Обучающая выборка выделяется путем «раскрашивания» примеров задачника в различные «цвета». Понятие цвета и способ работы с цветами описаны в разделе «Переменные типа цвет и операции с цветами».

Запросы последовательного перебора обучающей выборки:

«Инициировать выдачу примеров цвета К». По этому запросу происходит инициация выдачи примеров К-го цвета.

«Дать очередной пример». По этому запросу задачник возвращает предобработанные данные очередного примера и, при необходимости, правильные ответы, уровень достоверности и другие данные этого примера.

«Следующий пример». По этому запросу задачник переходит к следующему примеру обучающей выборки. Если такого примера нет, то возвращается признак отсутствия очередного примера.

Для непосредственного доступа к примерам задачника служит запрос «Дать пример номер N». Действия задачника в этом случае аналогичны выполнению запроса «Дать очередной пример».

Для изменения обучающей выборки служит запрос «Окрасить примеры в цвет К». Этот запрос используется редко, поскольку изменение обучающей выборки, как правило, осуществляется пользователем при редактировании задачника.

1.3.2 Запрос к предобработчику

Предобработчик сам никаких запросов не генерирует. Единственный запрос к предобработчику – «Предобработать пример» может быть выдан только задачником.

1.3.3 Запрос к исполнителю

«Обработать очередной пример». Вид ответа зависит от параметров запроса.

1.3.4 Запросы к учителю

«Начать обучение сети». По этому запросу учитель начинает процесс обучения сети.

«Прервать обучение сети». Этот запрос приводит к прекращению процесса обучения сети. Этот запрос требуется в случае необходимости остановить обучение сети до того, как будет удовлетворен критерий остановки обучения, предусмотренный в учителе.

«Провести N шагов обучения» – как правило, выдается контрастером, необходим для накопления показателей чувствительности.

1.3.5 Запрос к контрастеру

«Отконтрастировать сеть». Ответом является код завершения операции контрастирования.

1.3.6 Запрос к оценке

Оценка не генерирует никаких запросов. Она выполняет только один запрос – «Оценить пример». Результатом выполнения запроса является оценка примера и, при необходимости, вектор производных оценки по выходным сигналам сети.

1.3.7 Запрос к интерпретатору ответа

Интерпретатор ответа не генерирует никаких запросов. Он выполняет только один запрос – «Интерпретировать ответ». Ответом является результат интерпретации.

1.3.8 Запросы к сети

Сеть не генерирует никаких запросов. Набор исполняемых сетью запросов можно разбить на три группы.

Запрос, обеспечивающий тестирование.

«Провести прямое функционирование». На вход сети подаются данные примера. На выходе сети вычисляется ответ сети, подлежащий оцениванию или интерпретации.

Запросы, обеспечивающие обучение сети.

«Обнулить градиент». При исполнении этого запроса градиент оценки по обучаемым параметрам сети кладется равным нулю. Этот запрос необходим, поскольку при вычислении градиента по очередному примеру сеть *добавляет* его к ранее вычисленному градиенту по сумме других примеров.

«Вычислить градиент по примеру». Проводится обратное функционирование сети. Вычисленный градиент *добавляется* к ранее вычисленному градиенту по сумме других примеров.

«Изменить карту с шагами H1 и H2». Генерируется учителем во время обучения.

Запрос, обеспечивающие контрастирование.

«Изменить карту по образцу». Генерируется контрастером при контрастировании сети.

Таким образом, выделено семь основных компонент нейрокомпьютера, определены их функции и основные исполняемые ими запросы.

2. Общий стандарт

Эта глава посвящена описанию элементов стандарта, общих для всех компонентов нейрокомпьютера.

2.1 Стандарт типов данных

При описании запросов, структур данных, стандартов компонентов нейрокомпьютера необходимо использовать набор первичных типов данных. Поскольку в разных языках программирования типы данных называются по-разному, введем единый набор обозначений для них.

Таблица 1.

Типы данных для всех компонентов нейрокомпьютера

Тип	Длина	Значения	Описание
Real	4 байта	от 1.5 e- 45 до 3.4 e 38	Действительное число. Величина из указанного диапазона. Знак произвольный. В дальнейшем называется «действительное».
Integer	2 байта	от -32768 до 32767	Целое число из указанного диапазона. В дальнейшем называется «целое».
Long	4 байта	от -2147483648 до 2147483647	Целое число из указанного диапазона. В дальнейшем называется «длинное целое».
RealArray	4*N байт	Массив	действительных чисел.
PRealArray	4 байта	Используется	для передачи массивов между компонентами. Имеет значение адреса массива действительных чисел.
IntegerArray	2*N байт	Массив	целых чисел.
PIntegerArray	4 байта	Используется	для передачи массивов между компонентами. Имеет значение адреса массива целых чисел.
LongArray	4*N байт	Массив	длинных целых чисел.
PLongArray	4 байта	Используется	для передачи массивов между компонентами. Имеет значение адреса массива длинных целых чисел.
Logic	1 байт	True, False	Логическая величина. Далее называется «логическая».
LogicArray	N байт	Массив	логических переменных.
PLogicArray	4 байта	Используется	для передачи массивов между компонентами. Имеет значение адреса массива логических переменных.
Color	2 байта	Используется	для задания цветов. Является совокупностью из 16 элементарных (битовых) флагов. См. раздел «Цвет и операции с цветами».
FuncType	4 байта	Адрес	функции. Используется при необходимости передать функцию в качестве аргумента.
String	256 байт	Строка	символов.
PString	4 байта	Адрес	строки символов. Служит для передачи строк в запросах
Visual	4 байта	Отображаемый	элемент. Служит для адресации отображаемых элементов в интерфейсных функциях. Тип значений зависит от реализации библиотеки интерфейсных функций и не может изменяться пользователем иначе, чем через вызов интерфейсной функции.
Pointer	4 байта	Не	типизованный указатель (адрес). Этот тип совместим с любым типизованным указателям.

Числовые типы данных Integer, Long и Real предназначены для хранения различных чисел. Переменные числовых типов допускаются в языках описания всех компонентов нейрокомпьютера. При необходимости записать в один массив числовые переменные различного типа следует использовать функции приведения типов, описанные в разделе «Приведение типов»

Строка. Символьный тип данных предназначен для хранения комментариев, названий полей, имен сетей, оценок и другой текстовой информации. Все строковые переменные занимают 256 байт и могут включать в себя до 255 символов. Первый байт строки содержит длину строки. В переменных типа

строка возможен доступ к любому символу как к элементу массива. При этом длина имеет индекс ноль, первый символ – 1 и т.д.

Указатель на строку. При передаче данных между компонентами сети и процедурами в пределах одного компонента удобно вместо строки передавать указатель на строку, поскольку указатель занимает всего четыре байта. Для этой цели служит тип указатель на строку.

Логический тип используется для хранения логических значений. Значение истина задается предопределенной константой True, значение ложь – False.

Массивы. В данном стандарте предусмотрены массивы четырех типов – логических, целочисленных, длинных целых и действительных переменных. Длины массивов определяются при описании, но все массивы переменных одного типа относятся к одному типу, в отличие от языков типа Паскаль. Использование функций приведения и преобразования типов позволяют получать из этих массивов структуры произвольной сложности. Элементы массивов всегда нумеруются с единицы.

Вне зависимости от типа массива нулевой элемент массива имеет тип Long и содержит длину массива в элементах. На рис. 1 приведена схема распределения памяти всех типов массивов, каждый из которых содержит шесть элементов.

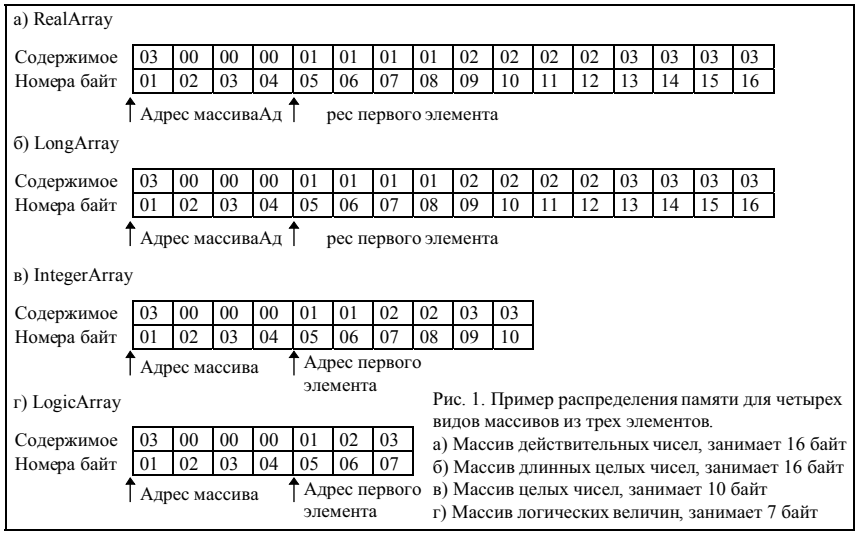


Рис. 1. Пример распределения памяти для четырех видов массивов из трех элементов.
а) Массив действительных чисел, занимает 16 байт
б) Массив длинных целых чисел, занимает 16 байт
в) Массив целых чисел, занимает 10 байт
г) Массив логических величин, занимает 7 байт

Все массивы, как правило, используется только в пределах одного компонента. При передаче массивов между компонентами или между процедурами в пределах одного компонента используются указатели на массивы.

Адрес функции. Этот тип используется для передачи функции в качестве аргумента. Переменная типа FuncType занимает четыре байта и является адресом функции. В зависимости от реализации по этому адресу может лежать либо начало машинного кода функции, либо начало текста функции. В случае передачи текста функции первые восемь байт по переданному адресу содержат слово «Function».

Отображаемый элемент. Переменные типа Visual (отображаемый элемент) Служат для адресации отображаемых элементов в интерфейсных функциях. Тип значений зависит от реализации библиотеки интерфейсных функций и не может изменяться пользователем иначе, чем через вызов интерфейсной функции. Особо следует отметить, что библиотека интерфейсных функций не является частью ни одного из компонентов.

2.2 Переменные типа цвет и операции с цветами

Использование цветов позволяет гибко разбивать множества на подмножества. В нейрокомпьютере возникает необходимость в разбиении на подмножества (раскрашивании) задачника. В этом разделе описывается стандарт работы с переменными типа цвет.

2.2.1 Значение переменной типа цвет (Color)

Переменная типа цвет представляет собой двухбайтовое беззнаковое целое. Однако основное использование предполагает работу не как с целым числом, а как с совокупностью однобитных флагов. При записи на диск используется символьное представление двоичной записи числа с ведущими нулями и разбиением на четверки символом «.» (точка), предваряемая заглавной буквой «B» латинского алфавита, или символьное представление шестнадцатеричной записи числа с ведущими нулями, предваряемая заглавной буквой «H» латинского алфавита. В таблице 2 приведена нумерация флагов (бит) переменной типа Color, их шестнадцатеричное, десятичное и двоичное значение. При использовании в учителе или других компонентах может возникнуть необходимость в присвоении некоторым из флагов или их комбинаций имен. На такое именование не накладывается никаких ограничений, хотя возможно будет выработан стандарт и на названия часто используемых цветов (масок, совокупностей флагов).

Таблица 2

Нумерация флагов (бит) переменной типа Color

Номер	Шестнадцатиричная запись	Десятичная запись	Двоичная запись
0	H0001	1	B.0000.0000.0000.0001
1	H0002	2	B.0000.0000.0000.0010
2	H0004	4	B.0000.0000.0000.0100
3	H0008	8	B.0000.0000.0000.1000
4	H0010	16	B.0000.0000.0001.0000
5	H0020	32	B.0000.0000.0010.0000
6	H0040	64	B.0000.0000.0100.0000
7	H0080	128	B.0000.0000.1000.0000
8	H0100	256	B.0000.0001.0000.0000
9	H0200	512	B.0000.0010.0000.0000
10	H0400	1024	B.0000.0100.0000.0000
11	H0800	2048	B.0000.1000.0000.0000
12	H1000	4096	B.0001.0000.0000.0000
13	H2000	8192	B.0010.0000.0000.0000
14	H4000	16384	B.0100.0000.0000.0000
15	H8000	32768	B.1000.0000.0000.0000

2.2.2 Операции с переменными типа цвет (Color)

В табл. 3 приведены операции с переменными типа Color. Первые пять операций могут использоваться только для сравнения переменных типа Color, а остальные четыре операции – для вычисления выражений типа Color.

Таблица 3

Предопределенные константы операций с переменными типа Цвет (Color)

Код	Обозначение	Вычисляемое выражение	Тип результата	Пояснение
1	CEqual	A = B	Logic	Полное совпадение.
2	CIn	A And B = A	Logic	A содержится в B.
3	CInclude	A And B = B	Logic	A содержит B.
4	CExclude	A And B = 0	Logic	A и B взаимоисключающие.
5	CIntersect	A And B <> 0	Logic	A и B пересекаются.
6	COr	A Or B	Color	Побитное включающее или.
7	CAnd	A And B	Color	Побитное и.
8	CXor	A Xor B	Color	Побитное исключающее или
9	CNot	Not A	Color	Побитное отрицание

В ряде запросов необходимо указать тип операции над цветом. Для передачи таких параметров используется переменная типа Integer. В качестве значений передается содержимое соответствующей ячейки столбца код табл. 3.

2.3 Приведение и преобразование типов

Есть два пути использовать переменную одного типа как переменную другого типа. Первый путь состоит в преобразовании значения к заданному типу. Так, для преобразования целочисленной переменной к действительному типу, достаточно просто присвоить переменной действительного типа целочисленное значение. С обратным преобразованием сложнее, поскольку не ясно что делать с дробной частью. В табл. 4 приведены все типы, которые можно преобразовать присваиванием переменной другого типа. В табл. 5 приведены все функции преобразования типов.

Таблица 4

Преобразование типов прямым присваиванием

Тип переменной, которой производится присваивание	Тип выражения, которое может быть присвоено	Пояснение
Real	Real, Integer, Long	Значение преобразуется к плавающему виду. При преобразовании значения выражения типа Long возможна потеря точности.
Long	Integer, Long	При преобразовании типа Integer, действуют следующие правила. Значение переменной помещается в два младших байта. Если значение выражения больше либо равно нолю, то старшие байты равны H0000, в противном случае старшие байты равны HFFFF.
Integer	Integer, Long	При преобразовании выражения типа Long значение двух старших байт отбрасывается.

При вычислении числовых выражений действуют следующие правила преобразования типов:

1. Выражения вычисляются слева на право.
2. Если два операнда имеют один тип, то результат имеет тот же тип.
3. Если аргументы имеют разные типы, то выражение имеет старший из двух типов. Список числовых типов по убыванию старшинства: Real, Long, Integer.
4. Результат операции деления действительных чисел (операция «/») всегда имеет тип Real, вне зависимости от типов аргументов.

В отличие от преобразования типов приведение типов позволяет по-разному интерпретировать одну область памяти. Функция приведения типа применима только к переменным или элементам массива (преобразование типов применимо и к выражениям). Рекомендуется использовать приведение типов только для типов, имеющих одинаковую длину. Например, Integer и Color или Real и Long. Список функций приведения типов приведен в табл. 6.

Таблица 5

Функции преобразования типов

Имя функции	Тип аргумента	Тип результата	Описание
Real	Real, Integer, Long	Real	Аналогично прямому присваиванию
Integer	Integer, Long	Integer	Аналогично прямому присваиванию
Long	Integer, Long	Long	Аналогично прямому присваиванию
Str	Real, Integer, Long	String	Представляет числовой аргумент в виде символьной строки в десятичном виде
Round	Real	Long	Округляет действительное значение до ближайшего длинного целого. Если значение действительного выражения выходит за диапазон длинного целого, то результат равен нулю.
Truncate	Real	Long	Преобразует действительное значение в длинное целое путем отбрасывания дробной части. Если значение действительного выражения выходит за диапазон длинного целого, то результат равен нулю.
LVal	String	Long	Преобразует длинное целое из символьного представления во внутреннее.
RVal	String	Real	Преобразует действительное число из символьного представления во внутреннее.
StrColor	Color	String	Преобразует внутреннее представление переменной типа Color в соответствии с разд. «Значение переменной типа цвет»
ValColor	String	Color	Преобразует символьное представление переменной типа Color во внутреннее.
Color	Integer	Color	Интерпретирует целое число как значение типа Color.

Следующие примеры иллюстрируют использование преобразования и приведения типов:

При вычислении следующих четырех выражений, получаются различные результаты

4096*4096=0

Таблица 6

Функции приведения типов

НазваниеТип	результата	Описание
TReal	Real	Четыре байта, адресуемые приводимой переменной, интерпретируются как действительное число.
TInteger	Integer	Два байта, адресуемые приводимой переменной, интерпретируются как целое число.
TLong	Long	Четыре байта, адресуемые приводимой переменной, интерпретируются как длинное целое.
TRealArray	RealArray	Область памяти, адресуемая приводимой переменной, интерпретируются как массив действительных чисел.
TPRealArray	PRealArray	Четыре байта, адресуемые приводимой переменной, интерпретируются как указатель на массив действительных чисел.
TIntegerArray	IntegerArray	Область памяти, адресуемая приводимой переменной, интерпретируются как массив целых чисел.
TPIntegerArray	PIntegerArray	Четыре байта, адресуемые приводимой переменной, интерпретируются как указатель на массив целых чисел.
TLongArray	LongArray	Область памяти, адресуемая приводимой переменной, интерпретируются как массив длинных целых.
TPLongArray	PLongArray	Четыре байта, адресуемые приводимой переменной, интерпретируются как указатель на массив длинных целых.
TLogic	Logic	Адресуемый приводимой переменной байт интерпретируются как логическая переменная.
TLogicArray	LogicArray	Область памяти, адресуемая приводимой переменной, интерпретируются как массив логических переменных.
TPLogicArray	LogicArray	Четыре байта, адресуемые приводимой переменной, интерпретируются как указатель на массив логических переменных.
TColor	Color	Два байта, адресуемые приводимой переменной, интерпретируются как переменная типа цвет.
TFuncType	FuncType	Четыре байта, адресуемые приводимой переменной, интерпретируются как адрес функции.
TString	String	256 бай области памяти, адресуемой приводимой переменной, интерпретируются как строка символов.
TPString	PString	Четыре байта, адресуемые приводимой переменной, интерпретируются как указатель на строку символов.
TVisual	Visual	Четыре байта, адресуемые приводимой переменной, интерпретируются как отображаемый элемент.
TPointer	Pointer	Четыре байта, адресуемые приводимой переменной, интерпретируются как адрес.

Поскольку константа 4096 имеет тип Integer, а 4096*4096=16777216=256*65536 , то есть младшие два байта результата равны нулю.

Long(4096*4096)=0

Поскольку оба сомножителя имеет тип Integer, то и выражение имеет тип Integer. Следовательно, результат умножения равен нулю, который затем преобразуется к типу Long.

Long(4096)*4096=16777216

Поскольку первый сомножитель имеет тип длинное целое, то и выражение имеет тип длинное целое.

4096.0*4096=1.677722E+7

Поскольку первый сомножитель имеет тип Real, то и выражение имеет тип Real. Из-за недостатка точности произошла потеря точности в седьмом знаке.

В следующем примере, используя приведение типов, в массив действительных чисел A размером в 66 элементов складываются: действительное число в первый элемент массива; длинное целое во второй элемент массива и символьную строку в элементы с 3 по 66.

A[1]= 1.677722E+7

TLong(A[2])= 16777216

TString(A[3])=‘Пример приведения типов’

Необходимо отметить, что элементы массива A, начиная со второго, после выполнения приведенного выше фрагмента программы не рекомендуется использовать как действительные числа, поскольку элемент A[2] содержит значение 2.350988E-38, а элемент A[5] – значение -4.577438E-18. Значение элементов, начиная с A[8] (символьная строка 'Пример приведения типов' содержит 23 символа и занимает 24 байта, то есть шесть элементов массива) вообще не зависят от приведенного фрагмента программы и содержат «мусор», который там находился ранее.

В списке типов определены только одномерные массивы. Однако, при необходимости, возможно использование двумерных массивов. Для этого в одномерный массив A необходимо поместить указатели на одномерные массивы. При этом I,J-й элемент двумерного массива записывается в виде:

TPRealArray(A[I])^[J]

В этом примере использована функция приведения типов TPRealArray, указывающая, что I-й элемент массива A нужно интерпретировать как указатель на одномерный массив действительных чисел, и операция «^» указывающая, что вместо указателя на массив TPRealArray(A[I]) используется массив, на который он указывает.

Таким образом, использование функций приведения типов позволяет из одномерных массивов строить структуры произвольной сложности. В языках программирования, таких как C и Паскаль, существует возможность строить пользовательские типы данных. При разработке стандарта эти возможности были исключены, поскольку использование пользовательских типов, облегчая написание программ, сильно затрудняет разработку компилятора или интерпретатора, а при использовании этого языка для описания компонентов нейрокомпьютера необходимость в пользовательских типах данных возникает чрезвычайно редко. Например, при описании примеров всех компонентов, приведенных в данной книге, такая необходимость ни разу не возникла.

2.4 Операции

В данном разделе приведены все операции, которые могут быть использованы при построении выражений различного типа. В табл. 7 приведены операции, которые допустимы в целочисленных выражениях (выражениях типа Integer или Long). В табл. 8 – список, дополняющий список операций из табл. 7 до полного списка операций, допустимых в выражениях действительного типа. В табл. 9 – операции, допустимые при построении логических выражений. В табл. 10 –для выражений типа символьная строка. В табл. 3 – для выражений типа Color.

Таблица 7

Операции, допустимые в целочисленных выражениях

Уровень приоритета	Обозначение	Тип 1-го операнда	Тип 2-го операнда	Тип результата	Название операции
1	*	Integer	Integer	Integer	Умножение
1	*	Long	Integer	Long	Умножение
1	*	Integer	Long	Long	Умножение
1	*	Long	Long	Long	Умножение
1	Div	Integer	Integer	Integer	Целочисленное деление
1	Div	Integer	Long	Long	Целочисленное деление
1	Div	Long	Integer	Long	Целочисленное деление
1	Div	Long	Long	Long	Целочисленное деление
1	Mod	Integer	Integer	Integer	Остаток от деления
1	Mod	Long	Integer	Long	Остаток от деления
1	Mod	Integer	Long	Long	Остаток от деления
1	Mod	Long	Long	Long	Остаток от деления
2	+	Integer	Integer	Integer	Сложение
2	+	Integer	Long	Long	Сложение
2	+	Long	Integer	Long	Сложение
2	+	Long	Long	Long	Сложение
2	–	Integer	Integer	Integer	Вычитание
2	–	Integer	Long	Long	Вычитание
2	–	Long	Integer	Long	Вычитание
2	–	Long	Long	Long	Вычитание
3	And	Integer	Integer	Integer	Побитное И
3	And	Long	Long	Long	Побитное И
3	Or	Integer	Integer	Integer	Побитное включающее ИЛИ

Таблица 7

Операции, допустимые в целочисленных выражениях (продолжение)

Уровень приоритета	Обозначение	Тип 1-го операнда	Тип 2-го операнда	Тип результата	Название операции
3	Or	Long	Long	Long	Побитное включающее ИЛИ
3	Xor	Integer	Integer	Integer	Побитное исключающее ИЛИ
3	Xor	Long	Long	Long	Побитное исключающее ИЛИ
3	Not	Integer	Integer	Integer	Побитное отрицание
3	Not	Long	Long	Long	Побитное отрицание

Таблица 8

Операции, дополняющие список операций из табл. 7 до полного списка операций, допустимых в выражениях действительного типа.

Уровень приоритета	Обозначение	Тип 1-го операнда	Тип 2-го операнда	Тип результата	Название операции
1	*	Real	Integer, Real, Long	Real	Умножение
1	/	Integer, Real, Long	Integer, Real, Long	Real	Деление
1	RMod	Integer, Real, Long	Integer, Real, Long	Real	Остаток от деления
2	+	Real	Integer, Real, Long	Real	Сложение
2	–	Real	Integer, Real, Long	Real	Вычитание

Таблица 9

Операции, допустимые при построении логических выражений

Уровень приорит.	Обозначение	Тип 1-го операнда	Тип 2-го операнда	Тип результата	Название операции
1	>	Integer, Real, Long	Integer, Real, Long	Logic	Больше
1	<	Integer, Real, Long	Integer, Real, Long	Logic	Меньше
1	>=	Integer, Real, Long	Integer, Real, Long	Logic	Больше или равно
1	<=	Integer, Real, Long	Integer, Real, Long	Logic	Меньше или равно
1	=	Integer, Real, Long	Integer, Real, Long	Logic	Равно
1	<>	Integer, Real, Long	Integer, Real, Long	Logic	Не равно
2	And	Logic	Logic	Logic	Логическое И
2	Or	Logic	Logic	Logic	Логическое включающее ИЛИ
2	Xor	Logic	Logic	Logic	Логическое исключающее ИЛИ
2	Not	Logic	Logic	Logic	Логическое отрицание

Таблица 10

Операции для выражений типа символьная строка

Уровень приоритета	Обозначение	Тип 1-го операнда	Тип 2-го операнда	Тип результата	Название операции
1	+	String	String	String	Конкатенация (сцепка) строк.

Во всех таблицах операции размещаются по убыванию приоритета. Для каждой операции указаны допустимые типы операндов, и тип результата, в зависимости от типов операндов.

В табл. 8 приводится необычная операция RMod – остаток от деления действительных чисел. Результат этой функции равен разности между первым операндом и вторым операндом, умноженным на целую часть отношения первого операнда ко второму.

Кроме операций, приведенных в табл. 3 и табл. 7–10, определены две взаимно обратные операции для работы с адресами и указателями:

^ – ставится после переменной типа указатель. Означает, что вместо указателя в выражении используется переменная или массив, на который указывает этот указатель. Не допускается после переменных типа Pointer.

@ – ставится перед именем переменной любого типа. Означает, что в выражении участвует не переменная, а адрес переменной. Используется при присвоении адресов переменных или массивов переменным типа указатель.

2.5 Предопределенные константы

При описании различных компонентов возникает необходимость в использовании некоторого набора стандартизированных констант. Стандартность набора констант особенно необходима при обмене между компонентами. Все константы, приведенные в табл. 11, описываются в тех разделах, где они используются. В табл. 11 для каждой константы указывается ее тип, значение и названия разделов, в которых она описывается.

Таблица 11

Предопределенные константы

Идентификатор	Тип	Значение		Раздел
		Шестнад.	Десятич.	
BackInSignals	Integer	H0005	5	Запросы к компоненту сеть
BackOutSignals	Integer	H0006	6	Запросы к компоненту сеть
BackParameters	Integer	H0007	7	Запросы к компоненту сеть
Binary	Integer	H0001	1	Запросы компонента интерпретатор ответа
BinaryPrep	Integer	H0000	0	Запросы компонента предобработчик
BynaryCoded	Integer	H0003	3	Запросы компонента интерпретатор ответа
CAnd	Integer	H0007	7	Операции с переменными типа цвет (Color)
Cascad	Integer	H0002	2	Запросы к компоненту сеть
CEqual	Integer	H0001	1	Операции с переменными типа цвет (Color)
CExclude	Integer	H0004	4	Операции с переменными типа цвет (Color)
CicleFor	Integer	H0003	3	Запросы к компоненту сеть
CicleUntil	Integer	H0004	4	Запросы к компоненту сеть
CIn	Integer	H0002	2	Операции с переменными типа цвет (Color)
CInclude	Integer	H0003	3	Операции с переменными типа цвет (Color)
CIntersect	Integer	H0005	5	Операции с переменными типа цвет (Color)
CNot	Integer	H0009	9	Операции с переменными типа цвет (Color)
COr	Integer	H0006	6	Операции с переменными типа цвет (Color)
CXor	Integer	H0008	8	Операции с переменными типа цвет (Color)
Element	Integer	H0000	0	Запросы к компоненту сеть
Empty	Integer	H0000	0	Запросы компонента интерпретатор ответа
EmptyPrep	Integer	H0003	3	Запросы компонента предобработчик
False	Logic	H00		
FuncPrep	Integer	H0005	5	Запросы компонента предобработчик
InSignalMask	Integer	H0003	3	Запросы к компоненту сеть
InSignals	Integer	H0000	0	Запросы к компоненту сеть
Layer	Integer	H0001	1	Запросы к компоненту сеть
MainVisual	Visible			Интерфейсные функции
Major	Integer	H0002	2	Запросы компонента интерпретатор ответа
mIntegerArray	Integer	H0002	2	Функции управления памятью
mLogicArray	Integer	H0001	1	Функции управления памятью
mLongArray	Integer	H0004	4	Функции управления памятью
ModPrep	Integer	H0004	4	Запросы компонента предобработчик
mRealArray	Integer	H0004	4	Функции управления памятью
Null	Pointer	H00000000	нет	
Ordered	Integer	H0002	2	Запросы компонента предобработчик
OutSignals	Integer	H0001	1	Запросы к компоненту сеть
Parameters	Integer	H0002	2	Запросы к компоненту сеть
ParamMask	Integer	H0004	4	Запросы к компоненту сеть
PositPrep	Integer	H0006	6	Запросы компонента предобработчик
tbAnswers	Integer	H0004	4	Язык описания задачника
tbCalcAnswers	Integer	H0006	6	Язык описания задачника
tbCalcReliability	Integer	H0007	7	Язык описания задачника
tbColor	Integer	H0001	1	Язык описания задачника
tbComment	Integer	H000A	10	Язык описания задачника
tbEstimation	Integer	H0009	9	Язык описания задачника

Таблица 11

Предопределенные константы (продолжение)

Идентификатор	Тип	Значение	Раздел	
		Шестнад.	Десятич.	
tbInput	Integer	H0002	2	Язык описания задачника
tbPrepared	Integer	H0003	3	Язык описания задачника
tbReliability	Integer	H0005	5	Язык описания задачника
tbWeight	Integer	H0008	8	Язык описания задачника
True	Logic	HFF	255 (-1)	
UnknownLong	Integer	H0000	0	Неопределенные значения
UnknownReal	Real	нет	$1E^{-40}$	Неопределенные значения
UnOrdered	Integer	H0001	1	Запросы компонента предобработчик
UserType	Integer	HFFFF	-1	Структурная единица, определенная пользователем.

Три предопределенные константы, приведенные в табл.11, не описываются ни в одном разделе данной книги. Это константы общего пользования. Их значение:

True – значение истина для присваивания переменным логического типа.

False – значение ложь для присваивания переменным логического типа.

Null – пустой указатель. Используется для сравнения или присваивания переменных всех типов указателей.

2.6 Интерфейсные функции

Часто при обучении и тестировании нейронных сетей возникает необходимость отображать на экран некоторую информацию. Например, число предъявлений примеров сети, максимальную оценку и т.п. Вряд ли разумно использовать язык описания компонентов нейрокомпьютера для описания интерфейса. Это противоречит требованию переносимости текста описания компонентов между разными платформами и операционными системами. Для облегчения создания интерфейса, с одной стороны, и обеспечения универсальности описания компонентов нейрокомпьютера, с другой, предложен набор интерфейсных функций. Поддержку этих функций несложно организовать в любой операционной среде и на любой платформе.

В основу набора интерфейсных функций положен принцип объектно-ориентированной машины потока событий. Примером таких систем может служить инструментальная библиотека объектов (классов) Turbo Vision фирмы Борланд, или оконный интерфейс Windows фирмы Майкрософт. Предлагаемый в данном разделе набор интерфейсных функций беднее любой из выше названных интерфейсных библиотек, но позволяет организовать достаточно красивый и удобный интерфейс.

2.6.1 Структура данных интерфейсных функций

Элементом данных в структуре интерфейса является отображаемый элемент. Каждый отображаемый элемент имеет свои координаты относительно начала владельца – отображаемого элемента, содержащего данный. Владельцем отображаемого элемента может являться одно из окон или диалогов или главный элемент. Главный отображаемый элемент является предопределенным и обозначается переменной MainVisual. В программах компонентов нейрокомпьютера не допускается изменение значения переменной MainVisual. Эту переменную стоит рассматривать как константу, однако, в отличие от всех остальных констант ее значение определяется не данным стандартом, а разработчиком библиотеки интерфейсных функций.

В данной работе не рассматривается способ реализации интерфейсных функций и не обсуждаются значения переменных типа Visible. По этому ставится единственное условие – переменные типа Visible могут изменять свои значения только при вызове интерфейсных функций или при присваивании им значений другой переменной того же типа.

Все отображаемые элементы создаются интерфейсными функциями, называющимися, так же как и сам отображаемый элемент. Интерфейсные функции создающие отображаемые элементы возвращают значения типа Visible. Если при вызове создающей отображаемый элемент функции элемент не был создан, то функция возвращает значение Null.

2.6.2 Соглашение о передаче значений отображаемым элементам

Отображаемые элементы могут быть связаны с обычными переменными. В следующих разделах описаны отображаемые элементы, для которых должна быть установлена такая связь. При изменении значения связанной переменной программным путем для отображения этого изменения должна быть вызвана функция Refresh, с переменной типа Visible данного элемента в качестве параметра. При связы-

вании переменной и отображаемого элемента необходимо совпадение типа переменной с связываемым переменной этого отображаемого элемента.

2.6.3 Перечень отображаемых элементов

Название элемента: Window (окно).

Параметры при создании:

BeginX, BeginY – Координаты верхнего левого угла окна относительно владельца.

SizeX, SizeY – Горизонтальный и вертикальный размеры окна.

ScrollX, ScrollY – Целочисленные параметры, задающие наличие у окна горизонтальной и вертикальной полосы прокрутки. Если значение параметра равно нулю, то соответствующая полоса прокрутки отсутствует, при любом другом значении параметра в окно включается соответствующая полоса прокрутки.

Text – Название окна.

Описание элемента. Элемент Window может являться владельцем любых других отображаемых элементов, кроме элементов типа Dialog. При вызове функции Refresh, с элементом типа Window в качестве параметра, обновляется изображение не только самого окна, но и всех отображаемых элементов, для которых это окно является владельцем. Для отображения созданного окна на экране необходимо вставить его (вызвать функцию Insert, с данным окном в качестве второго параметра.) в отображенное на экран окно, диалог или в отображаемый элемент MainVisible. Для того чтобы убрать окно с экрана, необходимо вызвать функцию Delete, с данным окном в качестве параметра. Для уничтожения окна необходимо вызвать функцию Erase, с данным окном в качестве параметра. При этом уничтожаются так же и все отображаемые элементы, для которых данное окно являлось владельцем.

Название элемента: Dialog (Диалог).

Параметры при создании:

BeginX, BeginY – Координаты верхнего левого угла окна диалога относительно владельца.

SizeX, SizeY – Горизонтальный и вертикальный размеры окна диалога.

ScrollX, ScrollY – Целочисленные параметры, задающие наличие у окна диалога горизонтальной и вертикальной полосы прокрутки. Если значение параметра равно нулю, то соответствующая полоса прокрутки отсутствует, при любом другом значении параметра в окно диалога включается соответствующая полоса прокрутки.

Text – Название окна диалога.

Описание элемента. Элемент Dialog может являться владельцем любых других отображаемых элементов. Этот элемент является модальным, то есть во время работы диалога невозможен вызов меню, переход в другие окна и диалоги и т.д. Если одновременно активно несколько диалогов, то модальным является последний по порядку отображения на экране. При закрытии текущего диалога модальность переходит к предыдущему и т.д. При вызове функции Refresh, с элементом типа Dialog в качестве параметра, обновляется изображение не только самого окна диалога, но и всех отображаемых элементов, для которых этот диалог является владельцем. Для отображения созданного диалога на экране необходимо вставить его (вызвать функцию Insert, с данным диалогом в качестве второго параметра.) в отображаемый элемент MainVisible. С этого момента диалог становится модальным. Он остается модальным либо до вставки в MainVisible другого диалога, либо до закрытия диалога. Если модальность потеряна диалогом из-за запуска следующего диалога, то при закрытии последнего диалога статус модальности восстанавливается. Для того чтобы закрыть диалог, необходимо вызвать функцию Delete, с данным диалогом в качестве параметра. Для уничтожения диалога необходимо вызвать функцию Erase, с данным диалогом в качестве параметра. При этом уничтожаются так же и все отображаемые элементы, для которых данный диалог являлся владельцем.

Название элемента: Label (метка).

Параметры при создании:

BeginX, BeginY – Координаты верхнего левого угла метки относительно владельца.

SizeX, SizeY – Горизонтальный и вертикальный размеры метки.

Text – текст метки.

Описание элемента. Элемент Label не может являться владельцем других отображаемых элементов. Этот элемент не связан с переменными. Как правило, он используется для организации отображения в окне или диалоге статической информации или в качестве подписи поля ввода. Если метка связана с полем ввода, то передача управления этой метке автоматически влечет за собой передачу управления связанному с ней полю. Для организации связи метки с полем необходимо вызвать функцию Link, передав ей в качестве первого параметра метку, а вторым параметром тот элемент, с которым необходи-

мо установить связь. Связь может быть установлена только с одним элементом. При повторном вызове функции Link устанавливается связь с новым элементом, а связь с прежним элементом разрывается. Для включения метки в окно или диалог, необходимо вызвать функцию Insert, с окном или диалогом в качестве первого параметра и меткой в качестве второго параметра. Для уничтожения метки необходимо вызвать функцию Erase, с данной меткой в качестве параметра.

Название элемента: StringVisible (строковый элемент).

Параметры при создании:

BeginX, BeginY – Координаты верхнего левого угла элемента относительно владельца.

SizeX, SizeY – Горизонтальный и вертикальный размеры элемента.

Size – размер поля.

Описание элемента. Элемент StringVisible не может являться владельцем других отображаемых элементов. Этот элемент должен быть связан с переменной типа String. Для установления связи используется функция Data со строковым элементом в качестве первого параметра и адресом переменной в качестве второго параметра. Как правило, строковый элемент связывают с меткой. Для организации связи метки со строковым элементом необходимо вызвать функцию Link, передав ей в качестве первого параметра метку, а вторым параметром строковый элемент. Для включения строкового элемента в окно или диалог, необходимо вызвать функцию Insert, с окном или диалогом в качестве первого параметра и строковым элементом в качестве второго параметра. Для уничтожения строки необходимо вызвать функцию Erase, с данным строковым элементом в качестве параметра. Параметр Size задает максимальный размер вводимой строки в символах.

Название элемента: RealVisible (LongVisible) (числовой элемент).

Параметры при создании:

BeginX, BeginY – Координаты верхнего левого угла элемента относительно владельца.

SizeX, SizeY – Горизонтальный и вертикальный размеры элемента.

Min, Max – минимальное и максимальное допустимые значения.

Size – размер поля.

Описание элемента. Элементы RealVisible и LongVisible служат для ввода действительных и длинных целых чисел, соответственно. Любой такой элемент должен быть связан с переменной соответствующего типа (Real или Long) и не может являться владельцем других отображаемых элементов. Для установления связи используется функция Data с числовым элементом в качестве первого параметра и адресом переменной в качестве второго параметра. Как правило, числовой элемент связывают с меткой. Для организации связи метки с числовым элементом необходимо вызвать функцию Link, передав ей в качестве первого параметра метку, а вторым параметром числовой элемент. Для включения числового элемента в окно или диалог, необходимо вызвать функцию Insert, с окном или диалогом в качестве первого параметра и числовым элементом в качестве второго параметра. Для уничтожения числового элемента необходимо вызвать функцию Erase, с данным числовым элементом в качестве параметра.

Название элемента: RadioButtons (переключатели).

Параметры при создании:

BeginX, BeginY – Координаты верхнего левого угла элемента относительно владельца.

SizeX, SizeY – Горизонтальный и вертикальный размеры элемента.

Описание элемента. Элемент RadioButtons служит для задания значения параметрам, которые могут принимать только несколько значений (например, два значения – истина или ложь). Процедура создания элемента RadioButtons сложнее, чем для ранее рассмотренных процедур. Кроме вызова функции RadioButtons, создающей элемент, необходимо несколько раз вызвать функцию AddItem, с элементом RadioButtons в качестве первого аргумента и подписью переключателя в качестве второго аргумента. Элемент RadioButtons должен быть связан с переменной типа Long и не может являться владельцем других отображаемых элементов. Для установления связи используется функция Data с элементом RadioButtons в качестве первого параметра и адресом переменной в качестве второго параметра. Элемент RadioButtons интерпретирует данные, содержащиеся в переменной, следующим образом: первому флагу соответствует младший бит переменной, второму следующий по старшинству и т. д. Элемент не может включать более 32 флагов. Биты с номерами большими числа флагов очищаются (заменяются нулями). Как правило, элемент RadioButtons связывают с меткой. Для организации связи метки с элементом RadioButtons необходимо вызвать функцию Link, передав ей в качестве первого параметра метку, а вторым параметром элемент RadioButtons. Для включения элемента RadioButtons в окно или диалог, необходимо вызвать функцию Insert, с окном или диалогом в качестве первого параметра и элементом

RadioButtons в качестве второго параметра. Для уничтожения элемента RadioButtons необходимо вызвать функцию Erase, с данным элементом RadioButtons в качестве параметра.

Название элемента: CheckBoxes (группа флагов).

Параметры при создании:

BeginX, BeginY – Координаты верхнего левого угла элемента относительно владельца.

SizeX, SizeY – Горизонтальный и вертикальный размеры элемента.

Описание элемента. Элемент CheckBoxes служит для задания значения параметрам, которые являются совокупностью битовых флагов. Процедура создания группы флагов аналогична созданию элемента RadioButtons. Кроме вызова функции CheckBoxes, создающей элемент, необходимо несколько раз вызвать функцию AddItem, с элементом CheckBoxes в качестве первого аргумента и названием флага в качестве второго аргумента. Элемент CheckBoxes должен быть связан с переменной типа Long и не может являться владельцем других отображаемых элементов. Для установления связи используется функция Data с элементом CheckBoxes в качестве первого параметра и адресом переменной в качестве второго параметра. Элемент CheckBoxes интерпретирует данные, содержащиеся в переменной, следующим образом: если значение переменной равно единице, то включен первый переключатель, если двум – то второй, трем – первые два и т. д. Если значение переменной меньше либо равно нулю или больше либо равно два в степени числа переключателей, то оно заменяется на единицу. Как правило, элемент CheckBoxes связывают с меткой. Для организации связи метки с элементом CheckBoxes необходимо вызвать функцию Link, передав ей в качестве первого параметра метку, а вторым параметром элемент CheckBoxes. Для включения группы флагов в окно или диалог, необходимо вызвать функцию Insert, с окном или диалогом в качестве первого параметра и элементом CheckBoxes в качестве второго параметра. Для уничтожения элемента CheckBoxes необходимо вызвать функцию Erase, с данным элементом CheckBoxes в качестве параметра.

Название элемента: Button(кнопка).

Параметры при создании:

BeginX, BeginY – Координаты верхнего левого угла элемента относительно владельца.

SizeX, SizeY – Горизонтальный и вертикальный размеры элемента.

Macro – Адрес функции, вызываемой при нажатии кнопки. В зависимости от реализации по этому адресу может лежать либо начало машинного кода функции, либо начало текста функции. В случае передачи текста функции первые восемь байт по переданному адресу содержат слово «Function».

Описание элемента. Кнопка служит для запуска макроса, который выполняет некоторые действия, являющиеся реакцией на нажатие кнопки. Кнопка не может быть связана с переменными и не может являться владельцем других отображаемых элементов. Для включения кнопки элемента в окно или диалог, необходимо вызвать функцию Insert, с окном или диалогом в качестве первого параметра и кнопкой в качестве второго параметра. Для уничтожения кнопки необходимо вызвать функцию Erase, с данной кнопкой в качестве параметра.

2.6.4 Перечень интерфейсных функций

В данном разделе дано описание всех интерфейсных функций. Приводится синтаксис описания на общем подмножестве языков описания компонентов нейрокомпьютера. Функции приведены в алфавитном порядке. Следует отметить, что, как и в языках описания всех компонентов нейрокомпьютера все аргументы передаются функциям по ссылке (передается не значение аргумента, а его адрес).

AddItem

Function AddItem(Elem : Visible; Text : String) : Logic;

Описание аргументов:

Elem – Отображаемый элемент типа CheckBoxes или RadioButtons.

Text – Название переключателя или флага.

Описание функции:

Эта функция добавляет название переключателя (если первый аргумент типа RadioButtons) или флага (CheckBoxes) к списку элемента, передаваемого функции первым аргументом. Если первый элемент не является элементом типа CheckBoxes или RadioButtons, то функция возвращает значение ложь (False). В случае успешного завершения операции добавления в список функция возвращает значение истина (True). В противном случае возвращается значение ложь (False).

Button

Function Button(BeginX, BeginY, SizeX, SizeY : Long; Macro : PString) : Visible;

Описание аргументов:

BeginX, BeginY – Координаты верхнего левого угла элемента относительно владельца.

SizeX, SizeY – Горизонтальный и вертикальный размеры элемента.

Macro – Адрес функции, вызываемой при нажатии кнопки.

Описание функции:

Эта функция создает отображаемый элемент типа Button. Если создание прошло успешно, то возвращается значение этого элемента (типы значений не оговариваются стандартом, но, как правило, это адрес соответствующей структуры). Если создание элемента завершилось не удачно, то возвращается значение Null.

CheckBoxes

Function CheckBoxes(BeginX, BeginY, SizeX, SizeY : Long) : Visible;

Описание аргументов:

BeginX, BeginY – Координаты верхнего левого угла элемента относительно владельца.

SizeX, SizeY – Горизонтальный и вертикальный размеры элемента.

Описание функции:

Эта функция создает отображаемый элемент типа CheckBoxes с пустым списком переключателей. Для добавления переключателей следует воспользоваться функцией AddItem. Если создание прошло успешно, то возвращается значение этого элемента (типы значений не оговариваются стандартом, но, как правило, это адрес соответствующей структуры). Если создание элемента завершилось не удачно, то возвращается значение Null.

Data

Function Data(Element : Visible; Var Datum) : Logic;

Описание аргументов:

Element – Отображаемый элемент, который связывается с переменной.

Datum – Адрес переменной.

Описание функции:

Эта функция связывает отображаемый элемент (Element) с перемнной Datum. Если элемент Element не допускает установления связи с переменной, то функция возвращает значение ложь (False). В противном случае она устанавливает связь между элементом и переменной и возвращает значение истина (True). Отметим, что функция не проверяет типа переменной. Если вместо адреса переменной типа длинное целое был дан адрес переменной действительного типа, то эта переменная будет интерпретироваться как длинное целое (см. разд. «Функции приведения типов»). Важно отметить, что производится приведение переменной, а не преобразование ее значения.

Delete

Function Delete(Owner, Element : Visible) : Logic;

Описание аргументов:

Owner – Отображаемый элемент типа окно или диалог, из которого происходит удаление.

Element – Удаляемый элемент.

Описание функции:

Эта функция удаляет отображаемый элемент (Element) из его владельца (Owner). Если элемент Owner не является окном или диалогом, или если он не является владельцем элемента Element, то функция возвращает значение ложь (False). В противном случае она удаляет элемент из владельца и возвращает значение истина (True). Отметим, что элемент удаляется, но не уничтожается. Если нет переменной, содержащей удаляемый элемент, то элемент «потеряется», то есть он станет недоступным из программы, но будет занимать память.

Dialog

Function Dialog(BeginX, BeginY, SizeX, SizeY, ScrollX, ScrollY : Long; Text : String) : Visible;

Описание аргументов:

BeginX, BeginY – Координаты верхнего левого угла элемента относительно владельца.

SizeX, SizeY – Горизонтальный и вертикальный размеры элемента.

ScrollX, ScrollY – Целочисленные параметры, задающие наличие у окна горизонтальной и вертикальной полосы прокрутки. Если значение параметра равно нулю, то соответствующая полоса

прокрутки отсутствует, при любом другом значении параметра в окно включается соответствующая полоса прокрутки.

Text – Название окна.

Описание функции:

Эта функция создает отображаемый элемент типа диалог. Если создание прошло успешно, то возвращается значение этого элемента (типы значений не оговариваются стандартом, но, как правило, это адрес соответствующей структуры). Если создание элемента завершилось не удачно, то возвращается значение Null. После создания диалог является пустым.

Erase

Function Erase(Element : Visible) : Logic;

Описание аргументов:

Element – Уничтожаемый элемент.

Описание функции:

Эта функция уничтожает отображаемый элемент (Element). Если аргумент Element является окном или диалогом, то уничтожаются так же все отображаемые элементы, для которых элемент Element является владельцем. Если операция завершена успешно, то функция возвращает значение истина (True). В противном случае – значение ложь (False). Если выполнение функции завершилось неуспешно (функция вернула значение ложь), то элемент может быть поврежден и его дальнейшее использование не гарантирует корректной работы.

Insert

Function Insert(Owner, Element : Visible) : Logic;

Описание аргументов:

Owner – Отображаемый элемент типа окно или диалог, в который производится вставка.

Element – Вставляемый элемент.

Описание функции:

Эта функция вставляет отображаемый элемент (Element) в элемент (Owner). Если элемент Owner не является окном или диалогом, или если Element является диалогом, то функция возвращает значение ложь (False). Такие же действия производятся, в случае, если аргумент Owner совпадает с MainVisible, а Element не является окном или диалогом. В противном случае она вставляет элемент в Owner и возвращает значение истина (True). Вставка окна или диалога в MainVisible вызывает отображение его на экране, а в случае, если вставляется диалог, то ему передается управление.

Label

Function Label(BeginX, BeginY, SizeX, SizeY : Long; Text : String) : Visible;

Описание аргументов:

BeginX, BeginY – Координаты верхнего левого угла элемента относительно владельца.

SizeX, SizeY – Горизонтальный и вертикальный размеры элемента.

Text – текст метки.

Описание функции:

Эта функция создает отображаемый элемент типа Label. Если создание прошло успешно, то возвращается значение этого элемента (типы значений не оговариваются стандартом, но, как правило, это адрес соответствующей структуры). Если создание элемента завершилось не удачно, то возвращается значение Null.

Link

Function Link(Element, Labels : Visible) : Logic;

Описание аргументов:

Owner – Отображаемый элемент, связываемый с меткой.

Element – Отображаемый элемент – метка.

Описание функции:

Эта функция устанавливает связь между меткой Labels и отображаемым элементом Element. Если элемент Labels не является меткой, то функция возвращает значение ложь (False). В противном случае она устанавливает связь и возвращает значение истина (True).

LongVisible

Function LongVisible(BeginX, BeginY, SizeX, SizeY, Min, Max, Size: Long) : Visible;

Описание аргументов:

BeginX, BeginY – Координаты верхнего левого угла элемента относительно владельца.
SizeX, SizeY – Горизонтальный и вертикальный размеры элемента.
Min, Max – минимальное и максимальное допустимые значения.
Size – размер поля в символах.

Описание функции:

Эта функция создает отображаемый элемент типа LongVisible для редактирования и ввода значений типа Long. Если создание прошло успешно, то возвращается значение этого элемента (типы значений не оговариваются стандартом, но, как правило, это адрес соответствующей структуры). Если создание элемента завершилось не удачно, то возвращается значение Null.

RadioButtons

Function RadioButtons(BeginX, BeginY, SizeX, SizeY : Long) : Visible;

Описание аргументов:

BeginX, BeginY – Координаты верхнего левого угла элемента относительно владельца.
SizeX, SizeY – Горизонтальный и вертикальный размеры элемента.

Описание функции:

Эта функция создает отображаемый элемент типа RadioButtons с пустым списком флагов. Для добавления переключателей следует воспользоваться функцией AddItem. Если создание прошло успешно, то возвращается значение этого элемента (типы значений не оговариваются стандартом, но, как правило, это адрес соответствующей структуры). Если создание элемента завершилось не удачно, то возвращается значение Null.

RealVisible

Function RealVisible (BeginX, BeginY, SizeX, SizeY : Long; Min, Max : Real; Size: Long) : Visible;

Описание аргументов:

BeginX, BeginY – Координаты верхнего левого угла элемента относительно владельца.
SizeX, SizeY – Горизонтальный и вертикальный размеры элемента.
Min, Max – минимальное и максимальное допустимые значения.
Size – размер поля в символах.

Описание функции:

Эта функция создает отображаемый элемент типа RealVisible для редактирования и ввода значений типа Real. Если создание прошло успешно, то возвращается значение этого элемента (типы значений не оговариваются стандартом, но, как правило, это адрес соответствующей структуры). Если создание элемента завершилось не удачно, то возвращается значение Null.

Refresh

Function Refresh(Element : Visible) : Logic;

Описание аргументов:

Element – Отображаемый элемент.

Описание функции:

Эта функция обновляет изображение элемента Element на экране. Если операция прошла успешно, то функция возвращает значение истина (True). В противном случае она возвращает значение ложь (False).

StringVisible

Function StringVisible (BeginX, BeginY, SizeX, SizeY, Size: Long) : Visible;

Описание аргументов:

BeginX, BeginY – Координаты верхнего левого угла элемента относительно владельца.
SizeX, SizeY – Горизонтальный и вертикальный размеры элемента.
Size – размер поля в символах.

Описание функции:

Эта функция создает отображаемый элемент типа StringVisible для редактирования и ввода символьных строк. Если создание прошло успешно, то возвращается значение этого элемента (типы значений не оговариваются стандартом, но, как правило, это адрес соответствующей структуры). Если создание элемента завершилось не удачно, то возвращается значение Null.

Window

Function Window(BeginX, BeginY, SizeX, SizeY, ScrollX, ScrollY : Long; Text : String) : Visible;

Описание аргументов:

BeginX, BeginY – Координаты верхнего левого угла элемента относительно владельца.

SizeX, SizeY – Горизонтальный и вертикальный размеры элемента.

ScrollX, ScrollY – Целочисленные параметры, задающие наличие у окна горизонтальной и вертикальной полосы прокрутки. Если значение параметра равно нулю, то соответствующая полоса прокрутки отсутствует, при любом другом значении параметра в окно включается соответствующая полоса прокрутки.

Text – Название окна.

Описание функции:

Эта функция создает отображаемый элемент типа окно. Если создание прошло успешно, то возвращается значение этого элемента (типы значений не оговариваются стандартом, но, как правило, это адрес соответствующей структуры). Если создание элемента завершилось не удачно, то возвращается значение Null. После создания окно является пустым.

2.7 Строковые функции

В этом разделе описан набор функций для работы со строками, которые могут использоваться в языках описания всех компонентов нейрокомпьютера.

Function SubStr(S : String; Origin, Leng : Integer) : String;

Описание аргументов

S – строка, из которой выделяется фрагмент.

Origin – начальная позиция выделяемого фрагмента в строке S

Leng – длина выделяемого фрагмента.

Выделяет из строки S фрагмент, начинающийся с позиции Origin и длиной Leng символов. Если строка короче чем Origin, то результатом является пустая строка. Если строка длиннее чем Origin символов, но короче чем Origin+Leng символов, то результатом является фрагмент строки S с символа Origin и до конца строки S.

Function Pos(S1, S2 : String) : Integer

Описание аргументов

S1 – строка, в которой ищется вхождение строки S2.

S2 – строка, вхождение которой ищется.

Функция Pos возвращает номер первого символа в строке S1, начиная с которого, в строке S1 полностью содержится строка S2. Если строка S2 ни разу не встретилась в строке S1, то результат равен нулю.

Function Len(S : String) : Integer

Описание аргументов

S – строка, длина которой вычисляется.

Функция Len возвращает длину (число символов) строки S

2.8 Описание языка описания компонентов

В табл. 12 приведен список ключевых слов, общих для всех языков описания компонентов нейрокомпьютера. Кроме того, к ключевым словам относятся типы данных, приведенные в табл. 1; обозначения операций, приведенные в табл. 3, 7, 8, 9, 10; названия функций преобразования (табл. 5) и приведения типов (табл. 6); идентификаторы предопределенных констант, приведенные в табл. 11; имена интерфейсных функций, приведенных в разделе «Перечень интерфейсных функций»; обозначения элементарных функций, приведенных в табл.13; обозначения строковых функций, приведенных в разделе «Строковые функции» и обозначения функций управления памятью из раздела «функции управления памятью».

2.8.1 Передача аргументов функциям

Во всех языках описания компонентов все параметры передаются по ссылке (передается не значение аргумента, а его адрес). Если в качестве фактического аргумента указано выражение, то значение выражения помещается интерпретатором (или компилятором) во временную переменную, имеющую тип, совпадающий с типом формального аргумента, а адрес временной переменной передается в качестве фактического аргумента.

Таблица 12.

Ключевые слова, общие для всех языков описания компонент нейрокомпьютера.

Ключевое слово	Краткое описание
Begin	Начало описания тела процедуры, или операторных скобок.
By	Часть оператора цикла с шагом. Предшествует шагу цикла.
Do	Завершающая часть операторов цикла.
Else	Часть условного оператора. Предшествует оператору, выполняемому, если условие ложно.
End	Конец описания тела процедуры или операторных скобок.
For	Заголовок оператора цикла с шагом.
Function	Заголовок описания функции.
Global	Начло блока описания глобальных переменных.
GoTo	Начало оператора перехода.
If	Начало условного оператора.
Include	Предшествует имени файла, целиком вставляемого в это место описания.
Label	Начало описания меток
Name	Предшествует имени статической переменной.
SetParameters	Признак раздела установления значений параметров.
Static	Начло блока описания статических переменных.
Then	Часть условного оператора, предшествующая оператору, выполняемому, если условие истинно.
To	Часть оператора цикла с шагом. Предшествует верхней границе цикла.
Var	Начло блока описания переменных.
While	Заголовок оператора цикла по условию.

Таблица 13

Элементарные функции, допустимые в языках описания компонент нейрокомпьютера

Обозначение	Значение	Обозначение	Значение
Sin	Синус	Cos	Косинус
Tan	Тангенс	Atan	Арктангенс
Sh	Гиперболический синус	Ch	Гиперболический косинус
Th	Гиперболический тангенс	Lg	Логарифм двоичный
Ln	Логарифм натуральный	Exp	Экспонента
Sqrt	Квадратный корень	Sqr	Квадрат
Abs	Абсолюеное значение	Sign	Знак аргумента (0 – минус)

2.8.2 Имена структурных единиц компонентов

Компоненты предобработчик, сеть, оценка и интерпретатор ответа имеют иерархическую структуру. Часть запросов может быть адресована не всему компоненту, а его структурной единице любого уровня. Для точного указания адресата запроса используется полное имя структурной единицы, которое строится по следующему правилу:

1. Имя компонента является полным именем компонента.
2. Полное имя младшей структурной единицы строится путем добавления справа к имени старшей структурной единицы точки, псевдонима младшей структурной единицы и номера экземпляра младшей структурной единицы, если младших структурных единиц с таким псевдонимом несколько.

Иногда при построении описания компонента требуется однозначное имя структурной единицы. В качестве однозначного имени можно использовать полное имя, но такой подход лишает возможности вставлять подготовленные структурные единицы в структуры более высокого уровня. Для этого вводится понятие однозначного имени структурной единицы: в описании структурной единицы *A* однозначным именем структурной единицы *B*, являющейся частью структурной единицы *A,* является полное имя структурной единицы *B*, из которого исключено полное имя структурной единицы *A*.

2.8.3 Описание синтаксических конструкций

Для описания синтаксиса языков описаний компонентов используется расширенная Бэкусова нормальная форма. В описании БНФ во всей книге приняты следующие обозначения:

- <Имя> – нетерминальный символ – понятие которое было раскрыто или будет раскрыто далее;

- *Имя* – терминальный символ – понятие, не требующее раскрытия;
- **Имя** – ключевое слово – подмножество терминальных символов;
- [Имя] – необязательный элемент – синтаксическая конструкция, заключенная в скобки может отсутствовать;
- {Имя1 | Имя2 | Имя3}– одно из значений – может присутствовать одно и только одно из разделенных символом | значений.

В данном разделе приведено описание общего подмножества языков описания компонентов. В некоторых случаях, когда БНФ описание понятия сложно, а неформальное описание просто и однозначно, в БНФ описание включаются фрагменты неформального описания таких понятий.

Список синтаксических конструкций общего назначения:

<Идентификатор> ::= <Буква> [<Символьная строка>]

<Буква> ::= {*a* | *b* | *c* | *d* | *e* | *f* | *g* | *h* | *i* | *j* | *k* | *l* | *m* | *n* | *o* | *p* | *q* | *r* | *s* | *t* | *u* | *v* | *w* | *x* | *y* | *z* | *A* | *B* | *C* | *D* | *E* | *F* | *G* | *H* | *I* | *J* | *K* | *L* | *M* | *N* | *O* | *P* | *Q* | *R* | *S* | *T* | *U* | *V* | *W* | *X* | *Y* | *Z*}

<Символьная строка> ::= {<Буква> | <Цифра> | _ } [<Символьная стока>]

<Цифра> ::= {*0* | *1* | *2* | *3* | *4* | *5* | *6* | *7* | *8* | *9*}

<Число> ::= {<Целое число> | <Действительное число>}

<Целое число> ::= [–] <Положительное целое число>

<Положительное целое число> ::= <Цифра> [<Положительное целое число>]

<Действительное число> ::= <Целое число>[**.**<Положительное целое число>][**e**<Целое число>]

<Целочисленная константа> ::= {<Предопределенная константа типа Integer> | < Предопределенная константа типа Long> | <Целое число>}

<Цветовая константа> ::= *H* <Шестнадцатеричная цифра> <Шестнадцатеричная цифра> <Шестнадцатеричная цифра> <Шестнадцатеричная цифра>

<Шестнадцатеричная цифра> ::= {*0* | *1* | *2* | *3* | *4* | *5* | *6* | *7* | *8* | *9* | *A* | *B* | *C* | *D* | *E* | *F* }

<Строковая константа> ::= “<Строка произвольных символов>”

<Логическая константа> ::= {**True** | **False**}

<Строка произвольных символов> – Последовательность произвольных символов из набора ANSI. В этой последовательности допускаются символы национальных алфавитов. При необходиости включить в эту конструкцию символ кавычек, он должен быть удвоен.

<Скалярный тип> ::= {**Long** | **Real** | **Integer** | **Color** | **Logic** | **String** | **PRealArray** | **PIntegerArray** | **PLongArray** | **PLogicArray** | **PString** | **Visual** | **Pointer** | **FuncType**}

<Тип массива> ::= { **RealArray** | **IntegerArray** | **LongArray** | **LogicArray**}

<Константа типа ***Тип***> – константа имеющая тип ***Тип***.

Список синтаксических конструкций для формальных аргументов:

<Список формальных аргументов> ::= <Формальный аргумент> [; <Список формальных аргументов>]

<Формальный аргумент> ::= <Список имен аргументов> : <Скалярный тип>

<Список имен аргументов> ::= <Имя аргумента> [,<Список имен аргументов>]

<Имя аргумента> ::= <Идентификатор>

<Аргумент типа ***Тип***> – одно из следующих понятий:

имя аргумента, который при описании формальных аргументов имел тип ***Тип***

имя элемента аргумента-массива, если элементы массива имеют тип ***Тип***

результат приведения произвольного аргумента или элемента аргумента-массива к типу ***Тип.***

Синтаксические конструкции описания переменных:

<Описание переменных> ::= **Var** <Список описаний однотипных переменных>

<Список описаний однотипных переменных> ::= <Тип переменной> <Список переменных>*;* [<Список описаний однотипных переменных>]

<Список переменных> ::= <Имя переменной> [, <Список переменных>]

<Имя переменной> ::= <Идентификатор>

<Тип переменной> ::= {<Скалярный тип> | <Тип массива>*[*<Целочисленное константное выражение>*]*}

<Переменная типа ***Тип***> – одно из следующих понятий:

имя переменной, которая при описании переменных имела тип ***Тип***

имя элемента массива, если элементы массива имеют тип ***Тип***

результат приведения произвольной переменной или элемента массива к типу ***Тип.***

Синтаксическая конструкция описания глобальных переменных (доступна только в языках описания компонентов учитель и контрастер):

<Описание глобальных переменных> ::= **Global** <Список описаний однотипных переменных>

Синтаксические конструкции описания статических переменных

Статические переменные, как правило, служат для описания параметров компонентов нейрокомпьютера. Использование в именах переменных только символов латинского алфавита и цифр делает идентификаторы универсальными, но неудобными для всех пользователей, кроме англо-говорящих. Для удобства всех остальных пользователей в описании статических переменных предусмотрена возможность использовать дополнительные имена для статических переменных. Однако эти имена служат только для построения интерфейса и не могут быть использованы в описании тела соответствующего компонента. Кроме того, статической переменной можно при описании задать значение по умолчанию.

<Описание статических переменных> ::= **Static** <Список описаний статических переменных>

<Список описаний статических переменных> ::= <Описание статической переменной>; [<Список описаний статических переменных>]

<Описание статической переменной> ::= <Тип переменной> <Имя переменной> [**Name** <Имя статической переменной>] [**Default** <Значение по умолчанию>]

<Имя статической переменной> ::= <Строковая константа>

<Значение по умолчанию> ::= <Константное выражение типа <Тип переменной>>

Синтаксические конструкции описания функций

<Описание функций> ::= <Описание функции> [<Описание функций>]

<Описание функции> ::= <Заголовок функции> <Описание переменных> <Описание меток> <Тело функции>

<Заголовок функции> ::= **Function** <Имя функции>[*(*<Список формальных аргументов>*)*] : <Скалярный тип>*;*

<Описание меток> ::= **Label** <Список меток>*;*

<Список меток> ::= <Имя метки> [, <Список меток>]

<Имя метки> ::= <Идентификатор>

<Тело функции> ::= **Begin** <Составной оператор> **End;**

<Составной оператор> ::= [<Имя метки>:] <Оператор> [; <Составной оператор>]

<Оператор> ::= {<Оператор присваивания> | <Оператор ветвления> | <Оператор цикла> | <Оператор перехода> | <Операторные скобки>}

<Оператор присваивания> ::= <Допустимое имя переменной> **=** <Выражение>

<Оператор ветвления> ::= **If** <Логическое выражение> **Then** <Оператор> [**Else** <Оператор>]

<Оператор цикла> ::= { <Цикл For> | <Цикл While> }

<Цикл For> ::= **For** <Имя переменной> = <Целочисленное выражение> **To** <Целочисленное выражение> [**By** <Целочисленное выражение>] **Do** <Оператор>

<Цикл While> ::= **While** <Логическое выражение> **Do** <Оператор>

<Оператор перехода> ::= **GoTo <**Имя метки**>**

<Операторные скобки> ::= **Begin** <Составной оператор> **End**

<Функция типа ***Tun*****>** – функция, возвращающая величину типа ***Tun***.

<Допустимое имя переменной> – допустимой переменной являются все переменные, описанные в данной функции или в данном процедурном блоке, глобальные переменные данного компонента. Для возвращения значения функции, в левой части оператора присваивания должно стоять имя функции.

Синтаксические конструкции описания выражений:

<Выражение> ::= { <Выражение типа ***Long***> | <Выражение типа ***Real***> | <Выражение типа ***Integer***> | <Выражение типа ***Color***> | <Выражение типа ***Logic***> | <Выражение типа ***String***> | <Выражение типа ***Pointer***>}

<Целочисленное выражение> ::= { <Выражение типа ***Long***> | <Выражение типа ***Integer***>}

<Выражение типа ***Tun***> ::= [<Префиксная операция типа ***Tun***>] <Операнд типа ***Tun***> [<Операция типа ***Tun***> <Операнд типа ***Tun***>]

<Операция типа ***Long***> ::= {**+** | **–** | ***** | **Div** | **Mod** | **And** | **Or** | **Xor**}

<Операция типа ***Real***>::= {**+** | **–** | ***** | **/** | **RMod** }

<Операция типа ***Integer***> ::= {**+** | **–** | ***** | **Div** | **Mod** | **And** | **Or** | **Xor**}

<Операция типа ***Color***> ::= {**COr** | **CAnd** | **CXor**}

<Операция типа ***Logic***> ::= {**And** | **Or** | **Xor**}
<Операция типа ***String***> ::= +
<Префиксная операция типа ***Long***> ::= { **–** | **Not** }
<Префиксная операция типа ***Real***>::= **–**
<Префиксная операция типа ***Integer***> ::= { **–** | **Not** }
<Префиксная операция типа ***Color***> ::= **CNot**
<Префиксная операция типа ***Logic***> ::= **Not**
<Операнд типа ***Logic***> ::= ::= {<Результат сравнения> | <Выражение типа ***Logic***> | *(*<Выражение типа ***Logic***>*)* | <Константа типа ***Logic***> | <Переменная типа ***Logic***> | <Аргумент типа ***Logic***> | <Вызов функции типа ***Logic***>}
<Результат сравнения типов ***Long, Integer, Real***> ::= *(*<Выражение типа ***Long, Integer, Real***> {> | < | >= | <= | = | <>} <Выражение типа ***Long, Integer, Real***> *)*
<Результат сравнения типа ***Color***> ::= *(*<Выражение типа ***Color***> {**CEqual** | **CIn** | **CInclude** | **CExclude** | **CIntersect**} <Выражение типа ***Color***> *)*
<Результат сравнения типа ***String***> ::= *(*<Выражение типа ***String***> {= | <>} <Выражение типа ***String***> *)*
<Операнд типа ***Tun***> ::= {<Выражение типа ***Tun***> | *(*<Выражение типа ***Tun***>*)* | <Константа типа ***Tun***> | <Переменная типа ***Tun***> | <Аргумент типа ***Tun***> | <Вызов функции типа ***Tun***>}
<Вызов функции типа ***Tun***> ::= <Имя функции типа ***Tun***> [*(*<Список фактических аргументов>*)*]
<Список фактических аргументов> ::= <Выражение> [,<Список фактических аргументов>]
<Константное выражение типа ***Tun***> – <Выражение типа ***Tun***> в операндах которого не могут фигурировать переменные и функции, описанные пользователем.
<Числовое выражение> ::= { <Выражение типа ***Long***> | <Выражение типа ***Real***> | <Выражение типа ***Integer***>}

Синтаксические конструкции задания значений статическим переменным

Эта конструкция служит для задания значений параметрам (статическим переменным) компонентов. Для компонента сеть она может встречаться не только при описании главной сети, но и при описании любой составной подсети. В специальных выражениях типа ***Tun*** могут участвовать только стандартные функции и аргументы той структурной единицы, в которой находится блок задания значений статическим переменным. При этом специальное выражение, задающее значение параметра должно иметь тип, совместимый с типом статической переменной, которой присваивается это значение.

<Установление параметров ***Структурной единицы***> ::= <Однозначное имя ***Структурной единицы***> [*[*[<Переменная цикла>:] <Начальный номер> [..<Конечный номер> [:<Шаг>] *]]*] **SetParameters <**Список значений параметров**>**
<Переменная цикла> ::= <Идентификатор>
<Начальный номер> ::= <Константное выражение типа ***Long***>
<Конечный номер> ::= <Константное выражение типа ***Long***>
<Шаг> ::= <Константное выражение типа ***Long***>
<Список значений параметров> ::= <Значение параметра> [,<Список значений параметров>]
<Значение параметра> ::= <Специальное выражение типа ***Tun***>
<Специальное выражение типа ***Tun***> ::= [<Префиксная операция типа ***Tun***>] <Специальный операнд типа ***Tun***> [<Операция типа ***Tun***> <Специальный операнд типа ***Tun***>]
<Специальный операнд типа ***Tun*** > ::= {<Специальное выражение типа ***Tun*** > | **<**Константа типа ***Tun***> | <Переменная цикла> | (<Специальное выражение типа ***Tun*** > | <Аргумент типа ***Tun***> | <Вызов функции типа ***Tun***>)>

Синтаксические конструкции описания распределения сигналов или параметров:

Данная конструкция имеет четыре аргумента, имеющих следующий смысл:

Данное – сигнал или параметр.
Объект – предобработчик, интерпретатор, оценка, сеть.
Подобъект – частный предобработчик, частный интерпретатор, частная оценка, подсеть.
<Идентификатор данных> – одно из ключевых слов Signals, Parameters, Data, InSignals, OutSignals.

<Описание распределения ***Данных, Объекта, Подобъекта, <Идентификатор данных>***> ::= **Connections** <Описание групп соответствий ***Данных***>
<Описание групп соответствий ***Данных***> ::= <Описание группы соответствий ***Данных***> [;<Описание групп соответствий ***Данных***>]
<Описание группы соответствий ***Данных***> ::= <Блок сигналов ***Подобъекта***> <=> {<Блок сигналов ***Объекта***> | <Блок сигналов ***Подобъекта***>}

<Блок сигналов ***Подобъекта***> ::= <Описатель сигналов ***Подобъекта***> [;<Блок сигналов ***Подобъекта***>]
<Описатель сигналов ***Подобъекта***> ::= { **For** <Переменная цикла> = <Начальный номер> **To** <Конечный номер> [**Step** <Шаг>] **Do** <Блок сигналов ***Подобъекта***> **End** | <Список ***Данных Подобъекта***>}
<Переменная цикла> ::= <Идентификатор>
<Список ***Данных Подобъекта***> ::= <***Данное Подобъекта***>[; <Список ***Данных Подобъекта***>]
<***Данное Подобъекта***> ::= <Псевдоним>[*[*<Номер экземпляра>*]*]**.<*Идентификатор данных*>** [*[*<Номер ***Данного***>*]*]
<Номер экземпляра> ::= {<Специальное выражение типа ***Long***> | [+:]<Начальный номер> [..<Конечный номер> [:<Шаг>]]}
<Номер ***Данного***> {<Специальное выражение типа ***Long***> | [+:]<Начальный номер> [..<Конечный номер> [:<Шаг>]]}
<Блок ***Данных Объекта***> ::= <Описатель ***Данных Объекта***> [; <Блок ***Данных Объекта***>]
<Описатель ***Данных Объекта***> ::= { **For** <Переменная цикла> = <Начальный номер> **To** <Конечный номер> [**Step** <Шаг>] **Do** <Блок ***Данных Объекта***> **End** | <Список ***Данных Объекта***> }
<Список ***Данных Объекта***> ::= <***Данное Объекта***>[; <Список ***Данных Объекта***>]
<***Данное Объекта***> ::= **<*Идентификатор данных*>** [*[*<Номер ***Данного***>*]*]

2.8.4 Комментарии

Для понятности описаний компонентов в них необходимо включать комментарии. Комментарием является любая строка (или несколько строк) символов, заключенных в фигурные скобки. Комментарий может находиться в любом месте описания компонента. При интерпретации или компиляции описания комментарии игнорируются (исключаются из текста).

2.8.5 Область действия переменных

Все идентификаторы состоят из произвольных комбинаций латинских букв, цифр и подчерков. Первым символом имени обязательно является буква. Использование букв только латинского алфавита связано с тем, что коды, используемые большинством компьютеров, имеют одинаковую кодировку для букв латинского алфавита, тогда как для букв национальных алфавитов других стран кодировка различна не только от компьютера к компьютеру но и от одной операционной системы к другой.

Заглавные и прописные буквы не различаются ни в именах, ни в ключевых словах.

Языки описания некоторых компонентов позволяют описывать глобальные переменные. Эти переменные доступны во всех функциях и процедурных блоках данного компонента. Функциям и процедурным блокам других компонентов эти переменные недоступны. Все остальные переменные (описанные в блоках Var и Static) являются локальными и доступны только в пределах той функции или процедурного блока, в котором они описаны. Статические переменные сохраняют свое значение между вызовами функций или процедурных блоков, тогда как переменные, описанные в блоках Var не сохраняют. В некоторых компонентах определены стандартные переменные и массивы (см. например описание языка описания нейронных сетей). В таких разделах область доступности предопределенных переменных оговаривается отдельно.

Переменная Error является глобальной для всех компонентов. Глобальной является также переменная ErrorManager. Однако не рекомендуется использование этих переменных путем прямого обращения к ним. Для получения значения переменной Error служит запрос GetError, исполняемый макрокомпонентом нейрокомпьютер.

2.8.6 Основные операторы

Оператор присваивания состоит из двух частей, разделенных знаком “=“. В левой части оператора присваивания могут участвовать имена любых переменных. В выражении, стоящем в правой части оператора присваивания могут участвовать любые переменные, аргументы процедурного блока и константы. В случае несоответствия типа выражения в правой части и типа переменной в левой части оператора присваивания производится приведение типа. Все выражения вычисляются слева на право с учетом старшинства операций.

Оператор ветвления. Оператор ветвления состоит из трех частей, каждая из которых начинается соответствующим ключевым словом. Первая часть – условие, начинается с ключевого слова If и содержит логическое выражение. В зависимости от значения вычисленного логического выражения выполняется Then часть (истина) или Else часть (ложь). Третья (Else) часть оператора может быть опущена. Каждая из выполняемых частей состоит из ключевого слова и оператора. При необходимости выполнить несколько операторов, необходимо использовать операторные скобки Begin End.

Цикл For имеет следующий вид:

For Переменная_цикла = Начальное_значение **To** Конечное_значение [**By** Шаг] **Do** <Оператор>

Переменная цикла должна быть одного из целочисленных типов. В ходе выполнения оператора она пробегает значения от Начальное_значение до Конечное_значение с шагом Шаг. Если описание шага опущено, то шаг равен единице. При каждом значении переменной цикла из диапазона выполняется оператор, являющийся телом цикла. Если в теле цикла необходимо выполнить несколько операторов, то необходимо воспользоваться операторными скобками. Допускается любое число вложенных циклов. Выполнение цикла в зависимости от соотношения между значениями Начальное_значение, Конечное_значение и Шаг приведено в табл. 14.

Таблица 14.

Способ выполнения цикла в зависимости от значений параметров цикла.

Конечное значение	Шаг	Способ выполнения
>Начального значения	>0	Цикл выполняется пока переменная цикла ≤ Конечного значения
<Начального значения	>0	Тело цикла не выполняется
=Начальному значению	≠0	Тело цикла выполняется один раз
>Начального значения	<0	Тело цикла не выполняется
<Начального значения	<0	Цикл выполняется пока переменная цикла ≥ Конечного значения
	=0	Тело цикла не выполняется

Цикл While. Тело цикла выполняется до тех пор, пока верно логическое выражение. Проверка истинности логического выражения производится перед выполнением тела цикла. Если тело цикла должно содержать более одного оператора, то необходимо использовать операторные скобки.

2.8.7 Описание распределения сигналов

Раздел описания распределения сигналов начинается с ключевого слова Connections. За ключевым словом Connections следует одна или несколько групп соответствий. Каждая группа соответствий состоит из правой и левой частей, разделенных символами «<=>« и описывает соответствие имен сигналов (параметров) различных структурных единиц. Каждая часть группы соответствий представляет собой список сигналов (параметров) или интервалов сигналов (параметров), разделенных между собой символом «;». Указанные в левой и правой частях сигналы (параметры) отождествляются. Если при указании сигнала (параметра) не указано имя подобъекта, то это сигнал (параметр) описываемого объекта. Использование интервала сигналов (параметров) в правой или левой части группы соответствий равносильно перечислению сигналов (параметров), с номерами, входящими в интервал, начиная с начального номера с шагом, указанным после символа «:». Если шаг не указан, то он полагается равным единице. Число сигналов в правой и левой частях группы соответствий должно совпадать. Если интервал пуст (например [2..1:1]), то описываемая им группа сигналов считается отсутствующей и пропускается. При использовании в описании соответствий явных циклов, во всех выражениях внутри цикла возможно использование переменной цикла. При этом подразумевается следующий порядок перечисления: Сначала изменяется номер в самом правом интервале, далее во втором справа, и т.д. В последнюю очередь изменяются значения переменных цикла явных циклов в порядке их вложенности (переменная самого внутреннего цикла меняется первой и т.д.). Рассмотрим следующий пример описания группы соответствий блока, содержащего две сети Net с 3 входами каждая. Ниже приведено две различных структуры связей по несколько эквивалентных вариантов описания.

Случай 1. Естественный порядок связей.

Вариант 1.

InSignals[1] <=> Net[1].**InSignals**[1]
InSignals[2] <=> Net[1].**InSignals**[2]
InSignals[3] <=> Net[1].**InSignals**[3]
InSignals[4] <=> Net[2].**InSignals**[1]
InSignals[5] <=> Net[2].**InSignals**[2]
InSignals[6] <=> Net[2].**InSignals**[3]

Вариант 2.

InSignals[1..6] <=> Net[1..2].**InSignals**[1..3]

Вариант 3.

InSignals[1]; **InSignals**[2]; **InSignals**[3]; **InSignals**[4]; **InSignals**[5]; **InSignals**[6] <=>
For I=1 **To** 3 **Do For** J=1 **To** 2 **Do** Net[J].**InSignals**[I] **End End**

Случай 2. Другой порядок связей.

Вариант 1.

InSignals[1] <=> Net[2].**InSignals**[3]
InSignals[2] <=> Net[1].**InSignals**[3]

InSignals[3] <=> Net[2].**InSignals**[2]
InSignals[4] <=> Net[1].**InSignals**[2]
InSignals[5] <=> Net[2].**InSignals**[1]
InSignals[6] <=> Net[1].**InSignals**[1]

Вариант 2.

InSignals[1..6] <=> **For** I=3 **To** 1 **Step** -1 **Do** Net[2..1:-1].**InSignals**[I] **End**

Вариант 3.

InSignals[6..1:-2]; **InSignals**[5..1:-2]<=>
For I=1 **To** 3 **Do For** J=1 **To** 2 **Do** Net[J].**InSignals**[I] **End End**

2.8.8 Функции управления памятью

Для создания массивов и освобождения занимаемой ими памяти используются следующие функции:

Создание массива.

Function NewArray(Type : Integer; Size : Long) : PRealArray;

Описание аргументов:

Type – задает размер элемента массива и является одной из предопределенных констант, приведенных в табл. 15.

Size – число элементов в массиве.

Описание исполнения.

1. Если аргумент Type не совпадает ни с одной из предопределенных констант, приведенных в табл. 15, то возвращается значение Null, исполнение функции завершается.
2. Создается массив, занимающий Size*Type+4 байта.
3. Адрес массива возвращается как результат.

Таблица 15.

Предопределенные константы типов элементов массивов

Идентификатор	Значение	Описание
mRealArray	4	Размер элемента – 4 байта
mIntegerArray	2	Размер элемента – 2 байта
mLongArray	4	Размер элемента – 4 байта
mLogicArray	1	Размер элемента – 1 байт

Освобождение массива.

Function FreeArray(Type : Integer; Array : PRealArray) : Logic;

Описание аргументов:

Type – задает размер элемента массива и является одной из предопределенных констант, приведенных в табл. 14.

Array – адрес массива. Память, занимаемая этим массивом, должна быть освобождена.

Описание исполнения.

1. Если аргумент Type не совпадает ни с одной из предопределенных констант, приведенных в табл. 15, то возвращается значение False, исполнение функции завершается.
2. Освобождается память размером TReal(Array[0])*Type+4 байта.
3. Аргументу Array присваивается значение Null

Пересоздание массива.

Function ReCreateArray(Type : Integer; Array : PRealArray; Size : Long) : Logic;

Описание аргументов:

Type – задает размер элемента массива и является одной из предопределенных констант, приведенных в табл. 15.

Array – адрес массива.

Size – число элементов в массиве.

Описание исполнения.

1. Если аргумент Type не совпадает ни с одной из предопределенных констант, приведенных в табл. 15, то возвращается значение False, исполнение функции завершается.
2. Если аргумент Array не равен Null, и TReal(Array[0]) равен Size, то возвращается значение True, выполнение функции завершается.
3. Если аргумент Array не равен Null, и TReal(Array[0]) не равен Size, то освобождается память размером TReal(Array[0])*Type+4 байта. Аргументу Array присваивается значение Null
4. Аргументу Array присваивается значение NewArray(Type,Size), возвращается значение True, исполнение функции завершается.

2.9 Использование памяти

Ряд запросов, исполняемых различными компонентами, возвращают в качестве ответа указатели на массивы. В этих случаях действуют следующие правила:

1. Если компонент получил пустой указатель (Null), то он сам создает массив необходимой длины.
2. Если передан непустой указатель, но существующей длины массива недостаточно, то компонент освобождает память, занятую под переданный массив и создает новый массив необходимой длины.
3. Освобождение памяти после использования массива лежит на вызывающем компоненте.

Если одному из компонентов не хватает памяти для выполнения запроса, то этот компонент может передать макрокомпоненту нейрокомпьютер запрос на дополнительную память. В этом случае макрокомпонент нейрокомпьютер передает всем компонентам запрос **FreeMemory.** При исполнении данного запроса каждый компонент должен освободить всю память, не являющуюся абсолютно необходимой для работы. Например, компонент задачник может для быстроты обработки держать в памяти все обучающее множество. Однако абсолютно необходимой является память, достаточная для хранения в памяти одного примера.

Запрос на освобождение памяти исполняется каждым компонентом и не включен в описания запросов компонентов, приведенные в следующих главах.

2.10 Обработка ошибок

Схема обработки ошибок достаточно проста по своей идее - каждый новый обработчик ошибок может обрабатывать только часть ошибок, а обработку остальных может передать ранее установленному обработчику. Пользователь может организовать обработку ошибок и не прибегая к установке обработчика ошибок - обработчик ошибок по умолчанию почти во всех случаях устанавливает номер последней ошибки в переменную Error, которая может быть считана с помощью запроса GetError и обработана прямо в компоненте, выдавшем запрос.

Если обработчик ошибок устанавливает номер последней ошибки в переменной Error, то все запросы, поступившие после момента установки, завершаются неуспешно. Это состояние сбрасывается при вызове запроса «дать номер ошибки».

2.10.1 Процедура обработки ошибок

Процедура обработки ошибок должна удовлетворять следующим требованиям:

I. Это должна быть процедура с дальним типом адресации. Формат описания процедуры обработки ошибок

Pascal:

```
Procedure ErrorFunc( ErrorNumber : Long ); Far;
```

C:

```
void far ErrorFunc(Long ErrorNumber)
```

II. После обработки ошибок процедура может вызвать ранее установленный обработчик ошибок. Адрес ранее установленного обработчика ошибок процедура обработки ошибок получает в ходе следующей процедуры:

A. Вызов процедуры с нулевым номером ошибки означает, что в следующем вызове будет передан адрес старой процедуры обработки ошибок.

B. Значение аргумента ErrorNumber при вызове, следующем непосредственно за вызовом с нулевым номером ошибки, должно интерпретироваться как адрес старой процедуры обработки ошибок.

Ниже приведено описание запросов, связанных с обработкой ошибок и исполняемых макрокомпонентом нейрокомпьютер.

2.10.2 Установить обработчик ошибок (OnError)

Описание запроса:

Pascal:

```
Function OnError( NewError : ErrorFunc ) : Logic;
```

C:

```
Logic OnError(ErrorFunc NewError)
```

Описание аргументов:

NewError - адрес новой процедуры обработки ошибок.

Назначение -

Описание исполнения.

1. Если Error <> 0, то выполнение запроса прекращается.
2. Вызов NewError с аргументом 0 - настройка на установку цепочки обработки ошибок.
3. Вызов NewError с аргументом ErrorManager (вместо длинного целого передается адрес старой процедуры обработки ошибок).
4. ErrorManager := NewError

2.10.3 Дать номер ошибки (GetError)

Описание запроса:
Pascal:

```
Function GetError : Integer;
```

C:

```
Integer GetError()
```

Назначение - возвращает номер последней необработанной ошибки и сбрасывает ее.
Описание исполнения.

1. GetError := Error
2. Error := 0

Списки ошибок, возникающих в различных компонентах, даны в разделах «Ошибки компоненты ...», в соответствующих главах. Все номера ошибок каждого компонента являются трехзначными числами и начинаются с номера компонента, указанного в колонке «Ошибка» табл. 16.

2.11 Запросы, однотипные для всех компонент

Ряд запросов обрабатывается всеми компонентами, кроме компонента исполнитель, носящего вспомогательный характер. Один из таких запросов – FreeMemory – был описан в разделе «Управление памятью», а два запроса, связанных с обработкой ошибок – в разделе «Обработка ошибок». В данном разделе приводятся описания остальных запросов, имеющих одинаковый смысл для всех компонентов. В отличие от ранее описанных запросов эти запросы опираются на структуру исполняющего компонента, поэтому к имени запроса добавляется префикс, задающий компонента. Список префиксов приведен в табл. 16. Единственным исключением из числа компонентов, исполняющих перечисленные в данном разделе запросы, является компонент исполнитель.

Таблица 16

Префиксы компонент

ПрефиксЗапросаОшибки	ПрефиксКомпонента	
ex	0	Исполнитель
tb	1	Задачник
pr	2	Предобработчик
nn	3	Сеть
es	4	Оценка
ai	5	Интерпретатор ответа
in	6	Учитель
cn	7	Контрастер

Все описываемые в данном разделе запросы можно разбить на четыре группы:

1. Установление текущего компонента.
2. Запросы работы со структурой компонента.
3. Запросы на получение или изменение параметров структурной единицы.
4. Запуск редактора компонента.

Все имена запросов начинаются с символов «xx», которые необходимо заменить на префикс из табл. 16 чтобы получить имя запроса для соответствующего компонента. При указании ошибок используется символ «n», который нужно заменить на соответствующий префикс ошибки из табл. 16.

Далее данном разделе компонентом также называются экземпляры компонента, а не только часть программы. Например, одна из загруженных нейронных сетей, а не только программный компонент сеть.

2.11.1 Запрос на установление текущего компонента

К этой группе запросов относится один запрос – xxSetCurrent – не исполняемый компонентом задачник.

2.11.1.1 Сделать текущей (xxSetCurrent)

Описание запроса:
Pascal:

```
Function xxSetCurrent( CompName : PString) : Logic;
```

C:

```
Logic xxSetCurrent(PString CompName)
```

Описание аргумента:

CompName – указатель на строку символов, содержащую имя компонента, которого надо сделать текущим..

Назначение – ставит указанного в параметре CompName компонента из списка загруженных компонентов на первое место в списке.

Описание исполнения.

1. Если список компонентов пуст или имя компонента, переданное в аргументе CompName, в этом списке не найдено, то возникает ошибка n01 – неверное имя компонента, управление передается обработчику ошибок, а обработка запроса прекращается.
2. Указанный в аргументе CompName компонент переносится в начало списка.

2.11.2 Запросы, работающие со структурой компонента.

К этой группе относятся запросы, позволяющие выяснить структуру компонента, прочитать ее или сохранить на диске.

2.11.2.1 Добавление нового экземпляра (xxAdd)

Описание запроса:

Pascal:

```
Function xxAdd( CompName : PString ) : Logic;
```

C:

```
Logic xxAdd(PString CompName)
```

Описание аргумента:

CompName – указатель на строку символов, содержащую имя файла компонента или адрес описания компонента.

Назначение – добавляет новый экземпляр компонента в список компонентов.

Описание исполнения.

1. Если в качестве аргумента CompName дана строка, первые четыре символа которой составляют слово File, то остальная часть строки содержит имя компонента и после пробела имя файла, содержащего компонента. В противном случае считается, что аргумент CompName содержит указатель на область памяти, содержащую описание компонента в формате для записи на диск. Если описание не вмещается в одну область памяти, то допускается включение в текст описания компонента ключевого слова Continue, за которым следует четыре байта, содержащие адрес следующей области памяти.
2. Экземпляр компонента считывается из файла или из памяти и добавляется *первым* в список компонентов (становится текущим).
3. Если считывание завершается по ошибке, то возникает ошибка n02 – ошибка считывания компонента, управление передается обработчику ошибок, а обработка запроса прекращается.

2.11.2.2 Удаление экземпляра компонента (xxDelete)

Описание запроса:

Pascal:

```
Function xxDelete( CompName : PString) : Logic;
```

C:

```
Logic xxDelete(PString CompName)
```

Описание аргумента:

CompName – указатель на строку символов, содержащую полное имя компонента.

Назначение – удаляет указанного в параметре CompName компонента из списка компонентов.

Описание исполнения.

1. Если список компонентов пуст или имя компонента, переданное в аргументе CompName, в этом списке не найдено, то возникает ошибка n01 – неверное имя компонента, управление передается обработчику ошибок, а обработка запроса прекращается.

Заметим, что попытка удаления младшей структурной единицы приводит к удалению всего компонента содержащего данную структурную единицу.

2.11.2.3 Запись компонента (xxWrite)

Описание запроса:
Pascal:

Function xxWrite(CompName : PString; FileName : PString) : Logic;

C:

Logic xxWrite(PString CompName, PString FileName)

Описание аргументов:
CompName – указатель на строку символов, содержащую имя компонента.
FileName – имя файла или адрес памяти, куда надо записать компонента.

Назначение – сохраняет в файле или в памяти компонента, указанного в аргументе CompName .
Описание исполнения.

1. Если в качестве аргумента CompName дан пустой указатель, или указатель на пустую строку, то исполняющим запрос объектом является текущий компонент.
2. Если список компонентов пуст или имя компонента, переданное в аргументе CompName, в этом списке не найдено, то возникает ошибка n01 – неверное имя компонента, управление передается обработчику ошибок, а обработка запроса прекращается.
3. Если в качестве аргумента FileName дана строка, первые четыре символа которой составляют слово File, то остальная часть строки содержит имя файла, для записи компонента. В противном случае FileName должен содержать пустой указатель. В этом случае запрос вернет в нем указатель на область памяти, куда будет помещено описание компонента в формате для записи на диск. Если описание не вмещается в одну область памяти, то в текст будет включено ключевое слово Continue, за которым следует четыре байта, содержащие адрес следующей области памяти.
4. Если во время сохранения компонента возникнет ошибка, то генерируется ошибка n03 – ошибка сохранения компонента, управление передается обработчику ошибок, а обработка запроса прекращается.

2.11.2.4 Вернуть имена структурных единиц (xxGetStructNames)

Описание запроса:
Pascal:

Function xxGetStructNames(CompName : PString; Var Names : PRealArray) : Logic;

C:

Logic xxGetStructNames(PString CompName, RealArray* Names)

Описание аргументов:
CompName – указатель на строку символов, содержащую имя компонента или полное имя его структурной единицы.
Names – массив указателей на имена структурных единиц.

Назначение – возвращает имена всех компонентов в списке компонентов или имена всех структурных единиц структурной единицы, указанной в аргументе CompName .
Описание исполнения.

1. Если в качестве аргумента CompName дан пустой указатель, или указатель на пустую строку, то исполняющим запрос объектом является соответствующий программный компонент. В качестве ответа в указателе Names возвращается массив, каждый элемент которого является указателем на *не подлежащую изменению* символьную строку, содержащую имя компонента из списка. После адреса имени последнего компонента следует пустой указатель. Выполнение запроса успешно завершается.
2. Если имя компонента, переданное в аргументе CompName, не найдено в списке компонентов, то возникает ошибка n01 – неверное имя компонента, управление передается обработчику ошибок, а обработка запроса прекращается.
3. Возвращается массив, каждый элемент которого является указателем на *не подлежащую изменению* символьную строку, содержащую псевдоним структурной единицы, являющейся частью структурной единицы, указанной в аргументе CompName. Имена структурных единиц перечисляются в порядке следования в разделе описания состава структурной единицы, имя которой указано в аргументе CompName. Если одна из структурных единиц задана в описании состава несколькими экземплярами, то имя каждого экземпляра возвращается отдельно. После указателя на имя последней структурной единицы следует пустой указатель.

2.11.2.5 Вернуть тип структурной единицы (xxGetType)

Описание запроса:

Pascal:

Function xxGetType(CompName , TypeName : PString; Var TypeId : Integer) : Logic;

C:

Logic xxGetType(PString CompName, PString TypeName, Integer TypeId)

Описание аргументов:

CompName – указатель на строку символов, содержащую полное имя структурной единицы.

TypeName – возвращает указатель на строку символов, содержащую имя структурной единицы, данное ей при описании.

TypeId – одна из предопределенных констант, соответствующая типу структурной единицы.

Назначение – возвращает имя и тип структурной единицы.

Описание исполнения.

1. Если список компонентов пуст или имя компонента, переданное в аргументе CompName, в этом списке не найдено, то возникает ошибка n01 – неверное имя компонента, управление передается обработчику ошибок, а обработка запроса прекращается.
2. В переменной TypeId возвращается тип структурной единицы. Значения предопределенных констант, соответствующих различным типам структурных единиц различных компонентов приведены в табл. 11 и в соответствующих разделах глав, содержащих описания компонентов.
3. Если структурная единица является стандартной, то указателю TypeName присваивается значение пустого указателя. Если структурная единица имеет пользовательский тип (значение аргумента TypeId равно -1), то указатель TypeName устанавливается на строку, содержащую имя, данное указанной в аргументе CompName структурной единице при ее описании.

2.11.3 Запросы на изменение параметров.

К группе запросов на изменение параметров относятся три запроса: xxGetData – получить параметры структурной единицы. xxGetName – получить названия параметров и xxSetData – установить значения параметров структурной единицы.

2.11.3.1 Получить параметры (xxGetData)

Описание запроса:

Pascal:

Function xxGetData(CompName : PString; Var Param : PRealArray) : Logic;

C:

Logic xxGetData(PString CompName, PRealArray* Param)

Описание аргументов:

CompName – указатель на строку символов, содержащую полное имя структурной единицы.

Param – адрес массива параметров.

Назначение – возвращает массив параметров структурной единицы, указанной в аргументе CompName .

Описание исполнения.

1. Если Error <> 0, то выполнение запроса прекращается.
2. Если список компонентов пуст или имя компонента, переданное в аргументе CompName, в этом списке не найдено, то возникает ошибка n01 – неверное имя компонента, управление передается обработчику ошибок, а обработка запроса прекращается.
3. В массив, адрес которого передан в аргументе Param, заносятся значения параметров. Параметры заносятся в массив в порядке описания в разделе описания статических переменных. Статические переменные, описанные вне описания структурных единиц, считаются параметрами компонента.

2.11.3.2 Получить имена параметров (xxGetName)

Описание запроса:

Pascal:

Function xxGetName(CompName : PString; Var Param : PRealArray) : Logic;

C:

Logic xxGetName(PString CompName, PRealArray* Param)

Описание аргументов:
CompName – указатель на строку символов, содержащую полное имя структурной единицы.
Param – адрес массива указателей на названия параметров.

Назначение – возвращает массив указателей на названия параметров структурной единицы, указанной в аргументе CompName .

Описание исполнения.

1. Если Error <> 0, то выполнение запроса прекращается.
2. Если список компонентов пуст или имя компонента, переданное в аргументе CompName, в этом списке не найдено, то возникает ошибка n01 – неверное имя компонента, управление передается обработчику ошибок, а обработка запроса прекращается.
3. В массив, адрес которого передан в аргументе Param, заносятся адреса символьных строк, содержащих названия параметров.

2.11.3.3 Установить параметры (xxSetData)

Описание запроса:
Pascal:

```
Function xxSetData( CompName : PString; Param : PRealArray ) : Logic;
```

C:

```
Logic xxSetData(PString CompName, PRealArray Param)
```

Описание аргументов:
CompName – указатель на строку символов, содержащую полное имя структурной единицы.
Param – адрес массива параметров.

Назначение – заменяет значения параметров структурной единицы, указанной в аргументе CompName , на значения, переданные, в аргументе Param.

Описание исполнения.

1. Если Error <> 0, то выполнение запроса прекращается.
2. Если список компонентов пуст или имя компонента, переданное в аргументе CompName, в этом списке не найдено, то возникает ошибка n01 – неверное имя компонента, управление передается обработчику ошибок, а обработка запроса прекращается.
3. Параметры, значения которых хранятся в массиве, адрес которого передан в аргументе Param, передаются указанной в аргументе CompName структурной единице.
4. Если исполняющим запрос компонентом является интерпретатор ответа (aiSetData), то генерируется запрос SetEstIntParameters к компоненту оценка. Аргументы генерируемого запроса совпадают с аргументами исполняемого запроса.

2.11.4 Инициация редактора компоненты.

К этой группе запросов относится запрос, который инициирует работу не рассматриваемых в данной работе компонентов – редакторов компонентов.

2.11.4.1 Редактировать компонента (xxEdit)

Описание запроса:
Pascal:

```
Procedure xxEdit(CompName : PString);
```

C:

```
void xxEdit(PString CompName )
```

Описание аргумента:
CompName – указатель на строку символов – имя файла или адрес памяти, содержащие описание редактируемого компонента.

Если в качестве аргумента CompName дана строка, первые четыре символа которой составляют слово File, то остальная часть строки содержит имя компонента и после пробела имя файла, содержащего описание компонента. В противном случае считается, что аргумент CompName содержит указатель на область памяти, содержащую описание компонента в формате для записи на диск. Если описание не вмещается в одну область памяти, то допускается включение в текст описания компонента ключевого слова Continue, за которым следует четыре байта, содержащие адрес следующей области памяти.

Если в качестве аргумента CompName передан пустой указатель или указатель на пустую строку, то редактор создает новый экземпляр компонента.

2.12 Описание задачи, используемой для примера

В главах, посвященных описанию предобработчика, задачника, интерпретатора ответа и оценки в качестве примера используется метеорологическая задача. Входная база данных содержит значения следующих показателей:

Температура воздуха – действительное число, изменяющееся от 273 до 393 градусов Кельвина.

Облачность – бинарный признак, означающий наличие (2) или отсутствие облачности (1).

Направление ветра – неупорядоченный качественный признак, принимающий одно из восьми значений: 1 – северный, 2 – северо-восточный, 3 – восточный, и т.д.

Осадки – упорядоченный качественный признак, принимающий следующие значения: 1 – без осадков, 2 – слабые осадки, 3 – сильные осадки.

В качестве ответов требуется предсказать значения тех же показателей через 8 часов. Такая постановка задачи не очень логична, но позволяет продемонстрировать возможности языков описания компонентов нейрокомпьютера.

3. Задачник и обучающее множество

Эта глава посвящена одному из наиболее важных и обделенных вниманием компонентов нейрокомпьютера – задачнику. Важность этого компонента определяется тем, что при обучении сетей всех видов с использованием любых алгоритмов обучения сети необходимо предъявлять примеры, на которых она обучается решению задачи. Источником данных для сети является задачник. Кроме того, задачник содержит правильные ответы для сетей, обучаемых с учителем.

3.1 Обсуждение Задачника

В этом разделе рассматриваются основные структуры и функции компонента задачник. Отметим, что задачник рассматривается только с точки зрения его использования нейронной сетью. Совершенно очевидно, что невозможно предусмотреть всех вариантов интерфейса между пользователем и задачником. Действительно, было бы странно, если бы в одном и том же интерфейсе обрабатывались задачники, содержащие только числовые поля, задачники, содержащие исключительно графическую информацию и задачники смешанного типа.

3.1.1 Структуры данных задачника

С точки зрения нейрокомпьютера задачник представляет собой прямоугольную таблицу, поля которой содержат информацию о входных данных примеров задачи, правильные ответы и другую информацию. На данный момент существует три основных способа хранения однотипных данных – базы данных, электронные таблицы, текстовые файлы. Какой из них является оптимальным для хранения данных в задачнике? Основными критериями являются удобство в использовании, компактность и универсальность. Поскольку задачник должен хранить однотипные данные и предоставлять их для обработки другим компонентам нейрокомпьютера, а не производить вычисления, то функционально задачник должен являться базой данных. Наиболее подходящим кажется формат табличных (реляционных) баз данных.

В современных операционных системах предусмотрены различные способы обмена данными между приложениями. Наиболее универсальным является обмен в символьном формате. Мы предлагаем использовать именно символьный формат данных для обмена между приложениями. Вопрос конкретной реализации обмена оставим за пределами рассмотрения, поскольку это чисто технический вопрос. Мы полагаем, что вне зависимости от того, каким путем и из какого приложения данные попали в задачник, их представление должно быть одинаковым (принятым в данной реализации задачника).

3.1.2 Поля задачника

Далее будем полагать, что задачник является реляционной базой данных из одной таблицы. Каждому примеру соответствует одна запись базы данных. Каждому данному – одно поле. В данном разделе рассмотрены допустимые типы полей, с точки зрения типа хранящихся в них данных. В разд. «Состав данных задачника» все поля разбиваются по смысловой нагрузке. Все поля базы данных можно разбить на четыре типа – числовые поля, текстовые поля, перечислимые поля и поля типа рисунок.

Числовые поля. Поля числовых типов данных Integer, Long и Real предназначены для хранения различных чисел. Отметим, поля числового типа могут нести любую смысловую нагрузку.

Перечислимые поля. Поля перечислимого типа служат для хранения качественных признаков – полей базы данных, содержащих, как правило, текстовую информацию, но имеющих малое число различных значений. Простейшим примером поля перечислимого типа является поле «пол» – это поле может принимать только два значения – «мужской» или «женский». Поле перечислимого типа не хранит соответствующего текстового значения, вместо него в поле содержится номер значения. Поля перечислимого типа могут быть только входными данными, комментариями или ответами.

Строки (текстовые поля). Поля текстового типа предназначены для хранения тестовой информации. Они могут быть только комментариями.

Рисунок. Поля типа рисунок предназначены для хранения графической информации. В данной работе не устанавливается стандарт хранения полей типа рисунок. Оговаривается только способ хранения полей типа рисунок на диске для файлов задачника, созданного в нейрокомпьютере. При передаче рисунков предобработчику используется формат, согласованный для предобработчика и задачника.

3.1.3 Состав данных задачника

Компонент задачник является необходимой частью нейрокомпьютера вне зависимости от типа применяемых в нем нейронных сетей. Однако в зависимости от решаемой задачи содержимое задачника

может меняться. Так, например, для решения задачи классификации без учителя используют нейросети, основанные на методе динамических ядер [223, 261] (наиболее известным частным случаем таких сетей являются сети Кохонена [98, 99]). Задачник для такой сети должен содержать только векторы входных данных и предобработанных входных данных. При использовании обучаемых сетей, основанных на принципе двойственности, к задачнику необходимо добавить вектор ответов сети. Кроме того, некоторые исследователи хотят иметь возможность просмотреть ответы, выданные сетью, вектор оценок примера, показатели значимости входных сигналов и, возможно, некоторые другие величины. Поэтому, стандартный задачник должен иметь возможность предоставить пользователю всю необходимую информацию. Но если задачник всегда будет отводить память для всех необходимых величин, то для простейших случаев использование памяти будет неэффективным. Таким образом, возникает противоречие, между необходимостью предоставить пользователю всю необходимую ему информацию и эффективностью использования памяти. Рассмотрим более подробно состав данных задачника.

3.1.3.1 Цвет примера и обучающая выборка

Довольно часто при обучении нейронных сетей возникает необходимость использовать в обучении не все примеры задачника, а только часть. Например, такая возможность необходима при использовании метода скользящего контроля для оценки качества обучения сети. Существует несколько способов реализации такой возможности. Кроме того, часто бывает полезно приписать примерам ряд признаков. Так, при просмотре задачника, пользователю полезно видеть степень обученности примера (например, отображать зеленым цветом примеры, которые решаются сетью идеально, желтым – те, которые сеть решает правильно, но не идеально, а красным – те, при решении которых сеть допускает ошибки).

Ту часть задачника, которая в данный момент используется в обучении нейронной сети, будем называть обучающей выборкой. Для выделения из задачника обучающей выборки предлагается использовать механизм «цветов». Если все примеры покрашены в некоторые цвета, то обучающую выборку можно задать, указав цвета примеров, которые необходимо использовать в обучении. В соответствии с предлагаемой схемой, каждый пример покрашен каким–то цветом, а при задании обучающей выборки можно задать комбинацию цветов. Схема работы с цветами детально рассмотрена в разделе «Переменные типа цвет и операции с цветами» главы «Общий стандарт».

Выделенную с помощью механизма цветов часть задачника будем далее называть текущей выборкой. Очевидно, что обучающая выборка является частным случаем текущей выборки.

3.1.3.2 Входные данные

Входные данные – данные, необходимые для решения сетью примера. Входные данные являются вектором. Существует всего несколько видов входных данных. Каждый компонент вектора входных данных может быть:

числом;

полем с ограниченным числом состояний;

рисунком.

3.1.3.3 Комментарии

Пользователю, при работе с задачником, часто бывает необходимо иметь возможность идентифицировать примеры не только по номерам. Например, при работе с медицинскими базами данных полезно иметь поле, содержащее фамилию больного или номер истории болезни. Для этих целей в задачнике может потребоваться хранить вектор комментариев, которые не могут быть использованы в обучении.

3.1.3.4 Предобработанные данные

Предобработанные данные – это вектор входных сигналов сети, полученный из входных данных после предобработки, выполняемой компонентом предобработчик. Хранение задачником этого вектора необязательно. Каждый элемент вектора предобработанных данных является действительным числом. Следует отметить, что любая нетривиальная предобработка, как правило, изменяет размерность вектора.

3.1.3.5 Правильные ответы

Правильные ответы – вектор ответов, которые должна выдать обученная нейронная сеть при решении примера. Этот вектор абсолютно необходим при обучении сетей с учителем. При использовании других видов сетей хранение задачником этого вектора необязательно. Компонентами вектора ответа могут быть как числа, так и поля с ограниченным набором состояний. В первом случае будем говорить о задаче аппроксимации функции, а во втором – о задаче классификации объектов.

3.1.3.6 Полученные ответы

Полученные ответы – вектор ответов, выданных сетью при решении примера. Для задачника хранение этой части примера не обязательно.

3.1.3.7 Оценки

Оценки – вектор оценок, полученный сетью за решение всех подзадач примера (число подзадач равно числу ответов примера). Хранение этого вектора задачником не обязательно.

3.1.3.8 Вес примера

Вес примера – скалярный параметр, позволяющий регулировать интенсивность участия примера в процессе обучения. Для не обучаемых нейронных сетей вес примера может использоваться для учета вклада данных примера в формируемую карту связей. Применение весов примеров зависит от типа используемой сети.

3.1.3.9 Достоверность ответа

При составлении задачника ответы довольно часто получаются как результат измерения или путем логических выводов в условиях нечеткой информации (например, в медицине). В этих случаях одни ответы имеют большую достоверность, чем другие. Некоторые способы построения оценки или формирования карты связей нейронной сети позволяют использовать эти данные. Достоверность ответа является вектором, поскольку ответ каждой подзадачи данного примера может иметь свою достоверность. Каждый элемент вектора достоверности ответа является действительным числом от нуля до единицы.

3.1.3.10 Уверенность в ответе

При использовании некоторых видов оценки (см. главу «Оценка и интерпретатор ответа») интерпретатор ответа способен оценить уверенность сети в полученном ответе. Вектор уверенности сети в ответах (для каждого ответа своя уверенность) может оказаться полезным для пользователя. Каждый элемент вектора уверенности в ответе является действительным числом от нуля до единицы.

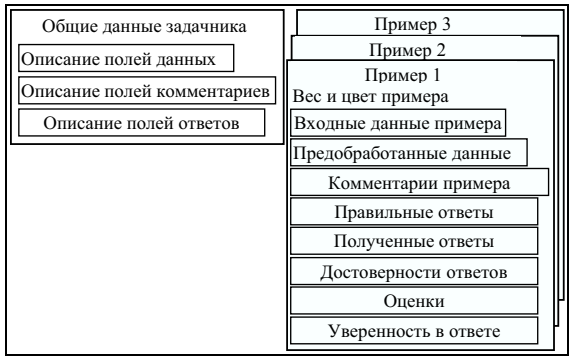


Рис. 1. Схема данных задачника.

Все перечисленные выше векторы можно разбить на четыре типа по структуре:

- Входные данные. Таких векторов обычно два – вектор описания полей данных (содержит описание полей данных: имя поля, его тип и возможно некоторую дополнительную информацию) и собственно вектор данных. Причем каждый пример имеет свой вектор данных, но вектор описания полей данных один для всех примеров задачника. Эти векторы имеют одинаковое число элементов, и их элементы попарно соответствуют друг другу.
- Вектор ответов. При обучении с учителем, в задачнике есть, по крайней мере, два вектора этого вида – вектор описания полей ответов и вектор правильных ответов. Кроме того, возможно хранение в задачнике векторов вычисленных ответов, достоверности ответов и уверенности в ответе. Вектор описания полей ответов – один для всех примеров задачника. Все остальные векторы данного типа хранятся по одному экземпляру каждого вектора на пример.
- Вектор комментариев. Таких векторов обычно только два – вектор описания полей комментариев и вектор комментариев. Вектор описания полей комментариев – один на весь задачник, а вектор комментариев – один на пример.

На рис. 1 приведено схематическое устройство задачника. Такое представление данных позволяет гибко использовать память. Однако следует учесть, что часть полей может переходить из одного вектора в другой. Например, при исключении одного входного данного из использования (см. главу «Контрастер»), соответствующее ему поле переходит из вектора входных данных в вектор комментариев.

3.1.4 Рекомендуемое управление памятью

Наиболее быстрым в работе является хранение всех примеров в памяти. Однако при большом числе примеров хранение всего задачника в памяти является невозможным. Но применение маски используемых данных позволяет задачнику держать в памяти компьютера только те векторы данных примеров, которые будет необходимо возвращать в запросах. Так, например, при обучении сети методом обратного распространения ошибки задачнику требуется хранить в памяти только векторы правильных ответов и предобработанных данных. Все остальные векторы можно хранить на дисках. Такое управление позволяет максимально эффективно использовать оперативную память компьютера. Задачник в ходе работы может выбирать всю свободную память для хранения данных. Однако он должен иметь экономный режим – режим работы при минимальном использовании оперативной памяти.

Запрос «Освободить память» (**FreeMemory**), описанный в главе «Общий стандарт», относится ко всему задачнику в целом. При обработке следующих запросов задачник может снова забирать свободную память. Такое использование памяти позволяет остальным компонентам нейрокомпьютера использовать излишки памяти, потребляемой задачником. Например, при запуске процедуры обучения компоненте нейронная сеть необходимо создать несколько копий синаптической карты. Если обнаружена нехватка памяти, она выдает макрокомпоненту нейрокомпьютер запрос на освобождение памяти, и все компоненты, выполняя этот запрос, оставляют себе только минимально необходимую память. После создания необходимого числа копий синаптической карты остаток памяти могут использовать те компоненты, которым она необходима.

3.2 Стандарт первого уровня компонента задачник

В этом разделе приводится описание хранения задачника на внешнем носителе.

3.2.1 Язык описания задачника

В языке описания задачника используется ряд ключевых слов, специфических для этого языка. Эти ключевые слова приведены в табл. 1.

Таблица 1.

Ключевые слова специфические для языка описания задачника

Ключевое слово	Краткое описание
TaskBook	Заголовок описания задачника
Structure	Заголовок описания структуры задачника
Field	Начало описания поля
Picture	Поле типа рисунок
Source	Описание источника данных
External	Описание внешнего источника данных

Список предопределенных констант языка описания задачника приведен в табл. 2. Эти константы используются при указании типа вектора, к которому принадлежит описываемое поле, при указании используемых векторов в запросе на открытие сеанса и при указании типа вектора в запросах на получение или занесение данных.

Таблица 2

Предопределенные константы

Идентификатор	Значение	Смысл
tbColor	1	Цвет примера
tbInput	2	Входной сигнал
tbPrepared	3	Предобработанные данные
tbAnswers	4	Правильные ответы
tbReliability	5	Достоверность ответа
tbCalcAnswers	6	Полученные ответы
tbCalcReliability	7	Уверенность в ответе
tbWeight	8	Вес примера
tbEstimation	9	Оценки
tbComment	10	Комментарии

3.2.1.1 БНФ языка описания задачника

Обозначения, принятые в данном расширении БНФ и описание ряда конструкций приведены в главе «Общий стандарт» в разделе «Описание языка описания компонентов».

<Описание задачника> ::= <Заголовок задачника> <Описание структуры задачника> <Описание источника данных> <Конец описания задачника>

<Заголовок задачника> ::= **TaskBook** <Имя задачника>

<Имя задачника> ::= <Идентификатор>

<Описание структуры задачника> ::= <Заголовок описания структуры> <Описание полей> <Описание цвета><Описание веса> <Конец описания структуры>

<Заголовок описания структуры> ::= **Structure**
<Описание цвета> ::= **Field** <Имя поля цвет> **tbColor Color End Field**
<Имя поля цвет> ::= <Константа типа ***String***>
<Описание веса> ::= **Field** <Имя поля вес> **tbWeight Real End Field**
<Имя поля вес> ::= <Константа типа ***String***>
<Описание полей> ::= <Описание поля> [<Описание полей>]
<Описание поля> ::= **Field** <Имя поля> <Тип вектора> {<Описание целого поля> | <Описание действительного поля> | <Описание перечислимого поля> | <Описание поля рисунка> | <Описание текстового поля>} **End Field**
<Имя поля> ::= <Константа типа ***String***>
<Тип вектора> ::= {**tbInput** | **tbAnswers** | **tbReliability** | **tbCalcAnswers** | **tbCalcReliability** | **tbEstimation**}
<Описание целого поля> ::= {**Long** | **Integer**}
<Описание действительного поля> ::= **Real**
<Описание перечислимого поля> ::= **Enumerated** <Список имен значений> *;*
<Список имен значений> ::= <Имя значения> [, <Список имен значений>]
<Имя значения> ::= <Константа типа ***String***>
<Описание текстового поля> ::= **String** <Максимальная длина строки>
<Максимальная длина строки> ::= <Константа типа ***Integer***>
<Описание поля рисунка> ::= **Picture** <Размер памяти для рисунка>
<Размер памяти для рисунка> ::= <Константа типа ***Long***>
<Конец описания структуры> ::= **End Structure**
<Описание источника данных> ::= **Source** {<Внешний источник> | <Подготовлено в задачнике>}
<Внешний источник> ::= <Имя приложения, которому нужно передать запрос> <SQL – запрос>
<Имя приложения, которому нужно передать запрос> ::= <Константа типа ***String***>
<SQL – запрос> ::= <Константа типа ***String***>
<Подготовлено в задачнике> – В соответствии с порядком описания полей выводятся символьные представления полей, разделенные символом табуляции (байтом содержащим код 9). Примеры (в терминологии баз данных – записи) разделяются символом конца абзаца (переводом строки – байтом, содержащим код 13). Поля рисунки выводятся в виде последовательности <Размер памяти для рисунка> целых чисел, разделенных пробелами, каждое из которых является десятичным представлением числа (от 0 до 255), содержащегося в соответствующем байте области памяти, отведенной для хранения рисунка.
<Конец описания задачника> ::= **End TaskBook**

3.2.1.2 Описание языка описания задачника

В этом разделе приведено подробное описание дополнительной информации (информации, следующей за типом данных поля) для полей, в блоках описания которых она используется.

Перечислимый тип поля. При использовании перечислимого типа поля в векторах данных хранятся не сами значения, а их номера. Для отображения в редакторе задачника значений полей их необходимо брать из блока описания поля. В списке имен значений блока описания перечислимого поля хранятся символьные константы, первая из которых содержит название состояния, соответствующее неопределенному значению поля; вторая – первому из значений, которые может принимать поле, и т.д.

Строка. Поля типа строка предназначены для хранения символьных строк фиксированной длины. Длина строки задается значением параметра <Максимальная длина строки>.

Рисунок. Поля типа рисунок предназначены для хранения графической информации. Первые семь байт поля имеют смысл, приведенный в табл. 3. В таблице принято обозначение Б1 – величина, хранящаяся в первом байте, Б2 – во втором и т.д. Рисунок разворачивается по строкам, начиная с левого верхнего угла, в непрерывный массив, размером (Б2*256+Б1)(Б4*256+Б3).

Таблица 3

Значение первых семи байт поля типа рисунок

ВеличинаЗначение	
Б2*256+Б1	Положительное целое число, задающее размер рисунка по горизонтали в пикселях.
Б4*256+Б3	Положительное целое число, задающее размер рисунка по вертикали в пикселях.
(Б7*256+Б6)*256+Б5	Число цветов, использованных в рисунке

Если число цветов равно единице (черно-белое изображение), то каждый следующий байт содержит восемь пикселей изображения. Самый младший бит восьмого байта соответствует левому верхнему пикселю рисунка. Если число цветов равно трем, то каждый байт, начиная с восьмого, содержит информацию о четырех пикселях. Младшие два бита задают левый верхний пиксель рисунка. Если число цветов от 4 до 15, то каждый байт, начиная с восьмого, содержит информацию о двух пикселях. Младшие четыре бита задают левый верхний пиксель рисунка. Если число цветов от 16 до 255, то каждый байт, начиная с восьмого, содержит информацию об одном пикселе. Значение в восьмом байте соответствует левому верхнему пикселю рисунка. При числе цветов от 256 до 65535 каждые два байта, начиная с восьмого, содержат информацию об одном пикселе (первый пиксель имеет цвет номер Б9*256+Б8). Значение в восьмом и девятом байтах соответствует левому верхнему пикселю рисунка. При числе цветов от 65535 до 16777215 каждые три байта, начиная с восьмого, содержат информацию об одном пикселе (первый пиксель имеет цвет номер (Б10*256+Б9)*256+Б8). Значение в восьмом, девятом и десятом байтах соответствует левому верхнему пикселю рисунка.

Альтернативный способ записи рисунка. Предложенный выше способ хорош своей простотой и плох большим объемом данных. Большинство графических форматов файлов (например GIF) обеспечивают высокую степень компрессии графической информации. Для использования этой возможности при записи поля графической информации на диск предлагается альтернативный формат. В этом формате первые два байта должны быть нулевыми. Поскольку в основном формате записи рисунков эти два байта формировали размер рисунка по горизонтали, нулевая ширина рисунка служит признаком использования альтернативного формата. Следующие пять байт хранят ASCII коды типа графического формата. Используются коды заглавных букв латинского алфавита. Если тип графического стандарта содержит менее пяти символов (PCX, GIF), то тип дополняется символами пробела справа. Следующие четыре байта, с восьмого по одиннадцатый, содержат число байт в графическом файле. Начиная с двенадцатого байта, идет информация, содержавшаяся в графическом файле.

Сопоставление элементов векторов правильных ответов, достоверности, вычисленных ответов, уверенности и оценки производится по следующему правилу. Первому полю типа правильный ответ соответствуют первое поле типа достоверность, первое поле типа вычисленный ответ, первое поле типа уверенность, первое поле типа оценка, второму – вторые и т.д.

3.2.1.3 Неопределенные значения

В практике работы большинство таблиц данных не полны. То есть, часть данных в примерах задачника не известна. Задачник должен однозначно указать предобработчику неизвестные данные. Для этих целей для каждого типа входных данных определено специальное значение - неопределенное. Для передачи неизвестных значений используются следующие величины: 10^{-40} для действительных чисел и 0 для всех типов качественных признаков.

3.2.1.4 Пример описания задачника

В этом разделе приведено описания простого задачника для прогнозирования курса американского доллара к рублю. Задачник содержит три примера. В разделе описания данных вместо символа табуляции использован символ «→», а вместо символа конца абзаца – «↵».

TaskBook CursValuty
Structure
Field "Цвет" **tbColor Color End Field**
Field "Вес" **tbWeight Real End Field**
Field "Дата" **tbComment Real End Field**
Field "Текущий курс" **tbInput Real End Field**
Field "Курс на следующий день" **tbAnswers Real End Field**
Field "Достоверность курса" **tbReliability Real End Field**
Field "Предсказанный курс" **tbCalcAnswers Real End Field**
Field "Надежность предсказания" **tbCalcReliability Real End Field**
Field "Оценка предсказания" **tbEstimation Real End Field**
End Structure
Source
HFFFF→1.0→01.01.97→5773→5774→1.0→5775→0.1→0.07↵
HFFFF→1.0→02.01.97→5774→5776→1.0→5777→0.01→0.7↵
HFFFF→1.0→03.01.97→5776→5778→1.0→5779→0.2→0.007↵
End TaskBook

3.3 Стандарт второго уровня компонента задачник

В этом разделе описаны все запросы компонента задачник в виде процедур и функций. При описании используется синтаксис языков Turbo Pascal и C. В Паскаль варианте приведены заголовки функций и процедур. В С варианте – прототипы функций. Большинство запросов, реализуется в виде функций, сообщающих о корректности завершения операции.

Предполагается возможность одновременной работы нескольких сеансов одного задачника. Например, допускается редактирование задачника и одновременное обучение сети по тому же задачнику.

Все запросы к компоненту задачник можно разбить на следующие группы.

1. Чтение и запись задачника.
2. Начало и конец сеанса.
3. Перемещение по примерам.
4. Определение, получение и изменение данных.
5. Окраска примеров.
6. Установление структуры **Задачника**.
7. Добавление и удаление примеров.
8. Обработка ошибок.

3.3.1 Чтение и запись задачника

К этой группе запросов относятся запросы, работающие со всем задачником в целом. Эти запросы считывают задачник, сохраняют задачник на диске или выгружают ранее считанный или созданный задачник.

3.3.1.1 Прочитать задачник (tbAdd)

Описание запроса:
Pascal:

```
Function tbAdd( CompName : PString ) : Logic;
```

C:

```
Logic tbAdd( PString CompName )
```

Описание аргумента:
CompName – указатель на строку символов, содержащую имя файла задачника.
Назначение – служит для считывания задачника.

Описание исполнения.

1. Если Error <> 0, то выполнение запроса прекращается.
2. Если в данный момент считан задачник, то генерируется запрос tbDelete. Если запрос tbDelete завершается неуспешно, то генерируется внутренняя ошибка 104 – попытка считывания задачника при открытых сеансах ранее считанного задачника. Управление передается обработчику ошибок. Выполнение запроса прекращается.
3. Первые четыре символа строки CompName составляют слово File. Остальная часть строки содержит имя компонента и после пробела имя файла, содержащего компонент.
4. Если во время выполнения запроса возникает ошибка, то генерируется внутренняя ошибка 102 – ошибка чтения задачника. Управление передается обработчику ошибок. Выполнение запроса прекращается. В противном случае выполнение запроса успешно завершается.

3.3.1.2 Записать задачник (tbWrite)

Описание запроса:
Pascal:

```
Function tbWrite( CompName, FileName : PString) : Logic;
```

C:

```
Logic tbWrite(PString CompName, PString FileName)
```

Описание аргументов:
CompName – указатель на строку символов, содержащую имя задачника.
FileName – имя файла, куда надо записать компонента.

Назначение – сохраняет задачник в файле.
Описание исполнения.

1. Если Error <> 0, то выполнение запроса прекращается.

2. Если в момент получения запроса отсутствует считанный задачник, то возникает ошибка 101 – запрос при отсутствии задачника, управление передается обработчику ошибок, а обработка запроса прекращается.
3. Задачник записывается в файл FileName под именем CompName.
4. Если во время выполнения запроса возникает ошибка, то генерируется внутренняя ошибка 103 – ошибка записи задачника. Управление передается обработчику ошибок. Выполнение запроса прекращается. В противном случае выполнение запроса успешно завершается.

3.3.1.3 Закрыть задачник (tbDelete)

Описание запроса:
Pascal:

Function tbDelete : Logic;

C:

Logic tbDelete()

Назначение – удаляет из памяти ранее считанный задачник.
Описание исполнения.

1. Если Error <> 0, то выполнение запроса прекращается.
2. Если есть открытые сеансы, то возникает ошибка 105 – закрытие задачника при открытых сеансах. Управление передается обработчику ошибок. Выполнение запроса прекращается.
3. Задачник закрывается. Запрос успешно завершается.

3.3.2 Начало и конец сеанса

К этой группе запросов относятся два запроса, открывающие и закрывающие сеансы работы с задачником.

3.3.2.1 Начало сеанса (InitSession)

Описание запроса:
Pascal:

Function InitSession(NewColor : Color; Oper : Integer; Var Handle: Integer) : Logic;

C:

Logic InitSession(Color NewColor, Integer Oper, Integer* Handle)

Описание аргументов:

NewColor – цвет для отбора примеров задачника в текущую выборку.

Oper – операция для отбора в текущую выборку. Должна быть одной из констант CEqual, CIn, CInclude, Cxclude, CIntersect

Handle – номер сеанса. Начальное значение не важно. В этом аргументе возвращается номер сеанса.

Назначение – начинает сеанс. Отбирает текущую выборку.
Описание исполнения.

1. Если Error <> 0, то выполнение запроса прекращается.
2. Если аргумент Oper является недопустимым, то возникает ошибка 106 – недопустимый код операции при открытии сеанса, управление передается обработчику ошибок. Сеанс не открывается. Возвращается значение ложь.
3. Создается новый сеанс (в одно-сеансовых задачниках просто инициируется сеанс). Номер сеанса заносится в аргумент Handle.
4. Значения аргументов NewColor и Oper сохраняются во внутренних переменных задачника
5. Указателю текущего примера присваивается состояние «до первого примера»
6. InitSession := Next(Handle) – результат выполнения запроса совпадает с результатом выполнения вызванного запроса «Следующий пример».

3.3.2.2 Конец сеанса (EndSession)

Описание запроса:
Pascal:

Procedure EndSession(Handle : Integer);

C:

void EndSession(Integer Handle)

Назначение – закрывает сеанс.
Описание аргументов:

Handle – номер сеанса.

Описание исполнения.

1. Если Error <> 0, то выполнение запроса прекращается.
2. Если аргумент Handle не корректен возникает ошибка 107 – неверный номер сеанса. Управление передается обработчику ошибок. Выполнение запроса прекращается.
3. Освобождается вся память, взятая для выполнения сеанса. После этого сеанс завершается.

3.3.3 Перемещение по примерам

В эту группу запросов входят запросы позволяющие управлять положением текущего указателя в текущей выборке.

3.3.3.1 В начало (Home)

Описание запроса:
Pascal:

```
Function Home( Handle : Integer ) : Logic;
```

C:

```
Logic Home(Integer Handle)
```

Описание аргументов:
Handle – номер сеанса.
Назначение – делает текущим первый пример текущей выборки.
Описание исполнения.

1. Если Error <> 0, то выполнение запроса прекращается.
2. Если аргумент Handle не корректен возникает ошибка 107 – неверный номер сеанса. Управление передается обработчику ошибок. Выполнение запроса прекращается.
3. Указателю на текущий пример присваивается значение «до первого примера»
4. Home := Next(Handle) – результат выполнения запроса совпадает с результатом выполнения вызванного запроса «Следующий»

3.3.3.2 В конец (End)

Описание запроса:
Pascal:

```
Function End( Handle : Integer ) : Logic;
```

C:

```
Logic End(Integer Handle)
```

Описание аргументов:
Handle – номер сеанса.
Назначение – делает текущим последний пример текущей выборки.
Описание исполнения.

1. Если Error <> 0, то выполнение запроса прекращается.
2. Если аргумент Handle не корректен возникает ошибка 107 – неверный номер сеанса. Управление передается обработчику ошибок. Выполнение запроса прекращается.
3. Указателю на текущий пример присваивается значение «после последнего примера»
4. Home := Prev(Handle) – результат выполнения запроса совпадает с результатом выполнения вызванного запроса «Предыдущий»

3.3.3.3 Следующий (Next)

Описание запроса:
Pascal:

```
Function Next( Handle : Integer ) : Logic;
```

C:

```
Logic Next(Integer Handle)
```

Описание аргументов:
Handle – номер сеанса.
Назначение – делает текущим следующий пример текущей выборки.
Описание исполнения.

1. Если Error <> 0, то выполнение запроса прекращается.
2. Если аргумент Handle не корректен возникает ошибка 107 – неверный номер сеанса. Управление передается обработчику ошибок. Выполнение запроса прекращается.
3. Если значение указателя равно «после последнего примера», то возникает ошибка 108 – переход за конечную границу текущей выборки, и управление передается обработчику ошибок. В случае

возврата управления в запрос, происходит немедленный выход из запроса с возвращением значения ложь.

4. Если значение указателя текущего примера равно «до первого примера», то присваиваем указателю адрес первого примера задачника. Если адрес в переменной в задачнике нет примеров, то возникает ошибка 108 – переход за конечную границу текущей выборки, и управление передается обработчику ошибок. В случае возврата управления в запрос, происходит немедленный выход из запроса с возвращением значения ложь. В противном случае переходим к шагу 6
5. Указатель перемещается на следующий пример задачника. Если следующего примера задачника нет, то указателю присваивается значение «после последнего примера».
6. Переходим к шагу 5, если не верно условие:

((GetColor Oper NewColor) And Last,

где Oper и NewColor – аргументы запроса InitSession, которым был открыт данный сеанс.

7. Next := Not Last (Переход к следующему примеру завершился удачно, если указатель не установлен в значение «после последнего примера»).

3.3.3.4 Предыдущий (Prev)

Описание запроса:

Pascal:

Function Prev(Handle : Integer): Logic;

C:

Logic Prev(Integer Handle)

Описание аргументов:

Handle – номер сеанса.

Назначение – делает текущим предыдущий пример текущей выборки.

Описание исполнения.

1. Если Error <> 0, то выполнение запроса прекращается.
2. Если аргумент Handle не корректен возникает ошибка 107 – неверный номер сеанса. Управление передается обработчику ошибок. Выполнение запроса прекращается.
3. Если значение указателя равно «до первого примера», то возникает ошибка 109 – переход за начальную границу текущей выборки, и управление передается обработчику ошибок. В случае возврата управления в запрос, происходит немедленный выход из запроса с возвращением значения ложь.
4. Если значение указателя равно «после последнего примера», то присваиваем указателю адрес последнего примера задачника. Если в задачнике нет примеров, то возникает ошибка 109 – переход за начальную границу текущей выборки, и управление передается обработчику ошибок. В случае возврата управления в запрос, происходит немедленный выход из запроса с возвращением значения ложь.
5. В противном случае шаг 7.
6. Указатель перемещается на предыдущий пример задачника. Если предыдущего примера задачника нет, то указателю присваивается значение «до первого примера».
7. Шаг 6 повторяется до тех пор, пока не выполнится условие:

((GetColor Oper NewColor) And First

8. Next := Not Last (Переход к следующему примеру завершился удачно, если указатель не установлен в значение «после последнего примера»).

3.3.3.5 Конец (Last)

Описание запроса:

Pascal:

Function Last(Handle : Integer) : Logic;

C:

Logic Last(Integer Handle)

Описание аргументов:

Handle – номер сеанса.

Назначение – возвращает значение истина, если текущим является состояние «после последнего примера», и ложь – в противном случае.

Описание исполнения.

1. Если Error <> 0, то выполнение запроса прекращается.
2. Если аргумент Handle не корректен возникает ошибка 107 – неверный номер сеанса. Управление передается обработчику ошибок. Выполнение запроса прекращается.

3. Возвращает значение истина, если текущим является состояние «после последнего примера», и ложь – в противном случае.

3.3.3.6 Начало (First)

Описание запроса:

Pascal:

Function First(Handle : Integer): Logic;

C:

Logic First(Integer Handle)

Описание аргументов:

Handle – номер сеанса.

Назначение – возвращает значение истина, если текущим является состояние «перед первым примером», и ложь в противном случае.

Описание исполнения.

1. Если Error <> 0, то выполнение запроса прекращается.
2. Если аргумент Handle не корректен возникает ошибка 107 – неверный номер сеанса. Управление передается обработчику ошибок. Выполнение запроса прекращается.
3. Возвращает значение истина, если текущим является состояние «перед первым примером», и ложь в противном случае.

3.3.3.7 Пример номер (Example)

Описание запроса:

Pascal:

Function Example(Number : Long; Handle : Integer) : Logic;

C:

Logic Example(Long Number, Integer Handle)

Описание аргументов:

Number – номер примера, который должен быть сделан текущим. Нумерация примеров ведется с единицы.

Handle – номер сеанса.

Назначение – делает текущим пример текущей выборки с указанным номером.

Описание исполнения.

1. Если Error <> 0, то выполнение запроса прекращается.
2. Если аргумент Handle не корректен возникает ошибка 107 – неверный номер сеанса. Управление передается обработчику ошибок. Выполнение запроса прекращается.
3. Указатель устанавливается в состояние «до первого примера».
4. Number раз выполняем запрос Next.
5. Example := Not Last (Если не установлено состояние «после последнего примера», то запрос выполнен успешно).

3.3.4 Определение, получение и изменение данных

К данной группе запросов относятся запросы позволяющие получать данные из задачника, заносить данные в задачник и сбросить предобработку (необходимо выполнить данный запрос после изменений в данных или предобработчике, если задачник хранит векторы предобработанных данных)

3.3.4.1 Дать пример (Get)

Описание запроса:

Pascal:

Function Get(Handle : Integer; Var Data : PRealArray; What : Integer) : Logic;

C:

Logic Get(Integer Handle, PRealArray* Data, Integer What)

Описание аргументов:

Handle – номер сеанса;

Data – указатель на массив, в котором должны быть возвращены данныt;

What – одна из предопределенных констант tbColor, tbInput, tbPrepared, tbAnswers, tbReliability, tbCalcAnswers, tbCalcReliability, tbWeight, tbEstimation, tbComment

Назначение – возвращает всю указанную в запросе «Параметры примера» информацию.

Описание исполнения.

1. Если Error <> 0, то выполнение запроса прекращается.
2. Если аргумент Handle не корректен возникает ошибка 107 – неверный номер сеанса. Управление передается обработчику ошибок. Выполнение запроса прекращается.
3. Если аргумент What имеет недопустимое значение, то возникает ошибка 110 – неверный тип вектора в запросе Get. Управление передается обработчику ошибок. Выполнение запроса прекращается.
4. Если текущий указатель указывает на одно из состояний «до первого примера» или «после последнего примера», то возникает ошибка 111 – попытка чтения до или после текущей выборки. Управление передается обработчику ошибок. Запрос завершается неуспешно.
5. Если в аргументе What указан вектор предобработанных данных, но в текущем примере он отсутствует, то генерируется запрос предобработать данные. Если предобработка завершается успешно, то полученный вектор предобработанных данных включается в пример, в противном случае выполнение запроса прекращается. Возвращается значение ложь.
6. В элементы массива, на который указывает аргумент Data, копируются данные из того вектора данных текущего примера, который указан в аргументе What. Если требуемый вектор в задачнике отсутствует, то возникает ошибка 112 – данные отсутствуют и запрос завершается со значением ложь. В противном случае запрос успешно завершается.

3.3.4.2 Обновить данные (Put)

Описание запроса:
Pascal:

```
Function Put( Handle : Integer; Data : PRealArray; What : Integer ) : Logic;
```

C:

```
Logic Put(Integer Handle, PRealArray Data, Integer What)
```

Описание аргументов:
Handle – номер сеанса
Data – указатель на массив, в котором переданы данные, которые должны быть занесены в задачник.
What – одна из предопределенных констант tbColor, tbInput, tbPrepared, tbAnswers, tbReliability, tbCalcAnswers, tbCalcReliability, tbWeight, tbEstimation, tbComment
Назначение – обновить данные текущего примера
Описание исполнения.

1. Если Error <> 0, то выполнение запроса прекращается.
2. Если аргумент Handle не корректен возникает ошибка 107 – неверный номер сеанса. Управление передается обработчику ошибок. Выполнение запроса прекращается.
3. Если аргумент What имеет недопустимое значение, то возникает ошибка 113 – неверный тип вектора в запросе Put. Управление передается обработчику ошибок. Выполнение запроса прекращается.
4. Если текущий указатель указывает на одно из состояний «до первого примера» или «после последнего примера», то возникает ошибка 111 – попытка чтения до или после текущей выборки. Управление передается обработчику ошибок. Запрос завершается неуспешно.
5. Если устанавливается вектор входных данных, то для текущего примера должен быть освобожден вектор предобработанных данных.
6. В данные примера копируются значения, указанные в массиве Data. Запрос успешно завершается.

3.3.4.3 Сбросить предобработку (RemovePrepare)

Описание запроса:
Pascal:

```
Procedure RemovePrepare;
```

C:

```
void RemovePrepare()
```

Назначение – отмена предобработки всех ранее предобработанных примеров.
Описание исполнения.

1. Если Error <> 0, то выполнение запроса прекращается.
2. У всех примеров задачника освобождаются вектора предобработанных данных.

3.3.5 Окраска примеров

В данный раздел помещены запросы для работы с цветами. Отметим, что цвет примера, возвращаемый запросом GetColor можно получить также с помощью запроса Get.

3.3.5.1 Дать цвет примера (GetColor)

Описание запроса:

Pascal:

Function GetColor(Handle : Integer) : Color;

C:

Logic GetColor(Integer Handle)

Описание аргументов:

Handle – номер сеанса

Назначение – возвращает цвет текущего примера.

Описание исполнения.

1. Если Error <> 0, то выполнение запроса прекращается.
2. Если аргумент Handle не корректен возникает ошибка 107 – неверный номер сеанса. Управление передается обработчику ошибок. Выполнение запроса прекращается.
3. Если текущий указатель указывает на одно из состояний «до первого примера» или «после последнего примера», то возникает ошибка 111 – попытка чтения до или после текущей выборки. Управление передается обработчику ошибок. Запрос завершается неуспешно.
4. Возвращается цвет текущего примера.

3.3.5.2 Покрасить пример (PaintCurrent)

Описание запроса:

Pascal:

Function PaintCurrent(Handle : Integer; NewColor, ColorMask : Color; Oper : Integer) : Logic;

C:

Logic PaintCurrent(Integer Handle, Color NewColor, Color ColorMask, Integer Oper)

Описание аргументов:

Handle – номер сеанса.

NewColor – новый цвет для окраски примера.

ColorMask – маска цвета для окраски примера.

Oper – операция, используемая при окраске примера. Должна быть одной из констант COr, CAnd, CXor, CNot.

Назначение – изменяет цвет текущего примера.

Описание исполнения.

1. Если Error <> 0, то выполнение запроса прекращается.
2. Если аргумент Handle не корректен возникает ошибка 107 – неверный номер сеанса. Управление передается обработчику ошибок. Выполнение запроса прекращается.
3. Если Oper не корректен, то возникает ошибка 114 – неверная операция окраски примера. Управление передается обработчику ошибок. Запрос завершается со значением ложь.
4. Новый цвет примера := (Старый цвет примера And ColorMask) Oper NewColor

3.3.5.3 Ошибки компонента задачника

В табл. 4 приведен полный список ошибок, которые могут возникать при выполнении запросов компонентом задачник, и действия стандартного обработчика ошибок.

Таблица 4.

Ошибки компонента задачник и действия стандартного обработчика ошибок.

№Название ошибкиСтандартная	обработка
101 Запрос при отсутствии задачникаЗанесение	номера в Error
102 Ошибка чтения задачникаЗанесение	номера в Error
103 Ошибка записи задачникаЗанесение	номера в Error
104 Попытка считывания задачника при открытых сеансах ранее считанного задачника	Занесение номера в Error
105 Закрытие задачника при открытых сеансахЗанесение	номера в Error
106 Недопустимый код операции при открытии сеансаЗанесение	номера в Error
107 Неверный номер сеансаЗанесение	номера в Error
108 переход за конечную границу текущей выборкиИгнорируется	
109 Переход за начальную границу текущей выборкиИгнорируется	
110 Неверный тип вектора в запросе Get	Занесение номера в Error
111 Попытка чтения до или после текущей выборкиЗанесение	номера в Error
112 Данные отсутствуютИгнорируется	
113 Неверный тип вектора в запросе Put	Занесение номера в Error
114 Неверная операция окраски примераЗанесение	номера в Error

4. Предобработчик

Данная глава посвящена компоненту предобработчик. В ней рассматриваются различные аспекты предобработки входных данных для нейронных сетей. Существует множество различных видов нейронных сетей (см. главу «Описание нейронных сетей»). Однако, для большинства нейронных сетей характерно наличие такого интервала входных сигналов, в пределах которого сигналы различимы. Для различных нейронных сетей эти интервалы различны. Большинство работающих с нейронными сетями прекрасно осведомлены об этом их свойстве, но до сих пор не предпринималось никаких попыток как-либо формализовать или унифицировать подходы к предобработке входных сигналов. В данной главе дан один из возможных формализмов этой задачи. За рамками рассмотрения осталась предобработка графической информации. Наиболее мощные и интересные способы обработки графической информации описаны в [90]

4.1 Описание предобработчика

В этой части главы будут описаны различные виды входных сигналов и способы их предобработки. В качестве примера будут рассмотрены сети с сигмоидными нелинейными преобразователями. Однако, описываемые способы предобработки применимы для сетей с произвольными нелинейными преобразователями. Единственным исключением является раздел «Оценка способности сети решить задачу», который применим только для сетей с нелинейными преобразователями, непрерывно зависящими от своих аргументов.

4.1.1 Нейрон

Нейроны, используемые в большинстве нейронных сетей, имеют структуру, приведенную на рис. 1. На рис. 1 использованы следующие обозначения:

x – вектор входных сигналов нейрона;
α – вектор синаптических весов нейрона;
Σ – входной сумматор нейрона;
$p=(\alpha,x)$ – выходной сигнал входного сумматора;
σ – функциональный преобразователь;
y – выходной сигнал нейрона.

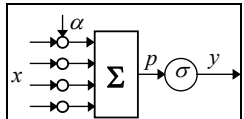


Рис. 1. Пример нейрона

Обычно нейронные сети называют по виду функции $\sigma(p)$. Хорошо известны и наиболее часто используются два вида сигмоидных сетей:

$$S_1: \quad \sigma(p)=1/(1+\exp(-cp)),$$
$$S_2: \quad \sigma(p)=p/(c+|p|),$$

где c - параметр, называемый «характеристикой нейрона». Обе функции имеют похожие графики.

Каждому типу нейрона соответствует свой интервал приемлемых входных данных. Как правило, этот диапазон либо совпадает с диапазоном выдаваемых выходных сигналов (например для сигмоидных нейронов с функцией S_1), либо является объединением диапазона выдаваемых выходных сигналов и отрезка, симметричного ему относительно нуля (например, для сигмоидных нейронов с функцией S_2), Этот диапазон будем обозначать как $[a,b]$

4.1.2 Различимость входных данных

Очевидно, что входные данные должны быть различимы. В данном разделе будут приведены соображения, исходя из которых, следует выбирать диапазон входных данных. Пусть одним из входных параметров нейронной сети является температура в градусах Кельвина. Если речь идет о температурах

Таблица 1

Величина входного сигнала	Нейрон типа S_1				Нейрон типа S_2			
	$c=0.1$	$c=0.5$	$c=1$	$c=2$	$c=0.1$	$c=0.5$	$c=1$	$c=2$
250	1.0	1.0	1.0	1.0	0.99960	0.99800	0.99602	0.99206
275	1.0	1.0	1.0	1.0	0.99964	0.99819	0.99638	0.99278
300	1.0	1.0	1.0	1.0	0.99967	0.99834	0.99668	0.99338

близких к нормальной, то входные сигналы изменяются от 250 до 300 градусов. Пусть сигнал подается прямо на нейрон (синаптический вес равен единице). Выходные сигналы нейронов с различными параметрами приведены в табл. 1.

Совершенно очевидно, что нейронная сеть просто неспособна научиться надежно различать эти сигналы (если вообще способна научиться их различать!). Если использовать нейроны с входными синапсами, не равными единице, то нейронная сеть сможет отмасштабировать входные сигналы так, чтобы они стали различимы, но при этом будет задействована только часть диапазона приемлемых входных данных - все входные сигналы будут иметь один знак. Кроме того, все подаваемые сигналы будут занимать лишь малую часть этого диапазона. Например, если мы отмасштабируем температуры так, чтобы 300 соответствовала величина суммарного входного сигнала равная 1 (величина входного синапса равна 1/300), то реально подаваемые сигналы займут лишь одну шестую часть интервала [0,1] и одну двенадцатую интервала [-1,1]. Получаемые при этом при этом величины выходных сигналов нейронов приведены в табл. 2.

Таблица 2

Величина входного сигнала	Нейрон типа S_1				Нейрон типа S_2			
	$c = 0.1$	$c = 0.5$	$c = 1$	$c = 2$	$c = 0.1$	$c = 0.5$	$c = 1$	$c = 2$
250 (0.83)	0.52074	0.60229	0.69636	0.84024	0.89286	0.62500	0.45455	0.29412
275 (0.91)	0.52273	0.61183	0.71300	0.86057	0.90164	0.64706	0.47826	0.31429
300 (1.0)	0.52498	0.62246	0.73106	0.88080	0.90909	0.66667	0.50000	0.33333

Сигналы, приведенные в табл. 2 различаются намного сильнее соответствующих сигналов из табл. 1. Таким образом, необходимо заранее позаботиться о масштабировании и сдвиге сигналов, чтобы максимально полно использовать диапазон приемлемых входных сигналов. Опыт использования нейронных сетей с входными синапсами свидетельствует о том, что в подавляющем большинстве случаев предварительное масштабирование и сдвиг входных сигналов сильно облегчает обучение нейронных сетей. Если заранее произвести операции масштабирования и сдвига входных сигналов, то величины выходных сигналов нейронов даже при отсутствии входных синапсов будут различаться еще сильнее (см. табл. 3).

Таблица 3

Величина входного сигнала	Нейрон типа S_1				Нейрон типа S_2			
	$c = 0.1$	$c = 0.5$	$c = 1$	$c = 2$	$c = 0.1$	$c = 0.5$	$c = 1$	$c = 2$
250 (-1)	0.47502	0.37754	0.26894	0.11920	-0.9091	-0.6667	-0.5000	-0.3333
275 (0)	0.50000	0.50000	0.50000	0.50000	0.0000	0.0000	0.0000	0.0000
300 (1)	0.52498	0.62246	0.73106	0.88080	0.9091	0.6667	0.5000	0.3333

Величину диапазона различимых входных сигналов можно определять различными способами. На практике в качестве диапазона различимых входных сигналов обычно используется диапазон приемлемых входных данных, исходя из того соображения, что если данные из этого интервала хороши для промежуточных нейронов, то они хороши и для входных.

Другой способ определения различимости входных сигналов приведен в разделе «Оценка способности сети решить задачу».

4.1.3 Классификация компонентов входных данных

Информация поступает к нейронной сети в виде набора ответов на некоторый список вопросов. Можно выделить три основных типа ответов (вопросов).

1. Бинарный признак (возможен только один из ответов – истина или ложь).
2. Качественный признак (принимает конечное число значений).
3. Число.

Ответ типа качественный признак - это ответ с конечным числом состояний. Причем нельзя ввести осмысленное расстояние между состояниями. Примером качественного признака может служить состояние больного - тяжелый, средний, легкий. Действительно, нельзя сказать, что расстояние от легкого больного до среднего больше, меньше или равно расстоянию от среднего больного до тяжелого.

Все качественные признаки можно в свою очередь разбить на три класса.

1. Упорядоченные признаки.
2. Неупорядоченные признаки.
3. Частично упорядоченные признаки.

Упорядоченным признаком называется такой признак, для любых двух состояний которого можно сказать, что одно из них предшествует другому. Тот факт, что состояние x предшествует состоянию y, будем обозначать следующим образом – $x < y$. Примером упорядоченного признака может служить состояние больного. Действительно, все состояния можно упорядочить по тяжести заболевания:

легкий больной < средний больной < тяжелый больной

Признак называют неупорядоченным, если никакие два состояния нельзя связать естественным в контексте задачи отношением порядка. Примером неупорядоченного признака может служить ответ на вопрос "Ваш любимый цвет?".

Признак называется частично упорядоченным, если для каждого состояния существует другое состояние, с которым оно связано отношением порядка. Примером частично упорядоченного признака является ответ на вопрос "Какой цвет Вы видите на экране монитора?", преследующий цель определение восприимчивости к интенсивностям основных цветов. Действительно, все множество из шестнадцати состояний разбивается на несколько цепочек:

Черный < Синий < Голубой < Белый;

Черный < Красный < Ярко красный < Белый;

Черный < Зеленый < Ярко зеленый < Белый;

Черный < Фиолетовый < Ярко фиолетовый < Белый

и т.д. Однако, между состояниями Синий и Красный отношения порядка нет.

Известно, что любой частично упорядоченный признак можно представить в виде комбинации нескольких упорядоченных и неупорядоченных признаков. Так, рассмотренный выше частично упорядоченный признак распадается на три упорядоченных признака: интенсивность синего, красного и зеленого цветов. Каждый из этих признаков является упорядоченным (цепочки порядка для этих признаков приведены в первых трех строчках рассмотрения примера). Каждое состояние исходного качественного признака описывается тройкой состояний полученных качественных признаков. Так, например, состояние Фиолетовый описывается в виде (Синий, Красный, Черный).

Исходя из вышесказанного, далее будет рассмотрено только кодирование упорядоченных и неупорядоченных признаков.

4.1.4 Кодирование бинарных признаков

Бинарные признаки характеризуются наличием только двух состояний – истина и ложь. Однако даже такие простые данные могут иметь два разных смысла. Значение истина означает наличие у описываемого объекта какого-либо свойства. А ответ ложь может означать либо (1) отсутствие этого свойства, либо (2) наличие другого свойства. В зависимости от смысловой нагрузки значения ложь, и учитывая заданный диапазон $[a,b]$, рекомендуемые способы кодирования бинарного признака приведены в табл. 4.

Таблица 4

Кодирование бинарного признака

Смысл значения ложьВеличина	входного сигнала для значения признака	
	ИстинаЛожь	
Отсутствие заданного свойства при $b = 0$	a	0
Отсутствие заданного свойства при $b \neq 0$	b	0
Наличие другого свойства	b	a

4.1.5 Кодирование неупорядоченных качественных признаков

Поскольку никакие два состояния неупорядоченного признака не связаны отношением порядка, то было бы неразумным кодировать их разными величинами одного входного сигнала нейронной сети. Поэтому, для кодирования качественных признаков рекомендуется использовать столько входных сигналов, сколько состояний у этого качественного признака. Каждый входной сигнал соответствует определенному состоянию. Так если набор всех состояний рассматриваемого признака обозначить через $\alpha_1, \alpha_2, \ldots, \alpha_n$, то рекомендуемая таблица кодировки имеет вид, приведенный в табл. 5.

Таблица 5.

Кодирование неупорядоченного качественного признака

СостояниеВектор	входных сигналов
α_1	$(b,a,a,\ldots,a)$
α_2	$(a,b,a,\ldots,a)$
α_n	$(a,a,\ldots,a,b)$

4.1.6 Кодирование упорядоченных частных признаков

Упорядоченные частные признаки, в отличие от неупорядоченных, имеют отношение порядка между состояниями. Однако кодирование их разными значениями одного входного сигнала неразумно из-за того, что расстояние между состояниями не определено, а такое кодирование эти расстояния задает явным образом. Поэтому, упорядоченные частные признаки рекомендуется кодировать в виде стольких входных сигналов, сколько состояний у признака. Но, в отличие от неупорядоченных признаков, накапливать число сигналов с максимальным значением. Для случая, когда все состояния обозначены через $\alpha_1 < \alpha_2 < \ldots < \alpha_n$, рекомендуемая таблица кодировки приведена в табл. 6.

Таблица 6.
Кодирование упорядоченного качественного признака

Состояние	Вектор входных сигналов
α_1	$(b,a,a,\ldots,a)$
α_2	$(b,b,a,\ldots,a)$
α_n	$(b,b,\ldots,b,b)$

4.1.7 Числовые признаки

При предобработке численных сигналов необходимо учитывать содержательное значение признака, расположение значений признака в интервале значений, точность измерения значений признака. Продемонстрируем это на примерах.

Содержательное значение признака. Если входными данными сети является угол между двумя направлениями, например, направление ветра, то ни в коем случае не следует подавать на вход сети значение угла (не важно в градусах или радианах). Такая подача приведет к необходимости "уяснения" сетью того факта, что 0 градусов и 360 градусов одно и тоже. Разумнее выглядит подача в качестве входных данных синуса и косинуса этого угла. Число входных сигналов сети увеличивается, но зато близкие значения признака кодируются близкими входными сигналами.

Точность измерения признака. Так в метеорологии используется всего восемь направлений ветра. Значит, при подаче входного сигнала сети необходимо подавать не угол, а всего лишь информацию о том, в какой из восьми секторов этот угол попадает. Но тогда имеет смысл рассматривать направление ветра не как числовой параметр, а как неупорядоченный качественный признак с восемью состояниями.

Расположение значений признака в интервале значений. Следует рассмотреть вопрос о равнозначности изменения значения признака на некоторую величину в разных частях интервала значений признака. Как правило, это связано с косвенными измерениями (вместо одной величины измеряется другая). Например, сила притяжения двух небесных тел при условии постоянства массы однозначно характеризуется расстоянием между ними. Пусть рассматриваются расстояния от 1 до 100 метров. Легко понять, что при изменении расстояния с 1 до 2 метров, сила притяжения изменится в четыре раза, а при изменении с 99 до 100 метров – в 1.02 раза. Следовательно, вместо подачи расстояния следует подавать обратный квадрат расстояния $c' = 1/c^2$.

4.1.8 Простейшая предобработка числовых признаков

Как уже отмечалось в разделе «Различимость входных данных» числовые сигналы рекомендуется масштабировать и сдвигать так, чтобы весь диапазон значений попадал в диапазон приемлемых входных сигналов. Эта предобработка проста и задается следующей формулой:

$$c' = \frac{(c - c_{\min})(b - a)}{(c_{\max} - c_{\min})} + a, \qquad (1)$$

где $[a,b]$ - диапазон приемлемых входных сигналов, $[c_{\min}, c_{\max}]$ – диапазон значений признака c, c' – предобработанный сигнал, который будет подан на вход сети. Предобработку входного сигнала по формуле (1) будем называть простейшей предобработкой.

4.1.9 Оценка способности сети решить задачу

В данном разделе рассматриваются только сети, все элементы которых непрерывно зависят от своих аргументов (см. главу «Описание нейронных сетей»). Предполагается, что все входные данные предобработаны так, что все входные сигналы сети лежат в диапазоне приемлемых входных сигналов $[a,b]$. Будем обозначать вектора входных сигналов через x^i, а требуемые ответы сети через f^i. Компоненты векторов будем обозначать нижним индексом, например, компоненты входного вектора

через x_j^i. Будем полагать, что в каждом примере ответ является вектором чисел из диапазона приемлемых сигналов $[a,b]$. В случае обучения сети задаче классификации требуемый ответ зависит от вида используемого интерпретатора ответа (см. главу «Оценка и Интерпретатор ответа»).

Нейронная сеть вычисляет некоторую вектор-функцию F от входных сигналов. Эта функция зависит от параметров сети. Обучение сети состоит в подборе такого набора параметров сети, чтобы величина $\sum_{i,j}\left(F_j\left(x^i\right)-f_j^i\right)^2$ была минимальной (в идеале равна нулю). Для того чтобы нейронная сеть могла хорошо приблизить заданную таблично функцию f необходимо, чтобы реализуемая сетью функция F при изменении входных сигналов с x^i на x^j могла изменить значение с f^i на f^j. Очевидно, что наиболее трудным для сети должно быть приближение функции в точках, в которых при малом изменении входных сигналов происходит большое изменение значения функции. Таким образом, наибольшую сложность будет представлять приближение функции f в точках, в которых достигает максимума выражение $\frac{\left\|f^i-f^j\right\|}{\left\|x^i-x^j\right\|}$. Для аналитически заданных функций величина $\sup_{x,y}\frac{\left\|f(x)-f(y)\right\|}{\|x-y\|}$ называется константой Липшица. Исходя из этих соображения можно дать следующее определение сложности задачи.

Сложность аппроксимации таблично заданной функции f, которая в точках x^i принимает значения f^i, задается выборочной оценкой константы Липшица, вычисляемой по следующей формуле:

$$\Lambda_t=\max_{i\neq j}\frac{\left\|f^i-f^j\right\|}{\left\|x^i-x^j\right\|} \tag{2}$$

Оценка (2) является оценкой константы Липшица аппроксимируемой функции снизу.

Для того, чтобы оценить способность сети заданной конфигурации решить задачу, необходимо оценить константу Липшица сети и сравнить ее с выборочной оценкой (2). Константа Липшица сети вычисляется по следующей формуле:

$$\Lambda_n=\sup_{x,y}\frac{\left\|F(x)-F(y)\right\|}{\|x-y\|} \tag{3}$$

В формулах (2) и (3) можно использовать произвольные нормы. Однако для нейронных сетей наиболее удобной является евклидова норма. Далее везде используется евклидова норма.

В следующем разделе описан способ вычисления оценки константы Липшица сети (3) сверху. Очевидно, что в случае $\Lambda_n<\Lambda_t$ сеть принципиально не способна решить задачу аппроксимации функции f.

4.1.9.1 Оценка константы Липшица сети

Оценку константы Липшица сети будем строить в соответствии с принципом иерархического устройства сети, описанным в главе «Описание нейронных сетей». При этом потребуются следующие правила.

1. Для композиции функций $f\circ g=f\left(g(x)\right)$ константа Липшица оценивается как произведение констант Липшица:
$$\Lambda_{f\circ g}\leq\Lambda_f\Lambda_g. \tag{4}$$
2. Для вектор-функции $f=\left(f_1,f_2,\ldots,f_n\right)$ константа Липшица равна:

$$\Lambda_f = \sqrt{\sum_{i=1}^{n} \Lambda_{f_i}^2} \ . \tag{5}$$

4.1.9.2 Способ вычисления константы Липшица

Для непрерывных функций константа Липшица является максимумом производной в направлении $r = (r_1, \ldots, r_n)$ по всем точкам и всем направлениям. При этом вектор направления имеет единичную длину: $\sum_{i=1}^{n} r_i^2 = 1$. Напомним формулу производной функции $f(x_1, \ldots, x_n)$ в направлении r:

$$\frac{\partial f}{\partial r} = \sum_{i=1}^{n} r_i \frac{\partial f}{\partial x_i} \tag{6}$$

4.1.9.3 Синапс

Обозначим входной сигнал синапса через x, а синаптический вес через α. Тогда выходной сигнал синапса равен αx. Поскольку синапс является функцией одной переменной, константа Липшица равна максимуму модуля производной – модулю синаптического веса:

$$\Lambda_s = |\alpha| \tag{7}.$$

4.1.9.4 Умножитель

Обозначим входные сигналы умножителя через x_1, x_2. Тогда выходной сигнал умножителя равен $f_* = x_1 x_2$. Используя (6) получаем $\Lambda_{f_*} = \sup_{x,r} |r_1 x_2 + r_2 x_1|$. Выражение $r_1 x_2 + r_2 x_1$ является скалярным произведением векторов (r_1, r_2) и, учитывая единичную длину вектора r, достигает максимума, когда эти векторы сонаправлены. То есть при векторе

$$r = \left(\frac{x_2}{\|x\|}, \frac{x_1}{\|x\|} \right), \ \|x\| = \sqrt{x_1^2 + x_1^2} \ .$$

Используя это выражение, можно записать константу Липшица для умножителя:

$$\Lambda_{f_*} = \sup_{x,r} |r_1 x_2 + r_2 x_1| = \sup_{x} \frac{\left| x_2^2 + x_1^2 \right|}{\|x\|} = \sup_{x} \|x\| \ . \tag{8}$$

Если входные сигналы умножителя принадлежат интервалу $[a, b]$, то константа Липшица для умножителя может быть записана в следующем виде:

$$\Lambda_{f_*} = \sqrt{2} \max\{|a|, |b|\} \ . \tag{9}$$

4.1.9.5 Точка ветвления

Поскольку в точке ветвления не происходит преобразования сигнала, то константа Липшица для нее равна единице.

4.1.9.6 Сумматор

Производная суммы по любому из слагаемых равна единице. В соответствии с (6) получаем:

$$\Lambda_\Sigma = \sup_{r} \left| \sum_{i=1}^{n} r_i \right| = \sqrt{n} \ , \tag{10}$$

поскольку максимум суммы при ограничении на сумму квадратов достигается при одинаковых слагаемых.

4.1.9.7 Нелинейный Паде преобразователь

Нелинейный Паде преобразователь или Паде элемент имеет два входных сигнала и один выходной. Обозначим входные сигналы через x_1, x_2. Используя (6) можно записать константу Липшица в следующем виде:

$$\Lambda_\pi = \sup_{r,x}\left|\frac{r_1 x_2}{x_2^2} - \frac{r_2 x_1}{x_2^2}\right| = \sup_{r,x}\left|\frac{r_1 x_2 - r_2 x_1}{x_2^2}\right|.$$

Знаменатель выражения под знаком модуля не зависит от направления, а числитель можно преобразовать так же, как и для умножителя. После преобразования получаем:

$$\Lambda_\pi = \sup_{x}\frac{\|x\|}{x_2^2} \tag{11}$$

4.1.9.8 Нелинейный сигмоидный преобразователь

Нелинейный сигмоидный преобразователь, как и любой другой нелинейный преобразователь, имеющий один входной сигнал x, имеет константу Липшица равную максимуму модуля производной:

$$\Lambda_\varphi = \max_x\left|\varphi'(x)\right|. \tag{12}$$

4.1.9.9 Адаптивный сумматор

Для адаптивного сумматора на n входов оценка константы Липшица, получаемая через представление его в виде суперпозиции слоя синапсов и простого сумматора, вычисляется следующим образом. Используя формулу (7) для синапсов и правило (5) для вектор-функции получаем следующую оценку константы Липшица слоя синапсов:

$$\Lambda_L = \sqrt{\sum_{i=1}^{n}\Lambda_{S_i}^2} = \sqrt{\sum_{i=1}^{n}|\alpha_i|^2} = \|\alpha\|.$$

Используя правило (4) для суперпозиции функций и оценку константы Липшица для простого сумматора (10) получаем:

$$\Lambda_A \le \Lambda_\Sigma \Lambda_L = \sqrt{n}\|\alpha\|. \tag{13}$$

Однако, если оценить константу Липшица адаптивного сумматора напрямую, то, используя (6) и тот факт, что при фиксированных длинах векторов скалярное произведение достигает максимума для сонаправленных векторов получаем:

$$\Lambda_A = \sup_{x,r}\left|\sum_{i=1}^{n} r_i\alpha_i\right| = \left|\sum_{i=1}^{n}\alpha_i\frac{\alpha_i}{\|\alpha\|}\right| = \|\alpha\|. \tag{14}$$

Очевидно, что оценка (14) точнее, чем оценка (13).

4.1.9.10 Константа Липшица сигмоидной сети

Рассмотрим слоистую сигмоидную сеть со следующими свойствами:

1. Число входных сигналов – n_0.
2. Число нейронов в i-м слое – n_i.
3. Каждый нейрон первого слоя получает все входные сигналы, а каждый нейрон любого другого слоя получает сигналы всех нейронов предыдущего слоя.
4. Все нейроны всех слоев имеют вид, приведенный на рис. 1 и имеют одинаковую характеристику.
5. Все синаптические веса ограничены по модулю единицей.
6. В сети m слоев.

В этом случае, учитывая формулы (4), (5), (12) и (14) константу Липшица i-о слоя можно оценить следующей величиной:

$$\Lambda_i \le \sqrt{\sum_{j=1}^{n_i}\left(\Lambda_\varphi\Lambda_{A_j}\right)^2} = \Lambda_\varphi\sqrt{\sum_{j=1}^{n_i}\left\|\alpha^j\right\|^2} \le \Lambda_\varphi\sqrt{n_{i-1}n_i}.$$

Используя формулу (4) получаем оценку константы Липшица всей сети:

$$\Lambda_n \le \prod_{i=1}^{m} \Lambda_i \le \Lambda_\varphi^m \sqrt{n_0 n_m} \prod_{i=1}^{m-1} n_i .$$

Если используется нейроны типа S_1, то $\Lambda_\varphi = c$ и оценка константы Липшица сети равна:

$$\Lambda_{S_1} \le c^m \sqrt{n_0 n_m} \prod_{i=1}^{m-1} n_i$$

Для нейронов типа S_2, то $\Lambda_\varphi = 1/c$ и оценка константы Липшица сети равна:

$$\Lambda_{S_1} \le c^{-m} \sqrt{n_0 n_m} \prod_{i=1}^{m-1} n_i$$

Обе формулы подтверждают экспериментально установленный факт, что чем круче характеристическая функция нейрона, тем более сложные функции (функции с большей константой Липшица) может аппроксимировать сеть с такими нейронами.

4.1.10 Предобработка, облегчающая обучение

При обучении нейронных сетей иногда возникают ситуации, когда дальнейшее обучение нейронной сети невозможно. В этом случае необходимо проанализировать причины. Возможно несколько видов анализа. Одной из возможных причин является высокая сложность задачи, определяемая как выборочная оценка константы Липшица.

Для упрощения задачи необходимо уменьшить выборочную оценку константы Липшица. Наиболее простой способ добиться этого – увеличить расстояние между входными сигналами. Рассмотрим пару примеров – x^i, x^j – таких, что $\Lambda_t = \dfrac{\left\| f^i - f^j \right\|}{\left\| x^i - x^j \right\|}$. Определим среди координат векторов x^i и x^j координату, в которой достигает минимума величина $\left| x_l^i - x_l^j \right|$, исключив из рассмотрения совпадающие координаты. Очевидно, что эта координата является «узким местом», определяющим сложность задачи. Следовательно, для уменьшения сложности задачи требуется увеличить расстояние между векторами x^i и x^j, а наиболее перспективной координатой для этого является l-я. Однако увеличение расстояние между x_l^i и x_l^j не всегда осмыслено. Дело в том, что все параметры, как правило, измеряются с конечной точностью. Поэтому, если величина $\left| x_l^i - x_l^j \right|$ меньше чем точность измерения l-го параметра, значения x_l^i и x_l^j можно считать совпадающими. Таким образом, для изменения масштаба надо выбирать тот из входных параметров, для которого значение $\left| x_l^i - x_l^j \right|$ минимально, но превышает точность измерения этого параметра.

Предположим, что все входные параметры предобработаны в соответствии с формулой (1). Перенумеруем примеры обучающего множества так, чтобы были верны следующие неравенства: $x_l^1 < x_l^2 <, \ldots, < x_l^N$, где N – число примеров в обучающем множестве. При этом, возможно, придется исключить ряд пар параметр-ответ с совпадающими значениями параметра. Если в какой-либо из таких пар значения ответов различаются, то это снижает возможную полезность данной процедуры.

Наиболее простой путь – разбить диапазон l-го параметра на два. Зададимся точкой x. Будем кодировать l-й

Таблица 7.

Кодирование параметра после разбиения на два сигнала

ЗначениеП	ервый сигналВторой	сигнал
$x_l^i < x$	$\dfrac{\left(x_l^i - a\right)\left(b-a\right)}{\left(x-a\right)} + a$	a
$x_l^i > x$	b	$\dfrac{\left(x_l^i - x\right)\left(b-a\right)}{\left(b-x\right)} + a$

параметр двумя входными сигналами в соответствии с табл. 7. При таком кодировании критерий Липшица, очевидно, уменьшится. Вопрос о выборе точки x может решаться по-разному. Простейший путь – положить $x = (a-b)/2$. Более сложный, но часто более эффективный – подбор x исходя из требования минимальности критерия Липшица.

Приведенный выше способ уменьшения критерия Липшица не единственный. В следующем разделе рассмотрен ряд способов предобработки, решающих ту же задачу.

4.1.11 Другие способы предобработки числовых признаков

В данном разделе будет рассмотрено три вида предобработки числовых признаков – модулярный, позиционный и функциональный. Основная идея этих методов предобработки состоит в том, чтобы сделать значимыми малые отличия больших величин. Действительно, пусть для ответа существенно изменение величины признака на единицу при значении признака порядка миллиона. Очевидно, что простейшая предобработка (1) сделает отличие в единицу неразличимым для нейронной сети при абсолютных значениях порядка миллиона.

Все эти виды предобработки обладают одним общим свойством – за счет кодирования входного признака несколькими сигналами они уменьшают сложность задачи (критерий Липшица).

4.1.11.1 Модулярная предобработка

Зададимся некоторым набором положительных чисел $y_1,\ldots,y_k$. Определим сравнение по модулю для действительных чисел следующим образом:

$$x \mod y = x - y \cdot Int(x/y), \tag{15}$$

где $Int(x)$ – функция, вычисляющая целую часть величины x путем отбрасывания дробной части. Очевидно, что величина $x \mod y$ лежит в интервале $(-y, y)$. Кодирование входного признака x при модулярной предобработке вектором z производится по следующей формуле:

$$z_i = \frac{((x \mod y_i) + y_i)(b-a)}{2y_i} + a. \tag{16}$$

Однако модулярная предобработка обладает одним отрицательным свойством – во всех случаях, когда $y_i \neq y_1^r$, при целом r, разрушается отношение предшествования чисел. В табл. 8 приведен пример векторов. Поэтому, модульная предобработка пригодна при предобработке тех признаков, у которых важна не абсолютная величина, а взаимоотношение этой величины с величинами y_1,K,y_k. Примером такого признака может служить угол между векторами, если в качестве величин y выбрать $y_i = \pi/i$.

Таблица 8.

Пример сигналов при модулярном вводе

x	x mod 3	x mod 5	x mod 7	x mod 11
5	2	0	5	5
10	1	0	3	10
15	0	0	1	3

4.1.11.2 Функциональная предобработка

Функциональная предобработка преследует единственную цель – снижение константы Липшица задачи. В разделе «Предобработка, облегчающая обучение», был приведен пример такой предобработки. Рассмотрим общий случай функциональной предобработки, отображающих входной признак x в k-мерный вектор z. Зададимся набором из k чисел, удовлетворяющих следующим условиям: $x_{\min} < y_1 < \ldots < y_{k-1} < y_k < x_{\max}$. Пусть φ – функция, определенная на интервале $[x_{\min} - y_k, x_{\max} - y_1]$, а $\varphi_{\min}, \varphi_{\max}$ – минимальное и максимальное значения функции φ на этом интервале. Тогда i-я координата вектора z вычисляется по следующей формуле:

$$z_i = \frac{(\varphi(x - y_i) - \varphi_{\min})(b-a)}{\varphi_{\max} - \varphi_{\min}} + a \tag{17}$$

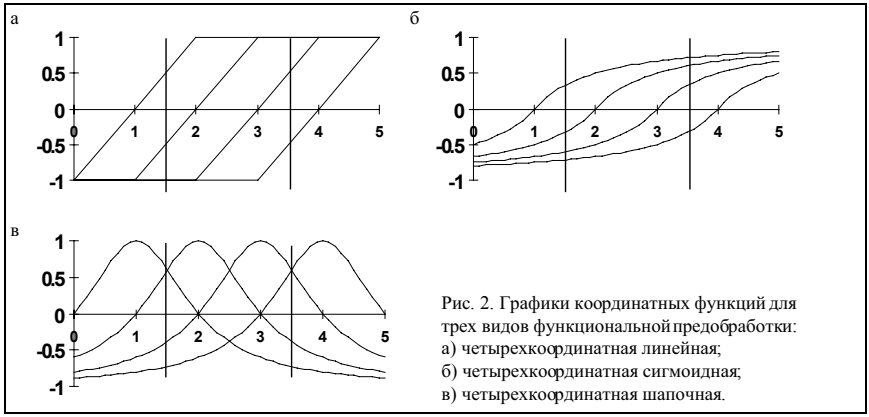


Рис. 2. Графики координатных функций для трех видов функциональной предобработки:
а) четырехкоординатная линейная;
б) четырехкоординатная сигмоидная;
в) четырехкоординатная шапочная.

Линейная предобработка. В линейной предобработке используется кусочно линейная функция:

$$z_i = \begin{cases} a, & x < a, \\ x, & a \le x \le b, \\ b, & b < x. \end{cases} \quad (18)$$

Графики функций $z_i(x)$ представлены на рис. 2а. Видно, что с увеличением значения признака x ни одна функция не убывает, а их сумма возрастает. В табл. 9 представлены значения этих функций для двух точек – $x_1 = 1.5$ и $x_2 = 3.5$.

Таблица 9

Пример функциональной предобработки числового признака $x \in [0,5]$, при условии, что сигналы нейронов принадлежат интервалу $[-1,1]$. В сигмоидной предобработке использована формула $\varphi(x) = x/(1+|x|)$, а в шапочной – $\varphi(x) = 2/(1+x^2) - 1$. Были выбраны четыре точки $y_i = i$.

x	$z_1(x)$	$z_2(x)$	$z_3(x)$	$z_4(x)$
Линейная предобработка				
1.5	0.5	-0.5	-1	-1
3.5	1	1	0.5	-0.5
Сигмоидная предобработка				
1.5	0.3333	-0.3333	-0.6	-0.7142
3.5	0.7142	0.6	0.3333	-0.3333
Шапочная предобработка				
1.5	0.6	0.6	-0.3846	-0.7241
3.5	-0.7241	-0.3846	0.6	0.6

Сигмоидная предобработка. В сигмоидной предобработке может использоваться любая сигмоидная функция. Если в качестве сигмоидной функции использовать функцию S_2, приведенную в разделе «Нейрон» этой главы, то формула (17) примет следующий вид:

$$z_i = \frac{\left(x/(c+|x|)-1\right)(b-a)}{2} + a .$$

Графики функций $z_i(x)$ представлены на рис. 2б. Видно, что с увеличением значения признака x ни одна функция не убывает, а их сумма возрастает. В табл. 9 представлены значения этих функций для двух точек $x_1 = 1.5,\ x_2 = 3.5$.

Шапочная предобработка. Для шапочной предобработки используются любые функции, имеющие график в виде «шапочки». Например, функция $\varphi(x) = 1/(1+x^2)$. Графики функций $z_i(x)$ представлены на рис. 2в. Видно, что с увеличением значения признака x ни одна из функций $z_i(x)$, ни их сумма не ведут себя монотонно. В табл. 9 представлены значения этих функций для двух точек $x_1 = 1.5,\ x_2 = 3.5$.

4.1.11.3 Позиционная предобработка

Основная идея позиционной предобработки совпадает с принципом построения позиционных систем счисления. Зададимся положительной величиной y такой, что $y^k \geq (x_{\min} - x_{\max})$. Сдвинем признак x так, чтобы он принимал только неотрицательные значения. В качестве сигналов сети будем использовать результат простейшей предобработки y-ичных цифр представления сдвинутого признака x. Формулы вычисления цифр приведены ниже:

$$\begin{aligned} z_0 &= (x - x_{\min}) \bmod y, \\ z_1 &= Int\left((x - x_{\min})/y\right) \bmod y, \\ &\dots \\ z_i &= Int\left((x - x_{\min})/y^i\right) \bmod y, \end{aligned} \tag{19}$$

где операция сравнения по модулю действительного числа определена в (15). Входные сигналы сети получаются из компонентов вектора z путем простейшей предобработки.

4.1.12 Составной предобработчик

Поскольку на вход нейронной сети обычно подается несколько входных сигналов, каждый из которых обрабатывается своим предобработчиком, то предобработчик должен быть составным. Представим предобработчик в виде совокупности независимых частных предобработчиков. Каждый частный предобработчик обрабатывает одно или несколько тесно связанных входных данных. Как уже отмечалось ранее, предобработчик может иметь один из четырех типов, приведенных в табл. 10. На входе предобработчик получает вектор входных данных (возможно, состоящий из одного элемента), а на выходе выдает вектор входных сигналов сети (так же возможно состоящий из одного элемента).

Таблица 10.

Типы предобработчиков

ТипОписание	
Number	Предобрабатывает числовые входные данные
Unordered	Предобрабатывает неупорядоченные качественные признаки
Ordered	Предобрабатывает упорядоченные качественные признаки
Binary	Обрабатывает бинарные признаки

Необходимость передачи предобработчику вектора входных данных и получения от него вектора входных сигналов связана с тем, что существуют предобработчики получающие несколько входных данных и выдающие несколько входных сигналов. Примером такого предобработчика может служить предобработчик, переводящий набор координат планеты из сферической в декартову.

Для качественных признаков принято кодирование длинными целыми числами. Первое значение равно 1, второе – 2 и т.д. Числовые признаки кодируются действительными числами.

4.2 Стандарт первого уровня компонента предобработчик

Данный раздел посвящен описанию стандарта языка описания и хранения на внешнем носителе компонента предобработчик. Поскольку крайне редко встречаются случаи, когда сеть получает один входной сигнал, предобработчик всегда является составным. Построение предобработчика происходит в редакторе предобработчика. Для описания предобработчика предлагается использовать специальный язык.

4.2.1 Неопределенные значения

В практике работы большинство таблиц данных не полны. То есть, часть данных в примерах задачника неизвестна. Задачник должен однозначно указать предобработчику неизвестные данные. Для этих целей для каждого типа входных данных определено специальное значение - неопределенное. Для передачи неизвестных значений используются следующие величины: 10^{-40} для действительных чисел и 0 для всех типов качественных признаков.

4.2.2 Стандартные предобработчики

В большинстве случаев достаточно использовать стандартные предобработчики, список которых приведен в табл. 11. Ниже в данном разделе приведено описание параметров стандартных предобработчиков.

Таблица 11

Стандартные предобработчики

Идентификатор	ПараметрыТипОписание		
BinaryPrep	MinSignals, MaxSignals : Real; Unknown: Real; Type : Logic.	Binary	Бинарный признак. Предобработка в соответствии с табл. 4.
UnOrdered	MinSignals, MaxSignals : Real; Unknown: Real; Num : Long	Unordered	Неупорядоченный качественный признак. Предобработка в соответствии с табл. 5.
Ordered	MinSignals, MaxSignals : Real; Unknown: Real; Num : Long	Ordered	Упорядоченный качественный признак. Предобработка в соответствии с табл. 6.
EmptyPrep	MinData, MaxData, Unnown, MinSignals, MaxSignals : Real	Number	Простейшая предобработка в соответствии с формулой (1).
ModPrep	MinSignals, MaxSignals : Real; Unknown: Real; Y : RealArray	Number	Модулярная предобработка в соответствии с формулой (16).
FuncPrep	MinSignals, MaxSignals, Unknown: Real; Y : RealArray; F : FuncType	Number	Функциональная предобработка в соответствии с формулой (17).
PositPrep	MinSignals, MaxSignals, Unnown, Y : Real; Num : Long	Number	Позиционная предобработка в соответствии с формулой (19).

Все стандартные предобработчики получают в качестве аргументов массивы входной информации и входных сигналов. Кроме того, они содержат различные наборы параметров. Алгоритмы выполнения стандартных предобработчиков приведены в разделе «Пример описания предобработчика». Далее описаны наборы параметров стандартных предобработчиков. Все параметры должны быть описаны как статические переменные.

Предобработка бинарного признака (BinaryPrep). Предобработка производится в соответствии с табл. 4. Принимает одно входное данное и генерирует один входной сигнал. Предобработчик содержит следующие параметры.

MinSignals, MaxSignals – значения нижней и верхней границ интервала приемлемых входных сигналов, соответственно. По умолчанию эти величины равны -1 и 1, соответственно.

Unknown– значение сигнала, который будет выдан, если значение входного признака не определено (0). По умолчанию эта величина равна $(MinSignals + MaxSignals)/2$.

Type – тип предобработки бинарного признака. Если значение параметра Type – истина, то производится предобработка по типу «Наличие другого свойства», если ложь, то по типу «Отсутствие заданного свойства». По умолчанию значение этого параметра равно истина.

Предобработка неупорядоченного качественного признака (UnOrdered). Предобработка производится в соответствии с табл. 5. Принимает одно входное данное и генерирует Num входных сигналов. Предобработчик содержит следующие параметры.

MinSignals, MaxSignals – значения нижней и верхней границ интервала приемлемых входных сигналов, соответственно. По умолчанию эти величины равны -1 и 1, соответственно.

Unknown– значение сигналов, которые будут выданы, если значение входного признака не определено (0). По умолчанию эта величина равна $(MinSignals + MaxSignals)/2$.

Num – число состояний качественного признака (число генерируемых входных сигналов). По умолчанию значение этого параметра равно 2.

Предобработка упорядоченного качественного признака (Ordered). Предобработка производится в соответствии с табл. 6. Принимает одно входное данное и генерирует Num входных сигналов. Предобработчик содержит следующие параметры.

MinSignals, MaxSignals – значения нижней и верхней границ интервала приемлемых входных сигналов, соответственно. По умолчанию эти величины равны -1 и 1, соответственно.

Unknown– значение сигналов, которые будут выданы, если значение входного признака не определено (0). По умолчанию эта величина равна $(MinSignals + MaxSignals)/2$

Num – число состояний качественного признака (число генерируемых входных сигналов). По умолчанию значение этого параметра равно 2.

Простейший предобработчик (EmptyPrep). Предобработка производится в соответствии с формулой (1). Принимает одно входное данное и генерирует один входной сигнал. Предобработчик содержит следующие параметры.

MinSignals, MaxSignals – значения нижней и верхней границ интервала приемлемых входных сигналов, соответственно. По умолчанию эти величины равны -1 и 1, соответственно.

Unknown– значение сигнала, который будет выдан, если значение входного признака не определено (10^{-40}). По умолчанию эта величина равна 0.

MinData, MaxData – значения нижней и верхней границ интервала изменения входных данных, соответственно. По умолчанию эти величины равны -1 и 1, соответственно. Эти значения могут быть определены поиском минимального и максимального значений по задачнику, однако предобработчик не может выполнить эту процедуру.

Модулярный предобработчик (ModPrep). Предобработка производится в соответствии с формулой (16). Принимает одно входное данное и генерирует столько входных сигналов, сколько элементов в массиве Y (нулевой элемент массива содержит число элементов). Предобработчик содержит следующие параметры.

MinSignals, MaxSignals – значения нижней и верхней границ интервала приемлемых входных сигналов, соответственно. По умолчанию эти величины равны -1 и 1, соответственно.

Unknown– значение сигналов, которые будут выданы, если значение входного признака не определено (10^{-40}). По умолчанию эта величина равна 0.

Y – массив величин, используемых для предобработки (см. раздел «Модулярная предобработка»).

Функциональный предобработчик (FuncPrep). Предобработка производится в соответствии с формулой (17). Принимает одно входное данное и генерирует столько входных сигналов, сколько элементов в массиве Y (нулевой элемент массива содержит число элементов). Предобработчик содержит следующие параметры.

MinSignals, MaxSignals – значения нижней и верхней границ интервала приемлемых входных сигналов, соответственно. По умолчанию эти величины равны -1 и 1, соответственно.

Unknown– значение сигналов, которые будут выданы, если значение входного признака не определено (10^{-40}). По умолчанию эта величина равна 0.

MinData, MaxData – значения нижней и верхней границ интервала изменения функции F от входных данных, соответственно. По умолчанию эти величины равны -1 и 1, соответственно. Эти значения могут быть определены поиском минимального и максимального значений функции по задачнику, однако предобработчик не может выполнить эту процедуру.

Y – массив величин, используемых для предобработки (см. раздел «Функциональная предобработка»).

F – имя однопараметрической функции действительного типа (ее адрес) используемой для предобработки.

Позиционный предобработчик (PositPrep). Предобработка производится в соответствии с формулой (19). Принимает одно входное данное и генерирует Num входных сигналов. Предобработчик содержит следующие параметры.

MinSignals, MaxSignals – значения нижней и верхней границ интервала приемлемых входных сигналов, соответственно. По умолчанию эти величины равны -1 и 1, соответственно.

Unknown– значение сигналов, которые будут выданы, если значение входного признака не определено (10^{-40}). По умолчанию эта величина равна 0.

Y – основание системы счисления (см. раздел «Функциональная предобработка»). По умолчанию эта величина равна 2.

Num – число цифр в представлении входного сигнала. По умолчанию эта величина равна 2.

4.2.3 Язык описания предобработчика

Предобработчик является составным объектом. В состав этого объекта входят частные предобработчики, правила распределения входных данных и входных сигналов сети между частными предоб-

Таблица 12

Ключевые слова языка описания предобработчика.

Идентификатор	Краткое описание
Connections	Начало блока описания распределения входных данных и сигналов.
Contents	Начало блока описания состава интерпретатора.
Data	Имя, по которому адресуются входные данные, начало блока описания входных данных
Include	Предшествует имени файла, целиком вставляемого в это место описания.
NumberOf	Функция. Возвращает число обрабатываемых частным предобработчиком входных данных или сигналов.
Prep	Начало заголовка описания частного предобработчика.
Preparator	Заголовок раздела файла, содержащий описание интерпретатора.
Signals	Имя, по которому адресуются входные сигналы; начало блока описания сигналов.

работчиками. Предобработчик при выполнении запроса на предобработку вектора входных данных получает на входе вектор исходных данных, а возвращает вектор входных сигналов сети.

Каждый частный интерпретатор ответа получает на входе вектор входных данных, которые он предобрабатывает, а на выходе дает вектор входных сигналов сети. Каждый частный интерпретатор описывается в виде процедурного блока.

В табл. 12 приведен список ключевых слов языка описания предобработчика, дополняющий список ключевых слов, приведенных в главе «Общий стандарт». Кроме того, ключевыми словами являются имена стандартных предобработчиков, приведенные в табл. 11.

4.2.3.1 БНФ языка описания предобработчика

Обозначения, принятые в данном расширении БНФ и описание ряда конструкций приведены в главе «Общий стандарт» в разделе «Описание языка описания компонентов».

<Описание предобработчика> ::= <Заголовок> [<Описание функций>] [<Описание частных предобработчиков>] <Описание состава> [<Установление параметров>] [<Описание сигналов>] [<Описание данных>] [<Описание распределения сигналов>] [<Описание распределения данных>] <Конец описания предобработчика>

<Заголовок> ::= **Preparator** <Имя предобработчика> *(* <Список формальных аргументов> *)*

<Имя предобработчика> ::= <Идентификатор>

<Описание частных предобработчиков> ::= <Описание частного предобработчика> [<Описание частных предобработчиков>]

<Описание частного предобработчика> ::= <Заголовок описания предобработчика> [<Описание статических переменных>] [<Описание переменных>] <Тело предобработчика>

<Заголовок описания предобработчика> ::= **Prep** <Имя частного предобработчика> ([(<Список формальных аргументов>)])

<Имя частного интерпретатора> ::= <Идентификатор>

<Тело предобработчика> ::= **Begin** <Составной оператор> **End**

<Описание состава> ::= **Contents** <Список имен предобработчиков> ;

<Список имен предобработчиков> ::= <Имя предобработчика> [,<Список имен предобработчиков>]

<Имя предобработчика> ::= <Псевдоним>*:* {<Имя ранее описанного интерпретатора> | <Имя стандартного интерпретатора>} [*/*<Число экземпляров >*/*] [(<Список фактических аргументов>)]

<Псевдоним> ::= <Идентификатор>

<Число экземпляров > ::= <Целое число>

<Имя ранее описанного интерпретатора> ::= <Идентификатор>

<Имя стандартного интерпретатора> ::= <Идентификатор>

<Установление параметров> ::= <Установление параметров ***Частного предобработчика***> [;<Установление параметров>]

<Описание сигналов> ::= **Signals <**Константное выражение типа ***Long*****>**

<Описание данных> ::= **Data <**Константное выражение типа ***Long*****>**

<Описание распределения сигналов> ::= <Описание распределения ***Сигналов, Предобработчика, Частного предобработчика, Signals***>

<Описание распределения данных> ::= <Описание распределения ***Данных, Предобработчика, Частного предобработчика, Data***>

<Конец описания предобработчика> ::= **End Preparator**

4.2.3.2 Описание языка описания предобработчика

Структура описания предобработчика имеет вид: заголовок; описание функций; описание частных предобработчиков; описание состава; установление параметров; описание сигналов; описание данных; описание распределения сигналов; описание распределения данных; конец описания предобработчика.

Заголовок состоит из ключевого слова **Preparator** и имени предобработчика и служит для обозначения начала описания предобработчика в файле, содержащем несколько компонентов нейрокомпьютера.

Описание функций – фрагмент описания, в котором описаны функции, необходимые для работы предобработчиков.

Описание частного предобработчика – это описание процедуры, вычисляющей входные сигналы нейронной сети по входным данным. Формальные аргументы служат для задания размерностей обрабатываемых векторов. При выполнении частный предобработчик получает в качестве аргументов два вектора – входных данных и входных сигналов. Формально, при исполнении частный предобработчик имеет описание следующего вида:

Pascal:

```
Procedure Preparator(Data, Signals : PRealArray);
```

C:

```
void Preparator(PRealArray Data; PRealArray Signals);
```

В разделе описания состава перечисляются частные предобработчики, входящие в состав предобработчика. Признаком конца раздела служит символ «;».

В необязательном разделе установления параметров производится задание значений параметров (статических переменных) частных предобработчиков. После ключевого слова SetParameters следует список значений параметров в том порядке, в каком параметры были объявлены при описании частного интерпретатора (для стандартных интерпретаторов порядок параметров соответствует порядку, приведенному в описании стандартных предобработчиков в разделе «Стандартные предобработчики»). При использовании одного оператора задания параметров для задания параметров нескольким экземплярам одного частного предобработчика после ключевого слова SetParameters указывается столько выражений, задающих значения параметров, сколько необходимо для одного экземпляра.

В необязательном разделе "описание сигналов" указывается число сигналов, вычисляемых предобработчиком. Если этот раздел опущен, то полагается, что число вычисляемых предобработчиком сигналов равно сумме сигналов, вычисляемых всеми частными предобработчиками. В константном выражении возможно использование функции NumberOf, аргументом которой является имя частного предобработчика (или его псевдоним) и ключевое слово Signals, в качестве второго аргумента.

В необязательном разделе "описание данных" указывается число входных данных, предобрабатываемых предобработчиком. Если этот раздел опущен, то полагается, что число предобрабатываемых предобработчиком данных равно сумме данных, предобрабатываемых всеми частными предобработчиками. В константном выражении возможно использование функции NumberOf, аргументом которой является имя частного предобработчика (или его псевдоним) и ключевое слово Data, в качестве второго аргумента.

В необязательном разделе описания распределения сигналов (данных) указывается для каждого частного предобработчика какие сигналы (входные данные) из общего вектора сигналов (данных) передаются ему для обработки.

Наиболее часто встречающиеся интерпретаторы объявлены стандартными. Для стандартных интерпретаторов описание частных интерпретаторов отсутствует.

Кроме того, в любом месте описания интерпретатора могут встречаться комментарии, заключенные в фигурные скобки.

4.2.3.3 Пример описания предобработчика

В этом разделе приведены два примера описания одного и того же предобработчика для метеорологической задачи. Используется следующий состав предобработчика: первый элемент вектора входных данных (температура воздуха) обрабатывается простейшим предобработчиком (EmptyPrep); второй (облачность) – бинарным предобработчиком (BinaryPrep); третий (направление ветра) – предобработчиком неупорядоченных качественных признаков (UnOrdered); четвертый (осадки) – предобработчиком неупорядоченных качественных признаков (Ordered).

В первом примере приведено описание дубликатов всех стандартных предобработчиков. Во втором – использованы стандартные предобработчики.

Пример 1.

```
Preparator Meteorology

Function Sigmoid( X Real ) : Real;
   Begin
      Sigmoid = X / (1 + Abs(X))
   End;

Prep BinaryPrep1 ()      {Предобработка бинарного признака}
   Static
      Real MinSignals Name "Нижняя граница интервала приемлемых сигналов";
      Real MaxSignals Name "Верхняя граница интервала приемлемых сигналов";
      Real Unknown Name "Значение сигнала, если значение входного признака не определено";
      Logic Type Name "Тип предобработки бинарного признака";
   Begin
      If TLong(Data[1]) = UnknownLong Then Signals[1] = Unknown
      Else Begin
```

```
        If Type Then Begin
            If TLong(Data[1]) = 1 Then Signals[1] = 0 Else Begin
                If MaxSignals =0 Then Signals[1] = MinSignals Else Signals[1] = MaxSignals
            End
        Else Begin
            If TLong(Data[1]) = 1 Then Signals[1] = MinSignals Else Signals[1] = MaxSignals
        End
    End
End

Prep UnOrdered1 ( Num : Long ) {Предобработка упорядоченного качественного признака}
    Static
        Real MinSignals Name "Нижняя граница интервала приемлемых сигналов";
        Real MaxSignals Name "Верхняя граница интервала приемлемых сигналов";
        Real Unknown Name "Значение сигнала, если значение входного признака не определено";
    Var
        Integer I;
    Begin
        If TLong(Data[1]) = UnknownLong Then Begin
            For I = 1 To Num Do
                Signals[I] = Unknown
        End Else Begin
            For I = 1 To Num Do
                Signals[I] = MinSignals
            Signals[TLong(Data[1])] = MaxSignals
        End
    End

Prep Ordered1 ( Num : Long ) {Предобработка упорядоченного качественного признака}
    Static
        Real MinSignals Name "Нижняя граница интервала приемлемых сигналов";
        Real MaxSignals Name "Верхняя граница интервала приемлемых сигналов";
        Real Unknown Name "Значение сигнала, если значение входного признака не определено";
    Var
        Integer I;
    Begin
        If TLong(Data[1]) = UnknownLong Then Begin
            For I = 1 To Num Do
                Signals[I] = Unknown
        End Else Begin
            For I = 1 To TLong(Data[1]) Do
                Signals[I] = MaxSignals
            For I = TLong(Data[1])+1 To Num Do
                Signals[I] = MinSignals
        End
    End

Prep EmptyPrep1 ()      {Предобработчик, осуществляющий масштабирование и сдвиг сигнала}
    Static
        Real MinSignals Name "Нижняя граница интервала приемлемых сигналов";
        Real MaxSignals Name "Верхняя граница интервала приемлемых сигналов";
        Real Unknown Name "Значение сигнала, если значение входного признака не определено";
        Real MinData Name "Значения нижней границы интервала изменения входных данных";
        Real MaxData Name "Значения верхней границы интервала изменения входных данных";
    Begin
        If Data[1] = UnknownReal Then Signals[1] = Unknown
        Else Signals[1] = (Data[1] – MinData) * (MaxSignals – MinSignals) / (MaxData – MinData)
            + MinSignals
    End

Prep ModPrep1 ( Num : Long )        {Модулярный предобработчик}
```

```
    Static
        Real MinSignals Name "Нижняя граница интервала приемлемых сигналов";
        Real MaxSignals Name "Верхняя граница интервала приемлемых сигналов";
        Real Unknown Name "Значение сигнала, если значение входного признака не определено";
        RealArray[Num] Y Name "Массив величин, используемых для предобработки"
    Var
        Integer I;
    Begin
        If Data[1] = UnknownReal Then Begin
            For I = 1 To Num Do
                Signals[I] = Unknown
        End Else Begin
            For I = 1 To Num Do
                Signals[I] = (Data[1] RMod Y[I] + Y[I]) * (MaxSignals – MinSignals) / (2 * Y[I])
                        + MinSignals
    End

Prep FuncPrep1(Num : Long; F : FuncType)          {Функциональный предобработчик}
    Static
        Real MinSignals Name "Нижняя граница интервала приемлемых сигналов";
        Real MaxSignals Name "Верхняя граница интервала приемлемых сигналов";
        Real Unknown Name "Значение сигнала, если значение входного признака не определено";
        Real MinData Name "Значения нижней границы интервала изменения значений функции F";
        Real MaxData Name "Значения верхней границы интервала изменения значений функции F";
        RealArray[Num] Y Name "Массив величин, используемых для предобработки"
    Var
        Integer I;
    Begin
        If Data[1] = UnknownReal Then Begin
            For I = 1 To Num Do
                Signals[I] = Unknown
        End Else Begin
            For I = 1 To Num Do
                Signals[1] = (F(Data[1] – Y[1] – MinData) * (MaxSignals – MinSignals) /
                        (MaxData – MinData) + MinSignals
    End

Prep PositPrep1( Num : Long )          {Позиционный предобработчик}
    Static
        Real MinSignals Name "Нижняя граница интервала приемлемых сигналов"
        Real MaxSignals Name "Верхняя граница интервала приемлемых сигналов"
        Real Unknown Name "Значение сигнала, если значение входного признака не определено"
        Real Y Name "Основание системы счисления"
    Var
        Integer I;
        Real W, Q;
    Begin
        If Data[1] = UnknownReal Then Begin
            For I = 1 To Num Do
                Signals[I] = Unknown
        End Else Begin
            W = Data[1];
            For I = 1 To Num Do Begin
                Q = W RMod Y;
                Signals[I] = Q * (MaxSignals – MinSignals) / Y + MinSignals;
                W = (W - Q) / Y
            End;
    End

Contents Temp : EmptyPrep1, Cloud : BinaryPrep1, Wind : UnOrdered1(8), Rain : Ordered1(3);
```

```
Temp SetParameters -1, 1, 1E-40, 273, 293;   {Для всех предобработчиков приемлемые значения вход-}
Cloud SetParameters -1, 1, 0, True;   {ных сигналов лежат в интервале от -1 до 1. В случае неопреде-}
Wind SetParameters -1, 1, 0;          {ленного значения во входных данных все сигналы данного}
Rain SetParameters -1, 1, 0           {предобработчика полагаются равными нулю. Входные данные}
                                      {первого предобработчика меняются от 273 до 293}
Signals NumberOf(Signals,Temp) + NumberOf(Signals, Cloud) + NumberOf(Signals, Wind(8)) +
        NumberOf(Signals, Rain(3))

Data NumberOf(Data,Temp) + NumberOf(Data, Cloud) + NumberOf(Data, Wind(8)) +
        NumberOf(Data, Rain(3))

Connections
    Temp.Data <=> Data[1];
    Cloud.Data <=> Data[2];
    Wind.Data <=> Data[3];
    Rain.Data <=> Data[4];
    Temp.Signals <=> Signals[1];
    Cloud.Signals <=> Signals[2];
    Wind.Signals[1..8] <=> Signals[3..10];
    Rain.Signals[1..3] <=> Signals[11..13]

End Preparator
```

Пример 2.

```
Preparator Meteorology

Contents Temp : EmptyPrep, Cloud : BinaryPrep, Wind : UnOrdered(8), Rain : Ordered(3);

Temp SetParameters -1, 1, 1E-40, 273, 293

End Preparator
```

4.3 Стандарт второго уровня компонента предобработчик

Запросы к компоненту предобработчик можно разбить на пять групп:

1. Предобработка.
2. Изменение параметров.
3. Работа со структурой.
4. Инициация редактора предобработчика.
5. Обработка ошибок.

Поскольку нейрокомпьютер может работать одновременно с несколькими сетями, то и компонент предобработчик должна иметь возможность одновременной работы с несколькими предобработчиками. Поэтому большинство запросов к предобработчику содержат явное указание имени предобработчика. Ниже приведено описание всех запросов к компоненту предобработчик. Каждый запрос является логической функцией, возвращающей значение истина, если запрос выполнен успешно, и ложь – при ошибочном завершении исполнения запроса.

В запросах второй и третьей группы при обращении к частным интерпретаторам используется следующий синтаксис:

<Полное имя частного интерпретатора> ::=
<Имя интерпретатора>.<Псевдоним частного интерпретатора> [*[*<Номер экземпляра>*]*]

Таблица 13.

Значения предопределенных констант компонента предобработчик

НазваниеВеличина	Значение
BinaryPrep	0	Стандартный предобработчик бинарных признаков
UnOrdered	1	Стандартный предобработчик неупорядоченных качественных признаков
Ordered	2	Стандартный предобработчик упорядоченных качественных признаков.
EmptyPrep	3	Стандартный простейший предобработчик
ModPrep	4	Стандартный модулярный предобработчик
FuncPrep	5	Стандартный функциональный предобработчик
PositPrep	6	Стандартный позиционный предобработчик
UserType	-1	Предобработчик, определенный пользователем.

При вызове ряда запросов используются предопределенные константы. Их значения приведены в табл. 13.

4.3.1 Запрос на предобработку

Единственный запрос первой группы выполняет основную функцию компонента предобработчик – предобрабатывает входные данные, вычисляя вектор входных сигналов.

4.3.1.1 Предобработать вектор сигналов (Prepare)

Описание запроса:

Pascal:

```
Function Prepare(CompName : PString; Data : PRealArray; Var Signals : PRealArray) : Logic;
```

C:

```
Logic Prepare(PString CompName, PRealArray Data; PRealArray* Signals)
```

Описание аргумента:

CompName – указатель на строку символов, содержащую имя предобработчика.

Data – массив входных данных.

Signals – вычисляемый массив входных сигналов.

Назначение – предобрабатывает массив входных данных Data, вычисляя массив входных сигналов Signals используя предобработчик, указанный в параметре CompName.

Описание исполнения.

1. Если Error <> 0, то выполнение запроса прекращается.
2. Если в качестве аргумента CompName дан пустой указатель, или указатель на пустую строку, то исполняющим запрос объектом является текущий предобработчик – первый в списке предобработчиков компонента предобработчик.
3. Если список предобработчиков компонента предобработчик пуст или имя предобработчика, переданное в аргументе CompName в этом списке не найдено, то возникает ошибка 201 – неверное имя предобработчика, управление передается обработчику ошибок, а обработка запроса прекращается.
4. Производится предобработка предобработчиком, имя которого было указано в аргументе CompName.
5. Если во время выполнения запроса возникает ошибка, то генерируется внутренняя ошибка 204 - ошибка предобработки. Управление передается обработчику ошибок. Выполнение запроса прекращается. В противном случае выполнение запроса успешно завершается.

4.3.2 Остальные запросы.

Ниже приведен список запросов к компоненту предобработчик, исполнение которых описано в главе «Общий стандарт»:

prSetCurrent – Сделать предобработчик текущим
prAdd – Добавление нового предобработчика
prDelete – Удаление предобработчика
prWrite – Запись предобработчика
prGetStructNames – Вернуть имена структурных единиц предобработчика
prGetType – Вернуть тип структурной единицы предобработчика
prGetData – Получить параметры предобработчика
prGetName – Получить имена параметров предобработчика
prSetData – Установить параметры предобработчика
prEdit – Редактировать предобработчик
OnError – Установить обработчик ошибок
GetError – Дать номер ошибки
FreeMemory – Освободить память

В запросе prGetType в переменной TypeId возвращается значение одной из предопределенных констант, перечисленных в табл. 13.

4.3.3 Ошибки компонента предобработчик

В табл. 14 приведен полный список ошибок, которые могут возникать при выполнении запросов компонентом предобработчик, и действия стандартного обработчика ошибок.

Таблица 14.

Ошибки компонента предобработчик и действия стандартного обработчика ошибок.

№Название ошибкиСтандартная обработка		
201	Неверное имя предобработчикаЗанесение	номера в Error
202	Ошибка считывания предобработчикаЗанесение	номера в Error
203	Ошибка сохранения предобработчикаЗанесение	номера в Error
204	Ошибка предобработкиЗанесение	номера в Error

5. Описание нейронных сетей

Эта глава состоит из четырех частей. В первой части описана система построения сетей из элементов. Описаны прямое и обратное функционирование сетей и составляющих их элементов. Во второй части приведены примеры различных парадигм нейронных сетей, описанные в соответствии с предложенной в первой части главы методикой. В третьей и четвертой частях главы описаны стандарты нейронных сетей.

Как уже говорилось в первой главе, на данный момент в нейросетевом сообществе принято описывать архитектуру нейронных сетей в неразрывном единстве с методами их обучения. Эта связь не является естественной. Так, в первой части этой главы будет рассматриваться только архитектура нейронных сетей. Во второй части будет продемонстрирована независимость ряда методов обучения нейронных сетей от их архитектуры. Однако, для удобства читателя, во второй части главы архитектуры всех парадигм нейронных сетей будут описаны вместе с методами обучения.

Нейронные сети можно классифицировать по разным признакам. Для описания нейронных сетей в данной главе существенной является классификация по типу времени функционирования сетей. По этому признаку сети можно разбить на три класса.

1. Сети с непрерывным временем (аналоговые сети).
2. Сети с дискретным асинхронным временем.
3. Сети с дискретным временем, функционирующие синхронно.

В данной работе рассматриваются только сети третьего вида, то есть сети, в которых все элементы каждого слоя срабатывают одновременно и затем передают свои сигналы нейронам следующего слоя.

5.1 Конструирование нейронных сетей

Впервые последовательное описание конструирования нейронных сетей из элементов было предложено в книге А.Н.Горбаня [64]. Однако за прошедшее время предложенный А.Н. Горбанем способ конструирования претерпел ряд изменений.

При описании нейронных сетей принято оперировать такими терминами, как нейрон и слой. Однако, при сравнении работ разных авторов выясняется, что если слоем все авторы называют приблизительно одинаковые структуры, то нейроны разных авторов совершенно различны. Таким образом, описание нейронных сетей, а значит и стандартизация, на уровне нейронов невозможна. Однако, возможна стандартизация на уровне составляющих нейроны элементов и процедур конструирования сложных сетей из простых.

5.1.1 Элементы нейронной сети

На рис. 1 приведены все элементы, необходимые для построения нейронных сетей. Естественно, что возможно расширение списка нелинейных преобразователей. Однако, это единственный вид элементов, который может дополняться. Вертикальными стрелками обозначены входы параметров (для синапса – синаптических весов или весов связей), а горизонтальными – входные сигналы элементов. С точки зрения функционирования элементов сети сигналы и входные параметры элементов равнозначны. Различие между этими двумя видами параметров относятся к способу их использования в обучении. Кроме того, удобно считать, что параметры каждого элемента являются его свойствами и хранятся при нем. Совокупность параметров всех элементов сети называют вектором параметров сети. Совокупность параметров всех синапсов называют вектором обучаемых параметров сети, картой весов связей или синаптической картой. Отметим, что необходимо различать входные сигналы элементов и входные сигналы сети. Они совпадают только для элементов входного слоя сети.

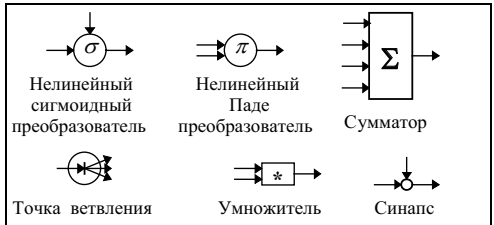


Рис. 1. Простейшие элементы сети

Из приведенных на рис. 1 элементов можно построить практически любую нейронную сеть. Вообще говоря, нет никаких правил, ограничивающих свободу творчества конструктора нейронных сетей.

Однако, есть набор структурных единиц построения сетей, позволяющий стандартизовать процесс конструирования. Детальный анализ различных нейронных сетей позволил выделить следующие структурные единицы:

1 элемент – неделимая часть сети, для которой определены методы прямого и обратного функционирования;
2 каскад – сеть составленная из последовательно связанных слоев, каскадов, циклов или элементов;
3 слой – сеть составленная из параллельно работающих слоев, каскадов, циклов или элементов;
4 цикл – каскад выходные сигналы которого поступают на его собственный вход.

Очевидно, что не все элементы являются неделимыми. В следующем разделе будет приведен ряд составных элементов.

Введение трех типов составных сетей связано с двумя причинами: использование циклов приводит к изменению правил остановки работы сети, описанных в разд. "Правила остановки работы сети"; разделение каскадов и слоев позволяет эффективно использовать ресурсы параллельных ЭВМ. Действительно, все сети, входящие в состав слоя, могут работать независимо друг от друга. Тем самым при конструировании сети автоматически закладывается база для использования параллельных ЭВМ.

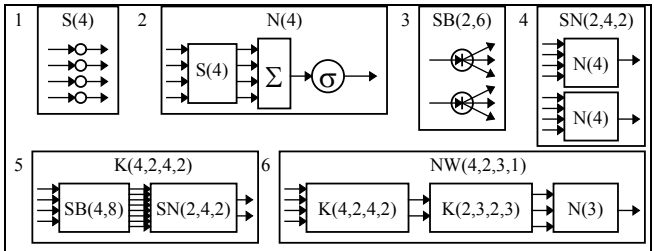


Рис. 2. Построение сети из простейших элементов. 1 - слой синапсов S4 (4 - означает число синапсов, входящих в слой). 2 - каскад-нейрон N4 (4 - означает число входных сигналов). 3 - слой точек ветвления SB(2,6) (первая цифра - число входных сигналов, вторая - выходных). 4 - слой нейронов SN(2,4,2) (первая цифра - число нейронов, вторая - число входных сигналов у каждого нейрона, третья - выходных). 5 - каскад точек ветвления и нейронов K(4,2,4,2) (первая цифра означает число входных сигналов, вторая - число нейронов, третья - число входных сигналов у каждого нейрона, четвертая - число выходных сигналов каскада). 6 - сеть NW(4,2,3,1) (первая цифра - число входных сигналов сети, вторая - число нейронов во входном слое, третья - число нейронов в скрытом слое, четвертая - число нейронов в выходном слое).

На рис. 2 приведен пример поэтапного конструирования трехслойной сигмоидной сети.

5.1.2 Составные элементы

Название «составные элементы» противоречит определению элементов. Это противоречие объясняется соображениями удобства работы. Введение составных элементов преследует цель упрощения конструирования. Как правило, составные элементы являются каскадами простых элементов.

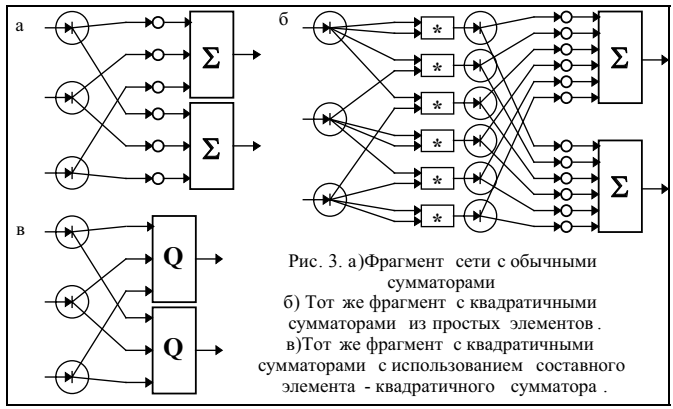
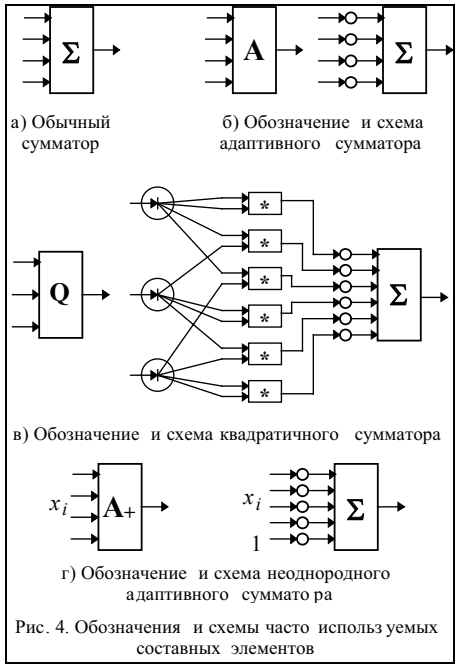


Рис. 3. а)Фрагмент сети с обычными сумматорами
б) Тот же фрагмент с квадратичными сумматорами из простых элементов .
в)Тот же фрагмент с квадратичными сумматорами с использованием составного элемента - квадратичного сумматора .

Хорошим примером полезно-

сти составных элементов может служить использование сумматоров. В ряде работ [36, 53, 106, 126, 288] интенсивно используются сети, нейроны которых содержат нелинейные входные сумматоры. Под нелинейным входным сумматором, чаще всего понимают квадратичные сумматоры – сумматоры, вычисляющие взвешенную сумму всех попарных произведений входных сигналов нейрона. Отличие сетей с квадратичными сумматорами заключается только в использовании этих сумматоров. На рис. 3а приведен фрагмент сети с линейными сумматорами. На рис. 3б – соответствующий ему фрагмент с квадратичными сумматорами, построенный с использованием элементов, приведенных на рис. 1. На (рис. 3в) – тот же фрагмент, построенный с использованием квадратичных сумматоров. При составлении сети с квадратичными сумматорами из простых элементов на пользователя ложится большой объем работ по проведению связей и организации вычисления попарных произведений. Кроме того, рис. 3в гораздо понятнее рис. 3б и содержит ту же информацию. Кроме того, пользователь может изменить тип сумматоров уже сконструированной сети, указав замену одного типа сумматора на другой. На рис. 4 приведены обозначения и схемы наиболее часто используемых составных элементов.



а) Обычный сумматор

б) Обозначение и схема адаптивного сумматора

в) Обозначение и схема квадратичного сумматора

г) Обозначение и схема неоднородного адаптивного суммато ра

Рис. 4. Обозначения и схемы часто использ уемых составных элементов

Таблица 1

Однородные и неоднородные сумматоры

Название	Однородный сумматор		Неоднородный	сумматор
	Обозначение	Значение	Обозначение	Значение
Обычный	Σ	$\sum_i x_i$	Σ_+	$1+\sum_i x_i$
Адаптивный	A	$\sum_i \alpha_i x_i$	A_+	$\alpha_0+\sum_i \alpha_i x_i$
Квадратичный	Q	$\sum_i q_{ij} x_i x_j$	Q_+	$\alpha_{0+}\sum_i \alpha_i x_i+\sum_i q_{ij}$

Необходимо отметить еще одну разновидность сумматоров, полезную при работе по конструированию сети – неоднородные сумматоры. Неоднородный сумматор отличается от однородного наличием еще одного входного сигнала, равного единице. На рис. 4г приведены схема и обозначения для неоднородного адаптивного сумматора. В табл. 1 приведены значения, вычисляемые однородными и соответствующими им неоднородными сумматорами.

5.1.3 Функционирование сети

Прежде всего, необходимо разделить процессы обучения нейронной сети и использования обученной сети. При использовании обученной сети происходит только решение сетью определенной задачи. При этом синаптическая карта сети остается неизменной. Работу сети при решении задачи будем далее называть **прямым функционированием**.

При обучении нейронных сетей методом обратного распространения ошибки нейронная сеть (и каждый составляющий ее элемент) должна уметь выполнять обратное функционирование. Во второй части этой главы будет показано, что обратное функционирование позволяет обучать также и нейросети, традиционно считающиеся не обучаемыми, а формируемыми (например, сети Хопфилда). **Обратным функционированием** называется процесс работы сети, когда на *выход* сети подаются определенные сигналы, которые далее распространяются по тем же связям, что и при прямом функционировании до входа сети. При прохождении сигналов обратного функционирования через элемент с обучаемыми параметрами вычисляются поправки к параметрам этого элемента. Если на выход сети с непрерывными элементами подается производная некоторой функции F от выходных сигналов сети, то вычисляемые сетью поправки должны быть элементами градиента функции F по обучаемым параметрам сети. Исходя из этого требования определим правила обратного функционирования для элементов сети, приведенных на рис. 1, в предположении, что сеть состоит только из одного элемента.

5.1.3.1 Синапс

У синапса два входа – вход сигнала и вход синаптического веса (рис. 5а). Обозначим входной сигнал синапса через x, а синаптический вес через α. Тогда выходной сигнал синапса равен αx. При обратном функционировании на выход синапса подается сигнал $\partial F/\partial(\alpha x)$. На входе синапса должен быть получен сигнал обратного функционирования, равный $\frac{\partial F}{\partial x}=\frac{\partial F}{\partial(\alpha x)}\frac{\partial(\alpha x)}{\partial x}=\alpha\frac{\partial F}{\partial(\alpha x)}$, а на входе синаптического веса – поправка к синаптическому весу, равная $\frac{\partial F}{\partial \alpha}=\frac{\partial F}{\partial(\alpha x)}\frac{\partial(\alpha x)}{\partial \alpha}=x\frac{\partial F}{\partial(\alpha x)}$ (рис. 5б).

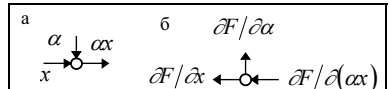


Рис. 5. Прямое (а) и обратное (б) функционирование синапса

5.1.3.2 Умножитель

Умножитель имеет два входных сигнала и не имеет параметров. Обозначим входные сигнал синапса через x_1, x_2. Тогда выходной сигнал умножителя равен x_1x_2 (рис. 6а). При обратном функционировании на выход умножителя подается сигнал $\partial F/\partial(x_1x_2)$. На входах сигналов x_1 и x_2 должны быть получены сигналы обратного функционирования, равные $\frac{\partial F}{\partial x_1}=\frac{\partial F}{\partial(x_1x_2)}\frac{\partial(x_1x_2)}{\partial x_1}=x_2\frac{\partial F}{\partial(x_1x_2)}$ и $\frac{\partial F}{\partial x_2}=\frac{\partial F}{\partial(x_1x_2)}\frac{\partial(x_1x_2)}{\partial x_2}=x_1\frac{\partial F}{\partial(x_1x_2)}$, соответственно (рис. 6б).

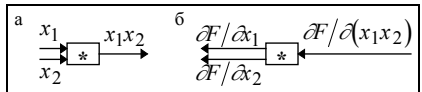


Рис. 6. Прямое (а) и обратное (б) функционирование умножителя

5.1.3.3 Точка ветвления

В отличие от ранее рассмотренных элементов, точка ветвления имеет только один вход и несколько выходов. Обозначим входной сигнал через x, а выходные через $x_1, x_2, \ldots, x_n$, причем $x_i = x$ (рис. 7а). При обратном функционировании на выходные связи точки ветвления подаются сигналы $\frac{\partial F}{\partial x_i}$ (рис. 7б). На входной связи должен получаться сигнал, равный $\frac{\partial F}{\partial x}=\sum_{i=1}^{n}\frac{\partial F}{\partial x_i}\frac{\partial x_i}{\partial x}=\sum_{i-1}^{n}\frac{\partial F}{\partial x_i}$. Можно сказать, что точка ветвления при обратном функционировании

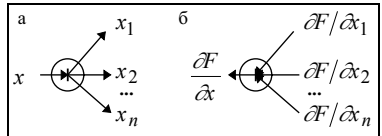


Рис. 7. Прямое (а) и обратное (б) функционирование точки ветвления

переходит в сумматор, или, другими словами, сумматор является двойственным по отношению к точке ветвления.

5.1.3.4 Сумматор

Сумматор считает сумму входных сигналов. Обычный сумматор не имеет параметров. При описании прямого и обратного функционирования ограничимся описанием простого сумматора, поскольку функционирование адаптивного и квадратичного сумматора может быть получено как прямое и обратное функционирование сети в соответствии с их схемами, приведенными на рис. 3б и 3в. Обозначим входные сигналы сумматора через $x_1, x_2, \ldots, x_n$ (рис. 8а). Выходной сигнал равен $\sum_{i=1}^{n} x_i$. При обратном функционировании на выходную связь сумматора подается сигнал $\frac{\partial F}{\partial \sum_{i=1}^{n} x_i}$ (рис. 8б). На входных связях должны получаться сигналы, равные

$$\frac{\partial F}{\partial x_j} = \frac{\partial F}{\partial \sum_{i=1}^{n} x_i} \frac{\partial \sum_{i=1}^{n} x_i}{\partial x_j} = \frac{\partial F}{\partial \sum_{i=1}^{n} x_i}$$

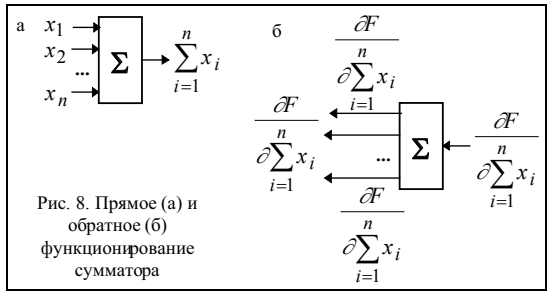


Рис. 8. Прямое (а) и обратное (б) функционирование сумматора

Из последней формулы следует, что все сигналы обратного функционирования, выдаваемые на входные связи сумматора, равны. Таким образом сумматор при обратном функционировании переходит в точку ветвления, или, другими словами, сумматор является двойственным по отношению к точке ветвления.

5.1.3.5 Нелинейный Паде преобразователь

Нелинейный Паде преобразователь или Паде элемент имеет два входных сигнала и один выходной. Обозначим входные сигналы через x_1, x_2. Тогда выходной сигнал Паде элемента равен x_1/x_2 (рис. 9а). При обратном функционировании на выход Паде элемента подается сигнал $\partial F / \partial(x_1/x_2)$. На входах сигналов x_1 и x_2 должны быть получены сигналы обратного функционирования, равные

$$\frac{\partial F}{\partial x_1} = \frac{\partial F}{\partial(x_1/x_2)} \frac{\partial(x_1/x_2)}{\partial x_1} = \frac{1}{x_2} \frac{\partial F}{\partial(x_1/x_2)}$$

и

$$\frac{\partial F}{\partial x_2} = \frac{\partial F}{\partial(x_1/x_2)} \frac{\partial(x_1/x_2)}{\partial x_2} = -\frac{x_1}{x_2^2} \frac{\partial F}{\partial(x_1/x_2)}$$

, соответственно (рис. 9б).

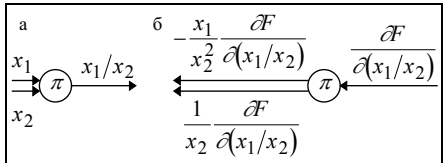


Рис. 9. Прямое (а) и обратное (б) функционирование нелинейного Паде преобразователя

5.1.3.6 Нелинейный сигмоидный пре-

образователь

Нелинейный сигмоидный преобразователь или сигмоидный элемент имеет один входной сигнал и один параметр. Сторонники чистого коннекционистского подхода считают, что обучаться в ходе обучения нейронной сети могут только веса связей. С этой точки зрения параметр сигмоидного элемента является не обучаемым и, как следствие, для него нет необходимости вычислять поправку. Однако, часть исследователей полагает, что нужно обучать все параметры всех элементов сети. Исходя из этого, опишем вычисление этим элементом поправки к содержащемуся в нем параметру.

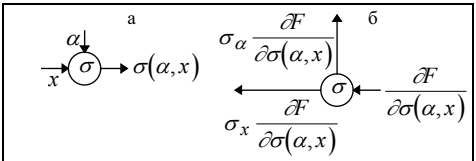


Рис. 10. Прямое (а) и обратное (б) функционирование нелинейного сигмоидного преобразователя

Обозначим входной сигнал через x, параметр через α, а вычисляемую этим преобразователем функцию через $\sigma(\alpha,x)$ (рис. 10а). При обратном функционировании на выход сигмоидного элемента подается сигнал $\partial F/\partial\sigma(\alpha,x)$. На входе сигнала должен быть получен сигнал обратного функционирования, равный $\frac{\partial F}{\partial x}=\frac{\partial F}{\partial\sigma(\alpha,x)}\frac{\partial\sigma(\alpha,x)}{\partial x}=\sigma_x\frac{\partial F}{\partial\sigma(\alpha,x)}$, а на входе параметра поправка, равная $\frac{\partial F}{\partial\alpha}=\frac{\partial F}{\partial\sigma(\alpha,x)}\frac{\partial\sigma(\alpha,x)}{\partial\alpha}=\sigma_\alpha\frac{\partial F}{\partial\sigma(\alpha,x)}$ (рис. 10б).

5.1.3.7 Произвольный непрерывный нелинейный преобразователь

Произвольный непрерывный нелинейный преобразователь имеет несколько входных сигналов, а реализуемая им функция зависит от нескольких параметров. Выходной сигнал такого элемента вычисляется как некоторая функция $\varphi(\mathbf{x},\alpha)$, где $\mathbf{x}$ – вектор входных сигналов, а α – вектор параметров. При обратном функционировании на выходную связь элемента подается сигнал обратного функционирования, равный $\partial F/\partial\varphi$. На входы сигналов выдаются сигналы обратного функционирования, равные $\frac{\partial F}{\partial x_i}=\frac{\partial F}{\partial\varphi}\frac{\partial\varphi}{\partial x_i}=\varphi_{x_i}\frac{\partial F}{\partial\varphi}$, а на входах параметров вычисляются поправки, равные $\frac{\partial F}{\partial\alpha_i}=\frac{\partial F}{\partial\varphi}\frac{\partial\varphi}{\partial\alpha_i}=\varphi_{\alpha_i}\frac{\partial F}{\partial\varphi}$.

5.1.3.8 Пороговый преобразователь

Пороговый преобразователь, реализующий функцию определения знака (рис. 11а), не является элементом с непрерывной функцией, и, следовательно, его обратное функционирование не может быть определено из требования вычисления градиента. Однако, при обучении сетей с пороговыми преобразователями полезно иметь возможность вычислять поправки к параметрам. Так как для порогового элемента нельзя определить однозначное поведение при обратном функционировании, предлагается доопределить его, исходя из соображений полезности при конструировании обучаемых сетей. Основным методом обучения сетей с пороговыми элементами является правило Хебба (подробно рассмотрено во второй части главы). Оно состоит из двух процедур, состоящих в изменении «весов связей между одновременно активными нейронами». Для этого правила пороговый элемент при обратном функционировании должен выдавать сигнал обратного функционирования, совпадающий с выданным им сигналом прямого функционирования (рис. 11б). Такой пороговый элемент будем называть зеркальным. При обучении сетей Хопфилда, подробно рассмотренном

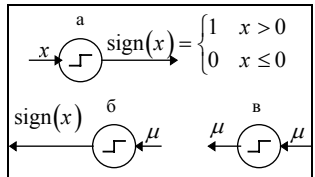


Рис. 11. Прямое (а) и обратное (б,в) функционирование порогового элемента.
б) “Зеркальный” пороговый элемент
в) “Прозрачный” пороговый элемент

во второй части главы, необходимо использовать «прозрачные» пороговые элементы, которые при обратном функционировании пропускают сигнал без изменения (рис. 11в).

5.1.4 Правила остановки работы сети

При использовании сетей прямого распространения (сетей без циклов) вопроса об остановке сети не возникает. Действительно, сигналы поступают на элементы первого (входного) слоя и, проходя по связям, доходят до элементов последнего слоя. После снятия сигналов с последнего слоя все элементы сети оказываются «обесточенными», то есть ни по одной связи сети не проходит ни одного ненулевого сигнала. Сложнее обстоит дело при использовании сетей с циклами. В случае общего положения, после подачи сигналов на входные элементы сети по связям между элементами, входящими в цикл, ненулевые сигналы будут циркулировать сколь угодно долго.

Существует два основных правила остановки работы сети с циклами. Первое правило состоит в остановке работы сети после указанного числа срабатываний каждого элемента. Циклы с таким правилом остановки будем называть ограниченными.

Второе правило остановки работы сети – сеть прекращает работу после установления равновесного распределения сигналов в цикле. Такие сети будем называть равновесными. Примером равновесной сети может служить сеть Хопфилда (см. разд. "Сети Хопфилда").

5.1.5 Архитектуры сетей

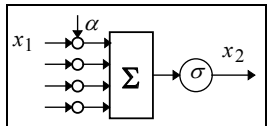


Рис. 12. Фрагмент немонотонной сети

Как уже отмечалось ранее, при конструировании сетей из элементов можно построить сеть любой архитектуры. Однако и при произвольном конструировании можно выделить наиболее общие признаки, существенно отличающие одну сеть от другой. Очевидно, что замена простого сумматора на адаптивный или даже на квадратичный не приведут к существенному изменению структуры сети, хотя число обучаемых параметров увеличится. Однако, введение в сеть цикла сильно изменяет как структуру сети, так и ее поведение. Таким образом можно все сети разбить на два сильно отличающихся класса: **ациклические** сети и **сети с циклами.** Среди сетей с циклами существует еще одно разделение, сильно влияющее на способ функционирования сети: **равновесные сети с циклами** и **сети с ограниченными циклами**.

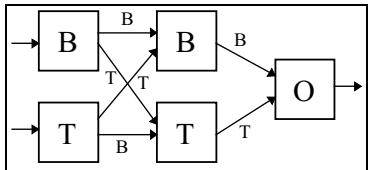


Рис. 13. Общая схема монотонной сети. Верхний ряд - возбуждающие блоки нейронов, нижний ряд - тормозящие. Буквой “Т” - помечены тормозящие связи, буквой “В” - возбуждающие.

Большинство используемых сетей не позволяют определить, как повлияет изменение какого-либо внутреннего параметра сети на выходной сигнал. На рис. 12 приведен пример сети, в которой увеличение параметра α приводит к неоднозначному влиянию на сигнал x_2: при отрицательных x_1 произойдет уменьшение x_2, а при положительных x_1 – увеличение. Таким образом, выходной сигнал такой сети немонотонно зависит от параметра α. Для получения монотонной зависимости выходных сигналов сети от параметров внутренних слоев (то есть всех слоев кроме входного) необходимо использовать специальную монотонную архитектуру нейронной сети. Принципиальная схема сетей монотонной архитектуры приведена на рис. 13.

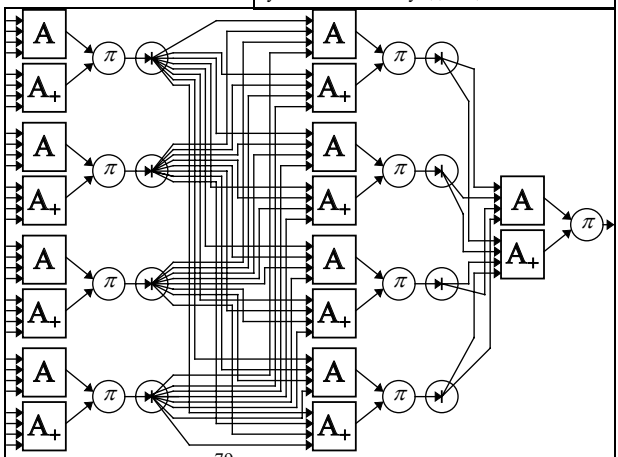


Рис. 14. Немонотонная сеть с Паде элементами

Основная идея построения монотонных сетей состоит в разделении каждого слоя сети на два – возбуждающий и тормозящий. При этом все связи в сети устроены так, что элементы возбуждающей части слоя возбуждают элементы возбуждающей части следующего слоя и тормозят тормозящие элементы следующего слоя. Аналогично, тормозящие элементы возбуждают тормозящие элементы и тормозят возбуждающие элементы следующего слоя. Названия «тормозящий» и «возбуждающий» относятся к влиянию элементов обеих частей на выходные элементы.

Отметим, что для сетей с сигмоидными элементами требование монотонности означает, что веса всех связей должны быть неотрицательны. Для сетей с Паде элементами требование не отрицательности весов связей является необходимым условием бессбойной работы. Требование монотонности для сетей с Паде элементами приводит к изменению архитектуры сети, не накладывая никаких новых ограничений на параметры сети. На рис. 14 приведены пример немонотонной сети, а на рис. 15 монотонной сети с Паде элементами.

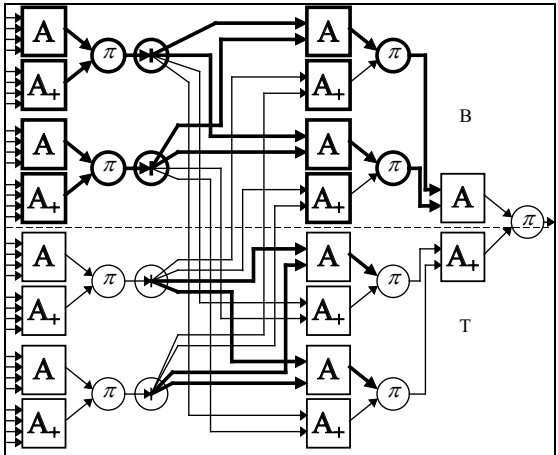


Рис. 15. Монотонная сеть с Паде элементами . Жирными линиями обозначены возбуждающие связи и элементы возбуждающей части сети

Особо отметим архитектуру еще одного класса сетей – сетей без весов связей. Эти сети, в противовес коннекционистским, не имеют обучаемых параметров связей. Любую сеть можно превратить в сеть без весов связей заменой всех синапсов на умножители. Легко заметить, что получится такая же сеть, только вместо весов связей будут использоваться сигналы. Таким образом в сетях без весов связей выходные сигналы одного слоя могут служить для следующего слоя как входными сигналами, так и весами связей. Заметим, что вся память таких сетей содержится в значениях параметров нелинейных преобразователей. Из разделов "Синапс" и "Умножитель" следует, что сети без весов связей способны вычислять градиент функции оценки и затрачивают на это ровно тоже время, что и аналогичная сеть с весами связей.

5.1.6 Модификация синаптической карты (обучение)

Кроме прямого и обратного функционирования, все элементы должны уметь выполнять еще одну операцию – модификацию параметров. Процедура модификации параметров состоит в добавлении к существующим параметрам вычисленных поправок (напомним, что для сетей с непрерывно дифференцируемыми элементами вектор поправок является градиентом некоторой функции от выходных сигналов). Если обозначить текущий параметр элемента через α, а вычисленную поправку через $\Delta\alpha$, то новое значение параметра вычисляется по формуле $\alpha' = h_1\alpha + h_2\Delta\alpha$. Параметры обучения h_1 и h_2 определяются компонентом учитель и передаются сети вместе с запросом на обучение. В некоторых случаях бывает полезно использовать более сложную процедуру модификации карты.

Во многих работах отмечается, что при описанной выше процедуре модификации параметров происходит неограниченный рост величин параметров. Существует несколько различных методов решения этой проблемы. Наиболее простым является жесткое ограничение величин параметров некоторыми минимальным и максимальным значениями. При использовании этого метода процедура модификации параметров имеет следующий вид:

$$\alpha' = \begin{cases} \alpha_{\min}, & h_1\alpha + h_2\Delta\alpha < \alpha_{\min}, \\ h_1\alpha + h_2\Delta\alpha, & \alpha_{\min} \le h_1\alpha + h_2\Delta\alpha < \alpha_{\max}, \\ \alpha_{\max}, & \alpha_{\max} < h_1\alpha + h_2\Delta\alpha. \end{cases}$$

5.1.7 Контрастирование и нормализация сети

В последние годы широкое распространение получили различные методы контрастирования или скелетонизации нейронных сетей. В ходе процедуры контрастирования достигается высокая степень разреженности синаптической карты нейронной сети, так как большинство связей получают нулевые веса (см. например [47, 99, 302. 303]).

Очевидно, что при такой степени разреженности ненулевых параметров проводить вычисления так, как будто структура сети не изменилась, неэффективно. Возникает потребность в процедуре нормализации сети, то есть фактического удаления нулевых связей из сети, а не только из обучения. Процедура нормализации состоит из двух этапов:

1. Из сети удаляются все связи имеющие нулевые веса и исключенные из обучения.
2. Из сети удаляются все подсети, выходные сигналы которых не используются другими подсетями в качестве входных сигналов и не являются выходными сигналами сети в целом.

В ходе нормализации возникает одна трудность: если при описании нейронной сети все нейроны одинаковы, и можно описать нейрон один раз, то после удаления отконтрастированных связей нейроны обычно имеют различную структуру. Компонент сеть должен отслеживать ситуации, когда два блока исходно одного и того же типа уже не могут быть представлены в виде этого блока с различными параметрами. В этих случаях компонент сеть порождает новый тип блока. Правила порождения имен блоков приведены в описании выполнения запроса на нормализацию сети.

5.2 Примеры сетей и алгоритмов их обучения

В этом разделе намеренно допущено отступление от общей методики – не смешивать разные компоненты. Это сделано для облегчения демонстрации построения нейронных сетей обратного распространения, позволяющих реализовать на них большинство известных алгоритмов обучения нейронных сетей.

5.2.1 Сети Хопфилда

Классическая сеть Хопфилда, функционирующая в дискретном времени, строится следующим образом. Пусть $\left\{\mathbf{e}^i\right\}$ – набор эталонных образов $(i=1,\ldots,m)$. Каждый образ, включая и эталоны, имеет вид n-мерного вектора с координатами, равными нулю или единице. При предъявлении на вход сети образа $\mathbf{x}$ сеть вычисляет образ, наиболее похожий на $\mathbf{x}$. В качестве меры близости образов выберем скалярное произведение соответствующих векторов. Вычисления проводятся по следующей формуле: $\mathbf{x}' = \operatorname{sign}\left(\sum_{i=\mathbf{1}}^{m}\left(\mathbf{x},\mathbf{e}^i\right)\mathbf{e}^i\right)$. Эта процедура выполняется до тех пор, пока после очередной итерации не окажется, что $\mathbf{x}=\mathbf{x}'$. Вектор $\mathbf{x}$, полученный в ходе последней итерации, считается ответом. Для нейросетевой реализации формула работы сети переписывается в следующем виде:

$$\mathbf{x}'_j = \operatorname{sign}\left(\sum_{i=1}^{m}\left(\mathbf{x},\mathbf{e}^i\right)e^i_j\right) = \operatorname{sign}\left(\sum_{i=1}^{m}\sum_{k=1}^{n}x_k e^i_k e^i_j\right) = \operatorname{sign}\left(\sum_{k=1}^{n}x_k\sum_{i=1}^{m}e^i_k e^i_j\right) = \operatorname{sign}\left(\sum_{k=1}^{n}a_{jk}x_k\right)$$

или

$$\mathbf{x}' = \operatorname{sign}(\mathbf{A}\mathbf{x}),$$

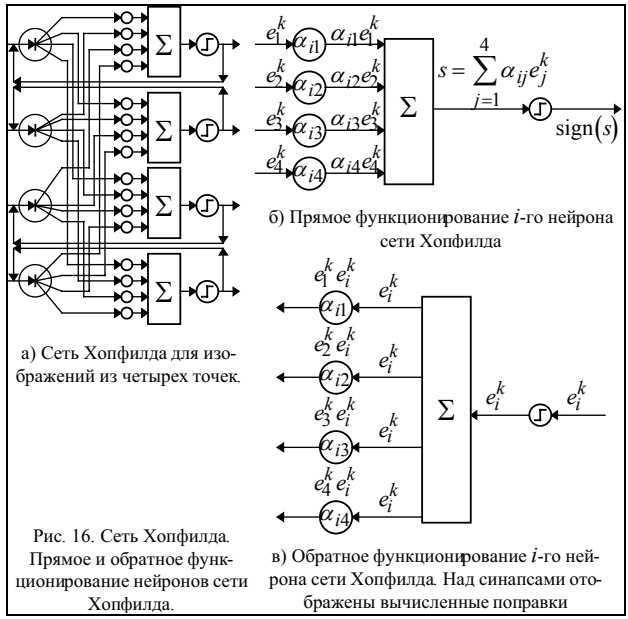
где $a_{jk} = a_{kj} = \sum_{i=1}^{m} e^i_j e^i_k$.

На рис. 16 приведена схема сети Хопфилда для распознавания четырехмерных образов. Обычно сети Хопфилда относят к сетям с формируемой синаптической картой. Однако, используя разработанный в первой части главы набор элементов, можно построить обучаемую сеть. Для построения такой сети используем «прозрачные» пороговые элементы. Ниже приведен алгоритм обучения сети Хопфилда.

1. Положим все синаптические веса равными нулю.

2. Предъявим сети первый эталон $\mathbf{e}^1$ и проведем один такт функционирования вперед, то есть цикл будет работать не до равновесия, а один раз (см. рис. 16б).

3. Подадим на выход каждого нейрона соответствующую координату вектора $\mathbf{e}^1$ (см. рис. 16в). Поправка,



а) Сеть Хопфилда для изображений из четырех точек.

б) Прямое функционирование i-го нейрона сети Хопфилда

в) Обратное функционирование i-го нейрона сети Хопфилда. Над синапсами отображены вычисленные поправки

Рис. 16. Сеть Хопфилда. Прямое и обратное функционирование нейронов сети Хопфилда.

вычисленная на j-ом синапсе i-го нейрона равна произведению сигнала прямого функционирования на сигнал обратного функционирования. Поскольку при обратном функционировании пороговый элемент прозрачен, а сумматор переходит в точку ветвления, то поправка равна $e_i^1 e_j^1$.

4. Далее проведем шаг обучения с параметрами обучения, равными единице. В результате получим $\alpha_{ij} = e_i^1 e_j^1$.

Повторяя этот алгоритм, начиная со второго шага, для всех эталонов получим $a_{ij} = \sum_{i=1}^{m} e_i^k e_j^k$, что полностью совпадает с формулой формирования синаптической карты сети Хопфилда, приведенной в начале раздела.

5.2.2 Сеть Кохонена

Сети Кохонена [98, 99] (частный случай метода динамических ядер [223, 261]) являются типичным представителем сетей решающих задачу классификации без учителя. Рассмотрим пространственный вариант сети Кохонена. Дан набор из m точек $\left\{\mathbf{x}^p\right\}$ в n-мерном пространстве. Необходимо разбить множество точек $\left\{\mathbf{x}^p\right\}$ на k классов близких в смысле квадрата евклидова расстояния. Для этого необходимо найти k точек α^l таких, что $D=\sum_{l=1}^{k}\sum_{x\in P_l}\left\|\alpha^l-x\right\|$, минимально; $P_l=\left\{x:\left\|\alpha^l-x\right\|<\left\|\alpha^q-x\right\|,\forall q\neq l\right\}$.

Существует множество различных алгоритмов решения этой задачи. Рассмотрим наиболее эффективный из них.

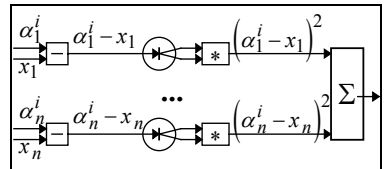
1. Зададимся некоторым набором начальных точек α^l.

2. Разобьем множество точек $\left\{\mathbf{x}^p\right\}$ на k классов по правилу $P_l=\left\{x:\left\|\alpha^l-x\right\|<\left\|\alpha^q-x\right\|,\forall l\neq q\right\}$.

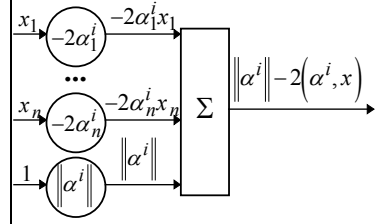
3. По полученному разбиению вычислим новые точки α^l из условия минимальности $D_l=\sum_{x\in P_l}\left\|\alpha^l-x\right\|$.

Обозначив через $\left|P_i\right|$ число точек в i-ом классе, решение задачи, поставленной на третьем шаге алгоритма, можно записать в виде $\alpha^i=\frac{1}{\left|P_i\right|}\sum_{x\in P_i}x$.

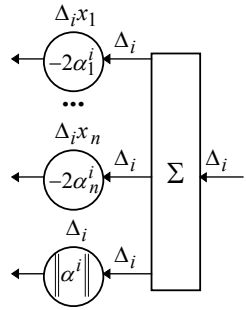
Второй и третий шаги алгоритма будем повторять до тех пор, пока набор точек α^l не перестанет изменяться. После окончания обучения получаем нейронную сеть, способную для произвольной точки x вычислить квадраты евклидовых расстояний от этой точки до всех точек α^l и, тем самым, отнести ее к одному из k классов. Ответом является номер нейрона, выдавшего минимальный сигнал.



а) Нейрон сети Кохонена, вычисляющий квадрат евклидова расстояния



б) Прямое функционирование i-го нейрона сети Кохонена



в) Обратное функционирование i-го нейрона сети Кохонена. Над синапсами отмечены вычисленные на последнем шаге поправки.

Рис. 17. Сеть Кохонена. Прямое и обратное функционирование нейронов сети Кохонена.

Теперь рассмотрим сетевую реализацию. Во первых, вычисление квадрата евклидова расстояния достаточно сложно реализовать в виде сети (рис. 17а). Однако заметим, что нет необходимости вычислять квадрат расстояния полностью. Действительно,

$$\left\|\alpha^l-x\right\|=\left(\alpha^l-x,\alpha^l-x\right)=\left\|\alpha^l\right\|-2\left(\alpha^l,x\right)+\left\|x\right\|.$$

Отметим, что в последней формуле первое слагаемое не зависит от точки x, второе вычисляется адаптивным сумматором, а третье одинаково для всех сравниваемых величин. Таким образом, легко получить нейронную сеть, которая вычислит для каждого класса только первые два слагаемых (рис. 17б).

Второе соображение, позволяющее упростить обучение сети, состоит в отказе от разделения второго и третьего шагов алгоритма.

Алгоритм классификации.

1. На вход нейронной сети, состоящей из одного слоя нейронов, приведенных на рис. 17б, подается вектор x.

2. Номер нейрона, выдавшего минимальный ответ, является номером класса, к которому принадлежит вектор x.

Алгоритм обучения.

1. Полагаем поправки всех синапсов равными нулю.
2. Для каждой точки множества $\left\{\mathbf{x}^p\right\}$ выполняем следующую процедуру.
   2.1. Предъявляем точку сети для классификации.
   2.2. Пусть при классификации получен ответ – класс l. Тогда для обратного функционирования сети подается вектор Δ, координаты которого определяются по следующему правилу: $\Delta_i = \begin{cases} 0, & i \neq l \\ 1, & i = l \end{cases}$.
   2.3. Вычисленные для данной точки поправки добавляются к ранее вычисленным.
3. Для каждого нейрона производим следующую процедуру.
   3.1. Если поправка, вычисленная последним синапсом равна 0, то нейрон удаляется из сети.
   3.2. Полагаем параметр обучения равным величине, обратной к поправке, вычисленной последним синапсом.
   3.3. Вычисляем сумму квадратов накопленных в первых n синапсах поправок и, разделив на -2, заносим в поправку последнего синапса.
   3.4. Проводим шаг обучения с параметрами $h_1 = 0,\ h_2 = -2$.
4. Если вновь вычисленные синаптические веса отличаются от полученных на предыдущем шаге, то переходим к первому шагу алгоритма.

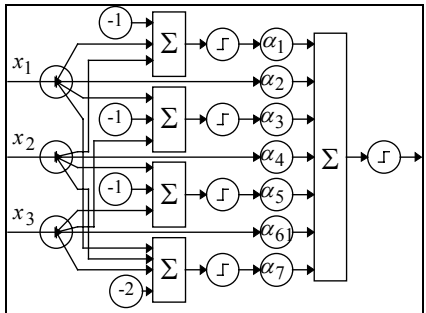
В пояснении нуждается только второй и третий шаги алгоритма. Из рис. 16в видно, что вычисленные на шаге 2.2 алгоритма поправки будут равны нулю для всех нейронов, кроме нейрона, выдавшего минимальный сигнал. У нейрона, выдавшего минимальный сигнал, первые n поправок будут равны координатам распознававшейся точки x, а поправка последнего синапса равна единице. После завершения второго шага алгоритма поправка последнего синапса i-го нейрона будет равна числу точек, отнесенных к i-му классу, а поправки остальных синапсов этого нейрона равны сумме соответствующих координат всех точек i-го класса. Для получения правильных весов остается только разделить все поправки первых n синапсов на поправку последнего синапса, положить последний синапс равным сумме квадратов полученных величин, а остальные синапсы – полученным для них поправкам, умноженным на -2. Именно это и происходит при выполнении третьего шага алгоритма.

5.2.3 Персептрон Розенблатта

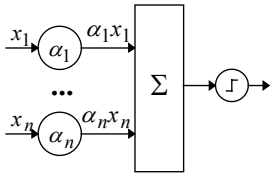
Персептрон Розенблатта является исторически первой обучаемой нейронной сетью. Существует несколько версий персептрона. Рассмотрим классический персептрон – сеть с пороговыми нейронами и входными сигналами, равными нулю или единице. Опираясь на результаты, изложенные в работе [145] можно ввести следующие ограничения на структуру сети.

1. Все синаптические веса могут быть целыми числами.
2. Многослойный персептрон по своим возможностям эквивалентен двухслойному. Все нейроны имеют синапс, на который подается постоянный единичный сигнал. Вес этого синапса далее будем называть порогом. Каждый нейрон первого слоя имеет единичные синаптические веса на всех связях, ведущих от входных сигналов, и его порог равен числу входных сигналов сумматора, уменьшенному на два и взятому со знаком минус.

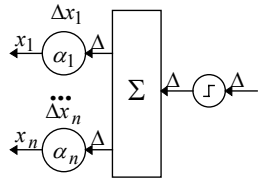
Таким образом, можно ограничиться рассмотрением только двухслойных персептронов с не обучаемым первым слоем. Заметим, что для построения полного первого слоя пришлось бы использовать 2^n нейронов, где n – число входных сигналов персептрона. На рис. 18а приведена схема полного персептрона для трехмерного вектора входных сигналов. Поскольку построение такой сети при достаточно большом n невозможно, то обычно используют некоторое подмножество нейронов первого слоя. К сожалению, только полностью решив задачу можно точно указать необходимое подмножество. Обычно используемое подмножество выбирается исследователем из каких-то содержательных соображений или случайно.



а) Полный персепрон Розенблатта с тремя входными сигналами.



б) Прямое функционирование второго слоя персептрона Розенблатта.



в) Обратное функционирование второго слоя персептрона Розенблатта.

Рис. 18. Персептрон Розенблатта. Прямое и обратное функционирование второго слоя персептрона Розенблатта.

Классический алгоритм обучения персептрона является частным случаем правила Хебба. Поскольку веса связей первого слоя персептрона являются не обучаемыми, веса нейрона второго слоя в дальнейшем будем называть просто весами. Будем считать, что при предъявлении примера первого класса персептрон должен выдать на выходе нулевой сигнал, а при предъявлении примера второго класса – единичный. Ниже приведено описание алгоритма обучения персептрона.

1. Полагаем все веса равными нулю.
2. Проводим цикл предъявления примеров. Для каждого примера выполняется следующая процедура.
   2.1. Если сеть выдала правильный ответ, то переходим к шагу 2.4.
   2.2. Если на выходе персептрона ожидалась единица, а был получен ноль, то веса связей, по которым прошел единичный сигнал, уменьшаем на единицу.
   2.3. Если на выходе персептрона ожидался ноль, а была получена единица, то веса связей, по которым прошел единичный сигнал, увеличиваем на единицу.
   2.4. Переходим к следующему примеру. Если достигнут конец обучающего множества, то переходим к шагу 3, иначе возвращаемся на шаг 2.1.
3. Если в ходе выполнения второго шага алгоритма хоть один раз выполнялся шаг 2.2 или 2.3 и не произошло зацикливания, то переходим к шагу 2. В противном случае обучение завершено.

В этом алгоритме не предусмотрен механизм отслеживания зацикливания обучения. Этот механизм можно реализовывать по разному. Наиболее экономный в смысле использования дополнительной памяти имеет следующий вид.

1. k=1; m=0. Запоминаем веса связей.

2. После цикла предъявлений образов сравниваем веса связей с запомненными. Если текущие веса совпали с запомненными, то произошло зацикливание. В противном случае переходим к шагу 3.
3. $m=m+1$. Если $m<k$, то переходим ко второму шагу.
4. $k=2k$; $m=0$. Запоминаем веса связей и переходим к шагу 2.

Поскольку длина цикла конечна, то при достаточно большом k зацикливание будет обнаружено.

Для использования в обучении сети обратного функционирования, необходимо переписать второй шаг алгоритма обучения в следующем виде.

2. Проводим цикл предъявления примеров. Для каждого примера выполняется следующая процедура.
   2.1. Если сеть выдала правильный ответ, то переходим к шагу 2.5.
   2.2. Если на выходе персептрона ожидалась единица, а был получен ноль, то на выход сети при обратном функционировании подаем $\Delta = -1$.
   2.3. Если на выходе персептрона ожидался ноль, а была получена единица, то на выход сети при обратном функционировании подаем $\Delta = 1$.
   2.4. Проводим шаг обучения с единичными параметрами.
   2.5. Переходим к следующему примеру. Если достигнут конец обучающего множества, то переходим к шагу 3, иначе возвращаемся на шаг 2.1.

На рис. 18в приведена схема обратного функционирования нейрона второго слоя персептрона. Учитывая, что величины входных сигналов этого нейрона равны нулю или единице, получаем эквивалентность модифицированного алгоритма исходному. Отметим также, что при обучении персептрона впервые встретились не обучаемые параметры – веса связей первого слоя.

5.3 Стандарт первого уровня компонента сеть

Данный раздел посвящен описанию стандарта хранения компонента сеть на внешних носителях.

5.3.1 Структура компонента

Рассмотрим более подробно структуры данных сети. Как уже было описано в первой части главы, сеть строится иерархически от простых подсетей к сложным. Простейшими подсетями являются элементы. Подсеть каждого уровня имеет свое имя и тип. Существуют следующие типы подсетей: элемент, каскад, слой, цикл с фиксированным числом тактов функционирования и цикл, функционирующий до тех пор, пока не выполнится некоторое условие. Последние четыре типа подсетей будем называть блоками. Имена подсетей определяются при конструировании. В разделе «Имена структурных единиц компонентов» главы «Общий стандарт» приведены правила построения полного и однозначного имен подсети. В качестве примера рассмотрим сеть, конструирование которой проиллюстрировано в первой части главына рис. 2. В описании сети NW однозначное имя первого нейрона второго слоя имеет вид K[2].SN.N[1]. При описании слоя однозначное имя первого нейрона записывается как N[1]. В квадратных скобках указываются номер экземпляра подсети, входящей в непосредственно содержащую ее структуру в нескольких экземплярах.

5.3.2 Сигналы и параметры

При использовании контрастирования для изменения структуры сети и значений обучаемых параметров другим компонентам бывает необходим прямой доступ к сигналам и параметрам сети в целом или отдельных ее подсетей. Для адресации входных и выходных сигналов используются имена InSignals и OutSignals, соответственно. Таким образом, для получения массива входных сигналов второго слоя сети, приведенной на рис. 2, необходимо запросить массив NW.K[2].InSignals, а для получения выходного сигнала всей сети можно воспользоваться любым из следующего списка имен:

- NW.OutSignals;
- NW.N.OutSignals.

Для получения конкретного сигнала из массива сигналов необходимо в конце в квадратных скобках указать номер сигнала. Например, для получения третьего входного сигнала второго слоя сети нужно указать следующее имя – NW.K[2].InSignals[3].

Для получения доступа к параметрам нужно указать имя подсети, к чьим параметрам нужен доступ и через точку ключевое слово Parameters. При необходимости получить конкретный параметр, его номер в квадратных скобках записывается после ключевого слова Parameters.

5.3.3 Обучаемые и не обучаемые параметры и сигналы

При обучении параметров и сигналов (использование обучения сигналов описано во введении) возникает необходимость обучать только часть из них. Так, например, при описании обучения персептрона во второй части этой главы было отмечено, что обучать необходимо только веса связей второго слоя. Для реализации этой возможности используются два массива логических переменных – маска обучаемых параметров и маска обучаемых входных сигналов.

5.3.4 Дополнительные переменные

При описании структуры сетей необходимо учитывать следующую дополнительные переменные, доступные в методах Forw и Back. Для каждой сети при прямом функционировании определен следующий набор переменных:

- InSignals[K] – массив из K действительных чисел, содержащих входные сигналы прямого функционирования.
- OutSignals[N] – массив из N действительных чисел, в которые заносятся выходные сигналы прямого функционирования.
- Parameters[M] – массив из M действительных чисел, содержащих параметры сети.

При выполнении обратного функционирования сети доступны еще три массива:

- Back.InSignals[K] – массив из K действительных чисел, параллельный массиву InSignals, в который заносятся выходные сигналы обратного функционирования.
- Back.OutSignals[N] – массив из N действительных чисел, параллельный массиву OutSignals, содержащий входные сигналы обратного функционирования.
- Back.Parameters[M] – массив из M действительных чисел, параллельный массиву Parameters, в который заносятся вычисленные при обратном функционировании поправки к параметрам сети.

При обучении (модификации параметров или входных сигналов) доступны все переменные обратного функционирования и еще два массива:

- InSignalMask[K] – массив из K логических переменных, параллельный массиву InSignals, содержащий маску обучаемости входных сигналов.
- ParamMask[M] – массив из M логических переменных, параллельный массиву Parameters, содержащий маску обучаемости параметров.

5.3.5 Стандарт языка описания сетей

Язык описания нейронных сетей предназначен для хранения сетей на диске. Следует отметить, что в отличии от таких компонентов, как предобработчик входных сигналов, оценка или задачник описание даже простой сети имеет большой размер. С другой стороны, многие подсети являются стандартными для большинства сетей. Для компонента сеть нет смысла вводить небольшой набор стандартных элементов и подсетей, поскольку этот набор может легко расширяться. Более эффективным является выделение часто употребляемых подсетей в отдельные библиотеки, подключаемые к описаниям конкретных сетей. В приведенных в этой главе примерах описания нейронных сетей выделен ряд библиотек.

5.3.5.1 Ключевые слова языка

В табл. 2 приведен список ключевых слов специфических для языка описания сетей.

Таблица 2.

Ключевые слова языка описания сетей.

Ключевое слово	Краткое описание
1. Back	Метод, осуществляющий обратное функционирование подсети. Префикс сигналов обратного функционирования.
2. Block	Тип аргумента подсети. Означает, что аргумент является подсетью.
3. Cascad	Тип подсети – каскад.
4. Connections	Начало блока описания связей подсети.
5. Contents	Начало блока описания состава подсети.
6. DefaultType	Тип параметров по умолчанию.
7. Element	Тип подсети – элемент.
8. Forw	Метод, осуществляющий прямое функционирования подсети.
9. InSignalMask	Имя, по которому адресуются маски обучаемости входных сигналов подсети.
10. InSignals	Имя, по которому адресуются входные сигналы подсети; начало блока описания входных сигналов.
11. Layer	Тип подсети – слой.
12. Loop	Тип подсети – цикл, выполняемый указанное число раз.
13. MainNet	Начало описания главной сети
14. NetLib	Начало описания библиотеки подсетей.
15. NetWork	Начало описания сети.
16. NumberOf	Функция (запрос). Возвращает число параметров или сигналов в подсети.
17. OutSignals	Имя, по которому адресуются выходные сигналы подсети; начало блока описания выходных сигналов.
18. ParamDef	Заголовок определения типа параметров.
19. Parameters	Имя, по которому адресуются параметры подсети; начало блока описания параметров.
20. ParamMask	Имя, по которому адресуются маски обучаемости параметров подсети.
21. ParamType	Заголовок описания типа параметров.
22. Until	Тип подсети – цикл, выполняемый до тех пор пока не выполнится условие.
23. Used	Начало списка подключаемых библиотек подсетей

5.3.5.2 БНФ языка описания сетей

Обозначения, принятые в данном расширении БНФ и описание ряда конструкций приведены в главе “Общий стандарт” в разделе “Описание языка описания компонентов”.

<Описание библиотеки подсетей> ::= <Заголовок библиотеки> <Описание подсетей> <Конец описания библиотеки>

<Заголовок библиотеки> ::= **NetLib** <Имя библиотеки> [**Used** <Список имен библиотек>]

<Имя библиотеки> ::= <Идентификатор>
<Список имен библиотек> ::= <Имя используемой библиотеки> [,<Список имен библиотек>]
<Имя используемой библиотеки> ::= <Идентификатор>
<Описание подсетей> ::= <Описание подсети> [<Описание подсетей>]
<Описание подсети> ::= {<Описание элемента> | <Описание блока> | <Описание функций>}
<Описание элемента> ::= <Заголовок описания элемента> <Описание сигналов и параметров> [<Описание типов параметров>] [<Определение типов параметров>] [<Описание статических переменных>] [<Установление значений статических переменных>] <Описание методов> <Конец описания элемента>
<Заголовок описания элемента> ::= **Element** <Имя элемента> [*(*<Список формальных аргументов>*)*]
<Имя элемента> ::= <Идентификатор>
<Описание сигналов и параметров> ::= <Описание входных сигналов> <Описание выходных сигналов> [<Описание параметров>]
<Описание входных сигналов> ::= **InSignals** <Константное выражение типа ***Long***>
<Описание выходных сигналов> ::= **OutSignals** <Константное выражение типа ***Long***>
<Описание параметров> ::= **Parameters** <Константное выражение типа ***Long***>
<Описание типов параметров> ::= <Описание типа параметров> [<Описание типов параметров>]
<Описание типа параметров> ::= **ParamType** <Имя типа параметра> <Список>
<Имя типа параметра> ::= <Идентификатор>
<Список> ::= {**Parameters/**<Начальный номер> [..<Конечный номер> [<Шаг>]]*/* | **InSignals/**<Начальный номер> [..<Конечный номер> [<Шаг>]]*/*} [;<Список>]
<Определение типов параметров> ::= <Определение типа параметра> [<Определение типов параметров>]
<Определение типа параметра> ::= **ParamDef** <Имя типа параметра> <Минимальное значение> <Максимальное значение>
<Минимальное значение> ::= <Константное выражение типа ***Real***>
<Максимальное значение> ::= <Константное выражение типа ***Real***>
<Установление значений статических переменных> ::= <Установление параметров ***Подсети***> [;<Установление значений статических переменных>]
<Описание методов> ::= <Описание функционирования вперед> <Описание функционирования назад>
<Описание функционирования вперед> ::= **Forw** [<Описание переменных>] <Тело метода>
<Тело метода> ::= **Begin** <Составной оператор> **End**
<Описание функционирования назад> ::= **Back** [<Описание переменных>] <Тело метода>
<Конец описания элемента> ::= **End** <Имя элемента>
<Описание блока> ::= <Заголовок описания блока> <Описание состава> <Описание сигналов и параметров> [<Описание статических переменных>] [<Установление значений статических переменных>] <Описание связей> [<Определение типов параметров>] <Конец описания блока>
<Заголовок описания блока> ::= {<Описание каскада> | <Описание слоя> | <Описание цикла с фиксированным числом шагов> | <Описание цикла по условию>}
<Описание каскада> ::= **Cascad** <Имя блока> [*(*<Список формальных аргументов блока>*)*]
<Имя блока> ::= <Идентификатор>
<Список формальных аргументов блока> ::= {<Список формальных аргументов> | <Аргумент – подсеть>} [;<Список формальных аргументов блока>]
<Аргумент – подсеть>::= <Список имен аргументов – подсетей> : **Block**
<Список имен аргументов – подсетей> ::= <Имя аргумента – подсети> [,<Список имен аргументов – подсетей>]
<Имя аргумента – подсети> ::= <Идентификатор>
<Описание слоя> ::= **Layer** <Имя блока> [*(*<Список формальных аргументов блока>*)*]
<Описание цикла с фиксированным числом шагов> ::= **Loop** <Имя блока> [*(*<Список формальных аргументов блока>*)*] <Число повторов цикла>
<Число повторов цикла> ::= <Константное выражение типа ***Long***>
<Описание цикла по условию> ::= **Until** <Имя блока> [*(*<Список формальных аргументов блока>*)*] : <Выражение типа ***Logic***>
<Описание состава> ::= **Contents** <Список имен подсетей>
<Список имен подсетей> ::= <Имя подсети> [,<Список имен подсетей>]
<Имя подсети> ::= <Псевдоним>: {<Имя ранее описанной подсети> [*(*<Список фактических аргументов блока>*)*] [*/*<Число экземпляров>*/*] | <Имя аргумента – подсети> [*/*<Число экземпляров >*/*]}
<Псевдоним> ::= <Идентификатор>

<Число экземпляров > ::= <Константное выражение типа ***Long***>
<Имя ранее описанной подсети> ::= <Идентификатор>
<Список фактических аргументов блока> ::= <Фактический аргумент блока> [,<Список фактических аргументов блока>]
<Фактический аргумент блока> ::= {<Фактический аргумент> | <Имя аргумента – подсети>}
<Описание связей> ::= {<Описание распределения ***Входных сигналов, Блока, Подсети, InSignals*** > | <Описание распределения ***Выходных сигналов, Блока, Подсети, OutSignals*** > | <Описание распределения ***Параметров, Блока, Подсети, Parameters*** >}
<Конец описания блока>::= **End** <Имя блока>
<Конец описания библиотеки> ::= **End NetLib**
<Описание сети> ::= <Заголовок описания сети> <Описание подсетей> <Описание главной сети> <Массивы параметров и масок сети> <Конец описания сети>
<Заголовок описания сети> ::= **NetWork** <Имя сети> [**Used** <Список имен библиотек>]
<Имя сети> ::= <Идентификатор>
<Описание главной сети> ::= **MainNet** <Имя ранее описанной подсети> [*(*<Список фактических аргументов блока>*)*]
<Массивы параметров и масок сети> ::= <Массив параметров> <Массив маски обучаемости параметров>
<Массив параметров> ::= **Parameters** <Значения параметров>;
<Значения параметров> ::= <Действительное число> [, <Значения параметров>]
<Массив маски обучаемости параметров> ::= **ParamMask** <Значения маски>;
<Значения маски> ::= <Константа типа ***Logic***> [,<Значения маски>]
<Конец описания сети> ::= **End NetWork**

5.3.5.3 Описание языка описания сетей

В этом разделе приводится детальное описание языка описания сетей, дополняющее БНФ, приведенную в предыдущем разделе и описание общих конструкций, приведенное в главе «Общий стандарт».

5.3.5.3.1 Описание и область действия переменных

Вспомогательные переменные могут потребоваться при описании прямого и обратного функционирования элементов. Переменная действует только в пределах той процедуры, в которой она описана. Кроме явно описанных переменных, в методе Forw доступны также сигналы прямого функционирования и параметры элемента, а в методе Back – входные и выходные сигналы прямого функционирования, выходные сигналы обратного функционирования, параметры элемента и градиент по параметрам элемента. Во всех методах доступны аргументы элемента.

Статические переменные, описываемые после ключевого слова Static, уникальны для каждого экземпляра элемента или блока, и доступны только в пределах блока. Эти переменные могут потребоваться для вычисления условий в цикле типа Until. Возможно использование таких переменных в элементах, например, для хранения предыдущего состояния элемента. Кроме того, в статической переменной можно хранить значения не обучаемых параметров.

5.3.5.3.2 Методы Forw и Back для блоков

Методы Forw и Back для блоков не описываются в языке описания сетей. Это связано с тем, что при выполнении метода Forw блоком происходит вызов метода Forw составляющих блок подсетей (для элементов – метода Forw) в порядке их описания в разделе описания состава блока. При выполнении метода Back происходит вызов методов Back составляющих блок подсетей в порядке обратном порядку их описания в разделе описания состава блока.

5.3.5.3.3 Описание элементов

Описание элемента состоит из следующих основных разделов: заголовка элемента, описания сигналов и параметров, описания статических переменных и описания методов. Заголовок элемента имеет следующий синтаксис:

Element Имя_Элемента (Аргументы элемента)

Аргументы элемента являются необязательной частью заголовка. В следующем разделе приведены описания нескольких элементов. Отметим, что сигмоидный элемент описан двумя способами: с принципиально не обучаемой (S_NotTrain) и с обучаемой (S_Train) характеристикой.

Раздел описания сигналов и параметров следует сразу после заголовка элемента и состоит из указания числа входных и выходных сигналов и числа параметров элемента. Если у элемента отсутствуют параметры, то указание числа параметров можно опустить. В следующем разделе приведены элементы как имеющие параметры (S_Train, Adaptiv_Sum, Square_Sum), так и элементы без параметров (Sum, S_NotTrain, Branch). Концом раздела описания сигналов и параметров служит одно из ключевых слов ParamType, ParamDef, Forw или Back.

Описание типов параметров является необязательной частью описания элемента и начинается с ключевого слова ParamType. Если раздел описания типов параметров отсутствует, то все параметры этого элемента считаются параметрами типа DefaultType. Если в сети должны присутствовать параметры разных типов (например с разными ограничениями на минимальное и максимальное значение) необходимо описать типы параметров. Концом этого раздела служит одно из ключевых слов ParamDef, Forw или Back.

Раздел определения типов параметров является необязательным разделом в описании элемента и начинается с ключевого слова ParamDef. В каждой строке этого раздела можно задать минимальную и масимальную границы изменения одного типа параметров. Если в описании сети встречаются параметры неопределенного типа то этот тип считается совпадающим с типом DefaultType. Описание типа *не обязано* предшествовать описанию параметров этого типа. Так например, определение типа параметров может находиться в описании главной сети. Концом этого раздела служит одно из ключевых слов Forw или Back.

Раздел описания методов состоит из описания двух методов: Forw и Back. Описание метода состоит из заголовка, раздела описания переменных и тела метода. Заголовок имеет вид ключевого слова Forw или Back для соответствующего метода. Раздел описания переменных состоит из ключевого слова Var, за которым следуют описания однотипных переменных, каждое из которых заканчивается символом «;». Необходимо понимать, что описание заголовков методов это не описание заголовка (прототипа) функции, выполняющей тело метода. Ниже приведен синтаксис заголовков методов Forw и Back на момент вызова:

Pascal:

```
Procedure Forw( InSignals, OutSignals, Parameters : PRealArray);
Procedure Back(InSignals, OutSignals, Parameters, Back.InSignals,
    Back.OutSignals, Back.Parameters : PRealArray);
```

C

```
void Forw(PRealArray InSignals, PRealArray OutSignals, PRealArray Parameters)
void Back(PRealArray InSignals, PRealArray OutSignals, PRealArray Parameters,
PRealArray Back.InSignals, PRealArray Back.OutSignals, PRealArray Back.Parameters)
```

В методе Forw в левой части оператора присваивания могут фигурировать имена любых переменных и элементов предопределенного массива выходных сигналов (OutSignals). В выражении, стоящем в правой части оператора присваивания могут участвовать любые переменные, аргументы элемента и элементы предопределенных массивов входных сигналов (InSignals) и параметров (Parameters).

В методе Back в левой части оператора присваивания могут фигурировать имена любых переменных, элементов предопределенных массивов входных сигналов обратного функционирования (Back.InSignals) и параметров (Back.Parameters). В выражении, стоящем в правой части оператора присваивания, могут участвовать любые переменные, аргументы элемента и элементы предопределенных массивов входных (InSignals) и выходных (OutSignals) сигналов и параметров (Parameters).Отметим важную особенность вычисления поправок к параметрам. Поскольку один и тот же параметр может использоваться несколькими элементами, при вычислении поправки к параметру вычисленное значение нужно не присваивать соответствующему элементу массива Back.Parameters, а добавлять. При этом в теле метода элементы массива Back.Parameters не могут фигурировать в правой части оператора присваивания. Эта особенность вычисления поправок к параметрам обрабатывается компонентом сеть.

Описание элемента завершается ключевым словом End за которым следует имя элемента.

5.3.5.3.4 Пример описания элементов

```
NetBibl Elements;                    {Библиотека элементов}

Element Synaps                       {Обычный синапс}
   InSignals 1                       {Один входной сигнал}
   OutSignals 1                      {Один выходной сигнал}
   Parameters 1                      {Один параметр – вес связи}
```

```
    Forw                                    {Начало описания прямого функционирования}
    Begin
            {Вычисление выходного сигнала как произведения входного сигнала на параметр}
        OutSignals[1] = InSignals[1] * Parameters[1]
    End                                     {Конец описания прямого функционирования}

    Back                                    {Начало описания обратного функционирования }
    Begin
            {Вычисление поправки к входному сигналу как произведения поправки к выходному
            сигналу на параметр}
        Back.InSignals[1] = Back.OutSignals[1] * Parameters[1];
            {Вычисление поправки к параметру как суммы ранее вычисленной поправки к
            параметру на произведение поправки к обратному  сигналу на входной сигнал}
        Back.Parameters[1] = Back.Parameters[1] + Back.OutSignals[1] * InSignals[1]
    End                                     {Конец описания обратного функционирования}
End Synaps                                  {Конец описания синапса}

Element Branch(N : Long)                    {Точка ветвления на N выходных сигналов}
    InSignals 1                             {Один входной сигнал}
    OutSignals N                            {N выходных сигналов}

    Forw                                    {Начало описания прямого функционирования}
        Var Long I;                     {I – локальная переменная типа длинное целое – индекс}
    Begin
        For I=1 To N Do                     {На каждый из N выходных сигналов}
            OutSignals[I] = InSignals[1]    {передаем входной сигнал}
    End                                     {Конец описания прямого функционирования}

    Back                                    {Начало описания обратного функционирования }
        Var                                 {Описание локальных переменных}
            Long I;                         {I – длинное целое – индекс}
            Real R;                         {R – действительное – для накопления суммы}
    Begin
        R = 0;
        For I=1 To N Do                     {Поправка ко входному сигналу равна}
            R = R + Back.OutSignals[I];     {сумме поправок выходных сигналов}
        Back. InSignals[1] = R
    End                                     {Конец описания обратного функционирования}
End Branch                                  {Конец описания точки ветвления}

Element Sum(N Long)                         {Простой сумматор на N входов}
    InSignals N                             {N входных сигналов}
    OutSignals 1                            {Один выходной сигнал}

    Forw                                    {Начало описания прямого функционирования}
        Var                                 {Описание локальных переменных}
            Long I;                         {I – длинное целое – индекс}
            Real R;                         {R – действительное – для накопления суммы}
    Begin
        R = 0;
        For I=1 To N Do                     {Выходной сигнал равен сумме входных}
            R = R + InSignals[I];
        OutSignals[1] = R
    End                                     {Конец описания прямого функционирования}

    Back                                    {Начало описания обратного функционирования}
        Var Long I;                     {I – локальная переменная типа длинное целое – индекс}
    Begin
        For I=1 To N Do                     {Поправка к каждому входному сигналу}
            Back.InSignals[I] = Back.OutSignals[1]    {равна поправке выходного сигнала}
    End                                     {Конец описания обратного функционирования}
End Sum                                     {Конец описания простого сумматора}
```

```
Element Mul                                  {Умножитель}
   InSignals 2                               {Два входных сигнала}
   OutSignals 1                              {Один выходной сигнал}

   Forw                                      {Начало описания прямого функционирования }
   Begin
            {Выходной сигнал равен произведению входных сигналов}
      OutSignals[1] = InSignals[1] * InSignals[2]
   End                                       {Конец описания прямого функционирования}

   Back                                      {Начало описания обратного функционирования }
   Begin
            {Поправка к каждому входному сигналу равна произведению поправки выходного
            сигнала на другой входной сигнал}
      Back.InSignals[1] = Back.OutSignals[1] * InSignals[2];
      Back.InSignals[2] = Back.OutSignals[1] * InSignals[1]
   End                                       {Конец описания обратного функционирования}
End Mul                                      {Конец описания умножителя}

Element S_Train          {Обучаемый гиперболический сигмоидный элемент}
   InSignals 1                               {Один входной сигнал}
   OutSignals 1                              {Один выходной сигнал}
   Parameters 1                                 {Один параметр – характеристика}

   Forw                                      {Начало описания прямого функционирования}
   Begin
            {Выходной сигнал равен отношению входного сигнала к сумме параметра и
            абсолютной величины входного сигнала}
      OutSignals[1] = InSignals[1] / (Parameters[1] + Abs(InSignals[1])
   End                                       {Конец описания прямого функционирования}

   Back                                      {Начало описания обратного функционирования}
      Var Real R;                            {R – действительное}
   Begin
            {R – вспомогательная величина для вычисления поправок, равная отношению
            поправки выходного сигнала к квадрату суммы параметра и абсолютной величины
            входного сигнала}
      R = Back.OutSignals[1] / Sqr(Parameters[1] + Abs(InSignals[1]);
            {Поправка к входному сигналу равна произведению вспомогательной величины на
            параметр}
      Back.InSignals[1] = R * Parameters[1];
            {Поправка к параметру равна сумме ранее вычисленной величины поправки и
            произведения вспомогательной величины на входной сигнал}
      Back.Parameters[1] = Back.Parameters[1] + R * InSignals[1]
   End                                       {Конец описания обратного функционирования}
End S_Train              {Конец описания обучаемого гиперболического сигмоидного элемента}

Element S_NotTrain( Char : Real)     {Не обучаемый гиперболический сигмоидный элемент}
                                             {Char – характеристика}
   InSignals 1                               {Один входной сигнал}
   OutSignals 1                              {Один выходной сигнал}

   Forw                                      {Начало описания прямого функционирования}
   Begin
            {Выходной сигнал равен отношению входного сигнала к сумме характеристики и
            абсолютной величины входного сигнала}
      OutSignals[1] = InSignals[1] / (Char + Abs(InSignals[1])
   End                                       {Конец описания прямого функционирования}

   Back                                      {Начало описания обратного функционирования}
   Begin
            {Поправка к входному сигналу равна отношению произведения поправки выходного
            сигнала на характеристику к квадрату суммы характеристики и абсолютной
```

```
            величины входного сигнала}
        Back.InSignals[1]  = Back.OutSignals[1] * Char /  Sqr(Char + Abs(InSignals[1]);
    End                                         {Конец описания обратного функционирования}
End S_NotTrain          {Конец описания не обучаемого гиперболического сигмоидного элемента}

Element Pade(Char : Real)                       {Паде преобразователь Char  – характеристика}
    InSignals 2                                 {Два входных сигнала}
    OutSignals 1                                {Один выходной сигнал}

    Forw                                        {Начало описания прямого функционирования}
    Begin
            {Выходной сигнал равен отношению первого входного сигнала к сумме
            характеристики и второго входного сигнала}
        OutSignals[1] = InSignals[1] / (Char+ InSignals[2])
    End                                         {Конец описания прямого функционирования}

    Back                                        {Начало описания обратного функционирования}
        Var Real R;                             {R – действительное}
    Begin
            {Вспомогательная величина равна поправке к первому входному сигналу –
            отношению поправки выходного сигнала к сумме характеристики и второго
            входного сигнала}
        R = Back.OutSignals[1] / (Char + InSignals[2]);
        Back.InSignals[1] = R;
            {Поправка ко второму входному сигналу равна минус отношению произведения
            первого входного сигнала на поправку выходного сигнала к квадрату суммы
            характеристики и второго входного сигнала}
        Back.InSignals[2] =  -R * OutSignals[1];
    End                                         {Конец описания обратного функционирования}
End Pade                                        {Конец описания Паде преобразователя}

Element Sign_Mirror                             {Зеркальный пороговый элемент}
    InSignals 1                                 {Один входной сигнал}
    OutSignals 1                                {Один выходной сигнал}

    Forw                                        {Начало описания прямого функционирования }
    Begin
        If InSignals[1] > 0 Then OutSignals[1] = 1  {Выходной сигнал равен 1, если входной сигнал}
                            Else OutSignals[1] = 0  {больше нуля, и нулю в противном случае}
    End                                         {Конец описания прямого функционирования}

    Back                                        {Начало описания обратного функционирования}
    Begin
        Back.InSignals[1] = OutSignals[1];  {Поправка к входному сигналу равна выходному сигналу}
    End                                         {Конец описания обратного функционирования}
End Sign_Mirror                                 {Конец описания зеркального порогового элемента}

Element Sign_ Easy                              {Прозрачный  пороговый элемент}
    InSignals 1                                 {Один входной сигнал}
    OutSignals 1                                {Один выходной сигнал}

    Forw                                        {Начало описания прямого функционирования }
    Begin
        If InSignals[1] > 0 Then OutSignals[1] = 1  {Выходной сигнал равен 1, если входной сигнал}
                            Else OutSignals[1] = 0  {больше нуля, и нулю в противном случае}
    End                                         {Конец описания прямого функционирования}

    Back                                        {Начало описания обратного функционирования}
    Begin
            {Поправка к входному сигналу равна поправке к выходному сигналу}
        Back.InSignals[1] = Back.OutSignals[1];
    End                                         {Конец описания обратного функционирования}
End Sign_Easy                                   {Конец описания прозрачного порогового элемента}
```

```
Element Adaptiv_Sum( N : Long)                {Адаптивный сумматор на N входов}
    InSignals N                                   {N входных сигналов}
    OutSignals 1                                  {Один выходной сигнал}
    Parameters N                                  {N параметров – весов связей}

    Forw                                          {Начало описания прямого функционирования}
        Var                                       {Описание локальных переменных}
            Long I;                               {I – длинное целое – индекс}
            Real R;                               {R – действительное – для накопления суммы}
    Begin
        R = 0;                                    {Выходной сигнал равен скалярному }
        For I=1 To N Do                           {произведению массива входных сигналов}
            R = R + InSignals[I] * Parameters[I];  {на массив параметров}
        OutSignals[1] = R
    End                                           {Конец описания обратного функционирования}

    Back                                          {Начало описания обратного функционирования}
        Var Long I;                               {I – локальная переменная типа}
    Begin                                         {длинное целое – индекс}
        For I=1 To N Do Begin
                {Поправка к I-у входному сигналу равна сумме ранее вычисленной поправки и
                произведения поправки выходного сигнала на I-й параметр}
            Back.InSignals[I] = Back.OutSignals[1] * Parameters[I];
                {Поправка к I-у параметру равна произведению поправки выходного сигнала на I-й
                входной сигнал}
            Back. Parameters[I] = Back. Parameters[I] + Back.OutSignals[1] * InSignals[I]
        End
    End                                           {Конец описания обратного функционирования}
End Adaptiv_Sum                                   {Конец описания адаптивного сумматора}

Element Adaptiv_Sum_Plus ( N : Long)          {Адаптивный неоднородный сумматор на N входов}
    InSignals N                                   {N входных сигналов}
    OutSignals 1                                  {Один выходной сигнал}
    Parameters N+1                              {N+1 параметр – веса связей}

    Forw                                          {Начало описания прямого функционирования }
        Var                                       {Описание локальных переменных}
            Long I;                               {I – длинное целое – индекс}
            Real R;                               {R – действительное – для накопления суммы}
    Begin
        R = Parameters[N+1];                      {Выходной сигнал равен сумме N+1 параметра}
        For I=1 To N Do                           {и скалярного произведения массива входных}
            R = R + InSignals[I] * Parameters[I]; {сигналов на массив параметров}
        OutSignals[1] = R
    End                                           {Конец описания прямого функционирования}

    Back                                          {Начало описания обратного функционирования }
        Var Long I;                               {I – локальная переменная типа}
    Begin                                         {длинное целое – индекс}
        For I=1 To N Do Begin
                {Поправка к I-у входному сигналу равна произведению поправки выходного сигнала
                на I-й параметр}
            Back.InSignals[I] = Back.OutSignals[1] * Parameters[I];
                {Поправка к I-у параметру равна сумме ранее вычисленной поправки и произведения
                поправки выходного сигнала на I-й входной
                сигнал}
            Back. Parameters[I] = Back. Parameters[I] + Back.OutSignals[1] * InSignals[I]
        End;
                {Поправка к (N+1)-у параметру равна сумме ранее вычисленной поправки и попраки
                к выходному сигналу}
        Back.Parameters[N+1] = Back.Parameters[N+1] + Back.OutSignals[1]
```

```
    End                                     {Конец описания обратного функционирования}
End Adaptiv_Sum_Plus                        {Конец описания неоднородного адаптивного сумматора}

Element Square_Sum( N : Long)               {Квадратичный сумматор на N входов}
    InSignals N                             {N входных сигналов}
    OutSignals 1                            {Один выходной сигнал}
    Parameters (Sqr(N) + N) Div 2           {N(N+1)/2 параметров – весов связей}

    Forw                                    {Начало описания прямого функционирования}
        Var                                 {Описание локальных переменных}
            Long I,J,K;                     {I,J,K – переменные типа длинное целое }
            Real R;                         {R – действительное – для накопления суммы}
    Begin
        K = 1;                              {K – номер обрабатываемого параметра}
        R = 0;
        For I = 1 To N Do                   {I,J – номера входных сигналов}
            For J = I To N Do Begin
                R = R + InSignals[I] * InSignals[J] * Parameters[K];
                K = K + 1
            End;
                {Выходной сигнал равен сумме всех попарных произведений входных сигналов,
                 умноженных на соответствующие параметры}
        OutSignals[1] = R
    End                                     {Конец описания прямого функционирования}

    Back                                    {Начало описания обратного функционирования }
        Var                                 {Описание локальных переменных}
            Long I, J, K;                   {I,J,K – переменные типа длинное целое }
            Real R;                         {R – действительное}
            Vector W;                       {Массив для накопления промежуточных величин}
    Begin
        For I = 1 To N Do
            W[I] = 0;
        K = 1;                              {K – номер обрабатываемого параметра}
        For I = 1 To N Do
            For J = I To N Do Begin
                {Поправка к параметру равна сумме ранее вычисленной поправки и произведения
                поправки к входному сигналу на произведение сигналов, прошедших через этот
                параметр при прямом функционировании}
                Back.Parameters[K] = Back.Parameters[K] +
                    Back.OutSignals[1] * InSignals[I] * InSignals[J];
                R = Back.OutSignals[1] * Parameters[K];
                W[I] = W[I] + R * InSignals[J];
                W[J] = W[J] + R * InSignals[I];
                K = K + 1
            End;
        For I = 1 To N Do
                {Поправка к входному сигналу равна произведению поправки к выходному сигналу
                на сумму всех параметров, через которые этот сигнал проходил при прямом
                функционировании, умноженных на другие входные сигналы, так же прошедшие
                через эти параметры при прямом функционировании}
            Back.InSignals[1] = W[I]
    End                                     {Конец описания прямого функционирования}
End Square_Sum                              {Конец описания квадратичного сумматора}

Element Square_Sum_Plus( N : Long)          {Адаптивный квадратичный сумматор на N входов}
    InSignals N                             {N входных сигналов}
    OutSignals 1                            {Один выходной сигнал}
    Parameters (Sqr(N) + 3 * N) Div 2 + 1   {N(N+3)/2+1 параметров – весов связей}

    Forw                                    {Начало описания прямого функционирования}
```

```
    Var                                  {Описание локальных переменных}
      Long I, J, K;                      {I,J,K – переменные типа длинное целое }
      Real R;                            {R – действительное – для накопления суммы}
  Begin
    K = 2 * N+1;                         {K – номер обрабатываемого параметра}
    R = Parameters[Sqr(N) + 3 * N) Div 2 + 1];
    For I = 1 To N Do Begin
      R = R + InSignals[I] * Parameters[I] + Sqr(InSignals[I]) * Parameters[N + I];
      For J = I + 1 To N Do Begin
        R = R + InSignals[I] * InSignals[J] * Parameters[K];
        K = K + 1
      End
    End
        {Выходной сигнал равен сумме всех попарных произведений входных сигналов,
      умноженных на соответствующие параметры, плюс сумме всех входных сигналов
    умноженных на соответствующие параметры, плюс последний параметр}
    OutSignals[1] = R
  End                                    {Конец описания прямого функционирования}

  Back                                   {Начало описания обратного функционирования }
    Var                                  {Описание локальных переменных}
      Long I, J, K;                      {I,J,K – переменные типа длинное целое }
      Real R;                            {R – действительное – для накопления суммы}
      Vector W;                          {Массив для накопления промежуточных величин}
  Begin
    For I = 1 To N Do
      W[I] = 0;
    K = 2 * N + 1;                       {K – номер обрабатываемого параметра}
    For I = 1 To N Do Begin
      Back.Parameters[I] = Back.Parameters[I] + Back.OutSignals[1] * InSignals[I];
      Back.Parameters[N + I] = Back.Parameters[N + I] + Back.OutSignals[1] * Sqr(InSignals[I]);
      W[I] = W[I] + Back.OutSignals[1] * (Parameters[I] + 2 * Parameters[N + I] * InSignals[I])
      For J = I + 1 To N Do Begin
        Back.Parameters[K] = Back.Parameters[K] +
          Back.OutSignals[1] * InSignals[I] * InSignals[J];
        R = Back.OutSignals[1] * Parameters[K];
        W[I] = W[I] + R * InSignals[J];
        W[J] = W[J] + R * InSignals[I];
        K = K + 1
      End
    End;
    For I = 1 To N Do
      Back.InSignals[1] = W[I]
  End                                    {Конец описания обратного функционирования}
End Square_Sum_Plus                      {Конец описания адаптивного квадратичного сумматора}
End NetBibl
```

5.3.5.3.5 Описание блоков

Описание блока состоит из пяти основных разделов: заголовка описания блока, описания сигналов и параметров, описания состава, описания связей и конца описания блока. Существует два типа блоков – каскад и слой (Layer). Различие между этими двумя типами блоков состоит в том, что подсети, входящие в состав слоя, функционируют параллельно и независимо друг от друга, тогда как составляющие каскад подсети функционируют последовательно, причем каждая следующая подсеть использует результаты работы предыдущих подсетей. В свою очередь существует три вида каскадов – простой каскад (Cascad), цикл с фиксированным числом шагов (Loop) цикл по условию (Until). Различие между тремя видами каскадов очевидно – простой каскад функционирует один раз, цикл Loop функционирует указанное в описании число раз, а цикл Until функционирует до тех пор, пока не выполнится указанное в описании условие. В условии, указываемом в заголовке цикла Until, возможно использование сравнений массивов или интервалов массивов сигналов. Например, запись

InSignals=OutSignals

эквивалентна следующей записи

InSignals[1..N]=OutSignals[1..N]

которая эквивалентна вычислению следующей логической функции:

```
Function Equal(InSignals, OutSignals : RealArray) : Logic;
  Var Long I;
    Logic L
  Begin
    L = True
    For I = 1 To N Do
      L = L And (InSignals[I] = OutSignals[I]);
    Equal = L
  End
```

Раздел описания состава следует сразу после заголовка блока за разделом описания сигналов и параметров и начинается с ключевого слова Contents, за которым следуют имена подсетей (блоков или элементов) со списками фактических аргументов, разделенные запятыми. Все имена подсетей должны предваряться псевдонимами. В дальнейшем указание псевдонима полностью эквивалентно указанию имени подсети со списком фактических аргументов или без, в зависимости от контекста. Признаком конца раздела описания состава подсети служит имя подсети за списком фактических аргументов которого не следует запятая.

Раздел описания сигналов и параметров следует за разделом описания состава и состоит из указания числа входных и выходных сигналов и числа параметров блока. В константных выражениях, указывающих число входных и выходных сигналов и параметров можно использовать дополнительно функцию NumberOf с двумя параметрами. Первым параметром является одно из ключевых слов InSignals, OutSignals, Parameters, а вторым – имя подсети со списком фактических аргументов. Функция NumberOf возвращает число входных или выходных сигналов или параметров (в зависимости от первого аргумента) в подсети, указанной во втором аргументе. Использование этой функции необходимо в случае использования блоком аргументов-подсетей. Концом раздела описания сигналов и параметров служит одно из ключевых слов ParamDef, Static или Connections.

Раздел определения типов параметров является необязательным разделом в описании блока и начинается с ключевого слова ParamDef. В каждой строке этого раздела можно задать минимальную и максимальную границы изменения одного типа параметров. Если в описании сети встречаются параметры неопределенного типа, то этот тип считается совпадающим с типом DefaultType. Описание типа *не обязано* предшествовать описанию параметров этого типа. Так, например, определение типа параметров может находиться в описании главной сети. Концом этого раздела служит одно из ключевых слов Connections.

Раздел описания связей следует за разделом описания сигналов и параметров и начинается с ключевого слова Connections. В разделе «Описание распределения сигналов» главы «Общий стандарт» детально описано распределение связей.

Раздел конца описания блока состоит из ключевого слова End, за которым следует имя блока.

5.3.5.3.6 Пример описания блоков

При описании блоков используются элементы, описанные в библиотеке Elements, приведенной в разд. «Пример описания элементов».

```
NetBibl SubNets Used Elements;          {Библиотека подсетей, использующая библиотеку Elements}

        {Сигмоидный нейрон с произвольным сумматором на N входов}
Cascad NSigm(aSum : Block; N : Long; Char : Real)
        {В состав каскада входит произвольный сумматор на N входов и сигмоидный нейрон
      с необучаемой характеристикой}
  Contents aSum(N), S_NotTrain(Char)
  InSignals NumberOf(InSignals, aSum(N))      {Число входных сигналов определяет сумматор}
  OutSignals 1                                {Один выходной сигнал}
  Parameters NumberOf(Parameters, aSum(N))    {Число параметров определяет сумматор}
  Connections
        {Входные сигналы нейрона – входные сигналы сумматора}
    InSignals[1.. NumberOf(InSignals, aSum(N))] <=>
      aSum.InSignals[1.. NumberOf(InSignals, aSum(N))]
    aSum.OutSignals <=> S_NotTrain.InSignals  {Выход сумматора – вход преобразователя}
```

```
        OutSignals <=> S_NotTrain.OutSignals
                {Параметры нейрона – параметры сумматора }
        Parameters[1.. NumberOf(Parameters, aSum(N))] <=>
            aSum.Parameters[1.. NumberOf(Parameters, aSum(N))]
End                             {Конец описания сигмоидного нейрона с произвольным сумматором}

                {Слой сигмоидных нейронов с произвольными сумматорами на N входов}
Layer Lay1(aSum : Block; N,M : Long; Char : Real)
    Contents Sigm: NSigm(aSum,N,Char)[M]          {В состав слоя входит M нейронов}
    InSignals M * NumberOf(InSignals, Sigm)
                {Число входных сигналов определяется как взятое M раз число
                входных сигналов нейронов. Вместо имени нейрона используем псевдоним}
    OutSignals M                                  {Один выходной сигнал на нейрон}
    Parameters M * NumberOf(Parameters, Sigm)
                {Число параметров определяется как взятое M раз число параметров нейронов}
    Connections
                {Первые NumberOf(InSignals, NSigm(aSum,N,Char)) сигналов первому нейрону, и т.д.}
        InSignals[1..M * NumberOf(InSignals, Sigm)] <=>
            Sigm[1..M].InSignals[1.. NumberOf(InSignals, Sigm)]
                {Выходные сигналы нейронов - выходные сигналы сети}
        OutSignals[1..M] <=> Sigm[1..M].OutSignals
                {Параметры слоя – параметры нейронов}
        Parameters[1..M * NumberOf(Parameters, Sigm)] <=>
            Sigm[1..M].Parameters[1.. NumberOf(Parameters, Sigm)]
End                     {Конец описания слоя сигмоидных нейронов с произвольным сумматором}

                {Слой точек ветвления}
Layer BLay( N,M : Long)
    Contents Branch(N)[M]               {В состав слоя входит M точек ветвления}
    InSignals M                         {По одному входному сигналу на точку ветвления}
    OutSignals M * N                    {N выходных сигналов у каждой точки ветвления}
    Connections
        InSignals[1..M] <=> Branch[1..M].InSignals     {По одному входу на точку ветвления}
                {Выходные сигналы в порядке первый с каждой точки ветвления, затем второй и т.д. }
        OutSignals[1..N * M] <=> Branch[+:1..M].OutSignals[1..N]
End                                     {Конец описания слоя Точек ветвления}

                {Полный слой сигмоидных нейронов с произвольными сумматорами на N входов}
Cascad FullLay(aSum : Block; N,M : Long; Char : Real)
    Contents Br: BLay1(M,N), Ne: Lay1(aSum,N,M,Char)  {Слой точек ветвления и слой нейронов}
    InSignals N                         {Число входных сигналов – число точек ветвления}
    OutSignals M                        {Один выходной сигнал на нейрон}
    Parameters NumberOf(Parameters, Ne)
                {Число параметров определяется как взятое M раз число параметров нейронов}
    Connections
        InSignals[1..N]<=> Br.InSignals[1..N]      {Входные сигналы – слою точек ветвления}
                {Выходные сигналы нейронов - выходные сигналы сети}
        OutSignals[1..M] <=> Ne.OutSignals[1..M]
                {Параметры слоя – параметры нейронов}
        Parameters[1..NumberOf(Parameters, Ne)] <=>
            Ne.Parameters[1.. NumberOf(Parameters, Ne)]
                {Выход слоя точек ветвления – вход слоя нейронов}
        Br.OutSignals[1..N * M] <=> Ne.InSignals[1..N * M]
End                     {Конец описания слоя сигмоидных нейронов с произвольным сумматором}

                {Сеть с сигмоидными нейронами и произвольными сумматорами, содержащая
                    Input – число нейронов на входном слое;
                    Output – число нейронов на выходном слое (число выходных сигналов);
                    Hidden – число нейронов на H>0 скрытых слоях;
                    N – число входных сигналов
                все входные сигналы подаются на все нейроны входного слоя}
```

```
Cascad  Net1(aSum : Block; Char : Real; Input, Output, Hidden, H, N : Long)
                {Под тремя разными псевдонимами используется одна и та же подсеть с разными
                параметрами}
    Contents
        In: FullLay(aSum,N,Input,Char),
        Hid1: FullLay(aSum,Input,Hidden,Char)
        Hid2: FullLay(aSum,Hidden,Hidden,Char)[H-1]              {Пусто при H=1}
        Out: FullLay(aSum,Hidden,Output,Char)
    InSignals N                                  {Число входных сигналов – N}
    OutSignals Output                              {Один выходной сигнал на нейрон}
                {Число параметров определяется как сумма чисел параметров всех подсетей}
    Parameters NumberOf(Parameters, In)+ NumberOf(Parameters, Hid1)+
        (H-1) * NumberOf(Parameters, Hid2)+ NumberOf(Parameters, Out)
    Connections
        InSignals[1..N]<=> In.InSignals[1..N]          {Входные сигналы – входному слою}
                {Выходные сигналы нейронов - с выходного слоя сети}
        OutSignals[1..Output] <=> Out.OutSignals[1.. Output]
                {Параметры сети последовательно всем подсетям}
        Parameters[1..NumberOf(Parameters,In)] <=> In.Parameters[1.. NumberOf(Parameters, In)]
        Parameters[NumberOf(Parameters,In)+1..NumberOf(Parameters,In)+
            NumberOf(Parameters, Hid1)] <=> Hid1.Parameters[1.. NumberOf(Parameters, Hid1)]
        Parameters[NumberOf(Parameters,In)+ NumberOf(Parameters, Hid1)]+1
            ..NumberOf(Parameters,In)+NumberOf(Parameters, Hid1)+
            (H-1) * NumberOf(Parameters, Hid2)] <=>
            Hid2[1..H-1].Parameters[1.. NumberOf(Parameters, Hid2)]
        Parameters[NumberOf(Parameters,In)+ NumberOf(Parameters, Hid1)]+
            (H-1) * NumberOf(Parameters, Hid2)+1..NumberOf(Parameters,In)+
            NumberOf(Parameters,Hid1)+(H-1)*NumberOf(Parameters,Hid2)+
            NumberOf(Parameters, Out)] <=> Out.Parameters[1.. NumberOf(Parameters, Out)]
                {Передача сигналов от слоя к слою}
        In.OutSignals[1..Input] <=> Hid1.InSignals[1..Input]     {От входного к первому скрытому слою}
        Hid1.OutSignals[1..Hidden] <=> Hid2[1].InSignals[1..Hidden]     {От первого скрытого слоя}
    {между скрытыми слоями. При H=1 эта запись пуста}
        Hid2[1..H-2].OutSignals[1.. Hidden] <=> Hid2[2..H-1].InSignals[1.. Hidden]
        Hid2[H-1].OutSignals[1.. Hidden] <=> Out.InSignals[1.. Hidden]{От скрытых – к выходному}
End

                {Полносвязная сеть с M сигмоидными нейронами на K тактов функционирования с
                невыделенным входным слоем на M сигналов}
Loop Circle(aSum : Block; Char : Real; M, K : Long) K
    Contents
        Net: FullLay(aSum,M,M,Char)
    InSignals M                                  {Число входных сигналов – N}
    OutSignals M                                 {Один выходной сигнал на нейрон}
    Parameters NumberOf(Parameters, Net)    {Число параметров определяется слоем FullLay}
    Connections
        InSignals[1..M]<=> Net.InSignals[1..M]          {Входные сигналы – на вход слоя}
        OutSignals[1..M] <=> Net.OutSignals[1.. M]          {Выходные сигналы – на выходе слоя}
                {Все параметры слою}
        Parameters[1..NumberOf(Parameters,Net)] <=> Net.Parameters[1.. NumberOf(Parameters,Net)]
        Net.OutSignals[1..M] <=> Net.InSignals[1..M]      {Замыкаем выход на вход}
End                     {Конец описания слоя сигмоидных нейронов с произвольным сумматором}

                {Полносвязная сеть с M сигмоидными нейронами на K тактов функционирования с
                выделенным входным слоем на N сигналов. Все входные сигналы подаются на вход
                каждого нейрона входного слоя. Все параметры ограничены по абсолютному
                значению единицей}
Cascad  Net2: (aSum : Block; Char : Real; M, K, N : Long)
    Contents
        In: FullLay(aSum,N,M,Char),                  {Входной слой}
```

```
        Net: Circle(aSum,Char,M,K)                    {Полносвязная сеть}
    InSignals N                                       {Число входных сигналов – N}
    OutSignals M                                      {Один выходной сигнал на нейрон}
            {Число параметров определяется как сумма чисел параметров всех подсетей}
    Parameters NumberOf(Parameters, In)+ NumberOf(Parameters, Net)
    ParamDef DefaultType -1 1
    Connections
        InSignals[1..N]<=> In.InSignals[1..N]         {Входные сигналы – входному слою}
            {Выходные сигналы нейронов - с выходного слоя сети}
        OutSignals[1..M] <=> Net.OutSignals[1.. M]
            {Параметры сети последовательно всем подсетям}
        Parameters[1..NumberOf(Parameters, In)] <=> In.Parameters[1.. NumberOf(Parameters, In)]
        Parameters[NumberOf(Parameters,In)+1..
            NumberOf(Parameters,In)+NumberOf(Parameters, Net)]
            <=> Net.Parameters[1.. NumberOf(Parameters, Net)]
            {Передача сигналов от слоя к слою}
        In.OutSignals[1..M] <=> Net.InSignals[1..M]       {От входного к циклу}
        Net.OutSignals[1..M] <=> Net.InSignals[1..M]      {От первого скрытого слоя}
End

            {Нейрон сети Хопфилда из N нейронов}
Cascad Hopf(N : Long)
    Contents Sum(N),Sign_Easy                         {Сумматор и пороговый элемент}
    InSignals N                                       {Число входных сигналов – N}
    OutSignals 1                                      {Число выходных сигналов – 1}
    Parameters NumberOf(Parameters,Sum(N))            {Число параметров – N}
    Connections
        InSignals[1..N]<=> Sum.InSignals[1..N]        {Входные сигналы – сумматору}
            {Выходной сигнал нейрона – выходной сигнал порогового элемета}
        OutSignals <=> Sign_Easy.OutSignals
            {Параметры нейрона – параметры сумматора}
        Parameters[1..NumberOf(Parameters, Sum(N))] <=>
            Sum.Parameters[1.. NumberOf(Parameters, Sum(N))]
        Sum.OutSignals <=> Sign_Easy.InSignals        {Выход сумматора на вход порога}
End

            {Слой нейронов Хопфилда}
Layer HLay(N : Long)
    Contents Hop: Hopf(N)[N]                  {В состав слоя входит N нейронов}
    InSignals N * N                               {N нейронов по N входных сигналов}
    OutSignals N                                  {Один выходной сигнал на нейрон}
    Parameters N * NumberOf(Parameters, Hop)
    Connections
            {Первые NumberOf(InSignals, Hop) сигналов первому нейрону, и т.д.}
        InSignals[1..Sqr(N)] <=> Hop[1..N].InSignals[1..N]
            {Выходные сигналы нейронов - выходные сигналы сети}
        OutSignals[1..N] <=> Hop[1..N].OutSignals
            {Параметры слоя – параметры нейронов}
        Parameters[1..N * NumberOf(Parameters, Hop)] <=>
            Hop[1..N].Parameters[1.. NumberOf(Parameters, Hop)]
End

            {Сеть Хопфилда из N нейронов}
Until Hopfield(N : Long) InSignals=OutSignals
    Contents BLay(N,N),HLay(N)                {Слой точек ветвления и слой нейронов}
    InSignals N                                   {Число входных сигналов – N}
    OutSignals N                                  {Число выходных сигналов – N}
    Parameters N * NumberOf(Parameters,HLay(N))   {Число параметров – N*N}
    Connections
        InSignals[1..N]<=> BLay.InSignals[1..N]   {Входные сигналы – точкам ветвления}
```

```
        {Выходные сигналы нейронов – выходные сигналы сети}
    OutSignals[1..N] <=> HLay.OutSignals[1..N]
    Parameters[1..N*NumberOf(Parameters, HLay(N))] <=>
        HLay.Parameters[1..N*NumberOf(Parameters, HLay(N))]
        {Выход точек ветвления на вход нейронов}
    BLay.OutSignals[1..Sqr(N)] <=> HLay.InSignals[1..Sqr(N)]
    HLay.OutSignals[1..N] <=> BLay.InSignals[1..N]      {Замыкаем конец на начало}
End

End NetLib

NetWork Hop Used SubNets;          {Сеть Хопфилда на пять нейронов}
MainNet Hopfield(5)
Parameters 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0;
ParamMask -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1;
End NetWork
```

5.3.5.4 Сокращение описания сети

Предложенный в предыдущих разделах язык описания многословен. В большинстве случаев за счет хорошей структуризации сети можно опустить все разделы описания блока кроме раздела состава. В данном разделе описывается генерация по умолчанию разделов описания сигналов и параметров, и описания связей. Использование механизмов умолчания позволяет сильно сократить текст описания сети.

5.3.5.4.1 Раздел описания сигналов и параметров

Для всех видов блоков число параметров определяется как сумма чисел параметров всех подсетей, перечисленных в разделе описания состава. Это может приводить к лишним записям, но не повлияет на работу сети. Примером лишней записи может служить генерируемая запись:

Parameters M * **NumberOf(Parameters**,Branch(N))

в описании слоя точек ветвления, поскольку точки ветвления не имеют параметров.

Число входных сигналов блока определяется по следующим правилам:

- для слоя число входных сигналов равно сумме числа входных сигналов всех подсетей, перечисленных в разделе описания состава;
- для каскадов всех видов число входных сигналов блока равно числу входных сигналов подсети, стоящей первой в списке подсетей в разделе описания состава

Число выходных сигналов блока определяется по следующим правилам:

- для слоя число выходных сигналов равно сумме числа выходных сигналов всех подсетей, перечисленных в разделе описания состава;
- для каскадов всех видов число выходных сигналов блока равно числу выходных сигналов подсети, стоящей последней в списке подсетей в разделе описания состава;

Описания всех сетей, приведенные в предыдущем разделе полностью соответствуют правилам генерации. В качестве более общего примера приведем раздел описания сигналов и параметров двух условных блоков.

```
Layer  A
  Contents Net1, Net2[K], Net3
  InSignals NumberOf(InSignals,Net1)+K*NumberOf(InSignals,Net2)
     +NumberOf(InSignals,Net3)
  OutSignals NumberOf(OutSignals,Net1)+K*NumberOf(OutSignals,Net2)
     +NumberOf(OutSignals,Net3)
  Parameters NumberOf(Parameters,Net1)+K*NumberOf(Parameters,Net2)
     +NumberOf(Parameters,Net3)

Cascad  B
  Contents Net1, Net2[K], Net3
  InSignals NumberOf(InSignals,Net1)
  OutSignals NumberOf(OutSignals,Net3)
  Parameters NumberOf(Parameters,Net1)+K*NumberOf(Parameters,Net2)
     +NumberOf(Parameters,Net3)
```

5.3.5.4.2 Раздел описания связей

Раздел описания связей может быть разбит на пять подразделов.

1. Установление связи входных сигналов блока с входными сигналами подсетей.
2. Установление связи выходных сигналов блока с выходными сигналами подсетей.
3. Установление связи параметров блока с параметрами подсетей.
4. Установление связи между выходными сигналами одних подсетей и входными сигналами других подсетей.
5. Замыкание выхода блока на вход блока.

Для слоя раздел описания связей строится по следующим правилам.

1. Все подсети получают входные сигналы в порядке перечисления подсетей в разделе описания состава – первая часть массива входных сигналов слоя отдается первой подсети, следующая – второй и т.д. Если какая-либо подсеть в разделе описания состава указана с некоторым не равным единице числом экземпляров, то считается, что экземпляры этой подсети перечислены в списке в порядке возрастания номера.
2. Выходные сигналы подсетей образуют массив выходных сигналов слоя также в порядке перечисления подсетей в разделе описания состава – первая часть массива выходных сигналов слоя состоит из выходных сигналов первой подсети, следующая – второй и т.д. Если какая-либо подсеть в разделе описания состава указана с некоторым не равным единице числом экземпляров, то считается, что экземпляры этой подсети перечислены в списке в порядке возрастания номера.
3. Подразделы установления связи между выходными сигналами одних подсетей и входными сигналами других подсетей и замыкания выхода блока на вход для слоя отсутствуют.

Для каскадов раздел описания связей строится по следующим правилам:

1. Входные сигналы блока связываются с входными сигналами первой подсети в списке подсетей в разделе описания состава. Если для первой подсети указано не единичное число экземпляров, то все входные сигналы связываются с входными сигналами первого экземпляра подсети.
2. Выходные сигналы блока связываются с выходными сигналами последней подсети в списке подсетей в разделе описания состава. Если для последней подсети указано не единичное число экземпляров, то все выходные сигналы связываются с выходными сигналами последнего (с максимальным номером) экземпляра подсети.
3. Массив параметров блока образуется из массивов параметров подсетей в порядке перечисления подсетей в разделе описания состава – первая часть массива параметров блока состоит из параметров первой подсети, следующая – второй и т.д. Если какая-либо подсеть в разделе описания состава указана с некоторым не равным единице числом экземпляров, то считается, что экземпляры этой подсети перечислены в списке в порядке возрастания номера.
4. Выходные сигналы каждой подсети, кроме последней связываются с входными сигналами следующей подсети в списке подсетей в разделе описания состава. Если какая-либо подсеть в разделе описания состава указана с некоторым не равным единице числом экземпляров, то считается, что экземпляры этой подсети перечислены в списке в порядке возрастания номера.
5. Для блоков типа Cascad замыкание выхода блока на вход блока отсутствует. Для блоков типов Loop и Until замыкание выхода блока на вход блока достигается путем установления связей между выходными сигналами последней подсети в списке подсетей в разделе описания состава с входными сигналами первой подсети в списке подсетей в разделе описания состава. Если какая-либо подсеть в разделе описания состава указана с некоторым не равным единице числом экземпляров, то считается, что экземпляры этой подсети перечислены в списке в порядке возрастания номера.

Описания всех сетей, приведенные в предыдущем разделе полностью соответствуют правилам генерации. В качестве более общего примера приведем раздел описания сигналов и параметров трех условных блоков.

```
Layer A
  Contents Net1, Net2[K], Net3
  InSignals[1..NumberOf(InSignals,Net1)+K*NumberOf(InSignals,Net2)
      +NumberOf(InSignals,Net3)] <=> Net1. InSignals[1..NumberOf(InSignals,Net1)],
      Net2[1..K].InSignals[1..NumberOf(InSignals,Net2)],
      Net3.InSignals[1..NumberOf(InSignals,Net3)]
  OutSignals[1..NumberOf(OutSignals,Net1)+K*NumberOf(OutSignals,Net2)
      +NumberOf(OutSignals,Net3)] <=> Net1. OutSignals[1..NumberOf(OutSignals,Net1)],
      Net2[1..K].OutSignals[1..NumberOf(OutSignals,Net2)],
      Net3.OutSignals[1..NumberOf(OutSignals,Net3)]
```

```
  Parameters[1..NumberOf(Parameters,Net1)+K*NumberOf(Parameters,Net2)
      +NumberOf(Parameters,Net3)] <=> Net1. Parameters[1..NumberOf(Parameters,Net1)],
      Net2[1..K].Parameters[1..NumberOf(Parameters,Net2)],
      Net3.Parameters[1..NumberOf(Parameters,Net3)]

Cascad B
  Contents Net1, Net2[K], Net3
  InSignals[1..NumberOf(InSignals,Net1)] <=> Net1. InSignals[1..NumberOf(InSignals,Net1)]
  OutSignals[1..NumberOf(OutSignals,Net3)] <=>
      Net3.OutSignals[1..NumberOf(OutSignals,Net3)]
  Parameters[1..NumberOf(Parameters,Net1)+K*NumberOf(Parameters,Net2)
      +NumberOf(Parameters,Net3)] <=> Net1. Parameters[1..NumberOf(Parameters,Net1)],
      Net2[1..K].Parameters[1..NumberOf(Parameters,Net2)],
      Net[3].Parameters[1..NumberOf(Parameters,Net3)]
  Net1. OutSignals[1..NumberOf(OutSignals,Net1)],
      Net2[1..K].OutSignals[1..NumberOf(OutSignals,Net2)] <=>
      Net2[1..K].InSignals[1..NumberOf(InSignals,Net2)],
      Net3.InSignals[1..NumberOf(InSignals,Net3)]

Loop  C N
  Contents Net1, Net2[K], Net3
  InSignals[1..NumberOf(InSignals,Net1)] <=> Net1. InSignals[1..NumberOf(InSignals,Net1)]
  OutSignals[1..NumberOf(OutSignals,Net3)] <=>
      Net3.OutSignals[1..NumberOf(OutSignals,Net3)]
  Parameters[1..NumberOf(Parameters,Net1)+K*NumberOf(Parameters,Net2)
      +NumberOf(Parameters,Net3)] <=> Net1. Parameters[1..NumberOf(Parameters,Net1)],
      Net2[1..K].Parameters[1..NumberOf(Parameters,Net2)],
      Net[3].Parameters[1..NumberOf(Parameters,Net3)]
  Net1. OutSignals[1..NumberOf(OutSignals,Net1)],
      Net2[1..K].OutSignals[1..NumberOf(OutSignals,Net2)] <=>
      Net2[1..K].InSignals[1..NumberOf(InSignals,Net2)],
      Net3.InSignals[1..NumberOf(InSignals,Net3)]
  Net3.OutSignals[1..NumberOf(OutSignals,Net3)] <=>
      Net1. InSignals[1..NumberOf(InSignals,Net1)]
```

5.3.5.4.3 Частично сокращенное описание

Если описываемый блок должен иметь связи, устанавливаемые не так, как описано в разд. «Раздел описания связей», то соответствующий раздел описания блока может быть описан явно полностью или частично. Если какой либо раздел описан частично, то действует следующее правило: те сигналы, параметры и их связи, которые описаны явно, берутся из явного описания, а те сигналы, параметры и их связи, которые не фигурируют в явном описании берутся из описания по умолчанию. Так, в приведенном в разд. «Пример описания блоков» описании слоя точек ветвления BLay невозможно использование генерируемого по умолчанию подраздела установления связи выходных сигналов блока с входными сигналами подсетей. Возможно следующее сокращенное описание.

```
                {Слой точек ветвления}
Layer BLay( N,M : Long)
   Contents Branch(N)[M]                {В состав слоя входит M точек ветвления}
   Connections
                {Выходные сигналы в порядке первый с каждой точки ветвления, затем второй и т.д. }
      OutSignals[1..N * M] <=> Branch[+:1..M].OutSignals[1..N]
End                                     {Конец описания слоя Точек ветвления}
```

5.3.5.4.4 Пример сокращенного описания блоков

При описании блоков используются элементы, описанные в библиотеке Elements, приведенной в разд. "Пример описания элементов".

```
NetBibl SubNets Used Elements;          {Библиотека подсетей, использующая библиотеку Elements}

                {Сигмоидный нейрон с произвольным сумматором на N входов}
Cascad NSigm(aSum : Block; N : Long; Char : Real)
```

```
            {В состав каскада входит произвольный сумматор на N входов и сигмоидный нейрон
        с необучаемой характеристикой}
    Contents aSum(N), S_NotTrain(Char)
End

            {Слой сигмоидных нейронов с произвольными сумматорами на N входов}
Layer Lay1(aSum : Block; N,M : Long; Char : Real)
    Contents Sigm: NSigm(aSum,N,Char)[M]                    {В состав слоя входит M нейронов}
End

            {Слой точек ветвления}
Layer BLay( N,M : Long)
    Contents Branch(N)[M]                   {В состав слоя входит M точек ветвления}
    Connections
            {Выходные сигналы в порядке первый с каждой точки ветвления, затем второй и т.д. }
        OutSignals[1..N * M] <=> Branch[+:1..M].OutSignals[1..N]
End

            {Полный слой сигмоидных нейронов с произвольными сумматорами на N входов}
Cascad FullLay(aSum : Block; N,M : Long; Char : Real)
    Contents BLay1(M,N), Lay1(aSum,N,M,Char)   {Слой точек ветвления и слой нейронов}
End                     {Конец описания слоя сигмоидных нейронов с произвольным сумматором}

            {Сеть с сигмоидными нейронами и произвольными сумматорами, содержащая
                Input – число нейронов на входном слое;
                Output – число нейронов на выходном слое (число выходных сигналов);
                Hidden – число нейронов на H>0 скрытых слоях;
                N – число входных сигналов
            все входные сигналы подаются на все нейроны входного слоя}
Cascad  Net1(aSum : Block; Char : Real; Input, Output, Hidden, H, N : Long)
            {Под тремя разными псевдонимами используется одна и таже подсеть с разными
            параметрами. Использование псевдонимов необходимо даже при сокращенном
            описании}
    Contents
        In: FullLay(aSum,N,Input,Char),
        Hid1: FullLay(aSum,Input,Hidden,Char)
        Hid2: FullLay(aSum,Hidden,Hidden,Char)[H-1]             {Пусто при H=1}
        Out: FullLay(aSum,Hidden,Output,Char)
End

            {Полносвязная сеть с M сигмоидными нейронами на K тактов функционирования с
            невыделенным входным слоем на M сигналов. Все параметры ограничены по
            абсолютному значению единицей}
Loop Circle(aSum : Block; Char : Real; M, K : Long) K
    Contents
        FullLay(aSum,M,M,Char)
    ParamDef DefaultType -1 1
End

            {Полносвязная сеть с M сигмоидными нейронами на K тактов функционирования с
            выделенным входным слоем на N сигналов. Все входные сигналы подаются на вход
            каждого нейрона входного слоя}
Cascad Net2: (aSum : Block; Char : Real; M, K, N : Long)
    Contents
        In: FullLay(aSum,N,M,Char),                 {Входной слой}
        Net: Circle(aSum,Char,M,K)                  {Полносвязная сеть}
End

Cascad Hopf(N : Long)                               {Нейрон сети Хопфилда из N нейронов}
    Contents Sum(N),Sign_Easy                       {Сумматор и пороговый элемент}
End

            {Слой нейронов Хопфилда}
```

```
Layer HLay(N : Long)
   Contents Hop: Hopf(N)[N]                    {В состав слоя входит N нейронов}
End

                {Сеть Хопфилда из N нейронов}
Until Hopfield(N : Long)  InSignals=OutSignals
   Contents BLay(N,N),HLay(N)                  {Слой точек ветвления и слой нейронов}
End

End NetLib
```

5.4 Стандарт второго уровня компонента сеть

В данном разделе главы рассмотрены все запросы, исполняемые компонентом сеть. Прежде чем приступать к описанию стандарта запросов компонента сеть следует выделить выполняемые им функции. Что должен делать компонент сеть? Очевидно, что прежде всего он должен уметь выполнять такие функции, как функционирование вперед (работа обученной сети) и назад (вычисление вектора поправок или градиента для обучения), модернизацию параметров (обучение сети) и входных сигналов (обучение примера). Кроме того компонент сеть должен уметь читать сеть с диска и записывать ее на диск. Необходимо так же предусмотреть возможность создавать сеть и редактировать ее структуру. Эти две функциональные возможности не связаны напрямую с работой (функционированием и обучением) сети. Таким образом, необходимо выделить сервисную компоненту – редактор сетей. Компонент редактор сетей позволяет создавать и изменять структуру сети, модернизировать обучаемые параметры в «ручном» режиме.

5.4.1 Запросы к компоненте сеть

Запросы к компоненту сеть можно разбить на пять групп:

1. Функционирование.
2. Изменение параметров.
3. Работа со структурой.
4. Инициация редактора и конструктора сетей.
5. Обработка ошибок.

Поскольку компонент сеть может работать одновременно с несколькими сетями, большинство запросов к сети содержат явное указание имени сети. Отметим, что при генерации запросов в качестве имени сети можно указывать имя любой подсети. Таким образом, иерархическая структура сети, описанная в стандарте языка описания сетей, позволяет работать с каждым блоком или элементом сети как с отдельной сетью. Ниже приведено описание всех запросов к компоненту сеть. Каждый запрос является логической функцией, возвращающей значение истина, если запрос выполнен успешно, и ложь – при

Таблица 3.

Значения предопределенных констант

НазваниеВеличинаЗначение		
InSignals	0	Входные сигналы прямого функционирования
OutSignals	1	Выходные сигналы прямого функционирования
Parameters	2	Параметры
InSignalMask	3	Маска обучаемости входных сигналов
ParamMask	4	Маска обучаемости параметров
BackInSignals	5	Входные сигналы обратного функционирования (поправки)
BackOutSignals	6	Выходные сигналы обратного функционирования (поправки)
BackParameters	7	Поправки к параметрам
Element	0	Тип подсети – элемент
Layer	1	Тип подсети – слой
Cascad	2	Тип подсети – простой каскад
CicleFor	3	Тип подсети – цикл с заданным числом проходов
CicleUntil	4	Тип подсети – цикл по условию

ошибочном завершении исполнения запроса.

При вызове ряда запросов используются предопределенные константы. Их значения приведены в табл. 3.

5.4.2 Запросы на функционирование

Два запроса первой группы позволяют проводить прямое и обратное функционирование сети. По сути эти запросы эквивалентны вызову методов Forw и Back сети или ее элемента.

5.4.2.1 Выполнить прямое Функционирование (Forw)

Описание запроса:
Pascal:

```
Function Forw ( Net : PString; InSignals : PRealArray ) : Logic;
```

C:

```
Logic Forw(PString Net, PRealArray InSignals)
```

Описание аргумента:
Net – указатель на строку символов, содержащую имя сети.
InSignals – массив входных сигналов сети.
Назначение – проводит прямое функционирование сети, указанной в параметре Net.
Описание исполнения.

1. Если Error <> 0, то выполнение запроса прекращается.
2. Если в качестве аргумента Net дан пустой указатель, или указатель на пустую строку, то исполняющим запрос объектом является первая сеть в списке сетей компонента сеть.
3. Если список сетей компонента сеть пуст или имя сети, переданное в аргументе Net в этом списке не найдено, то возникает ошибка 301 – неверное имя сети, управление передается обработчику ошибок, а обработка запроса прекращается.
4. Вызывается метод Forw сети, имя которой было указано в аргументе Net.
5. Если во время выполнения запроса возникает ошибка, то генерируется внутренняя ошибка 304 - ошибка прямого функционирования. Управление передается обработчику ошибок. Выполнение запроса прекращается. В противном случае выполнение запроса успешно завершается.

5.4.2.2 Выполнить обратное Функционирование (Back)

Описание запроса:
Pascal:

```
Function Back( Net : PString; BackOutSignals : PRealArray) : Logic;
```

C:

```
Logic Back(PString Net, PRealArray BackOutSignals)
```

Описание аргумента:
Net – указатель на строку символов, содержащую имя сети.
BackOutSignals – массив производных функции оценки по выходным сигналам сети.
Назначение – проводит обратное функционирование сети, указанной в параметре Net.
Описание исполнения.

1. Если Error <> 0, то выполнение запроса прекращается.
2. Если в качестве аргумента Net дан пустой указатель, или указатель на пустую строку, то исполняющим запрос объектом является первая сеть в списке сетей компонента сеть.
3. Если список сетей компонента сеть пуст или имя сети, переданное в аргументе Net в этом списке не найдено, то возникает ошибка 301 – неверное имя сети, управление передается обработчику ошибок, а обработка запроса прекращается.
4. Вызывается метод Back сети, имя которой было указано в аргументе Net.
5. Если во время выполнения запроса возникает ошибка, то генерируется внутренняя ошибка 305 - ошибка обратного функционирования. Управление передается обработчику ошибок. Выполнение запроса прекращается. В противном случае выполнение запроса успешно завершается.

5.4.3 Запросы на изменение параметров.

Ко второй группе запросов относятся четыре запроса: Modify – модификация параметров, обычно называемая обучением, ModifyMask – модификация маски обучаемых синапсов, NullGradient – обнуление градиента и RandomDirection – сгенерировать случайное направление спуска.

5.4.3.1 Провести обучение (Modify)

Описание запроса:
Pascal:

Function Modify(Net : PString; OldStep, NewStep : Real; Tipe : Integer; Grad : PRealArray) : Logic;

C:

Logic Modify(PString Net, Real OldStep, Real NewStep, Integer Tipe, PRealArray Grad)

Описание аргументов:

Net – указатель на строку символов, содержащую имя сети.

OldStep, NewStep – параметры обучения.

Tipe – одна из констант InSignals или Parameters.

Grad – адрес массива поправок или пустой указатель.

Назначение – проводит обучение параметров или входных сигналов сети, указанной в параметре Net.

Описание исполнения.

1. Если Error <> 0, то выполнение запроса прекращается.
2. Если в качестве аргумента Net дан пустой указатель, или указатель на пустую строку, то исполняющим запрос объектом является первая сеть в списке сетей компонента сеть.
3. Если список сетей компонента сеть пуст или имя сети, переданное в аргументе Net в этом списке не найдено, то возникает ошибка 301 – неверное имя сети, управление передается обработчику ошибок, а обработка запроса прекращается.
4. Если аргумент Grad содержит пустой указатель, то поправки берутся из массива Back.Parameters или Back.InputSignals в зависимости от значения аргумента Tipe.
5. В зависимости от значения аргумента Tipe для каждого параметра или входного сигнала P, при условии, что соответствующий ему элемент маски обучаемости, соответствующей аргументу Tipe равен -1 (значение истина) выполняется следующая процедура:

   P1=P*OldStep+DP*NewStep.

   Если для типа, которым описан параметр P, заданы минимальное и максимальное значения, то:

   P2=Pmin, при P1<Pmin

   P2=Pmax, при P1>Pmax

   P2=P1 в противном случае

5.4.3.2 Изменить маску обучаемости (ModifyMask)

Описание запроса:

Pascal:

Function ModifyMask(Net : PString; Tipe : Integer; NewMask: PLogicArray) : Logic;

C:

Logic Modify(PString Net, Integer Tipe, PLogicArray NewMask)

Описание аргументов:

Net – указатель на строку символов, содержащую имя сети.

Tipe – одна из констант InSignals или Parameters.

NewMask – новая маска обучаемости.

Назначение – Заменяет маску обучаемости параметров или входных сигналов сети, указанной в параметре Net.

Описание исполнения.

1. Если Error <> 0, то выполнение запроса прекращается.
2. Если в качестве аргумента Net дан пустой указатель, или указатель на пустую строку, то исполняющим запрос объектом является первая сеть в списке сетей компонента сеть.
3. Если список сетей компонента сеть пуст или имя сети, переданное в аргументе Net в этом списке не найдено, то возникает ошибка 301 – неверное имя сети, управление передается обработчику ошибок, а обработка запроса прекращается.
4. В зависимости от значения параметра Tipe заменяет маску обучаемости параметров или входных сигналов на переданную в параметре NewMask.

5.4.3.3 Обнулить градиент (NullGradient)

Описание запроса:

Pascal:

Function NullGradient(Net : PString) : Logic;

C:

Logic NullGradient(PString Net)

Описание аргументов:
Net – указатель на строку символов, содержащую имя сети.

Назначение – производит обнуление градиента сети, указанной в параметре Net.
Описание исполнения.

1. Если Error <> 0, то выполнение запроса прекращается.
2. Если в качестве аргумента Net дан пустой указатель, или указатель на пустую строку, то исполняющим запрос объектом является первая сеть в списке сетей компонента сеть.
3. Если список сетей компонента сеть пуст или имя сети, переданное в аргументе Net в этом списке не найдено, то возникает ошибка 301 – неверное имя сети, управление передается обработчику ошибок, а обработка запроса прекращается.
4. Обнуляются массивы Back.Parameters и Back.OutSignals.

5.4.3.4 Случайное направление спуска (RandomDirection)

Описание запроса:
Pascal:

```
Function RandomDirection( Net : PString; Range : Real ) : Logic;
```

C:

```
Logic RandomDirection(PString Net, Real Range)
```

Описание аргументов:
Net – указатель на строку символов, содержащую имя сети.
Range – относительная ширина интервала, на котором должны быть распределены значения случайной величины.

Назначение – генерирует вектор случайных поправок к параметрам сети.
Описание исполнения.

1. Если Error <> 0, то выполнение запроса прекращается.
2. Если в качестве аргумента Net дан пустой указатель, или указатель на пустую строку, то исполняющим запрос объектом является первая сеть в списке сетей компонента сеть.
3. Если список сетей компонента сеть пуст или имя сети, переданное в аргументе Net в этом списке не найдено, то возникает ошибка 301 – неверное имя сети, управление передается обработчику ошибок, а обработка запроса прекращается.
4. Замещают все значения массива Back.Parameters на случайные величины. Интервал распределения случайной величины зависит от типа параметра, указанного при описании сети (ParamType) и аргумента Range. Полуширина интервала определяется как произведение полуширины интервала допустимых значений параметра, указанных в разделе ParamDef описания сети на величину Range. Интервал распределения случайной величины определяется как [-Полуширина; Полуширина].

5.4.4 Запросы, работающие со структурой сети.

К третьей группе относятся запросы, позволяющие изменять структуру сети. Часть запросов этой группы описана в разд. "Остальные запросы".

5.4.4.1 Вернуть параметры сети (nwGetData)

Описание запроса:
Pascal:

```
Function nwGetData(Net : PString; DataType : Integer; Var Data : PRealArray) : Logic;
```

C:

```
Logic nwGetData(PString Net, Integer DataType, PRealArray* Data)
```

Описание аргументов:
Net – указатель на строку символов, содержащую имя сети.
DataType – одна из восьми предопределенных констант, описывающих тип данных сети.
Data – возвращаемый массив параметров сети.

Назначение – возвращает параметры, входные или выходные сигналы сети, указанной в аргументе Net.
Описание исполнения.

1. Если в качестве аргумента Net дан пустой указатель, или указатель на пустую строку, то исполняющим запрос объектом является перавя сеть в списке сетей компонента сеть.

2. Если имя сети, переданное в аргументе Net не найдено в списке сетей компонента сеть или этот список пуст, то возникает ошибка 301 – неверное имя сети, управление передается обработчику ошибок, а обработка запроса прекращается.
3. Если значение, переданное в аргументе DataType больше семи или меньше нуля, то возникает ошибка 306 – ошибочный тип параметра сети, управление передается обработчику ошибок, а обработка запроса прекращается.
4. В массиве Data возвращаются указанные в аргументе DataType параметры сети.

5.4.4.2 Установить параметры сети (nwSetData)

Описание запроса:
Pascal:

```
Function nwSetData(Net : PString; DataType : Integer; Var Data : RealArray) : Logic;
```

C:

```
Logic nwSetData(PString Net, Integer DataType, RealArray* Data)
```

Описание аргументов:
Net – указатель на строку символов, содержащую имя сети.
DataType – одна из восьми предопределенных констант, описывающих тип данных сети.
Data – массив параметров для замещения текущего массива параметров сети.

Назначение – замещает параметры, входные или выходные сигналы сети, указанной в аргументе Net на значения из массива Data.
Описание исполнения.

1. Если в качестве аргумента Net дан пустой указатель, или указатель на пустую строку, то исполняющим запрос объектом является первая сеть в списке сетей компонента сеть.
2. Если имя сети, переданное в аргументе Net не найдено в списке сетей компонента сеть или этот список пуст, то возникает ошибка 301 – неверное имя сети, управление передается обработчику ошибок, а обработка запроса прекращается.
3. Если значение, переданное в аргументе DataType больше семи или меньше нуля, то возникает ошибка 306 – ошибочный тип параметра сети, управление передается обработчику ошибок, а обработка запроса прекращается.
4. Значения параметров (входных или выходных сигналов) сети заменяются на значения из массива Data. Если длинны массива Data недостаточно для замены значений всех параметров (входных или выходных сигналов), то замещаются только столько элементов массива параметров (входных или выходных сигналов) сколько элементов в массиве Data. Если длинна массива Data больше длинны массива параметров (входных или выходных сигналов), то заменяются все элементы вектора параметров (входных или выходных сигналов), а лишние элементы массива Data игнорируются.

5.4.4.3 Нормализовать сеть (NormalizeNet)

Описание запроса:
Pascal:

```
Function NormalizeNet(Net : PString) : Logic;
```

C:

```
Logic NormalizeNet(PString Net)
```

Описание аргумента:
Net – указатель на строку символов, содержащую имя сети.

Назначение – нормализация сети, указанной в аргументе Net.
Описание исполнения.

1. Если в качестве аргумента Net дан пустой указатель, или указатель на пустую строку, то исполняющим запрос объектом является первая сеть в списке сетей компонента сеть.
2. Если имя сети, переданное в аргументе Net не найдено в списке сетей компонента сеть или этот список пуст, то возникает ошибка 301 – неверное имя сети, управление передается обработчику ошибок, а обработка запроса прекращается.
3. Из сети удаляются связи, имеющие нулевой вес и исключенные из обучения. Нумерация сигналов и параметров сохраняется.
4. Из структуры сети удаляются «немые» участки – элементы и блоки, выходные сигналы которых не являются выходными сигналами сети в целом и не используются в качестве входных сигналов другими подсетями. Нумерация сигналов и параметров сохраняется.

5. Производится замена элементов, ставших «прозрачными» – путем замыкания входного сигнала на выходной, удаляются простые однородные сумматоры с одним входом и точки ветвления с одним выходом; адаптивные однородные сумматоры с одним входом заменяются синапсами. Нумерация сигналов и параметров сохраняется.
6. В каждом блоке производится замена имен подсетей на псевдонимы.
7. Производится изменение нумерации сигналов и параметров сети.

5.4.5 Остальные запросы

Ниже приведен список запросов, исполнение которых описано в главе "Общий стандарт":

nwSetCurrent – Сделать сеть текущей
nwAdd – Добавление сети
nwDelete – Удаление сети
nwWrite – Запись сети
nwGetStructNames – Вернуть имена подсетей
nwGetType – Вернуть тип подсети
nwEdit – Редактировать компоненту сеть
OnError – Установить обработчик ошибок
GetError – Дать номер ошибки
FreeMemory – Освободить память

В запросе nwGetType в переменной TypeId возвращается значение одной из предопределенных констант, перечисленных в табл. 3.

Следует заметить, что два запроса nwGetData (Получить параметры) и nwSetData (Установить параметры) имеют название, совпадающее с названием запросов, описанных в главе "Общий стандарт", но они имеют другой набор аргументов.

5.4.5.1 Ошибки компонента сеть

В табл. 4 приведен полный список ошибок, которые могут возникать при выполнении запросов компонентом сеть, и действия стандартного обработчика ошибок.

Таблица 4

Ошибки компонента сеть и действия стандартного обработчика ошибок.

№Название ошибкиСтандартная обработка		
301	Неверное имя сетиЗанесение	номера в Error
302	Ошибка считывания сетиЗанесение	номера в Error
303	Ошибка сохранения сетиЗанесение	номера в Error
304	Ошибка прямого функционированияЗанесение	номера в Error
305	Ошибка обратного функционированияЗанесение	номера в Error
306	Ошибочный тип параметра сетиЗанесение	номера в Error

6. Оценка и интерпретатор ответа

Эта глава посвящена обзору различных видов оценок, способам их вычисления. В ней так же рассмотрен способ определения уровня уверенности сети в выданном ответе и приведен способ построения оценок, позволяющих определять уровень уверенности. Приведен основной принцип проектирования оценки - надо учить сеть тому, что мы хотим от нее получить.

Напомним основные функции, которые должна выполнять оценка:

1. Вычислять оценку решения, выданного сетью.
2. Вычислять производные этой оценки по выходным сигналам сети.

Кроме оценок, в первом разделе этой главы рассмотрен другой, тесно связанный с ней объект - интерпретатор ответа. Основное назначение этого объекта - интерпретировать выходной вектор сети как ответ, понятный пользователю. Однако, при определенном построении интерпретатора и правильно построенной по нему оценке, интерпретатор ответа может также оценивать уровень уверенности сети в выданном ответе.

6.1 Интерпретатор ответа

Как было показано в главе «Описание нейронных сетей», ответ, выдаваемый нейронной сетью, как правило, является числом, из диапазона $[a,b]$. Если ответ выдается несколькими нейронами, то на выходе сети мы имеем вектор, каждый компонент которого лежит в интервале $[a,b]$. Если в качестве ответа требуется число из этого диапазона, то мы можем его получить. Однако, в большинстве случаев это не так. Достаточно часто требуемая в качестве ответа величина лежит в другом диапазоне. Например, при предсказании температуры воздуха 25 июня в Красноярске ответ должен лежать в интервале от 5 до 35 градусов Цельсия. Сеть не может дать на выходе такого сигнала. Значит, прежде чем обучать сеть необходимо решить в каком виде будем требовать ответ. В данном случае ответ можно требовать в виде $\alpha = (b-a)(T - T_{\min}) / (T_{\max} - T_{\min}) + a$, где T - требуемая температура, $T_{\min}$ и $T_{\max}$ минимальная и максимальная температуры, α - ответ, который будем требовать от сети. При интерпретации ответа необходимо проделать обратное преобразование. Если сеть выдала сигнал α, то ответом является величина $T = (\alpha - a)(T_{\max} - T_{\min}) / (b-a) + T_{\min}$. Таким образом, можно интерпретировать выдаваемый сетью сигнал, как величину из любого, наперед заданного диапазона.

Если при составлении обучающего множества ответ на примеры определялся с некоторой погрешностью, то от сети следует требовать не точного воспроизведения ответа, а попадания в интервал заданной ширины. В этом случае интерпретатор ответа может выдать сообщение о правильности (попадании в интервал) ответа.

Другим, часто встречающимся случаем, является предсказание сетью принадлежности входного вектора одному из заданных классов. Такие задачи называют задачами классификации, а решающие их сети - классификаторами. В простейшем случае задача классификации ставится следующим образом: пусть задано N классов. Тогда нейросеть выдает вектор из N сигналов. Однако, нет единого универсального правила интерпретации этого вектора. Наиболее часто используется интерпретация по максимуму: номер нейрона, выдавшего максимальный по величине сигнал, является номером класса, к которому относится предъявленный сети входной вектор. Такие интерпретаторы ответа называются интерпретаторами, кодирующими ответ **номером канала** (номер нейрона - номер класса). Все интерпретаторы, использующие кодирование номером канала, имеют один большой недостаток - для классификации на N классов требуется N выходных нейронов. При большом N требуется много выходных нейронов для получения ответа. Однако существуют и другие виды интерпретаторов.

Двоичный интерпретатор. Основная идея двоичного интерпретатора - получение на выходе нейронной сети двоичного кода номера класса. Это достигается двухэтапной интерпретацией:

1. Каждый выходной сигнал нейронной сети интерпретируется как 1, если он больше $(a+b)/2$, и как 0 в противном случае.
2. Полученная последовательность нулей и единиц интерпретируется как двоичное число.

Двоичный интерпретатор позволяет интерпретировать N выходных сигналов нейронной сети как номер одного из 2^N классов.

Порядковый интерпретатор. Порядковый интерпретатор кодирует номер класса подстановкой. Отсортируем вектор выходных сигналов по возрастанию. Вектор, составленный из номеров нейронов последовательно расположенных в отсортированном векторе выходных сигналов, будет подстанов-

кой. Если каждой подстановке приписать номер класса, то такой интерпретатор может закодировать *N!* классов используя *N* выходных сигналов.

6.2 Уровень уверенности

Часто при решении задач классификации с использованием нейронных сетей недостаточно простого ответа «входной вектор принадлежит *K-му* классу». Хотелось бы также оценить уровень уверенности в этом ответе. Для различных интерпретаторов вопрос определения уровня уверенности решается поразному. Однако, необходимо учесть, что от нейронной сети нельзя требовать больше того, чему ее обучили. В этом разделе будет рассмотрен вопрос об определении уровня уверенности для нескольких интерпретаторов, а в следующем будет показано, как построить оценку так, чтобы нейронная сеть позволяла его определить.

1. Кодирование номером канала. Знаковый интерпретатор. Знаковый интерпретатор работает в два этапа.

1. Каждый выходной сигнал нейронной сети интерпретируется как 1, если он больше $(a+b)/2$, и как 0 в противном случае.
2. Если в полученном векторе только одна единица, то номером класса считается номер нейрона, сигнал которого интерпретирован как 1. В противном случае ответом считается неопределенный номер класса (ответ «не знаю»).

Для того чтобы ввести уровень уверенности для этого интерпретатора потребуем, чтобы при обучении сети для всех примеров было верно неравенство: $|\alpha_i - (a+b)/2| \le \varepsilon$, где $i = 1,\mathrm{K}\ ,N$; α_i - i-ый выходной сигнал. ε - уровень надежности (насколько сильно сигналы должны быть отделены от $(a+b)/2$ при обучении). В этом случае уровень уверенности R определяется следующим образом: $R = \min\left\{1; \min \frac{|\alpha_i - (a+b)/2|}{\varepsilon}\right\}$. Таким образом, при определенном ответе уровень уверенности показывает, насколько ответ далек от неопределенного, а в случае неопределенного ответа - насколько он далек от определенного.

2. Кодирование номером канала. Максимальный интерпретатор. Максимальный интерпретатор в качестве номера класса выдает номер нейрона, выдавшего максимальный сигнал. Для такого интерпретатора в качестве уровня уверенности естественно использовать некоторую функцию от разности между максимальным и вторым по величине сигналами. Для этого потребуем, чтобы при обучении для всех примеров обучающего множества разность между максимальным и вторым по величине сигналами была не меньше уровня надежности ε. В этом случае уровень уверенности вычисляется по следующей формуле: $R = \max\left\{1; (\alpha_i - \alpha_j)/\varepsilon\right\}$, где α_i - максимальный, а α_j - второй по величине сигналы.

3. Двоичный интерпретатор. Уровень надежности для двоичного интерпретатора вводится так же, как и для знакового интерпретатора при кодировании номером канала.

4. Порядковый интерпретатор. При использовании порядкового интерпретатора в качестве уровня уверенности естественно брать функцию от разности двух соседних сигналов в упорядоченном по возрастанию векторе выходных сигналов. Для этого потребуем, чтобы при обучении для всех примеров обучающего множества в упорядоченном по возрастанию векторе выходных сигналов разность между двумя соседними элементами была не меньше уровня надежности ε. В этом случае уровень уверенности можно вычислить по формуле $R = \min_i \left\{1; (\alpha_{i+1} - \alpha_i)/\varepsilon\right\}$, причем вектор выходных сигналов предполагается отсортированным по возрастанию.

В заключение заметим, что для ответа типа число, ввести уровень уверенности подобным образом невозможно. Пожалуй, единственным способом оценки достоверности результата является консилиум нескольких сетей - если несколько сетей обучены решению одной и той же задачи, то в качестве ответа можно выбрать среднее значение, а по отклонению ответов от среднего можно оценить достоверность результата.

6.3 Построение оценки по интерпретатору

Если в качестве ответа нейронная сеть должна выдать число, то естественной оценкой является квадрат разности выданного сетью выходного сигнала и правильного ответа. Все остальные оценки для обучения сетей решению таких задач являются модификациями данной. Приведем пример такой моди-

фикации. Пусть при составлении задачника величина $\overline{\alpha}$, являющаяся ответом, измерялась с некоторой точностью ε. Тогда нет смысла требовать от сети обучиться выдавать в качестве ответа именно величину $\overline{\alpha}$. Достаточно, если выданный сетью ответ попадет в интервал $\left[\overline{\alpha}-\varepsilon, \overline{\alpha}+\varepsilon\right]$. Оценка, удовлетворяющая этому требованию, имеет вид:

$$H=\begin{cases}0, & \text{при } \left|\alpha-\overline{\alpha}\right|\le\varepsilon,\\ \left(\alpha-\overline{\alpha}-\varepsilon\right)^2, & \text{при } \alpha>\overline{\alpha}+\varepsilon,\\ \left(\alpha-\overline{\alpha}+\varepsilon\right)^2, & \text{при } \alpha<\overline{\alpha}-\varepsilon.\end{cases}$$

Эту оценку будем называть оценкой числа с допуском ε.

Для задач классификации также можно пользоваться оценкой типа суммы квадратов отклонений выходных сигналов сети от требуемых ответов. Однако, эта оценка плоха тем, что во-первых, требования при обучении сети не совпадают с требованиями интерпретатора, во-вторых - такая оценка не позволяет оценить уровень уверенности сети в выданном ответе. Достоинством такой оценки является ее универсальность. Опыт работы с нейронными сетями, накопленный красноярской группой НейроКомп, свидетельствует о том, что при использовании оценки, построенной по интерпретатору, в несколько раз возрастает скорость обучения. Рассмотрим построение оценок по интерпретатору для четырех рассмотренных в предыдущем разделе интерпретаторов ответа.

1. Кодирование номером канала. Знаковый интерпретатор. Пусть для рассматриваемого примера правильным ответом является *k*-ый класс. Тогда вектор выходных сигналов сети должен удовлетворять следующей системе неравенств:

$$\begin{cases}\alpha_i<(a+b)/2-\varepsilon, & i\ne k\\ \alpha_k>(a+b)/2+\varepsilon,\end{cases}$$

где ε- уровень надежности.

Оценку, вычисляющую расстояние от точки α в пространстве выходных сигналов до множества точек, удовлетворяющих этой системе неравенств, можно записать в виде:

$$H=\sum_{i\ne k}\left(\alpha_i-(a+b)/2+\varepsilon\right)^2+\left(\alpha_k-(a+b)/2-\varepsilon\right)^2.$$

Производная оценки по *i*-му выходному сигналу равна $\dfrac{\partial H}{\partial\alpha_i}=\begin{cases}2(\alpha_i-(a+b)/2+\varepsilon), i\ne k\\ 2(\alpha_k-(a+b)/2-\varepsilon).\end{cases}$.

2. Кодирование номером канала. Максимальный интерпретатор. Пусть для рассматриваемого примера правильным ответом является *k*-ый класс. Тогда вектор выходных сигналов сети должен удовлетворять следующей системе неравенств: $\alpha_k-\varepsilon\ge\alpha_i$, при $i\ne k$. Оценкой решения сетью данного примера является расстояние от точки α в пространстве выходных сигналов до множества точек, удовлетворяющих этой системе неравенств. Для записи оценки, исключим из вектора выходных сигналов сигнал α_k, а остальные сигналы отсортируем по убыванию. Обозначим величину $\alpha_k-\varepsilon$ через β_0, а вектор отсортированных сигналов через $\beta_1\ge\beta_2\ge\text{K}\ge\beta_{N-1}$. Система неравенств в этом случае приобретает вид $\beta_0\ge\beta_i$, при $i>1$. Множество точек удовлетворяющих этой системе неравенств обозначим через ***D***. Очевидно, что если $\beta_0\ge\beta_1$, то точка β принадлежит множеству ***D***. Если $\beta_0<\beta_1$, то найдем проекцию точки β на гиперплоскость $\beta_0=\beta_1$. Эта точка имеет координаты $\beta^1=\left(\dfrac{\beta_0+\beta_1}{2},\dfrac{\beta_0+\beta_1}{2},\beta_2,\text{K},\beta_{N-1}\right)$. Если $\dfrac{\beta_0+\beta_1}{2}>\beta_2$, то точка β^1 принадлежит множеству ***D***. Если нет, то точку β нужно проектировать на гиперплоскость $\beta_0=\beta_1=\beta_2$. Найдем эту точку. Ее координаты можно записать в следующем виде $\left(b,b,b,\beta_3,\text{K},\beta_{N-1}\right)$. Эта точка обладает тем свойством, что расстояние от нее до точки β минимально. Таким образом, для нахождения величины b достаточно взять производную от расстояния по b и приравнять ее к нулю:

$$\frac{d}{db}\left(\left(b-\beta_0\right)^2+\left(b-\beta_1\right)^2+\left(b-\beta_2\right)^2\right)=2\left(\left(b-\beta_0\right)+\left(b-\beta_1\right)+\left(b-\beta_2\right)\right)=$$
$$=3b-\beta_0-\beta_1-\beta_2=0$$

Из этого уравнения находим b и записываем координаты точки β^2:

$$\beta^2=\left(\frac{\beta_0+\beta_1+\beta_2}{3},\frac{\beta_0+\beta_1+\beta_2}{3},\frac{\beta_0+\beta_1+\beta_2}{3},\beta_3,\text{K},\beta_{N-1}\right).$$

Эта процедура продолжается дальше, до тех пор, пока при некотором l не выполнится неравенство $\frac{\sum_{i=0}^{l}\beta_i}{l+1}\geq\beta_{l+1}$ или пока l не окажется равной N-1. Оценкой является расстояние от точки β до точки $\beta^l=\left(\frac{\sum_{i=0}^{l}\beta_i}{l+1},\text{K},\frac{\sum_{i=0}^{l}\beta_i}{l+1},\beta_{l+1},\text{K},\beta_{N+1}\right)$. Она равна следующей величине: $H=\sum_{j=0}^{l}\left(\frac{\sum_{i=0}^{l}\beta_i}{l+1}-\beta_j\right)^2$. Производная оценки по выходному сигналу β_m равна:

$$\frac{\partial H}{\partial\beta_m}\begin{cases}\left(\frac{\sum_{i=0}^{l}\beta_i}{l+1}-\beta_m\right), & m\leq l,\\ 0, & m>l.\end{cases}$$

Для перехода к производным по исходным выходным сигналам α_i необходимо обратить сделанные на первом этапе вычисления оценки преобразования.

3. Двоичный интерпретатор. Оценка для двоичного интерпретатора строится точно также как и для знакового интерпретатора при кодировании номером канала. Пусть правильным ответом является k-ый класс, тогда обозначим через K множество номеров сигналов, которым в двоичном представлении k соответствуют единицы. При уровне надежности оценка задается формулой:

$$H=\sum_{i\notin K}\left(\alpha_i-\left(a+b\right)/2+\varepsilon\right)^2+\sum_{i\in K}\left(\alpha_i-\left(a+b\right)/2-\varepsilon\right)^2.$$

Производная оценки по i-му выходному сигналу равна:

$$\frac{\partial H}{\partial\alpha_i}=\begin{cases}2(\alpha_i-(a+b)/2+\varepsilon), i\notin K\\ 2(\alpha_k-(a+b)/2-\varepsilon), i\in K.\end{cases}.$$

4. Порядковый интерпретатор. Для построения оценки по порядковому интерпретатору необходимо предварительно переставить компоненты вектора α в соответствии с подстановкой, кодирующей правильный ответ. Обозначим полученный в результате вектор через β^0. Множество точек, удовлетворяющих условию задачи, описывается системой уравнений $\beta_i^0+\varepsilon\leq\beta_{i+1}^0$, где ε - уровень надежности. Обозначим это множество через ***D.*** Оценка задается расстоянием от точки β до проекции этой точки на множество ***D***. Опишем процедуру вычисления проекции.

1. Просмотрев координаты точки β^0, отметим те номера координат, для которых нарушается неравенство $\beta_i^0+\varepsilon\leq\beta_{i+1}^0$.

2. Множество отмеченных координат либо состоит из одной последовательности последовательных номеров $i, i+1, \ldots, i+l$, или из нескольких таких последовательностей. Найдем точку β^1, которая являлась бы проекцией точки β^0 на гиперплоскость, определяемую уравнениями $\beta_i^1 + \varepsilon = \beta_{i+1}^1$, где i пробегает множество индексов отмеченных координат. Пусть множество отмеченных координат распадается на n последовательностей, каждая из которых имеет вид

$$\beta^1 = \left(\beta_1^0, \ldots, \beta_{i_1-1}^0, \gamma_1, \gamma_1+\varepsilon, \ldots, \gamma_1+l_1\varepsilon, \beta_{i_1+l_1+1}^0, \ldots, \beta_{i_2-1}^0, \gamma_2, \gamma_2+\varepsilon, \ldots, \gamma_2+l_2\varepsilon, \ldots, \beta_N^0\right)$$

$\beta_{i_m}, \ldots, \beta_{i_m+l_m}$, где m - номер последовательности. Тогда точка β^1 имеет вид:

1. Точка β^1 является проекцией, и следовательно, расстояние от β^0 до β^1 должно быть минимальным. Это расстояние равно $\sum_{m=1}^{n}\left[\sum_{j=0}^{l_m}\left(\beta_{i_m+j}^0 - \gamma_m - j\varepsilon\right)^2\right]$. Для нахождения минимума этой функции необходимо приравнять к нулю ее производные по γ_m. Получаем систему уравнений $\sum_{j=0}^{l_m}\left(\beta_{i_m+j}^0 - \gamma_m - j\varepsilon\right) = 0$. Решая ее, находим $\gamma_m = \sum_{j=0}^{l_m}\beta_{i_m+j}^0 \Big/ \left(l_m+1\right) - l_m\varepsilon/2$.
2. Если точка β^1 удовлетворяет неравенствам, приведенным в первом пункте процедуры, то расстояние от нее до точки β^0 является оценкой. В противном случае, повторяем первый шаг процедуры, используя точку β^1 вместо β^0; Объединяем полученный список отмеченных компонентов со списком, полученным при поиске предыдущей точки; находим точку β^2, повторяя все шаги процедуры, начиная со второго.

Отметим, что в ходе процедуры число отмеченных последовательностей соседних индексов не возрастает. Некоторые последовательности могут сливаться, но новые возникать не могут.

После нахождения проекции можно записать оценку:

$$H = \sum_{m=1}^{n}\left[\sum_{j=0}^{l_m}\left(\beta_{i_m+j}^0 - \sum_{j=0}^{l_m}\beta_{i_m+j}^0 \Big/ \left(l_m+1\right) - (l_m-2j)\varepsilon/2\right)^2\right].$$

Обозначим через I_m m-ую последовательность соседних координат, выделенную при последнем исполнении первого шага процедуры вычисления оценки: $I_m = \left\{i_m, i_m+1, \ldots, i_m+l_m\right\}$. Тогда производную оценки по выходному сигналу β_i^0 можно записать в следующем виде:

$$\frac{\partial H}{\partial \beta_i} = \begin{cases} \left(\beta_i^0 - \sum_{j=0}^{l_m}\beta_{i_m+j}^0 \Big/ \left(l_m+1\right) - (l_m-2(i-i_m))\varepsilon/2\right), \exists m: i \in I_m; \\ 0, i \notin \bigcup_{m=1}^{n} I_m. \end{cases}$$

Таким образом, построение оценки по интерпретатору сводится к следующей процедуре.

1. Определяем множество допустимых точек, то есть таких точек в пространстве выходных сигналов, которые интерпретатор ответа будет интерпретировать как правильный ответ со стопроцентным уровнем уверенности.
2. Находим проекцию выданной сетью точки на это множество. Проекцией является ближайшая точка из множества.
3. Записываем оценку как расстояние от точки, выданной сетью, до ее проекции на множество допустимых точек.

6.4 Оценка обучающего множества. Вес примера

В предыдущем разделе был рассмотрен ряд оценок, позволяющих оценить решение сетью конкретного примера. Однако, ситуация, когда сеть хотят обучить решению только одного примера, достаточно редка. Обычно сеть должна научиться решать все примеры обучающего множества. Ряд алгоритмов обучения, которые будут рассматриваться в главе "учитель", требуют возможности обучать сеть решению всех примеров одновременно и, соответственно, оценивать решение сетью всех примеров обучающего множества. Как уже отмечалось, обучение нейронной сети - это процесс минимизации в пространстве обучаемых параметров функции оценки. Большинство алгоритмов обучения используют способность нейронных сетей быстро вычислять вектор градиента функции оценки по обучаемым параметрам. Обозначим оценку отдельного примера через H_i. а оценку всего обучающего множества через H_{OM}. Простейший способ получения H_{OM} из H_i - простая сумма. При этом вектор градиента вычисляется очень просто:

$$H_{OM} = \sum H_i ,$$
$$\nabla H_{OM} = \nabla\left(\sum H_i\right) = \sum \nabla H_i .$$

Таким образом, используя способность сети вычислять градиент функции оценки решения одного примера, можно получить градиент функции оценки всего обучающего множества.

Обучение по всему обучающему множеству позволяет задействовать дополнительные механизмы ускорения обучения. Большинство этих механизмов будет рассмотрено в главе ???. В этом разделе будет рассмотрен только один из них - использование весов примеров. Использование весов примеров может быть вызвано одной из следующих причин.

1. Один из примеров плохо обучается.
2. Число примеров разных классов в обучающем множестве сильно отличаются друг от друга.
3. Примеры в обучающем множестве имеют различную достоверность.

Рассмотрим первую причину - пример плохо обучается. Под «плохо обучается» будем понимать медленное снижение оценки данного примера по отношению к снижению оценки по обучающему множеству. Для того чтобы ускорить обучение данного примера, ему можно приписать вес, больший, чем у остальных примеров. При этом оценка по обучающему множеству и ее градиент можно записать в следующем виде: $H_{OM} = \sum w_i H_i ; \nabla H_{OM} = \sum w_i \nabla H_i$. где w_i - вес *i*-го примера. Эту функцию оценки будем называть оценкой взвешенных примеров. При этом градиент, вычисленный по оценке решения сетью этого примера, войдет в суммарный градиент с большим весом, и, следовательно, сильнее повлияет на выбор направления обучения. Этот способ применим также и для коррекции проблем, связанных со второй причиной - разное число примеров разных классов. Однако в этом случае увеличиваются веса всем примерам того класса, в котором меньше примеров. Опыт показывает, что использование весов в таких ситуациях позволяет улучшить обобщающие способности сетей.

В случае различной достоверности примеров в обучающем множестве функция взвешенных примеров не применима. Действительно, если известно, что достоверность ответа в *k*-ом примере в два раза ниже, чем в *l*-ом, хотелось бы, чтобы обученная сеть выдавала для *k*-ого примера в два раза меньший уровень уверенности. Этого можно достичь, если при вычислении оценки *k*-ого примера будет использоваться в два раза меньший уровень надежности. Оценка обучающего множества в этом случае вычисляется по формуле без весов, а достоверность учитывается непосредственно при вычислении оценки по примеру. Такую оценку будем называть оценкой взвешенной достоверности.

Таким образом, каждый пример может иметь два веса: вес примера и достоверность примера. Кроме того, при решении задач классификации каждый класс может обладать собственным весом. Окончательно функцию оценки по обучающему множеству и ее градиент можно записать в следующем виде:

$$H_{OM}(\varepsilon) = \sum w_i H_i(\delta_i \varepsilon),$$
$$\nabla H_{OM}(\varepsilon) = \sum w_i \nabla H_i .(\delta_i \varepsilon),$$

где w_i - вес примера, δ_i - его достоверность.

6.5 Глобальные и локальные оценки

В предыдущих разделах был рассмотрен ряд оценок. Эти оценки обладают одним общим свойством - для вычисления оценки по примеру, предъявленному сети, достаточно знать выходной вектор, выданный сетью при решении этого примера, и правильный ответ. Такие оценки будем называть локальными. Приведем точное определение.

Определение. Локальной называется любая оценка, являющаяся линейной комбинацией произвольных непрерывно дифференцируемых функций, каждая из которых зависит от оценки только одного примера.

Использование локальных оценок позволяет обучать сеть решению как отдельно взятого примера, так и всего обучающего множества в целом. Однако существуют задачи, для которых невозможно построить локальную оценку. Более того, для некоторых задач нельзя построить даже обучающее множество. Использование нелокальных оценок возможно даже при решении задач классификации.

Приведем два примера нелокальных оценки.

Кинетическая оценка для задачи классификации. Пусть в обучающее множество входят примеры k классов. Требуется обучить сеть так, чтобы в пространстве выходных сигналов множества примеров разных классов были попарно линейно разделимы.

Пусть сеть выдает N выходных сигналов. Для решения задачи достаточно, чтобы в ходе обучения все точки в пространстве выходных сигналов, соответствующие примерам одного класса, собирались вокруг одной точки - центра концентрации класса, и чтобы центры концентрации разных классов были как можно дальше друг от друга. В качестве центра концентрации можно выбрать барицентр множества точек, соответствующих примерам данного класса.

Таким образом, функция оценки должна состоять из двух компонентов: первая реализует притяжение между примерами одного класса и барицентром этого класса, а вторая отвечает за отталкивание барицентров разных классов. Обозначим точку в пространстве выходных сигналов, соответствующую m-му примеру, через α^m, множество примеров i-го класса через I_i, барицентр точек, соответствующих примерам этого класса, через B^i ($B^i = \frac{1}{|I_i|}\sum_{m\in I_i}\alpha^m$), число примеров в i-ом классе через $\left|B^i\right|$, а расстояние между точками a и b через $\mathrm{dist}(a,b) = \sum_j \left(a_j - b_j\right)^2$. Используя эти обозначения, можно записать притягивающий компонент функции оценки для всех примеров i-го класса в виде:

$$H_i^P = \sum_{j\in I_i} \mathrm{dist}\left(\alpha^j, B^i\right)$$

Функция оценки H_i^P обеспечивает сильное притяжение для примеров, находящихся далеко от барицентра. Притяжение ослабевает с приближением к барицентру. Компонент функции оценки, отвечающий за отталкивание барицентров разных классов, должен обеспечивать сильное отталкивание близких барицентров и ослабевать с удалением барицентров друг от друга. Такими свойствами обладает гравитационное отталкивание. Используя гравитационное отталкивание можно записать второй компонент функции оценки в виде: $H^O = \sum_{i<j} \mathrm{dist}\left(B^i, B^j\right)^{-1}$. Таким образом, оценку, обеспечивающую сближение точек, соответствующих примерам одного класса, и отталкивание барицентров, можно записать в виде:

$$H = \sum_i H_i^P + H^O = \sum_i \sum_{j\in I_i} \mathrm{dist}\left(\alpha^j, B^i\right) + \sum_{i<j} \mathrm{dist}\left(B^i, B^j\right).$$

Вычислим производную оценки по j-му выходному сигналу, полученному при решении i-го примера. Пусть i-ый пример принадлежит l-му классу. Тогда производная имеет вид:

$$\frac{dH}{d\alpha_j^i} = 2\left(\alpha_j^i - B_j^l\right) - \frac{2}{|I_l|}\sum_{k\neq l} \frac{B_j^l - B_j^k}{\mathrm{dist}^2\left(B^l, B^k\right)}.$$

Эту оценку будем называть кинетической. Существует одно основное отличие этой оценки от всех других, ранее рассмотренных, оценок для решения задач классификации. При использовании традиционных подходов, сначала выбирают интерпретатор ответа, затем строят по выбранному интерпретатору функцию оценки, и только затем приступают к обучению сети. Для кинетической оценки такой

подход не применим. Действительно, до того как будет закончено обучение сети невозможно построить интерпретатор. Кроме того, использование кинетической оценки, делает необходимым обучение сети решению всех примеров обучающего множества одновременно. Это связанно с невозможностью вычислить оценку одного примера. Кинетическая оценка, очевидно, не является локальной: для вычисления производных оценки по выходным сигналам примера необходимо знать барицентры всех классов, для вычисления которых, в свою очередь, необходимо знать выходные сигналы, получаемые при решении *всех* примеров обучающего множества.

Интерпретатор для кинетической оценки строится следующим образом. Для построения разделителя *i*-го и *j*-го классов строим плоскость, перпендикулярную к вектору $\left(B^i - B^j\right)$. Уравнение этой плоскости можно записать в виде

$$\frac{\sum_{h=1}^{N}\left(B_h^i - B_p^j\right)\alpha_p}{\sum_{h=1}^{N}\left(B_h^i - B_p^j\right)^2} + D = 0.$$

Для определения константы D находим среди точек *i*-го класса ближайшую к барицентру *j*-го класса. Подставляя координаты этой точки в уравнение гиперплоскости, получаем уравнение на D. Решив это уравнение, находим величину D_1. Используя ближайшую к барицентру *i*-го класса точку *j*-го класса, находим величину D_2. Искомая константа D находится как среднее арифметическое между D_1 и D_2. Для отнесения произвольного вектора к *i*-му или j-му классу достаточно подставить его значения в левую часть уравнения разделяющей гиперплоскости. Если значение левой части уравнения получается больше нуля, то вектор относится к j-му классу, в противном случае - к *i*-му.

Интерпретатор работает следующим образом: если для *i*-го класса все разделители этого класса с остальными классами выдали ответ *i*-ый класс, то окончательным ответом является *i*-ый класс. Если такого класса не нашлось, то ответ «не знаю». Ситуация, когда для двух различных классов все разделители подтвердили принадлежность к этому классу, невозможна, так как разделитель этих двух классов должен был отдать предпочтение одному из них.

Рассмотренный пример решения задачи с использованием нелокальной оценки позволяет выделить основные черты обучения с нелокальной оценкой:

1. Невозможность оценить решение одного примера.
2. Невозможность оценить правильность решения примера до окончания обучения.
3. Невозможность построения интерпретатора ответа до окончания обучения.

Этот пример является отчасти надуманным, поскольку его можно решить с использованием более простых локальных оценок. Ниже приведен пример задачи, которую невозможно решить с использованием локальных оценок.

Генератор случайных чисел. Необходимо обучить сеть генерировать последовательность случайных чисел из диапазона $[0,1]$ с заданными k первыми моментами. Напомним, что для выборки роль первого момента играет среднее значение, второго - средний квадрат, третьего - средний куб и так далее. Есть два пути решения этой задачи. Первый - используя стандартный генератор случайных чисел подготовить задачник и обучить по нему сеть. Этот путь плох тем, что такой генератор будет просто воспроизводить последовательность чисел, записанную в задачнике. Для получения такого результата можно просто хранить задачник.

Второй вариант - обучать сеть без задачника! Пусть нейросеть принимает один входной сигнал и выдает один выходной. При использовании сети выходной сигнал первого срабатывания сети (первое случайное число) будет служить входным сигналом для второго срабатывания сети и так далее.

Для построения оценки зададимся тремя наборами чисел: M_i - необходимое значение *i*-го момента, L_i - длина последовательности, на которой *i*-ый момент сгенерированной последовательности должен не более чем на ε_i отличаться от M_i. ε_i - точность вычисления *i*-го момента.

Выборочная оценка совпадения *i*-го момента в сгенерированной последовательности на отрезке, начинающемся с *j*-го случайного числа, вычисляется по следующей формуле: $\overline{M_i^j} = \frac{1}{L_i}\sum_{l=j}^{j+L_i-1}\alpha_l^i$, где α_l - выходной сигнал, полученный на l-ом срабатывании сети. Для оценки точности совпадения *i*-го

момента в сгенерированной последовательности на отрезке, начинающемся с j-го случайного числа, воспользуемся оценкой числа с допуском ε_i:

$$H_j^i = \begin{cases} 0, & \text{при } \left|\overline{M_i^j} - \mathrm{M}_i\right| \le \varepsilon_i, \\ \left(\overline{M_i^j} - \mathrm{M}_i - \varepsilon_i\right)^2, & \text{при } \overline{M_i^j} > \mathrm{M}_i +, \varepsilon_i \\ \left(\overline{M_i^j} - \mathrm{M}_i + \varepsilon_i\right)^2, & \text{при } \overline{M_i^j} < \mathrm{M}_i - \varepsilon_i. \end{cases}$$

Таким образом, при обучении сети генерации последовательности из N случайных чисел оценку можно записать в следующем виде:

$$H = \sum_{i=1}^{k} \sum_{j=1}^{N-L_i} H_j^i.$$

Производная оценки по выходному сигналу l-го срабатывания сети можно записать в следующем виде:

$$\frac{dH}{d\alpha_l} = \sum_{i=1}^{k} \sum_{j=l-L_i+1}^{l+L_i-1} \begin{cases} 0, & \text{при } \left|\overline{M_i^j} - \mathrm{M}_i\right| \le \varepsilon_i, \\ \dfrac{i\alpha^{i-1}}{L_i}\left(\overline{M_i^j} - \mathrm{M}_i - \varepsilon_i\right), & \text{при } \overline{M_i^j} > \mathrm{M}_i + \varepsilon_i, \\ \dfrac{i\alpha^{i-1}}{L_i}\left(\overline{M_i^j} - \mathrm{M}_i + \varepsilon_i\right), & \text{при } \overline{M_i^j} < \mathrm{M}_i - \varepsilon_i. \end{cases}$$

Используя эту оценку можно обучать сеть генерировать случайные числа. Удобство этого подхода к решению задачи обучения генератора случайных чисел в том, что можно достаточно часто менять инициирующий сеть входной сигнал, что позволит сети генерировать не одну, а много различных последовательностей, обладающих всеми необходимыми свойствами.

При использовании предложенной оценки нет никаких гарантий того, что в генерируемой сетью последовательности не появятся сильно скоррелированные подпоследовательности. Для удаления корреляций можно модифицировать оценку так, чтобы она возрастала при появлении корреляций. Рассмотрим две подпоследовательности длинны L, первая из которых начинается с α_i, а другая с α_{i+h}. Коэффициент корреляции этих последовательностей записывается в виде:

$$r_{ih} = \frac{\dfrac{1}{L}\sum_{j=0}^{L-1} \alpha_{i+j}\alpha_{i+j+h} - \overline{\alpha_i}\,\overline{\alpha_{i+h}}}{\sqrt{\left(\overline{\alpha_i^2} - \overline{\alpha_i}^2\right)\left(\overline{\alpha_{i+h}^2} - \overline{\alpha_{i+h}}^2\right)}}.$$

В этой формуле приняты следующие обозначения: $\overline{\alpha_i}$ - среднее по последовательности, начинающейся с α_i; $\overline{\alpha_i^2}$ - средний квадрат последовательности начинающейся с α_i. Вычисление такого коэффициента корреляции довольно долгий процесс. Однако вместо выборочных моментов в формулу можно подставить значения моментов, которые последовательность должна иметь. В этом случае формула сильно упрощается:

$$r_{ih} = \frac{\dfrac{1}{L}\sum_{j=0}^{L-1} \alpha_{i+j}\alpha_{i+j+h} - M_1^2}{M_2 - M_1^2}.$$

Добавку для удаления корреляций последовательностей длиной от L_1 до L_2 и смещенных друг относительно друга на смещения от h_1 до h_2 можно записать в виде:

$$H_r = \sum_{L=L_1}^{L_2} \sum_{h=h_1}^{h_2} \sum_{i=1}^{N-h-L+1} \left(\frac{\frac{1}{L} \sum_{j=0}^{L-1} \alpha_{i+j} \alpha_{i+j+h} - M_1^2}{M_2 - M_1^2} \right)^2 .$$

При необходимости можно ввести и другие поправки, учитывающие требования к генератору случайных чисел.

6.6 Составные интерпретатор ответа и оценка

При использовании нейронных сетей для решения различных задач возникает необходимость получать от сети не один ответ, а несколько. Например, при обучении сети решению задачи диагностики отклонений в реакции на стресс нейронная сеть должна была определить наличие или отсутствие тринадцати различных патологий. Если одна сеть может выдавать только один ответ, то для решения задачи необходимо задействовать тринадцать сетей. Однако в этом нет необходимости. Поскольку каждый ответ, который должна выдавать сеть, имеет только два варианта, то можно использовать для его получения классификатор на два класса. Для такого классификатора необходимо два выходных сигнала. Тогда для решения задачи достаточно получать 26 выходных сигналов: первые два сигнала - для определения первой патологии, третий и четвертый - для второй и так далее. Таким образом, интерпретатор ответа для этой задачи состоит из тринадцати интерпретаторов, а оценка из тринадцати оценок. Более того, нет никаких ограничений на типы используемых интерпретаторов или оценок. Возможна комбинация, например, следующих ответов.

1. Число с допуском.
2. Классификатор на восемь классов.
3. Случайное число.

При использовании таких составных оценок и интерпретаторов каждый из этих компонентов должен следить за тем, чтобы каждая частная оценка или интерпретатор получали на вход те данные, которые им необходимы.

6.7 Стандарт первого уровня компонента интерпретатор ответа

Данный раздел посвящен описанию стандарта записи на диск компонента интерпретатор ответов. Построение интерпретатора происходит в редакторе интерпретаторов ответа. Интерпретатор ответа всегда является составным, даже если выходом является один ответ. В состав этого объекта входят частные интерпретаторы. Кроме того, описание интерпретатора должно включать в себя правила распределения выходных сигналов сети между частными интерпретаторами и расположения ответов частных интерпретаторов в едином массиве ответов.

Таким образом, интерпретатор ответа при выполнении запроса на интерпретацию массива выходных сигналов сети получает на входе массив выходных сигналов сети, а возвращает два массива – ответов и коэффициентов уверенности.

Каждый частный интерпретатор ответа получает на входе массив сигналов (возможно из одного элемента), которые он интерпретирует, а на выходе возвращает два числа – ответ и коэффициент уверенности в этом ответе.

Таблица 1.

Ключевые слова языка описания интерпретаторов ответа.

Ключевое слово	Краткое описание
1. Answer	Ответ.
2. Connections	Начало блока описания распределения сигналов и ответов.
3. Contents	Начало блока описания состава интерпретатора.
4. Include	Предшествует имени файла, целиком вставляемого в это место описания.
5. Interpretator	Заголовок раздела файла, содержащий описание интерпретатор.
6. NumberOf	Функция. Возвращает число интерпретируемых частным интерпретатором сигналов.
7. Reliability	Коэффициент уверенности.
8. Signals	Имя, по которому адресуются интерпретируемые сигналы; начало блока описания сигналов.
9. SetParameters	Процедура установления значений параметров.

Таблица 2.

Стандартные частные интерпретаторы.

НазваниеПараметрыАргументыОписание			
Empty	B – множитель C – смещение		Интерпретирует один сигнал A. Ответом является величина O=A*B+C
Binary	E – уровень надежности	N – число сигналов (классов)	Кодирование номером канала. Знаковый интерпретатор
Major	E – уровень надежности	N – число сигналов (классов)	Кодирование номером канала. Максимальный интерпретатор.
BynaryCoded	E – уровень надежности	N – число сигналов (классов)	Двоичный интерпретатор.

В табл. 1 приведен список ключевых слов, специфических для языка описания интерпретатора ответов. Наиболее часто встречающиеся интерпретаторы объявлены стандартными. Для стандартных интерпретаторов описание частных интерпретаторов отсутствует. Список стандартных интерпретаторов приведен в табл. 2.

6.7.1 БНФ языка описания интерпретатора

Обозначения, принятые в данном расширении БНФ и описание ряда конструкций приведены в главе «Общий стандарт» в разделе «Описание языка описания компонентов».

<Описание интерпретатора> ::= <Заголовок> [<Описание функций>] <Описание частных интерпретаторов> <Описание состава> [<Установление параметров>] [<Описание сигналов>] [<Описание распределения сигналов>] [<Описание распределения ответов>] <Конец описания интерпретатора>

<Заголовок> ::= **Interpretator** <Имя интерпретатора>

<Имя интерпретатора>::= <Идентификатор>

<Описание частных интерпретаторов> ::= <Описание частного интерпретатора> [<Описание частных интерпретаторов>]

<Описание частного интерпретатора> ::= <Заголовок описания интерпретатора> [<Описание статических переменных >] [<Описание переменных>] <Тело интерпретатора>

<Заголовок описания интерпретатора> ::= **Inter** <Имя частного интерпретатора> : (<Список формальных аргументов>)

<Имя частного интерпретатора> ::= <Идентификатор>

<Тело интерпретатора> ::= **Begin** <Составной оператор> **End**

<Описание состава> ::= **Contents** <Список имен интерпретаторов> ;

<Список имен интерпретаторов> ::= <Имя интерпретатора> [,<Список имен интерпретаторов >]

<Имя интерпретатора> ::= <Псевдоним>: {<Имя ранее описанного интерпретатора> | <Имя стандартного интерпретатора>} [*[*<Число экземпляров >*]*][*(*<Список фактических аргументов>*)*]

<Псевдоним> ::= <Идентификатор>

<Число экземпляров > ::= <Целое число>

<Имя ранее описанного интерпретатора> ::= <Идентификатор>

<Имя стандартного интерпретатора> ::= <Идентификатор>

<Установление параметров> ::= <Установление параметров ***Частного интерпретатора***> [;<Установление параметров>]

<Описание сигналов> ::= **Signals <**Константное выражение типа ***Long>***

<Описание распределения сигналов> ::= <Описание распределения ***Сигналов, Интерпретатора, Частного интерпретатора, Signals***>

<Описание распределения ответов> ::= <Описание распределения ***Ответов, Интерпретатора, Частного интерпретатора, Answer***>

<Конец описания интерпретатора> ::= **End Interpretator**

6.7.2 Описание языка описания интерпретаторов

Структура описания интерпретатора имеет вид: заголовок, описание частных интерпретаторов, описание состава, описание сигналов, описание распределения сигналов, описание распределения ответов, конец описания интерпретатора.

Заголовок состоит из ключевого слова Interpretator и имени интерпретатора и служит для обозначения начала описания интерпретатора в файле, содержащем несколько компонентов нейрокомпьютера.

Описание частного интерпретатора – это описание процедуры, вычисляющей две величины: ответ и уверенность в ответе. Отметим, что уверенность в ответе имеет смысл только для оценок с уровнем надежности. В остальных случаях интерпретатор ответа может вычислять аналогичную величину, но эта величина не является коэффициентом уверенности в ответе в точном смысле. Отметим, что при описании частного интерпретатора его аргументом, как правило, является число интерпретируемых сигналов. При выполнении частный интерпретатор получает в качестве аргументов массив интерпретируемых сигналов и две действительные переменные для возвращения вычисленных ответа и уверенности в ответе. Формально, при исполнении, частный интерпретатор имеет описание следующего вида:
Pascal:

```
Procedure Interpretator(Signals : PRealArray; Var Answer, Reliability : Real);
```

C:

```
void Interpretator(PRealArray Signals, Real* Answer, Real* Reliability);
```

В разделе описания состава перечисляются частные интерпретаторы, входящие в состав интерпретатора. Признаком конца раздела служит символ «;».

В необязательном разделе установления параметров производится задание значений параметров (статических переменных) частных интерпретаторов. После ключевого слова SetParameters следует список значений параметров в том порядке, в каком параметры были объявлены при описании частного интерпретатора (для стандартных интерпретаторов порядок параметров указан в табл. 2). При использовании одного оператора задания параметров для задания параметров нескольким экземплярам одного частного интерпретатора после ключевого слова SetParameters указывается столько выражений, задающих значения параметров, сколько необходимо для одного экземпляра. Например, если в блоке описания состава содержится 10 экземпляров двоичного интерпретатора на 15 интерпретируемых сигналов – MyInt : BinaryCoded(15)[10], то после ключевого слова SetParameters должно быть только одно выражение:

```
MyInt[I:1..10] SetParameters 0.01*I
```

В данном примере первый интерпретатор будет иметь уровень надежности равный 0.01, второй – 0.02 и т.д.

В необязательном разделе описание сигналов указывается число сигналов, интерпретируемых интерпретатором. Если этот раздел опущен, то полагается, что число интерпретируемых интерпретатором сигналов равно сумме сигналов, интерпретируемых всеми частными интерпретаторами. В константном выражении может вызываться функция NumberOf, аргументом которой является имя частного интерпретатора (или его псевдоним) с указанием фактических аргументов.

В необязательном разделе описания распределения сигналов для каждого частного интерпретатора указывается, какие сигналы из общего интерпретируемого массива передаются ему для интерпретации. Если этот раздел отсутствует, то считается, что каждый следующий частный интерпретатор получает следующий фрагмент общего вектора выходных сигналов. В примере 1 данный раздел описывает распределение сигналов по умолчанию.

В необязательном разделе описания распределения ответов для каждого частного интерпретатора указывается, какой элемент массива ответов он вычисляет. Если этот раздел опущен, то считается, что первый частный интерпретатор вычисляет первый элемент массива ответов, второй – второй элемент и т.д. Массив уровней надежностей всегда параллелен массиву ответов. В примере 1 данный раздел описывает распределение ответов по умолчанию.

Кроме того, в любом месте описания интерпретатора могут встречаться комментарии, заключенные в фигурные скобки.

6.7.3 Пример описания интерпретатора

В этом разделе приведены два примера описания одного и того же интерпретатора следующего состава: первый сигнал интерпретируется как температура путем умножения на 10 и добавления 273; следующие два сигнала интерпретируются как наличие облачности, используя знаковый интерпретатор; следующие три сигнала интерпретируются как направление ветра, используя двоичный интерпретатор (восемь румбов); последние три сигнала интерпретируются максимальным интерпретатором как сила осадков (без осадков, слабые осадки, сильные осадки). В первом примере приведено описание дубликатов всех стандартных интерпретаторов. Во втором – использованы стандартные интерпретаторы.

Пример 1.

```
Interpretator Meteorology

Inter Empty1 ()              {Интерпретатор осуществляющий масштабирование и сдвиг сигнала}
  Static
    Real B Name "Масштабный множитель";
```

```
    Real C Name "Сдвиг начала отсчета";
  Begin
    Answer = Signals[1] * B + C;
    Reliability = 0
  End

Inter Binary1 : ( N : Long )                { Кодирование номером канала. Знаковый интерпретатор}
  Static
    Real E Name "Уровень надежности";
  Var
    Long A, B, I;
    Real Dist;
  Begin
    Dist = E;
    B = 0;          {Число единиц}
    A = 0;          {Номер единицы}
    For I = 1 To N Do Begin
      If Abs(Signals[I]) < Dist Then Dist = Abs(Signals[I]);
      If Signals[I] > 0 Then Begin A = I; B = B + 1; End;
    End;
    If B <> 1 Then Answer = 0 Else Answer = A
    Reliability = Abs(Dist / E)
  End

Inter Major1 : ( N : Long)                  { Кодирование номером канала. Максимальный интерпретатор.}
  Static
    Real E Name "Уровень надежности";
  Var
    Real A, B;
    Long I, J;
  Begin
    A = -1.E+30;          {Максимальный сигнал}
    B = -1.E+30;              {Второй по величине сигнал}
    J = 0;                    {Номер максимального сигнала}
    For I = 1 To N Do Begin
      If Signals[I] > A Then Begin B = A; A = Signals[I]; J=I; End
      Else If Signals[I] > B Then B = Signals[I];
    End;
    Answer = J;
    If A - B > E Then Reliability = 1 Else Reliability = (A - B) / E;
  End

Inter BynaryCoded1 : ( N : Long )
  Static
    Real E Name "Уровень надежности";
  Var
    Long A, I;
    Real Dist;
  Begin
    Dist = E;
    A = 0;          {Ответ}
    For I = 1 To N Do Begin
      If Abs(Signals[I]) < Dist Then Dist = Abs(Signals[I]);
      A = A * 2;
      If Signals[I] > 0 Then A = A + 1;
    End;
    Answer = A;
    Reliability = Abs(Dist / E)
  End

Contents Temp : Empty1, Cloud : Binary1(2), Wind : BynaryCoded1(3), Rain : Major1(3);
```

Temp **SetParameters** 10, 273;
Cloud **SetParameters** 0.1;
Wind **SetParameters** 0.2;
Rain **SetParameters** 0.15

Signals NumberOf(**Signals**,Temp) + **NumberOf**(**Signals**, Cloud) + **NumberOf**(**Signals**, Wind) +
NumberOf(**Signals**, Rain)

Connections
Temp.**Signals** <=> **Signals**[1];
Cloud.**Signals**[1..2] <=> **Signals**[2; 3];
Wind.**Signals**[1..3] <=> **Signals**[4..6];
Rain.**Signals**[1..3] <=> **Signals**[7..9]
Temp.**Answer** <=> **Answer**[1];
Cloud.**Answer**[1..2] <=> **Answer**[2];
Wind.**Answer**[1..3] <=> **Answer**[3];
Rain.**Answer**[1..3] <=> **Answer**[4]

End Interpretator

Пример 2.
Interpretator Meteorology

Contents Temp : Empty, Cloud : Binary(2), Wind : BynaryCoded(3), Rain : Major(3);

Temp **SetParameters** 10, 273;
Cloud **SetParameters** 0.1;
Wind **SetParameters** 0.2;
Rain **SetParameters** 0.15

End Interpretator

6.8 Стандарт второго уровня компонента интерпретатор ответа

Запросы к компоненту интерпретатор ответа можно разбить на пять групп:

1. Интерпретация.
2. Изменение параметров.
3. Работа со структурой.
4. Инициация редактора и конструктора интерпретатора ответа.
5. Обработка ошибок.

Поскольку нейрокомпьютер может работать одновременно с несколькими сетями, то и компонент интерпретатор ответа должен иметь возможность одновременной работы с несколькими интерпретаторами. Поэтому большинство запросов к интерпретатору содержат явное указание имени интерпретатора ответа. Ниже приведено описание всех запросов к компоненту интерпретатор ответа. Каждый запрос является логической функцией, возвращающей значение истина, если запрос выполнен успешно, и ложь – при ошибочном завершении исполнения запроса.

В запросах второй и третьей группы при обращении к частным интерпретаторам используется следующий синтаксис:

<Полное имя частного интерпретатора> ::=
<Имя интерпретатора>.<Псевдоним частного интерпретатора> [*[*<Номер экземпляра>*]*]

Таблица 3.

Значения предопределенных констант компонента интерпретатор ответа и оценка

НазваниеВеличинаЗначение		
Empty	0	Интерпретирует один сигнал как действительное число.
Binary	1	Кодирование номером канала. Знаковый интерпретатор
Major	2	Кодирование номером канала. Максимальный интерпретатор.
BynaryCoded	3	Двоичный интерпретатор.
UserType	-1	Интерпретатор, определенный пользователем.

При вызове ряда запросов используются предопределенные константы. Их значения приведены в табл. 3.

6.8.1 Запрос на интерпретацию

Единственный запрос первой группы выполняет основную функцию компонента интерпретатор ответа – интерпретирует массив сигналов.

6.8.1.1 Интерпретировать массив сигналов (Interpretate)

Описание запроса:

Pascal:

```
Function Interpretate( IntName : PString; Signals : PRealArray;
    Var Reliability, Answers : PRealArray ) : Logic;
```

C:

```
Logic Interpretate(PString IntName, PRealArray Signals, PRealArray* Reliability, PRealArray* Answers)
```

Описание аргумента:

IntName – указатель на строку символов, содержащую имя интерпретатора ответа.

Signals – массив интерпретируемых сигналов.

Answers – массив ответов.

Reliability – массив коэффициентов уверенности в ответе.

Назначение – интерпретирует массив сигналов Signals, используя интерпретатор ответа, указанный в параметре IntName.

Описание исполнения.

1. Если Error <> 0, то выполнение запроса прекращается.
2. Если в качестве аргумента IntName дан пустой указатель, или указатель на пустую строку, то исполняющим запрос объектом является первый интерпретатор ответа в списке интерпретаторов компонента интерпретатор.
3. Если список интерпретаторов компонента интерпретатор пуст или имя интерпретатора ответа, переданное в аргументе IntName в этом списке не найдено, то возникает ошибка 501 – неверное имя интерпретатора ответа, управление передается обработчику ошибок, а обработка запроса прекращается.
4. Производится интерпретация ответа интерпретатором ответа, имя которого было указано в аргументе IntName.
5. Если во время выполнения запроса возникает ошибка, то генерируется внутренняя ошибка 504 - ошибка интерпретации. Управление передается обработчику ошибок. Выполнение запроса прекращается. В противном случае выполнение запроса успешно завершается.

6.8.2 Остальные запросы

Ниже приведен список запросов, исполнение которых описано в главе "Общий стандарт":

aiSetCurrent – Сделать интерпретатор ответа текущим
aiAdd – Добавление нового интерпретатора ответа
aiDelete – Удаление интерпретатора ответа
aiWrite – Запись интерпретатора ответа
aiGetStructNames – Вернуть имена частных интерпретаторов
aiGetType – Вернуть тип частного интерпретатора
aiGetData – Получить параметры частного интерпретатора
aiGetName – Получить имена параметров частного интерпретатора
aiSetData – Установить параметры частного интерпретатора
aiEdit – Редактировать интерпретатор ответа
OnError – Установить обработчик ошибок
GetError – Дать номер ошибки
FreeMemory – Освободить память

В запросе aiGetType в переменной TypeId возвращается значение одной из предопределенных констант, перечисленных в табл. 3.

При исполнении запроса aiSetData генерируется запрос SetEstIntParameters к компоненте оценка. Аргументы генерируемого запроса совпадают с аргументами исполняемого запроса

6.8.3 Ошибки компонента интерпретатор ответа

В табл. 4 приведен полный список ошибок, которые могут возникать при выполнении запросов компонентом интерпретатор ответа, и действия стандартного обработчика ошибок.

Таблица 4.

Ошибки компонента интерпретатор ответа и действия стандартного обработчика ошибок.

№Название ошибкиСтандартная	обработка
501 Неверное имя интерпретатора ответаЗанесение	номера в Error
502 Ошибка считывания интерпретатора ответаЗанесение	номера в Error
503 Ошибка сохранения интерпретатора ответаЗанесение	номера в Error
504 Ошибка интерпретацииЗанесение	номера в Error

6.9 Стандарт первого уровня компонента оценка

Данный раздел посвящен описанию стандарта хранения на диске описания компонента оценка. Построение оценки происходит в редакторе оценок. В данном стандарте предлагается ограничиться рассмотрением только локальных оценок, поскольку использование нелокальных (глобальных) оценок сильно усложняет компонент оценка, а область применения нелокальных оценок узка по сравнению с локальными оценками.

Оценка всегда является составной, даже если ответом сети является одна величина. В состав этого объекта входят частные оценки. Кроме того, в описание оценки включаются правила распределения выходных сигналов сети между частными оценками и расположения оценок, вычисляемых частными оценками, в едином массиве оценок. Кроме того, различные частные оценки могут иметь разную значимость. В этом случае общая оценка определяется как сумма частных оценок с весами, задающими значимость.

Таким образом, оценка при выполнении запроса на оценивание массива выходных сигналов сети получает на входе массив выходных сигналов сети, массив правильных ответов и массив их достоверностей, а возвращает два массива – массив оценок и массив производных оценки по выходным сигналам сети – и величину суммарной оценки. Возможны два режима оценивания: оценивание без вычисления массива производных оценки по выходным сигналам сети, и оценивание с вычислением массива производных.

Таблица 5

Ключевые слова языка описания оценок.

Ключевое словоКраткое описание	
1. Answer	Правильный ответ.
2. Back	Массив производных оценки по оцениваемым сигналам.
3. Contents	Начало блока описания состава оценки.
4. Direv	Признак необходимости вычисления производных.
5. Est	Заголовок описания частной оценки.
6. Estim	Переменная действительного типа, для возвращения вычисленной оценки.
7. Estimation	Заголовок раздела файла, содержащий описание оценки.
8. Include	Предшествует имени файла, целиком вставляемого в это место описания.
9. Link	Указывает интерпретатор ответа, связанный с оценкой.
10. NumberOf	Функция. Возвращает число интерпретируемых частным интерпретатором сигналов.
11. Reliability	Достоверность правильного ответа.
12. Signals	Имя, по которому адресуются интерпретируемые сигналы; начало блока описания сигналов.
13. Weight	Вес частной оценки.
14. Weights	Начало блока описания весов частных оценок.

Каждая частная оценка получает на входе свой массив сигналов (возможно из одного элемента), правильный ответ и его достоверность, а на выходе вычисляет оценку и, при необходимости, массив производных оценки по выходным сигналам сети.

В табл. 5 приведен список ключевых слов специфических для языка описания оценок. Наиболее часто встречающиеся частные оценки объявлены стандартными. Для стандартных оценок описание частных оценок отсутствует. Список стандартных оценок приведен в табл. 6.

Таблица 6

Стандартные частные оценки.

НазваниеПараметрыАргументыОписание			
Empty	B – множитель C – смещение		Оценивает один сигнал A, вычисляя расстояние до правильного ответа с учетом нормировки.
Binary	E – уровень надежности	N – число сигналов.	Кодирование номером канала. Соответствует знаковому интерпретатору.
Major	E – уровень надежности	N – число сигналов.	Кодирование номером канала. Соответствует максимальному интерпретатору.
BynaryCoded	E – уровень надежности	N – число сигналов.	Соответствует двоичному интерпретатору.

6.9.1 БНФ языка описания оценок

Обозначения, принятые в данном расширении БНФ и описание ряда конструкций приведены в главе «Общий стандарт» в разделе «Описание языка описания компонент».

<Описание оценки> ::= <Заголовок> [<Описание функций>] <Описание частных оценок> <Описание состава> [<Связывание с интерпретаторами>] [<Установление параметров>] [<Описание весов>] [<Описание сигналов>] [<Описание распределения сигналов>] [<Описание распределения оценок>] <Конец описания оценки>

<Заголовок> ::= **Estimation** <Имя оценки>

<Имя оценки>::= <Идентификатор>

<Описание частных оценок> ::= <Описание частной оценки> [<Описание частных оценок>]

<Описание частной оценки> ::= <Заголовок описания оценки> [<Описание статических переменных>] [<Описание переменных>] <Тело оценки>

<Заголовок описания оценки> ::= **Est** <Имя частной оценки> (<Список формальных аргументов>)

<Имя частной оценки> ::= <Идентификатор>

<Тело оценки> ::= **Begin** <Составной оператор> **End**

<Описание состава> ::= **Contents** <Список имен оценок> ;

<Список имен оценок> ::= <Имя оценки> [,<Список имен оценок >]

<Имя оценки> ::= <Псевдоним>: {<Имя ранее описанной оценки> | <Имя стандартной оценки>} [*(*<Список фактических аргументов>*)*] [*[*<Число экземпляров>*]*]

<Псевдоним> ::= <Идентификатор>

<Число экземпляров > ::= <Целое число>

<Имя ранее описанной оценки> ::= <Идентификатор>

<Имя стандартной оценки> ::= <Идентификатор>

<Установление параметров> ::= <Установление параметров ***Частной оценки***> [;<Установление параметров>]

<Связывание с интерпретаторами> ::= <Псевдоним> [*[*<Начальный номер> [..<Конечный номер> [:<Шаг>] *]*] **Link** <Псевдоним интерпретатора> [*[*<Начальный номер> [..<Конечный номер> [:<Шаг>] *]*]

<Псевдоним интерпретатора> ::= <Идентификатор>

<Описание весов> ::= **Weights** <Список весов>;

<Список весов> ::= <Вес> [,<Список весов>]

<Вес> ::= <Действительное число>

<Описание сигналов> ::= **Signals <**Константное выражение типа ***Long>***

<Описание распределения сигналов> ::= <Описание распределения ***Сигналов, Оценки, Частной оценки, Signals***>

<Описание распределения ответов> ::= <Описание распределения ***Ответов, Оценки, Частной оценки, Answer***>

<Конец описания оценки> ::= **End Estimation**

6.9.2 Описание языка описания оценок

Структура описания оценки имеет вид: заголовок, описание функций, описание частных оценок, описание состава, описание связей с интерпретаторами, описание сигналов, описание распределения сигналов, описание распределения ответов, конец описания оценки.

Заголовок состоит из ключевого слова Estimation и имени оценки и служит для обозначения начала описания оценки в файле, содержащем несколько компонент нейрокомпьютера.

Описание частной оценки – это описание процедуры, вычисляющей оценку и, при необходимости, массив производных оценки по выходным сигналам сети. Отметим, что при описании частной оценки его аргументом, как правило, является число оцениваемых сигналов. При выполнении частная оценка получает в качестве аргументов массив оцениваемых сигналов, признак необходимости вычисления производных, правильный ответ, достоверность правильного ответа, действительную переменную для возвращения вычисленной оценки и массив для возвращения производных. Формально, при исполнении частная оценка имеет описание следующего вида:

Pascal:

```
Procedure Estimation(Signals, Back : PRealArray; Direv : Logic; Answer, Reliability : Real; Var Estim : Real);
```

C:

```
void Estimation(PRealArray Signals, PRealArray Back,
                Logic Direv, Real Answer, Real Reliability, Real* Estim);
```

Отметим одну важную особенность выполнения тела частной оценки. Оператор присваивания значения элементу массива производных, означает добавление этого значения к величине, ранее находившейся в этом массиве. Например, запись **Back**[I] = A, означает выполнение следующего оператора **Back**[I] = **Back**[I] + A. Это связано с тем, что один и тот же сигнал может быть задействован в нескольких частных оценках и производная общей функции оценки равна сумме производных частных оценок по этому сигналу.

В разделе описания состава перечисляются частные оценки, входящие в состав оценки. Признаком конца раздела служит символ «;».

В необязательном разделе установления параметров производится задание значений параметров частных оценок. После ключевого слова SetParameters следует список значений параметров в том порядке, в каком параметры (статические переменные) были объявлены при описании частной оценки (для стандартных оценок порядок параметров указан в табл. 6). При использовании одного оператора задания параметров для задания параметров нескольким экземплярам одной частной оценки после ключевого слова SetParameters указывается столько выражений, задающих значения параметров, сколько необходимо для одного экземпляра. Например, если в блоке описания состава содержится 10 экземпляров двоичной оценки на 15 оцениваемых сигналов – MyEst : BinaryCoded(15)[10], то после ключевого слова SetParameters должно быть только одно выражение:

```
MyEst[I:1..10] SetParameters 0.01*I
```

В данном примере первая оценка будет иметь уровень надежности равный 0.01, вторая – 0.02 и т.д.

В необязательном разделе описания связей с интерпретаторами можно указать интерпретатор ответа, связанный с данной оценкой. Для связи интерпретатор и оценка должны иметь одинаковое число параметров и одинаковый порядок их описания. Так, в приведенном ниже примере, невозможно связывание оценки Temp с одноименным интерпретатором из-за различия в числе параметров. Если в левой части выражения Link указан диапазон оценок, то в правой части должен быть указан диапазон, содержащий столько же интерпретаторов. Указание связи влечет идентичность параметров оценки и связанного с ней интерпретатора ответов. Идентичность обеспечивается при исполнении запросов aiSetData и esSetData.

В необязательном разделе описания весов указываются веса частных оценок. Если этот раздел опущен, то все частные оценки равны единице, то есть все частные оценки имеют равную значимость.

В необязательном разделе описания сигналов указывается число сигналов, оцениваемых всеми частными оценками. Если этот раздел опущен, то полагается, что число оцениваемых оценкой сигналов равно сумме сигналов, оцениваемых всеми частными оценками. В константном выражении может вызываться функция NumberOf, аргументом которой является имя частной оценки (или ее псевдоним) с указанием фактических аргументов.

В необязательном разделе описания распределения сигналов для каждой частной оценки указывается, какие сигналы из общего оцениваемого массива передаются ей для оценивания. Если этот раздел опущен, то считается, что каждая следующая частная оценка получает следующий фрагмент массива сигналов. Порядок следования частных оценок соответствует порядку их перечисления в разделе описания состава. В примере 1 раздел описания распределения сигналов задает распределение сигналов по умолчанию. Массив производных оценки по выходным сигналам сети параллелен массиву сигналов.

В необязательном разделе описания распределения ответов для каждой частной оценки указывается какой элемент массива ответов будет ей передан. Если этот раздел опущен, то считается, что каждая следующая частная оценка получает следующий элемент массива ответов. Порядок следования частных оценок соответствует порядку их перечисления в разделе описания состава. В примере 1 раздел описания распределения ответов задает распределение сигналов по умолчанию. Массивы достоверностей ответов и вычисленных оценок параллельны массиву ответов.

Кроме того, в любом месте описания оценки могут встречаться комментарии, заключенные в фигурные скобки.

6.9.3 Пример описания оценки

В этом разделе приведены два примера описания одной и той же оценки следующего состава: первый сигнал интерпретируется как температура путем умножения на 10 и добавления 273; следующие два сигнала интерпретируются как наличие облачности, используя знаковый интерпретатор; следующие три сигнала интерпретируются как направление ветра, используя двоичный интерпретатор (восемь румбов); последние три сигнала интерпретируются максимальным интерпретатором как сила осадков (без осадков, слабые осадки, сильные осадки). Для трех последних интерпретаторов используются соответствующие им оценки типа расстояние до множества. В первом примере приведено описание дубликатов всех стандартных оценок. Во втором – использованы стандартные оценки.

Пример 1.

```
Estimation Meteorology

Est Empty1 ()     {Оценка для интерпретатора, осуществляющего масштабирование и сдвиг сигнала}
   Static
      Real B Name "Масштабный множитель";
      Real C Name "Сдвиг начала отсчета";
      Real E Name "Требуемая точность совпадения";
   Var
      Real A;
   Begin
      A = Signals[1] - (Answer - C) / B;
      D = E * Reliability;      {Уровень надежности с поправкой на достоверность}
      If Abs(A)<D Then Estim = 0
      Else
         If A > 0 Then Begin
            Estim = Weight * Sqr(A - D) / 2;
            If Direv Then Back[1] = Weight * (A - D);
         End Else Begin
            Estim = Weight * Sqr(A + D) / 2;
            If Direv Then Back[1] = Weight * (A + D);
         End
   End

Est Binary1 ( N : Long)      { Кодирование номером канала. Оценка для знакового интерпретатора.}
   Static
      Real E Name "Уровень надежности;
   Var
      Long I, J;
      Real A, B, C;
   Begin
      J = Answer;               {Правильный ответ – номер правильного класса}
      B = 0;
      C = E * Reliability;      {Уровень надежности с поправкой на достоверность}
      For I = 1 To N Do
         If  I = J Then Begin
            If Signals[I] < C Then Begin
               B = B + Sqr(Signals[I] - C);
               If Direv Then Back[I] = 2 * Weight * (Signals[I]-C);
            End;
         End Else Begin
            If Signals[I] > -C Then Begin
               B = B + Sqr(Signals[I] + C);
               If Direv Then Back[I] = 2 * Weight * (Signals[I] + C);
            End
         End;
      Estim = Weight*B
   End
```

```
Est Major1 ( N : Long) {Кодирование номером канала. Оценка для максимального интерпретатора.}
    Static
        Real E Name "Уровень надежности;
    Var
        Real A, B;
        Long I, J, K, Ans;
        RealArray[N+1] Al,Ind;
    Begin
        Ans = Answer;
        Ind[1] = Ans;
        Al[1] = Signals[Ans] - E * Reliability;
        Ind[N+1] = 0;
        Al[N+1] = -1.e40;
        K:=1;
        For I = 1 To N Do
            If I <> Ans Then Begin
                Al[K] = Signals[I];
                Ind[K] = I;
                K = K + 1;
             End;                          {Подготовлен массив сигналов}
        For I = 2 To N-1 Do Begin
            A = Al[I];
            K = I;
            For J = I+1 To N Do
                If Al[J] > A Then Begin
                    K = J;
                    A = Al[J];
                End;                       {Найден следующий по величине}
            Al[K] = Al[I];
            Al[I] = A;
            J = Ind[K];
            Ind[K] = Ind[I];
            Ind[I] = J;
        End;                               {Массивы отсортированы}
        A = Al[1];                      {Сумма первых I членов}
        I = 1;
        While (A / I <= Al[I+1]) Do Begin
            A = A + Al[I];
            I = I + 1;
        End;                               {В конце цикла I-1 равно числу корректируемых сигналов}
        B = A / I;                         {B – величина, к которой должны стремиться}
        A = 0;                             {корректируемые сигналы}
        For J = 1 To I Do Begin
            A = A + Sqr(Al[J] - B);
            If Direv Then Back[Ind[J]] = -2* Weight * (Al[J] - B);
        End;
        Estim = Weight * A
    End;

Est BynaryCoded1 : ( N : Long)              {Оценка для кодирования номером канала}
    Static
        Real E Name "Уровень надежности;
    Var
        Long I, J, A, K;
        Real B, C;
    Begin
        A = Answer;
        B = 0;
        C = E * Reliability;               {Уровень надежности с поправкой на достоверность}
```

```
        For I = N To 1 By -1 Do Begin
            J = A / 2;
            K = A - 2 * J;
            A = J;
            If  A = 1 Then Begin
                If Signals[I] < C Then Begin
                    B = B + Sqr(Signals[I] - C);
                    If Direv Then Back[I] = 2 * Weight * (Signals[I]-C);
                End;
            End Else Begin
                If Signals[I] > -C Then Begin
                    B = B + Sqr(Signals[I] + C);
                    If Direv Then Back[I] = 2 * Weight * (Signals[I] + C);
                End;
            End;
        Estim = Weight*B
    End

Contents Temp : Empty1, Cloud : Binary1(2), Wind : BynaryCoded1(3), Rain : Major1(3);

Cloud Link Meteorology.Cloud          {Связываем оценки с интерпретаторами}
Wind Link Meteorology.Wind
Rain Link Meteorology.Rain

Temp SetParameters 10, 273;           {Устанавливаем значения параметров оценок}
Cloud SetParameters 0.1;              {и интерпретаторов}
Wind SetParameters 0.2;
Rain SetParameters 0.15

Weights 1, 1, 1, 1

Signals NumberOf(Signals,Temp) + NumberOf(Signals, Cloud) + NumberOf(Signals, Wind) +
        NumberOf(Signals, Rain)

Connections
    Temp.Signals <=> Signals[1];
    Cloud.Signals[1..2] <=> Signals[2; 3];
    Wind.Signals[1..3] <=> Signals[4..6];
    Rain.Signals[1..3] <=> Signals[7..9]
    Temp.Answer <=> Answer[1];
    Cloud.Answer[1..2] <=> Answer[2];
    Wind.Answer[1..3] <=> Answer[3];
    Rain.Answer[1..3] <=> Answer[4]

End Interpretator
```

Пример 2.

```
Estimation Meteorology

Contents Temp : Empty, Cloud : Binary(2), Wind : BynaryCoded(3), Rain : Major(3);

Cloud Link Meteorology.Cloud          {Связываем оценки с интерпретаторами}
Wind Link Meteorology.Wind
Rain Link Meteorology.Rain

Temp SetParameters 10, 273;           {Устанавливаем значения параметров оценок}
Cloud SetParameters 0.1;              {и интерпретаторов}
Wind SetParameters 0.2;
Rain SetParameters 0.15

End Interpretator
```

6.10 Стандарт второго уровня компонента оценка

Запросы к компоненте оценка можно разбить на пять групп:

1. Оценивание.
2. Изменение параметров.
3. Работа со структурой.
4. Инициация редактора и конструктора оценки.
5. Обработка ошибок.

Поскольку нейрокомпьютер может работать одновременно с несколькими сетями, то и компонент оценка должен иметь возможность одновременной работы с несколькими оценками. Поэтому большинство запросов к оценке содержат явное указание имени оценки. Ниже приведено описание всех запросов к компоненту оценка. Каждый запрос является логической функцией, возвращающей значение истина, если запрос выполнен успешно, и ложь – при ошибочном завершении исполнения запроса.

В запросах второй и третьей группы при обращении к частным оценкам используется следующий синтаксис:

<Полное имя частной оценки> ::=
<Имя оценки>.<Псевдоним частной оценки> [/<Номер экземпляра>/]

При вызове ряда запросов используются предопределенные константы. Их значения приведены в табл. 3.

6.10.1 Запрос на оценивание

Единственный запрос первой группы выполняет основную функцию компонента оценка – вычисляет оценку и, если требуется, массив производных оценки по оцениваемым сигналам.

6.10.1.1 Оценить массив сигналов (Estimate)

Описание запроса:

Pascal:

```
Function Estimate( EstName : PString; Signals, Back, Answers, Reliability: PRealArray;
                   Direv : Logic; Var Estim : Real ) : Logic;
```

C:

```
Logic Estimate(PString EstName, PRealArray Signals, PRealArray* Back, PRealArray Answers,
               PRealArray Reliability, Logic Direv, Real* Estim)
```

Описание аргумента:

EstName – указатель на строку символов, содержащую имя оценки.
Signals – указатель на массив оцениваемых сигналов.
Back – указатель на массив производных оценки по оцениваемым сигналам.
Answers – указатель на массив правильных ответов.
Reliability – указатель на массив достоверностей правильных ответов.
Direv – признак необходимости вычисления производных (False – не вычислять).
Estim – вычисленная оценка.

Назначение – вычисляет оценку массива сигналов Signals, используя оценку, указанную в параметре EstName.

Описание исполнения.

1. Если Error <> 0, то выполнение запроса прекращается.
2. Если в качестве аргумента EstName дан пустой указатель, или указатель на пустую строку, то исполняющим запрос объектом является первая оценка в списке оценок компонента оценка.
3. Если список оценок компонента оценка пуст или имя оценки, переданное в аргументе EstName, в этом списке не найдено, то возникает ошибка 401 – неверное имя оценки, управление передается обработчику ошибок, а обработка запроса прекращается.
4. Производится вычисление оценки оценкой, имя которой было указано в аргументе EstName.
5. Если во время выполнения запроса возникает ошибка, то генерируется внутренняя ошибка 404 – ошибка оценивания. Управление передается обработчику ошибок. Выполнение запроса прекращается. В противном случае выполнение запроса успешно завершается.

6.10.2 Остальные запросы

Ниже приведен список запросов, исполнение которых описано в главе "Общий стандарт":

esSetCurrent – Сделать оценку текущим
esAdd – Добавление новой оценки
esDelete – Удаление оценки
esWrite – Запись оценки
esGetStructNames – Вернуть имена частных оценок

esGetType – Вернуть тип частной оценки
esGetData – Получить параметры частной оценки
esGetName – Получить имена параметров частной оценки
esSetData – Установить параметры частной оценки
esEdit – Редактировать оценку
OnError – Установить обработчик ошибок
GetError – Дать номер ошибки
FreeMemory – Освободить память

В запросе esGetType в переменной TypeId возвращается значение одной из предопределенных констант, перечисленных в табл. 3.

Кроме того, во второй группе запросов есть запрос SetEstIntParameters аналогичный запросу esSetData, но определяющий частную оценку, параметры которой изменяются, по полному имени связанного с ней интерпретатора ответа.

6.10.2.1 Установить параметры (SetEstIntParameters)

Описание запроса:

Pascal:

```
Function SetEstIntParameters( IntName : PString; Param : PRealArray ) : Logic;
```

C:

```
Logic SetEstIntParameters(PString IntName, PRealArray Param)
```

Описание аргументов:

IntName – указатель на строку символов, содержащую полное имя частного интерпретатора ответа.

Param – адрес массива параметров.

Назначение – заменяет значения параметров частной оценки, связанной с интерпретатором ответа, указанного в аргументе IntName, на значения, переданные, в аргументе Param.

Описание исполнения.

1. Запрос передается всем частным оценкам всех оценок в списке оценок компонента оценка.
2. Если частная оценка связана с частным интерпретатором ответа, имя которого указано в аргументе IntName, то текущие значения параметров частной оценки заменяются на значения, хранящиеся в массиве, адрес которого передан в аргументе Param,.

6.10.3 Ошибки компонента оценка

В табл. 7 приведен полный список ошибок, которые могут возникать при выполнении запросов компонентом оценка, и действия стандартного обработчика ошибок.

Таблица 7.

Ошибки компонента оценка и действия стандартного обработчика ошибок.

№Название ошибкиСтандартная	обработка
401 Неверное имя оценкиЗанесение	номера в Error
402 Ошибка считывания оценкиЗанесение	номера в Error
403 Ошибка сохранения оценкиЗанесение	номера в Error
404 Ошибка вычисления оценкиЗанесение	номера в Error

7. Исполнитель

Компонент исполнитель является служебным. Это означает, что он универсален и невидим для пользователя. В отличие от всех других компонентов исполнитель не выполняет ни одной явной функции в обучении нейронных сетей, а является вспомогательным для компонентов учитель и контрастер. Задача этого компонента – упростить работу компонентов учитель и контрастер. Этот компонент выполняет всего несколько запросов, преобразуя каждый из них в последовательность запросов к различным компонентам. В первой части главы содержательно рассмотрены алгоритмы исполнения всех запросов исполнителя, а во второй части приведено их формальное описание. Отметим, что ввиду универсальности компонента исполнитель стандарт первого уровня отсутствует.

7.1 Описание запросов исполнителя.

Как было описано в главе «Функциональные компоненты», исполнитель выполняет четыре вида запросов.

1. Тестирование решения примера.
2. Оценивание решения примера.
1. Оценивание решения примера с вычислением градиента.
2. Оценивание и тестирование решения примера.

Все перечисленные запросы работают с текущей сетью и текущим примером задачника. Однако компоненту задачник необходимо указать, какой пример подлежит обработке. Кроме того, в главе «Оценка и интерпретатор ответа» введен класс оценок, вычисляемых по всему обучающему множеству. Такие оценки позволяют существенно улучшить обучаемость сети и ускорить ее обучение. Нет смысла возлагать перебор примеров на учителя, поскольку это снижает полезность компонента исполнитель. Таким образом, возникает еще четыре вида запросов.

5. Тестирование решения всех примеров обучающего множества.
6. Оценивание решения всех примеров обучающего множества.
5. Оценивание решения всех примеров обучающего множества с вычислением градиента.
6. Оценивание и тестирование решения всех примеров обучающего множества.

Как уже отмечалось в главе «Функциональные компоненты», каждую из приведенных четверок запросов можно объединить в один запрос с параметрами. В табл. 1 приведен полный список параметров для первой четверки запросов, а в табл. 2 – для второй.

Символ «+» означает, что в запросе, номер которого указан в первой строке колонки, возможность, задаваемая данным параметром, должна быть использована. Символ «–» – что связанная с данным параметром возможность не используется. Символы «+/–» означают, что запрос может, как использовать, так и не использовать дан-

Таблица 1

Параметры запроса для позадачной работы

Название параметра	1	2	3	4
Перейти к следующему примеру	+/–	+/–	+/–	+/–
Остановиться в конце обучающего множества	+/–	+/–	+/–	+/–
Вычислять оценку	–	+	+	+
Интерпретировать ответ	+	–	–	+
Вычислять градиент	–	–	+	–
Подготовка к контрастированию	–	–	+/–	–

Таблица 2

Параметры запроса для обучающего множества в целом

Название параметра	5	6	7	8
Вычислять оценку	–	+	+	+
Интерпретировать ответ	+	–	–	+
Вычислять градиент	–	–	+	–
Подготовка к контрастированию	–	–	+/–	–

Таблица 3

Предопределенные константы компонента исполнитель

НазваниеИдентификаторЗначение		Десят.	Шестн.
Вычислять оценку	Estimate	1	0001
Интерпретировать ответ	Interpret	2	0002
Вычислять градиент	Gradient	4	0004
Подготовка к контрастированию	Contrast	8	0008
Перейти к следующему примеру	NextExample	16	0010
Остановиться в конце обучающего множества	StopOnEnd	32	0020
Устанавливать ответы	PutAnswers	64	0040
Устанавливать оценки	PutEstimations	128	0080
Устанавливать уверенность в ответе	PutReliability	256	0100

ную возможность. Отметим, что подготовка к контрастированию может быть задействована, только если производится вычисление градиента, а вычисление градиента невозможно без вычисления оценки. Остальные параметры независимы.

Отбор примеров в обучающее множество, открытие сеанса работы с задачником должны выполняться учителем или контрастером. Исполнитель только организует перебор примеров в обучающем множестве.

7.2 Стандарт компонента исполнитель второго уровня

В данном разделе описаны запросы исполнителя с алгоритмами их исполнения. При описании запросов используется аргумент Instruct, являющийся целым числом, принимающим значение одной из предопределенных констант, приведенных в табл. 3., или суммы любого числа этих констант. Аргумент Instruct является совокупностью шести битовых флагов.

В запросах не указываются используемые сеть, оценка и интерпретатор ответа, поскольку компонент исполнитель всегда использует текущие сеть, оценку и интерпретатор ответа.

7.2.1 Позадачная обработка (TaskWork)

Описание запроса:

Pascal:

```
Function TaskWork(Instruct, Handle : Integer; Var Answers, Reliability : PRealArray; Var Estim : Real) :
                                                                                    Logic;
```

C:

```
Logic TaskWork(Integer Instruct, Integer Handle, PRealArray* Answers, PRealArray* Reliability; Real* Estim)
```

Описание аргументов:

Instruct – содержит инструкции о способе исполнения.

Handle – номер сеанса в задачнике.

Answers – указатель на массив вычисленных ответов.

Reliability – указатель на массив коэффициентов уверенности сети в ответах.

Estim – оценка решения примера.

Назначение – производит обработку одного примера.

Переменные, используемые при исполнении запроса

InArray, RelArray – адреса массивов для обменов с задачником.

Back – адрес массива для обменов с оценкой.

Описание исполнения.

Если в любой момент исполнения запроса возникает ошибка при исполнении запросов к другим компонентам, то исполнение запроса прекращается, возвращается значение ложь, ошибка компонента исполнитель не генерируется.

1. Если в аргументе Instruct установлен бит Gradient и не установлен бит Estimate, то выполнение запроса прекращается, и генерируется ошибка 001 – Некорректное сочетание флагов в аргументе Instruct.
2. Если в аргументе Instruct установлен бит Gradient, то генерируется запрос к сети NullGradient с аргументом Null.
3. Если в аргументе Instruct установлен бит NextExample, то генерируется запрос к задачнику Next с аргументом Handle. (Переход к следующему примеру)
4. Генерируется запрос к задачнику Last с аргументом Handle. (Проверка, существует ли пример)
5. Если запрос Last вернул значение истина, то
   - 5.1. Если в аргументе Instruct установлен бит StopOnEnd, то исполнение запроса прекращается, возвращается значение ложь. (Примера нет, переход на начало не нужен)
   - 5.2. Генерируется запрос к задачнику Home с аргументом Handle. (Переход на начало обучающего множества)
6. Переменной InArray присваивается значение Null и генерируется запрос к задачнику Get с аргументами Handle, InArray, tbPrepared (Получает от задачника предобработанные входные сигналы)
7. Генерируется запрос к сети Forw, с аргументами Null, InArray (выполняется прямое функционирование сети).
8. Освобождается массив InArray

9. Присваивает переменной Data значение Null и генерирует запрос к сети GetNetData с аргументами Null, OutSignals, Data (Получает от сети выходные сигналы).
10. Если в аргументе Instruct установлен бит Interpret, то
    10.1. Генерируется запрос к интерпретатору ответа Interpretate с аргументами Data, Answers, Reliability. (Производит интерпретацию ответа)
    10.2. Если в аргументе Instruct установлен бит PutAnswers, то генерируется запрос к задачнику Put с аргументами Handle, Answers, tbCalcAnswers (Передает задачнику вычисленные ответы)
    10.3. Если в аргументе Instruct установлен бит PutReliability, то генерируется запрос к задачнику Put с аргументами Handle, Reliability, tbCalcReliability (Передает задачнику вычисленные коэффициенты уверенности в ответе)
11. Если в аргументе Instruct установлен бит Gradient, то создается массив Back того же размера, что и Data. В противном случае переменной Back присваивается значение Null.
12. Если в аргументе Instruct установлен бит Estimate, то
    12.1. Переменной InArray присваивается значение Null и генерируется запрос к задачнику Get с аргументами Handle, InArray, tbAnswers (Получает от задачника правильные ответы)
    12.2. Переменной RelArray присваивается значение Null и генерируется запрос к задачнику Get с аргументами Handle, RelArray, tbCalcReliability(Получает от задачника достоверности ответов)
    12.3. Генерируется запрос к оценке Estimate с аргументами Data, Back, InArray, RelArray, Direv, Estim. Вместо Direv передается ноль, если в аргументе Instruct установлен бит Gradient, и 1 в противном случае. (Вычисляет оценку примера и, возможно, производные)
    12.4. Если в аргументе Instruct установлен бит PutEstimations, то генерируется запрос к задачнику Put с аргументами Handle, Estim, tbEstimations (Передает задачнику оценку примера)
    12.5. Освобождает массивы InArray и RelArray.
13. Если в аргументе Instruct установлен бит Gradient, то генерируется запрос к сети Back, с аргументами Null, Back. Освобождает массив Back. (Выполняется обратное функционирование сети)
14. Освобождается массив Data.
15. Если в аргументе Instruct установлен бит Contrast, то генерируется запрос к контрастеру ContrastExample с аргументом истина.
16. Завершает исполнение, возвращая значение истина

7.2.2 Обработка обучающего множества (TaskSetWork)

Описание запроса:

Pascal:

```
Function TaskSetWork(Instruct, Handle : Integer; Var Tasks : Integer; Var Correct : PRealArray; Var Estim :
                                                                    Real) : Logic;
```

C:

```
Logic TaskSetWork(Integer Instruct, Integer Handle, Integer* Tasks, PRealArray* Correct, Real* Estim)
```

Описание аргументов:

Instruct – содержит инструкции о способе исполнения.

Handle – номер сеанса в задачнике.

Tasks – число примеров в обучающем множестве.

Correct – указатель на массив, первый элемент которого равен числу правильных ответов на первую подзадачу и т.д.

Estim – средняя оценка решения всех примеров обучающего множества.

Назначение – производит обработку всех примеров обучающего множества.

Переменные, используемые при исполнении запроса

InArray, AnsArray, RelArray – адреса массивов для обменов с задачником.

Answers – указатель на массив вычисленных ответов.

Reliability – указатель на массив коэффициентов уверенности сети в ответах.

Back – адрес массива для обменов с оценкой.

Work – рабочая переменная типа Real для подсчета суммарной оценки.

Weight – рабочая переменная типа Real для веса примера.

Описание исполнения.

Если в любой момент исполнения запроса возникает ошибка при исполнении запросов к другим компонентам, то исполнение запроса прекращается, освобождаются все созданные в нем массивы, возвращается значение ложь, ошибка компонента исполнитель не генерируется.

Значение бит NextExample и StopOnEnd в аргументе Instruct игнорируются.

1. Если в аргументе Instruct установлен бит Gradient и не установлен бит Estimate, то выполнение запроса прекращается, и генерируется ошибка 001 – Некорректное сочетание флагов в аргументе Instruct.
2. Если в аргументе Instruct установлен бит Interpret, то создаются массивы Answers и Reliability того же размера, что и Correct
3. Выполняется следующий фрагмент программы (Обнуление массива количеств правильных ответов)
   3.1. For I = 1 To TLong(Correct[0]) Do
   3.2. Correct[I] = 0
4. Обнуляем счетчик числа примеров: Tasks = 0
5. Обнуляем суммарную оценку: Work = 0
6. Переменной Back присваивается значение Null.
7. Присваивает переменной Data значение Null и генерирует запрос к сети GetNetData с аргументами Null, OutSignals, Data. (Получает от сети выходные сигналы, для выяснения размерности массива Data. Сами значения сигналов не нужны)
8. Если в аргументе Instruct установлен бит Gradient, то
   8.1. Генерируется запрос к сети NullGradient с аргументом Null.
   8.2. Создается массив Back того же размера, что и Data.
9. Генерируется запрос к задачнику Home с аргументом Handle. (Переход на начало обучающего множества)
10. Переменной InArray присваивается значение Null и генерируется запрос к задачнику Get с аргументами Handle, InArray, tbPrepared (Создаем массив InArray для получения от задачника предобработанных входных сигналов)
11. Переменной AnsArray присваивается значение Null и генерируется запрос к задачнику Get с аргументами Handle, AnsArray, tbAnswers (Создаем массив AnsArray для получения от задачника правильных ответов)
12. Если в аргументе Instruct установлен бит Estimate, то создается массив RelArray того же размера, что и AnsArray.
13. Генерируется запрос к задачнику Last с аргументом Handle. (Проверка, существует ли пример)
14. Если запрос Last вернул значение ложь, то
    14.1. Tasks = Tasks + 1
    14.2. Генерируется запрос к задачнику Get с аргументами Handle, InArray, tbPrepared (Получает от задачника предобработанные входные сигналы)
    14.3. Генерируется запрос к сети Forw, с аргументами Null, InArray. (Выполняется прямое функционирование сети)
    14.4. Генерирует запрос к сети GetNetData с аргументами Null, OutSignals, Data. (Получает от сети выходные сигналы)
    14.5. Если в аргументе Instruct установлен бит Interpret, то
        14.5.1. Генерируется запрос к интерпретатору ответа Interpretate с аргументами Data, Answers, Reliability. (Производит интерпретацию ответа)
        14.5.2. Если в аргументе Instruct установлен бит PutAnswers, то генерируется запрос к задачнику Put с аргументами Handle, Answers, tbCalcAnswers (Передает задачнику вычисленные ответы)
        14.5.3. Если в аргументе Instruct установлен бит PutReliability, то генерируется запрос к задачнику Put с аргументами Handle, Reliability, tbCalcReliability (Передает задачнику вычисленные коэффициенты уверенности в ответе)
        14.5.4. Генерируется запрос к задачнику Get с аргументами Handle, AnsArray, tbAnswers (Получает от задачника правильные ответы)
        14.5.5. Выполняется следующий фрагмент программы (Подсчитываются правильно полученные ответы)
        14.5.5.1.For I = 1 To TLong(Correct[0]) Do
        14.5.5.2. If Answers[I] = AnsArray[I] Then TLong(Correct[I]) = TLong(Correct[I]) + 1
    14.6. Если в аргументе Instruct установлен бит Estimate, то
        14.6.1. Если в аргументе Instruct не установлен бит Interpret, то генерируется запрос к задачнику Get с аргументами Handle, AnsArray, tbAnswers (Получает от задачника правильные ответы)

14.6.2. Генерируется запрос к задачнику Get с аргументами Handle, RelArray, tbCalcReliability (Получает от задачника достоверности ответов)
14.6.3. Генерируется запрос к оценке Estimate с аргументами Data, Back, AnsArray, RelArray, Direv, Estim. Вместо Direv передается ноль, если в аргументе Instruct установлен бит Gradient, и 1 в противном случае. (Вычисляет оценку примера и, возможно, производные)
14.6.4. Генерируется запрос к задачнику Get с аргументами Handle, Weight, tbWeight (Получает от задачника вес примера)
14.6.5. Work = Work + Estim * Weight (Подсчитываем суммарную оценку)
14.6.6. Если в аргументе Instruct установлен бит PutEstimations, то генерируется запрос к задачнику Put с аргументами Handle, Estim, tbEstimations (Передает задачнику оценку примера)
14.7. Если в аргументе Instruct установлен бит Gradient, то генерируется запрос к сети Back, с аргументами Null, Back. (Выполняется обратное функционирование сети)
14.8. Если в аргументе Instruct установлен бит Contrast, то генерируется запрос к контрастеру ContrastExample с аргументом ложь.
14.9. Генерируется запрос к задачнику Next с аргументом Handle. (Переход к следующему примеру)
14.10. Переход к шагу 13 алгоритма.
15. Вычисляем среднюю оценку: If Tasks = 0 Then Estim = 0 Else Estim = Work / Task
16. Если в аргументе Instruct установлен бит Contrast, то генерируется запрос к контрастеру ContrastExample с аргументом истина.
17. Освобождаются массивы Data, AnsArray и InArray.
18. Если в аргументе Instruct установлен бит Estimate, то освобождается массив и RelArray.
19. Если в аргументе Instruct установлен бит Interpret, то освобождаются массивы Answers и Reliability.
20. Если Back <> Null освобождается массив Back.
21. Завершает исполнение, возвращая значение истина

7.2.3 Ошибки компонента исполнитель

В табл. 4 приведен полный список ошибок, которые могут возникать при выполнении запросов компонентом исполнитель, и действия стандартного обработчика ошибок.

Таблица 4.

Ошибки компонента исполнитель и действия стандартного обработчика ошибок.

№Название ошибкиСтандартная	обработка
001 Некорректное сочетание флагов в аргументе Instruct.	Занесение номера в Error

8. Учитель

Этот компонент не является столь универсальным как задачник, оценка или нейронная сеть, поскольку существует ряд алгоритмов обучения жестко привязанных к архитектуре нейронной сети. Примерами таких алгоритмов могут служить обучение (формирование синаптической карты) сети Хопфилда, обучение сети Кохонена и ряд других аналогичных сетей. Однако в главе «Описание нейронных сетей» приводится способ формирования сетей, позволяющий обучать сети Хопфилда и Кохонена методом обратного распространения ошибки. Описываемый в этой главе стандарт компонента учитель ориентирован в первую очередь на обучение двойственных сетей (сетей обратного распространения ошибки).

8.1 Что можно обучать методом двойственности

Как правило, метод двойственности (обратного распространения ошибки) используют для подстройки параметров нейронной сети. Однако, как было показано в главе «Описание нейронных сетей», сеть может вычислять не только градиент функции оценки по обучаемым параметрам сети, но и по входным сигналам сети. Используя градиент функции оценки по входным сигналам сети можно решать задачу, обратную по отношению к обучению нейронной сети.

Рассмотрим следующий пример. Пусть есть сеть, обученная предсказывать по текущему состоянию больного и набору применяемых лекарств состояние больного через некоторый промежуток времени. Поступил новый больной. Его параметры ввели сети и она выдала прогноз. Из прогноза следует ухудшение некоторых параметров состояния больного. Возьмем выданный сетью прогноз, заменим значения параметров, по которым наблюдается ухудшение, на желаемые значения. Полученный вектор ответов объявим правильным ответом. Имея правильный ответ и ответ, выданный сетью, вычислим градиент функции оценки по входным сигналам сети. В соответствии со значениями элементов градиента изменим значения входных сигналов сети так, чтобы оценка уменьшилась. Проделав эту процедуру несколько раз, получим вектор входных сигналов, порождающих правильный ответ. Далее врач должен определить, каким способом (какими лекарствами или процедурами) перевести больного в требуемое (полученное в ходе обучения входных сигналов) состояние. В большинстве случаев часть входных сигналов не подлежит изменению (например пол или возраст больного). В этом случае эти входные сигналы должны быть помечены как не обучаемые (см. использование маски обучаемости входных сигналов в главе «Описание нейронных сетей»).

Таким образом, способность сетей вычислять градиент функции оценки по входным параметрам сети позволяет решать вполне осмысленную обратную задачу: так подобрать входные сигналы сети, чтобы выходные сигналы удовлетворяли заданным требованиям.

Кроме того, использование нейронных сетей позволяет ставить новые вопросы перед исследователем. В практике группы «НейроКомп» был следующий случай. Была поставлена задача обучить сеть ставить диагноз вторичного иммунодефицита по данным анализов крови и клеточного метаболизма. Вся обучающая выборка была разбита на два класса: больные и здоровые. При анализе базы данных стандартными статистическими методами значимых отличий обнаружить не удалось. Сеть оказалась не способна обучиться. Далее у исследователя было два пути: либо увеличить число нейронов в сети, либо определить, что мешает обучению. Исследователи выбрали второй путь. При обучении сети была применена следующая процедура: как только обучение сети останавливалось из-за невозможности дальнейшего уменьшения оценки, пример, имеющий наихудшую оценку, исключался из обучающего множества. После того, как сеть обучилась решению задачи на усеченном обучающем множестве, был проведен анализ исключенных примеров. Выяснилось, что исключено около половины больных. Тогда множество больных было разбито на два класса – больные1 (оставшиеся в обучающем множестве) и больные2 (исключенные). При таком разбиении обучающей выборки стандартные методы статистики показали значимые различия в параметрах классов. Обучение сети классификации на три класса быстро завершилось полным успехом. При содержательном анализе примеров, составляющих классы больные1 и больные2, было установлено, что к классу болные1 относятся больные на завершающей стадии заболевания, а к классу больные2 – на начальной. Ранее такое разбиение больных не проводилось. Таким образом, обучение нейронной сети решению прикладной задачи поставило перед исследователем содержательный вопрос, позволивший получить новое знание о предметной области.

Подводя итоги этого раздела, можно сказать, что, используя метод двойственности в обучении нейронных сетей можно:

1. Обучать сеть решению задачи.
2. Подбирать входные данные так, чтобы на выходе нейронной сети был заданный ответ.
3. Ставить вопросы о соответствии входных данных задачника постановке нейросетевой задачи.

8.2 Описание алгоритмов обучения

Все алгоритмы обучения сетей методом обратного распространения ошибки опираются на способность сети вычислять градиент функции ошибки по обучающим параметрам. Даже правило Хебба использует вектор псевдоградиента, вычисляемый сетью при использовании зеркального порогового элемента (см. раздел «Пороговый элемент» главы «Описание нейронных сетей»). Таким образом, акт обучения состоит из вычисления градиента и собственно обучения сети (модификации параметров сети). Однако, существует множество не градиентных методов обучения, таких, как метод покоординатного спуска, метод случайного поиска и целое семейство методов Монте-Карло. Все эти методы могут использоваться при обучении нейронных сетей, хотя, как правило, они менее эффективны, чем градиентные методы. Некоторые варианты методов обучения описаны далее в этой главе.

Поскольку обучение двойственных сетей с точки зрения используемого математического аппарата эквивалентно задаче многомерной оптимизации, то в данной главе рассмотрены только несколько методов обучения, наиболее используемых при обучении сетей. Более полное представление о методах оптимизации, допускающих использование в обучении нейронных сетей, можно получить из книг по методам оптимизации (см. например [48, 103, 142]).

8.2.1 Краткий обзор макрокоманд учителя

При описании методов используется набор макросов, приведенный в табл. 2. В табл. 2 дано пояснение выполняемых макросами действий. Все макрокоманды могут оперировать с данными как пространства параметров, так и пространства входных сигналов сети. В первой части главы полагается, что объект обучения установлен заранее. В макросах используются понятия и аргументы, приведенные в табл. 1. Список макрокоманд приведен в табл. 2. При описании методов обучения все аргументы имеют

Таблица 1

Понятия и аргументы макрокоманд учителя (предварительный список)

НазваниеСмысл	
ТочкаТочка	в пространстве параметров или входных сигналов. Аналогична вектору.
ВекторВектор	в пространстве параметров или входных сигналов. Аналогичен точке.
Вектор_минимумовВектор	минимальных значений параметров или входных сигналов.
Вектор_максимумовВектор	максимальных значений параметров или входных сигналов.
Указатель_на_векторАдрес	вектора. Используется для передачи векторов в макрокоманды.
Пустой_указательУказатель	на отсутствующий вектор.

Таблица 2

Список макрокоманд учителя (предварительный).

НазваниеАргументы	(типы)	Выполняемые действия
Создать_векторУказатель	_на_вектор	Создает экземпляр вектора с неопределенными значениями. Адрес вектора помещается в Указатель_на_вектор.
Освободить_векторУказатель	_на_вектор	Освобождает память занятую вектором, расположенным по адресу Указатель_на_вектор.
Случайный_векторУказатель	_на_вектор	В векторе, на который указывает Указатель_на_вектор, генерируется вектор, каждая из координат которого является случайной величиной, равномерно распределенной на интервале между значениями соответствующих координат векторов Вектор_минимумов и Вектор_максимумов.
Модификация_вектора	Указатель _на_вектор Старый_Шаг Новый_Шаг	Генерирует запрос на модификацию вектора (см. раздел «Провести обучение (Modify)» главы «Описание нейронных сетей»).
Оптимизация_шагаУказатель	_на_вектор Начальный_Шаг	Производит подбор оптимального шага (см. рис. 3).
Сохранить_векторУказатель	_на_вектор	Скопировать текущий вектор в вектор, указанный в аргументе Указатель_на_вектор.
Устано- вить_параметры	Указатель_на_вектор	Скопировать вектор, указанный в аргументе Указатель_на_вектор, в текущий вектор.
Вычислить_оценкуОценка	Вычисляет	оценку текущего вектора. Вычисленную величину складывает в аргумент Оценка.
Вычислить_градиент	Вычисляет	градиент функции оценки.

тип, определяемый типом аргумента макрокоманды. Если в описании макрокоманды в табл. 2 тип аргумента не соответствует ни одному из типов, приведенных в табл. 1, то эти аргументы имеют числовой тип.

8.2.2 Неградиентные методы обучения

Среди неградиентных методов рассмотрим следующие методы, каждый из которых является представителем целого семейства методов оптимизации:

1. Метод случайной стрельбы (представитель семейства методов Монте-Карло).
2. Метод покоординатного спуска (псевдоградиентный метод).
3. Метод случайного поиска (псевдоградиентный метод).
4. Метод Нелдера-Мида.

8.2.2.1 Метод случайной стрельбы

Идея метода случайной стрельбы состоит в генерации большой последовательности случайных точек и вычисления оценки в каждой из них. При достаточной длине последовательности минимум будет найден. Запись этой процедуры на макроязыке приведена на рис. 1

Остановка данной процедуры производится по команде пользователя или при выполнении условия, что О1 стало меньше некоторой заданной величины. Существует огромное разнообразие модификаций этого метода. Наиболее простой является метод случайной стрельбы с уменьшением радиуса. Пример процедуры, реализующей этот метод, приведен на рис. 2. В этом методе есть два параметра, задаваемых пользователем:

Число_попыток – число неудачных пробных генераций вектора при одном радиусе.

Минимальный_радиус – минимальное значение радиуса, при котором продолжает работать алгоритм.

Идея этого метода состоит в следующем. Зададимся начальным состоянием вектора параметров. Новый вектор параметров будем искать как сумму начального и случайного, умноженного на радиус, векторов. Если после Число_попыток случайных генераций не произошло уменьшения оценки, то уменьшаем радиус. Если произошло уменьшение оценки, то полученный вектор объявляем начальным и продолжаем процедуру с тем же шагом. Важно, чтобы последовательность уменьшающихся радиусов образовывала расходящийся ряд. Примером такой последовательности может служить использованный в примере на рис. 2 ряд $1/n$.

Отмечен ряд случаев, когда метод случайной стрельбы с уменьшением радиуса работает быстрее градиентных методов, но это скорее исключение, чем правило.

1. Создать_вектор В1
2. Создать_вектор В2
3. Вычислить_оценку О1
4. Сохранить_вктор В1
5. Установить_параметры В1
6. Случайный_вектор В2
7. Модификация_вектора В2, 0, 1
8. Вычислить_оценку О2
9. Если О2<О1 то переход к шагу 11
10. Переход к шагу 5
11. О1=О2
12. Переход к шагу 4
13. Установить_параметры В1
14. Освободить_вектор В1
15. Освободить_вектор В2

Рис. 1. Простейший алгоритм метода случайной стрельбы

1. Создать_вектор В1
2. Создать_вектор В2
3. Вычислить_оценку О1
4. Число_Смен_Радиуса=1
5. Радиус=1/ Число_Смен_Радиуса
6. Попытка=0
7. Сохранить_вктор В1
8. Установить_параметры В1
9. Случайный_вектор В2
10. Модификация_вектора В2, 1, Радиус
11. Вычислить_оценку О2
12. Попытка=Попытка+1
13. Если О2<О1 то переход к шагу 16
14. Если Попытка<=Число_попыток то переход к шагу 8
15. Переход к шагу 18
16. О1=О2
17. Переход к шагу 6
18. Число_Смен_Радиуса= Число_Смен_Радиуса+1
19. Радиус=1/ Число_Смен_Радиуса
20. Если радиус>= Минимальный_радиус то переход к шагу 6
21. Установить_параметры В1
22. Освободить_вектор В1
23. Освободить_вектор В2

Рис. 2. Алгоритм метода случайной стрельбы с уменьшением радиуса

8.2.2.2 Метод покоординатного спуска

Идея этого метода состоит в том, что если в задаче сложно или долго вычислять градиент, то можно построить вектор, обладающий приблизительно теми же свойствами, что и градиент следующим путем. Даем малое положительное приращение первой координате вектора. Если оценка при этом уве-

личилась, то пробуем отрицательное приращение. Далее так же поступаем со всеми остальными координатами. В результате получаем вектор, в направлении которого оценка убывает. Для вычисления такого вектора потребуется, как минимум, столько вычислений функции оценки, сколько координат у вектора. В худшем случае потребуется в два раза большее число вычислений функции оценки. Время же необходимое для вычисления градиента в случае использования двойственных сетей можно оценить как 2-3 вычисления функции оценки. Таким образом, учитывая способность двойственных сетей быстро вычислять градиент, можно сделать вывод о нецелесообразности применения метода покоординатного спуска в обучении нейронных сетей.

8.2.2.3 Подбор оптимального шага

Данный раздел посвящен описанию макрокоманды Оптимизация_Шага. Эта макрокоманда часто используется в описании процедур обучения и не столь очевидна как другие макрокоманды. Поэтому ее текст приведен на рис. 3. Идея подбора оптимального шага состоит в том, что при наличии направления в котором производится спуск (изменение параметров) задача многомерной оптимизации в пространстве параметров сводится к одномерной оптимизации – подбору шага. Пусть заданы начальный шаг (Ш2) и направление спуска (антиградиент или случайное) (Н). Тогда вычислим величину О1 – оценку в текущей точке пространства параметров. Изменив параметры на вектор направления, умноженный на величину пробного шага, вычислим величину оценки в новой точке – О2. Если О2 оказалось меньше либо равно О1, то увеличиваем шаг и снова вычисляем оценку. Продолжаем эту процедуру до тех пор, пока не получится оценка, большая предыдущей. Зная три последних значения величины шага и оценки, используем квадратичную оптимизацию – по трем точкам построим параболу и следующий шаг сделаем в вершину параболы. После нескольких шагов квадратичной оптимизации получаем приближенное значение оптимального шага.

Если после первого пробного шага получилось О2 большее О1, то уменьшаем шаг до тех пор, пока не получим оценку, меньше чем О1. После этого производим квадратичную оптимизацию.

```
1.  Создать_вектор В
2.  Сохранить_вктор В
3.  Вычислить_оценку О1
4.  Ш1=0
5.  Модификация_вектора Н, 1, Ш2
6.  Вычислить_оценку О2
7.  Если О1<О2 то переход к шагу 15
8.  Ш3=Ш2*3
9.  Установить_параметры В
10. Модификация_вектора Н, 1, Ш3
11. Вычислить_оценку О3
12. Если О3>О2 то переход к шагу 21
13. О1=О2  О2=О3  Ш1=Ш2  Ш2=Ш3
14. Переход к шагу 8
15. Ш3=Ш2  О3=О2
16. Ш2=Ш3/3
17. Установить_параметры В
18. Модификация_вектора Н, 1, Ш2
19. Вычислить_оценку О3
20. Если О2>=О1 то переход к шагу 15
21. Число_парабол=0
22. Ш=((Ш3Ш3-Ш2Ш2)О1+(Ш1Ш1-Ш3Ш3)О2  +(Ш2Ш2-
    Ш1Ш1)О3)/(2((Ш3-Ш2)О1+(Ш1-Ш3)О2 +(Ш2-Ш1)О3))
23. Установить_параметры В
24. Модификация_вектора Н, 1, Ш
25. Вычислить_оценку О
26. Если Ш>Ш2 то переход к шагу 32
27. Если О>О2 то переход к шагу 30
28. Ш3=Ш2  О3=О2  О2=О  Ш2=Ш
29. Переход к шагу 36
30. Ш1=Ш О1=О
31. Переход к шагу 36
32. Если О>О2 то переход к шагу 35
33. Ш3=Ш2  О3=О2  О2=О  Ш2=Ш
34. Переход к шагу 36
35. Ш1=Ш О1=О
36. Число_парабол= Число_парабол+1
37. Если Число_парабол<Максимальное_Число_Парабол то
    переход к шагу 22
38. Установить_параметры В
39. Модификация_вектора Н, 1, Ш2
40. Освободить_вектор В
```

Рис. 3. Алгоритм оптимизации шага

8.2.2.4 Метод случайного поиска

Этот метод похож на метод случайной стрельбы с уменьшением радиуса, однако в его основе лежит другая идея – сгенерируем случайный вектор и будем использовать его вместо градиента. Этот метод использует одномерную оптимизацию – подбор шага. Одномерная оптимизация описана в разделе

«Одномерная оптимизация». Процедура случайного поиска приведена на рис. 4. В этом методе есть два параметра, задаваемых пользователем.

Число_попыток – число неудачных пробных генераций вектора при одном радиусе.

Минимальный_радиус – минимальное значение радиуса, при котором продолжает работать алгоритм.

Идея этого метода состоит в следующем. Зададимся начальным состоянием вектора параметров. Новый вектор параметров будем искать как сумму начального и случайного, умноженного на радиус, векторов. Если после Число_попыток случайных генераций не произошло уменьшения оценки, то уменьшаем радиус. Если произошло уменьшение оценки, то полученный вектор объявляем начальным и продолжаем процедуру с тем же шагом. Важно, чтобы последовательность уменьшающихся радиусов образовывала расходящийся ряд. Примером такой последовательности может служить использованный в примере на рис. 4 ряд $1/n$.

1. Создать_вектор H
2. Число_Смен_Радиуса=1
3. Попытка=0
4. Радиус=1/ Число_Смен_Радиуса
5. Случайный_вектор H
6. Оптимизация шага H Радиус
7. Попытка=Попытка+1
8. Если Радиус=0 то Попытка=0
9. Если Попытка<=Число_попыток то переход к шагу 4
10. Число_Смен_Радиуса= Число_Смен_Радиуса+1
11. Радиус=1/ Число_Смен_Радиуса
12. Если Радиус>= Минимальный_радиус то переход к шагу 3
13. Освободить_вектор H

Рис. 4. Алгоритм метода случайного поиска

8.2.2.5 Метод Нелдера-Мида

Этот метод является одним из наиболее быстрых и наиболее надежных не градиентных методов многомерной оптимизации. Идея этого метода состоит в следующем. В пространстве оптимизируемых параметров генерируется случайная точка. Затем строится n-мерный симплекс с центром в этой точке, и длиной стороны l. Далее в каждой из вершин симплекса вычисляется значение оценки. Выбирается вершина с наибольшей оценкой. Вычисляется центр тяжести остальных n вершин. Проводится оптимизация шага в направлении от наихудшей вершины к центру тяжести остальных вершин. Эта процедура повторяется до тех пор, пока не окажется, что оптимизация не изменяет положения вершины. После этого выбирается вершина с наилучшей оценкой и вокруг нее снова строится симплекс с меньшими размерами (например $l/2$). Процедура продолжается до тех пор, пока размер симплекса, который необходимо построить, не окажется меньше требуемой точности.

Однако, несмотря на свою надежность, применение этого метода к обучению нейронных сетей затруднено большой размерностью пространства параметров.

8.2.3 Градиентные методы обучения

Изучению градиентных методов обучения нейронных сетей посвящено множество работ [47, 64, 90] (сослаться на все работы по этой теме не представляется возможным, поэтому дана ссылка на работы, где эта тема исследована наиболее детально). Кроме того, существует множество публикаций, посвященных градиентным методам поиска минимума функции [48, 103] (как и в предыдущем случае, ссылки даны только на две работы, которые показались наиболее удачными). Данный раздел не претендует на какую-либо полноту рассмотрения градиентных методов поиска минимума. В нем приведены только несколько методов, применявшихся в работе группой «НейроКомп». Все градиентные методы объединены использованием градиента как основы для вычисления направления спуска.

8.2.3.1 Метод наискорейшего спуска

Наиболее известным среди градиентных методов является метод наискорейшего спуска. Идея этого метода проста: поскольку вектор градиента указывает направление наискорейшего возрастания функции, то минимум следует искать в обратном направлении. Последовательность действий приведена на рис. 5.

1. Вычислить_оценку O2
2. O1=O2
3. Вычислить_градиент
4. Оптимизация шага Пустой_указатель Шаг
5. Вычислить_оценку O2
6. Если O1-O2<Точность то переход к шагу 2

Рис. 5. Метод наискорейшего спуска

Этот метод работает, как правило, на порядок быстрее методов случайного поиска. Он имеет два параметра – Точность, показывающий, что если изменение оценки за шаг метода меньше чем Точность, то обучение останавливается; Шаг – начальный шаг для оптимизации шага. Заметим, что шаг постоянно изменяется в ходе оптимизации шага.

Остановимся на основных недостатках этого метода. Во-первых, эти методом находится тот минимум, в область притяжения которого попадет начальная точка. Этот минимум может не быть глобальным. Существует несколько способов выхода из этого положения. Наиболее простой и действенный – случайное изменение параметров с дальнейшим повторным обучение методом наискорейшего спуска. Как правило, этот метод позволяет за несколько циклов обучения с последующим случайным изменением параметров найти глобальный минимум.

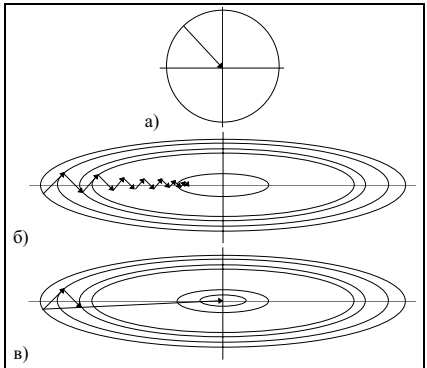


Рис. 6. Траектории спуска при различных конфигурациях окрестности минимума и разных методах оптимизации.

Вторым серьезным недостатком метода наискорейшего спуска является его чувствительность к форме окрестности минимума. На рис. 6а проиллюстрирована траектория спуска при использовании метода наискорейшего спуска, в случае, если в окрестности минимума линии уровня функции оценки являются кругами (рассматривается двумерный случай). В этом случае минимум достигается за один шаг. На рис. 6б приведена траектория метода наискорейшего спуска в случае эллиптических линий уровня. Видно, что в этой ситуации за один шаг минимум достигается только из точек, расположенных на осях эллипсов. Из любой другой точки спуск будет происходить по ломаной, каждое звено которой ортогонально к соседним звеньям, а длина звеньев убывает. Легко показать что для точного достижения минимума потребуется бесконечное число шагов метода градиентного спуска. Этот эффект получил название овражного, а методы оптимизации, позволяющие бороться с этим эффектом – антиовражных.

8.2.3.2 kParTan

Одним из простейших антиовражных методов является метод kParTan. Идея метода состоит в том, чтобы запомнить начальную точку, затем выполнить k шагов оптимизации по методу наискорейшего спуска, затем сделать шаг оптимизации по направлению из начальной точки в конечную. Описание метода приведено на рис 7. На рис 6в приведен один шаг оптимизации по методу 2ParTan. Видно, что после шага вдоль направления из первой точки в третью траектория спуска привела в минимум. К сожалению, это верно только для двумерного случая. В многомерном случае направление kParTan не ведет прямо в точку минимума, но спуск в этом направлении, как правило, приводит в окрестность минимума меньшего радиуса, чем при еще одном шаге метода наискорейшего спуска (см. рис. 6б). Кроме того, следует отметить, что для выполнения третьего шага не потребовалось вычислять градиент, что экономит время при численной оптимизации.

1. Создать_вектор B1
2. Создать_вектор B2
3. Шаг=1
4. Вычислить_оценку O2
5. Сохранить_вектор B1
6. O1=O2
7. N=0
8. Вычислить_градиент
9. Оптимизация_шага Пустой_указатель Шаг
10. N=N+1
11. Если N<k то переход к шагу 8
12. Сохранить_вектор B2
13. B2=B2-B1
14. ШагParTan=1
15. Оптимизация шага B2 ШагParTan
16. Вычислить_оценку O2
17. Если O1-O2<Точность то переход к шагу 5

Рис. 7. Метод kParTan

8.2.3.3 Квазиньютоновские методы

Существует большое семейство квазиньютоновских методов, позволяющих на каждом шаге проводить минимизацию в направлении минимума квадратичной формы. Идея этих методов состоит в том, что функция оценки приближается квадратичной формой. Зная квадратичную форму, можно вычислить ее минимум и проводить оптимизацию шага в направлении этого минимума. Одним из наиболее часто используемых методов из семейства одношаговых квазиньютоновских методов является BFGS метод. Этот метод хорошо зарекомендовал себя при обучении нейронных сетей (см. [29]). Подробно ознакомиться с методом BFGS и другими квазиньютоновскими методами можно в работе [48].

8.3 Стандарт первого уровня компонента учитель

В этом разделе приводится стандарт языка описания компонента учитель. Поскольку часть алгоритмов обучения жестко привязана к архитектуре сети, то в следующем разделе предложен способ опознания «своих» сетей.

8.3.1 Способ опознания сети для методов, привязанных к архитектуре сети

Для опознания типа сети рекомендуется использовать первый параметр сети. Для этого архитектуре сети приписывается уникальный номер, типа Long. Уникальность может поддерживаться, например, за счет использования генератора случайных чисел. Кроме того, при описании параметров сети следует задать отдельный тип параметров для первого параметра и указать минимальную границу равной максимальной и равной номеру архитектуры сети. Также необходимо указать в маске параметров, что этот параметр является необучаемым. Учитель, прежде чем выполнить любую операцию с сетью, читает параметры сети, и проверяет первый параметр сети, интерпретируемый как переменная типа Long, на совпадение с хранимым в учителе номером архитектуры. В случае несовпадения номера в параметрах сети с номером в учителе, учитель генерирует внутреннюю ошибку 601 – несовместимость сети и учителя.

Если учитель работает с сетями любой архитектуры, то процедура опознания архитектуры сети не нужна.

8.3.2 Список стандартных функций

В этом разделе описаны стандартные функции, специфические для компонента учитель. Эти функции соответствуют макросам, использованным в первой части главы. Заголовки функций даны на языке описания учителя.

8.3.2.1 Установить объект обучения (SetInstructionObject)

Заголовок функции:

Function SetInstructionObject (What : Integer; Net : PString) : Logic;

Описание аргументов

What может принимать следующие значения (предопределенные константы, приведенные в табл. 11 главы «Общий стандарт»):

Parameters – для обучения параметров сети;

InSignals – для обучения входных сигналов.

Net – имя нейронной сети, которая будет обучаться.

Возможно обучение одного из двух объектов – параметров сети или входных сигналов. Объект обучения должен быть задан до начала собственно обучения. По умолчанию обучается первая сеть в списке нейронных сетей компонента сеть. При необходимости в качестве объекта обучения может быть задана часть сети (см. раздел «Описание нейронных сетей»). При сохранении учителя в файле сети объект обучения хранится вместе с учителем. Функция возвращает значение истина, если ее выполнение завершено успешно. В противном случае (например, указанная сеть отсутствует в списке сетей компонента сеть) возвращается значение ложь.

8.3.2.2 Создание массива (CreateArray)

Заголовок функции:

Function CreateArray : PRealArray;

Аргументов нет.

Функция возвращает указатель на массив, пригодный для хранения массива обучаемых параметров (входных сигналов) сети. Если массив создать не удалось, то возвращается пустой указатель.

8.3.2.3 Освободить массив (EraseArray)

Заголовок функции:

Function EraseArray(Vec : PRealArray) : Logic;

Описание аргументов

Vec – указатель на массив. При вызове содержит адрес освобождаемого массива.

После выполнения функции в аргументе Vec содержится пустой указатель. В случае невозможности освобождения памяти функция генерирует внутреннюю ошибку 604 – некорректная работа с памятью, передает управление обработчику ошибок, выполнение функции завершается, возвращается значение ложь. В противном случае возвращается значение истина.

8.3.2.4 Случайный массив (RandomArray)

Заголовок функции:

Function RandomArray(Vec : PRealArray) : Logic;

Описание аргументов

Vec – указатель на массив. При входе в макрос содержит адрес существующего массива.

В ходе выполнения функции для каждого элемента массива параметров генерируется случайное значение. Для генерации используется генератор случайных чисел, равномерно распределенных на отрезке от нуля до единицы. После получения случайной величины a она преобразуется по формуле $a' = a(a_{\max} - a_{\min}) - a_{\min}$ к случайной величине, распределенной на отрезке $[a_{\min}, a_{\max}]$. Величины $a_{\min}$ и $a_{\max}$ для параметров сети определяются их типом (см. раздел «Описание элементов»). Для входных сигналов принимается $a_{\min} = -1$, $a_{\max} = 1$. Если обучаемым объектом являются параметры, то генерация случайного массива производится путем генерации запроса RandomDirection компонента сеть. Если при выполнении функции возникла ошибка, то генерируется внутренняя ошибка 605 – ошибка при исполнении внешнего запроса, управление передается обработчику ошибок, функция возвращает значение ложь. В противном случае возвращается значение истина.

8.3.2.5 Модификация массива (Modify)

Заголовок функции:

Function Modify(Direct : PRealArray; OldStep, NewStep : Real) : Logic;

Описание аргументов

Direct – указатель на массив направления модификации сети.

OldStep – вес старого массива параметров в модифицированном.

NewStep – вес массива направления модификации в модифицированном массиве параметров.

Эта функция генерирует запрос на модификацию параметров сети (см. раздел «Провести обучение (Modify)» главы «Описание нейронных сетей»). Вызов запроса имеет вид:

Modify(Net, OldStep, NewStep, Tipe, Direct)

Аргументами запроса являются:

Net – указатель на пустую строку (используется сеть по умолчанию).

OldStep, NewStep – аргументы функции.

Tipe – значение аргумента What в запросе InstructionObject.

Direct – аргумент функции.

Аргумент функции Direct может быть пустым указателем. В этом случае для модификации используется массив градиента, хранящийся вместе с сетью. В случае возникновения ошибки в ходе модификации сети (запрос Modify возвращает значение ложь) генерируется внутренняя ошибка 605 – ошибка при исполнении внешнего запроса, управление передается обработчику ошибок, функция возвращает значение ложь. В противном случае возвращается значение истина.

8.3.2.6 Оптимизация шага (Optimize)

Заголовок функции:

Function Optimize (Direct : PRealArray; Step : Real) : Real;

Описание аргументов

Direct – указатель на массив направления модификации сети.

Step – начальный шаг в направлении Direct.

Действия, выполняемые функцией Optimize, описаны в разделе «Подбор оптимального шага» этой главы. В случае возникновения ошибки при выполнении функции она генерирует внутреннюю ошибку 605 – ошибка при исполнении внешнего запроса, передает управление обработчику ошибок, функция возвращает значение 0. В противном случае возвращается значение оценки при оптимальном

шаге. Следует отметить, что после завершения выполнения функции, параметры сети соответствуют результату выполнения функции Modify(Direct, 1, Step), где Step – значение оптимального шага.

8.3.2.7 Сохранить массив (SaveArray)

Заголовок функции:

Function SaveArray(Vec : PRealArray) : Logic;

Описание аргументов

Vec – указатель на массив.

Функция генерирует запрос nwGetData. После выполнения функции в массиве, на который указывает аргумент Vec, содержится текущий массив параметров. В случае возникновения ошибки в ходе выполнения функции генерируется внутренняя ошибка 605 – ошибка при исполнении внешнего запроса, управление передается обработчику ошибок, функция возвращает значение ложь. В противном случае возвращается значение истина.

8.3.2.8 Установить параметры (SetArray)

Заголовок функции:

Function SetArray(Vec : PRealArray) : Logic;

Описание аргументов

Vec – указатель на массив, содержащий параметры, которые необходимо установить.

Функция генерирует запрос nwSetData.После выполнения функции параметры сети совпадают с параметрами, содержащимися в массиве, на который указывает аргумент Vec. В случае возникновения ошибки в ходе выполнения функции генерируется внутренняя ошибка 605 – ошибка при исполнении внешнего запроса, управление передается обработчику ошибок, функция возвращает значение ложь. В противном случае возвращается значение истина.

8.3.2.9 Вычислить оценку (Estimate)

Заголовок функции:

Function Estimate(Handle : Integer; All : Logic) : Real;

Описание аргументов

Handle – номер сеанса задачника.

All – признак обучения по всему обучающему множеству.

Функция генерирует запрос к исполнителю на вычисление оценки. Если аргумент All содержит значение истина, то обучение производится по всему обучающему множеству, в противном случае – позадачно. В случае возникновения ошибки при выполнении функции он генерирует внутреннюю ошибку 605 – ошибка при исполнении внешнего запроса, передает управление обработчику ошибок, функция возвращает значение 0. В противном случае возвращается значение вычисленной оценки.

8.3.2.10 Вычислить градиент (CalcGradient)

Заголовок функции:

Function CalcGradient(Handle : Integer; All : Logic) : Real;

Описание аргументов

Handle – номер сеанса задачника.

All – признак обучения по всему обучающему множеству.

Функция генерирует запрос к исполнителю на вычисление градиента. Если аргумент All содержит значение истина, то обучение производится по всему обучающему множеству, в противном случае – позадачно. В случае возникновения ошибки при выполнении функции он генерирует внутреннюю ошибку 605 – ошибка при исполнении внешнего запроса, передает управление обработчику ошибок, функция возвращает значение 0. В противном случае возвращается значение вычисленной оценки.

8.3.2.11 Запустить запрос (GenerateQuest)

Заголовок функции:

Function GenerateQuest(Name : PString; Arguments : PRealArray) : Logic

Описание аргументов

Name – указатель на символьную строку, содержащую имя запроса.

Arguments – массив, содержащий адреса аргументов запроса.

Функция генерирует запрос к макрокомпоненту нейрокомпьютер на исполнение запроса, имя которого указано в аргументе Name, с аргументами, адреса которых указаны в аргументе Arguments. Действуют следующие ограничения. В строке, содержащей имя запроса должно содержаться только одно слово – имя запроса. Ведущие и хвостовые пробелы подавляются. В массиве Arguments должно

содержаться ровно столько элементов, сколько аргументов у генерируемого запроса. В массив Arguments всегда складываются адреса аргументов, даже если в запрос данный аргумент передается по значению.

8.3.3 Язык описания учителя

В отличие от таких компонентов как оценка, сеть и интерпретатор ответа, учитель не является составным объектом. Однако учитель может состоять из множества функций, вызывающих друг друга. Собственно учитель – это процедура, управляющая обучением сети. Ключевые слова, специфические для языка описания учителя приведены в табл. 3

Таблица 3.

Ключевые слова специфические для языка описания учителя

Ключевое слово	Краткое описание
1. Main	Начало главной процедуры
2. Instructor	Заголовок описания учителя
3. InstrLib	Заголовок описания библиотеки функций
4. Used	Подключение библиотек функций
5. Init	Начало блока инициации
6. InstrStep	Начало блока одного шага обучения
7. Close	Начало блока завершения обучения

8.3.3.1 Библиотеки функций учителя

Библиотеки функций учителя содержат описание функций, необходимых для работы одного или нескольких учителей. Использование библиотек позволяет избежать дублирования функций в различных учителях. Описание библиотеки функций аналогично описанию учителя, но не содержит главной процедуры.

8.3.3.2 БНФ языка описания учителя

Обозначения, принятые в данном расширении БНФ и описание ряда конструкций приведены в главе «Общий стандарт» в разделе «Описание языка описания компонент».

<Описание библиотеки> ::= <Заголовок библиотеки> <Описание глобальных переменных> <Описание функций> <Конец описания библиотеки>

<Заголовок библиотеки> ::= **InstrLib** <Имя библиотеки> [**Used** <Список имен библиотек>]

<Имя библиотеки> ::= <Идентификатор>

<Список имен библиотек> ::= <Имя используемой библиотеки> [,<Список имен библиотек>]

<Имя используемой библиотеки> ::= <Идентификатор>

<Конец описания библиотеки> ::= **End InstrLib**

<Описание учителя> ::= <Заголовок учителя> <Описание глобальных переменных> <Описание функций> <Главная процедура> <Конец описания учителя>

<Заголовок учителя> ::= **Instructor** <Имя библиотеки> [**Used** <Список имен библиотек>]

<Главная процедура> ::= **Main** <Описание статических переменных> <Описание переменных> <Блок инициации> <Блок шага обучения> <Блок завершения>

<Блок инициации> ::= **Init** <Тело функции>

<Блок шага обучения> ::= **InstrStep** <Выражение типа ***Logic***> <Тело функции>

<Блок завершения> ::= **Close** <Тело функции>

<Конец описания учителя> **End Instructor**

8.3.3.3 Описание языка описания учителя

Язык описания учителя является наиболее простым из всех языков описания компонент. Фактически все синтаксические конструкции этого языка описаны в главе «Общий стандарт». В теле функции, являющемся частью главной процедуры недопустим оператор возврата значения, поскольку главная процедура не является функцией. Три раздела главной функции – блок инициации, блок одного шага обучения и блок завершения являются фрагментами одной процедуры. Выделение этих разделов необходимо для выполнения запроса «Выполнить N шагов обучения». Выполнение главной процедуры происходит следующим образом. Выполняется блок инициации. Выполнение блока одного шага обучения сети производится до тех пор, пока не наступит одно из следующих событий:

1. программа выйдет из блока одного шага обучения сети прямым переходом на метку в другом разделе;
2. нарушится условие, указанное в конструкции InstStep;
3. компонент учитель получит запрос «Прервать обучение сети»;
4. в случае выполнения запроса «Выполнить N шагов обучения» блок одного шага обучения сети выполнен N раз.

   Далее выполняется блок завершения обучения.

8.3.3.4 Пример описания учителя

В данном разделе приведены описания некоторых методов обучения, описанных в разделе «Описание алгоритмов обучения».

Пример 1.

```
Instructor RandomFire;{Метод случайной стрельбы с уменьшением радиуса}
Main                        {Обучение ведется по всему обучающему множеству}
  Label Exit, Exit1;
  Static
    Integer Try Name "Число попыток при одном радиусе" Default 5;
    Real MinRadius Name "Минимальный радиус, при котором продолжается работа" Default 0.001;
    String NetName Name "Имя сети" Default "";
    Integer What Name "Что обучать" Default Parameters;
    Color InstColor Name "Цвет примеров обучающего множества" Default HFFFF; {По умолчанию}
    Integer OperColor Name "Операция для отбора цветов" Default CIn; {все примеры, в цвете ко-}
  Var                                                    {торых есть хоть один единичный бит}
    PRealArray Map, DirectMap;  {Для хранения текущего и случайного массивов параметров}
    Real Est1, Est2;            {Для хранения текущей и случайной оценки}
    Real Radius;                {Текущий радиус}
    Integer TryNum, RadiusNum;  {Число попыток, номер использованного радиуса}
    Integer Handle;             {Номер сеанса задачника}
    String QName;               {Имя запроса}
Init
  Begin
    If Not SetInstructionObject (What, @NetName) Then GoTo Exit;      {Задаем объекты обучения}
    QName = "InitSession";                                  {Задаем имя запроса}
    Map = NewArray(mRealArray, 3);                          {Создаем массив для аргументов запроса}
    If Map = Null Then GoTo Exit;
    TPointer(Map^[1]) = @InstColor;                         {Заносим адрес первого аргумента}
    TPointer(Map^[2]) = @OperColor;                         {Заносим адрес второго аргумента}
    TPointer(Map^[3]) = @Handle;                            {Заносим адрес третьего аргумента}
    If Not GenerateQuest(@QName, Map) Then GoTo Exit;{Открываем сеанс работы с задачником}
    If Not FreeArray(mRealArray, Map) Then GoTo Exit;  {Освобождаем массив для аргументов}
{Собственно начало обучения}
    Map = CreateArray;                                      {Создаем вспомогательные массивы}
    DirectMap= CreateArray;
    If Map = Null Then GoTo Exit;
    If DirectMap= Null Then GoTo Exit;
    Est1 = Estimate(Handle, True);
    If Error <> 0 Then GoTo Exit;
    RadiusNum = 1;                                          {Обрабатываем первый радиус}
    Radius = 1 / RadiusNum;                                 {Вычисляем первый радиус}
    If Not SaveArray(Map) Then GoTo Exit;                   {Сохраняем начальный массив параметров}
  End
InstrStep Radius > MinRadius                {Обработка с одним радиусом – один шаг обучения}
  Begin
    TryNum = 0;
    While TryNum < Try Do Begin
      If Not SetArray(Map) Then GoTo Exit;                  {Устанавливаем лучший массив параметров}
      If Not RandomArray(DirectMap) Then GoTo Exit;  {Генерируется новый массив параметров}
      If Not Modify(DirectMap, 1, Radius) Then GoTo Exit; {Модифицируем массив параметров}
      Est2 = Estimate(Handle, True);
      If Error <> 0 Then GoTo Exit;
      If Est1>Est2 Then Begin
        If Not SaveArray(Map) Then GoTo Exit; {Сохраняем лучший массив параметров}
        Est1 = Est2;
        TryNum = 0;
      End Else TryNum = TryNum + 1;                         {Увеличиваем счетчик отказов}
    End
```

```
        RadiusNum = RadiusNum + 1;                  {Обрабатываем следующий радиус}
        Radius = 1 / RadiusNum;                     {Вычисляем следующий радиус}
    End
Close
    Begin
Exit:
        If Not SetArray(Map) Then;          {Восстанавливаем лучший массив параметров}
        If Not EraseArray(Map1) Then;       {Освобождаем вспомогательные массивы}
        If Not EraseArray(Map2) Then;
        QName = "CloseSession";                         {Задаем имя запроса}
        Map = NewArray(mRealArray, 1);              {Создаем массив для аргументов запроса}
        If Map = Null Then GoTo Exit1;
        TPointer(Map^[1]) = @Handle;                    {Заносим адрес единственного аргумента}
        If Not GenerateQuest(@QName, Map) Then;{Открываем сеанс работы с задачником}
        If Not FreeArray(mRealArray, Map) Then;   {Освобождаем массив для аргументов}
Exit1:
    End
End Instructor
```

Пример 2. Библиотека функций

```
InstrLib Library1;                          {Библиотека содержит функции для следующего учителя}
    Function SDM( Handle : Integer; Step : Real) : Real;        {Метод наискорейшего спуска}
    Label Exit, Endd;
    Var
        Real Est;
    Begin
        Est = CalcGradient(Handle, True);
        If Error <> 0 Then GoTo Exit;
        Est = Optimize(Null, Step);         {Вызываем функцию подбора оптимального шага}
        If Error <> 0 Then GoTo Exit;
        SDM = Est;
        GoTo Endd;
Exit:
        SDM = 0;
Endd:
    End

    Function RDM( Handle : Integer; Step : Real) : Real;        {Метод случайного поиска}
    Label Exit, Endd;
    Var
        Real Est;
        PRealArray : Direction;
    Begin
        Direction = CreateArray;                        {Создаем вспомогательный массив}
        If Direction = Null Then GoTo Exit;
        If Not RandomArray(Direction) Then GoTo Exit;   {Генерируется новый массив параметров}
        If Error <> 0 Then GoTo Exit;
        Est = Optimize(Direction, Step);    {Вызываем функцию подбора оптимального шага}
        If Error <> 0 Then GoTo Exit;
        RDM = Est;
        GoTo Endd;
Exit:
        RDM = 0;
Endd:
    End
End InstrLib
```

Пример 3. Антиовражная процедура обучения.

```
Instructor kParTan Used Library1;       { Антиовражная процедура обучения kParTan }
Main                                    {Обучение ведется по всему обучающему множеству}
    Label Exit, Exit1;
```

```
    Static
        Color InstColor Name "Цвет примеров обучающего множества" Default HFFFF; {По умолчанию}
        Integer OperColor Name "Операция для отбора цветов" Default CIn; {все примеры, в цвете ко-}
        String NetName Name "Имя сети" Default ""; {торых есть хоть один единичный бит}
        Integer What Name "Что обучать" Default Parameters;
        Integer k Name "Число шагов между ParTan шагами" Default 2; {По умолчанию kParTan}
        Real Accuracy Name "Требуемый минимум оценки" Default 0.00001;
        Logic Direction Name "Случайное направление или антиградиент" Default True; {Если истина,}
    Var                                                                    {то антиградиент}
        Integer Handle;                 {Номер сеанса задачника}
        String QName;                   {Имя запроса}
        PRealArray Map1, DirectMap;     {Для текущего массива параметров и ParTan направления}
        Real Step, ParTanStep;          {Длины шагов для оптимизации шага}
        Real Est1, Est2;                {Для хранения текущей и случайной оценки}
        Long I;
Init
    Begin
        If Not SetInstructionObject (What, @NetName) Then GoTo Exit;    {Задаем объекты обучения}
        QName = "InitSession";                                          {Задаем имя запроса}
        Map1 = NewArray(mRealArray, 3);                                 {Создаем массив для аргументов запроса}
        If Map = Null Then GoTo Exit;
        TPointer(Map^[1]) = @InstColor;                                 {Заносим адрес первого аргумента}
        TPointer(Map^[2]) = @OperColor;                                 {Заносим адрес второго аргумента}
        TPointer(Map^[3]) = @Handle;                                    {Заносим адрес третьего аргумента}
        If Not GenerateQuMap(@QName, Map) Then GoTo Exit;{Открываем сеанс работы с задачником}
        If Not FreeArray(mRealArray, Map) Then GoTo Exit;    {Освобождаем массив для аргументов}
{Собственно начало обучения}
        Map = CreateArray;                                  {Создаем вспомогательные массивы}
        DirectMap= CreateArray;
        If Map = Null Then GoTo Exit;
        If DirectMap = Null Then GoTo Exit;
        Est1 = Accuracy*10;             {Задаем оценку, не удовлетворяющую требованию точности}
        Step = 0.005;                       {Задаем начальное значение шагу}
    End
InstrStep Est > Accuracy
    Begin
        If Not SaveArray(Map1) Then GoTo Exit;          {Сохраняем начальный массив параметров}
        For I = 1 To k Do Begin                         {Выполняем k межпартанных шагов}
            If Direct Then Est = SDM(Handle, Step) Else Est = RDM(Handle, Step);
            If Error <> 0 Then GoTo Exit;
        End;
        If Not SaveArray(DirectMap) Then GoTo Exit;     {Сохраняем конечный массив параметров}
        For I = 1 To TLong(Map^[0]) Do
            DirectMap^[I] = DirectMap^[I] - Map^[I];    {Вычисляем направление ParTan шага}
        ParTanStep = 1;                                 {Задаем начальное значение ParTan шагу}
        Est = Optimize(DirectMap, ParTanStep);      {Вызываем функцию подбора оптимального шага}
        If Error <> 0 Then GoTo Exit;
    End
Close
    Begin
Exit:
        If Not EraseArray(Map) Then;        {Освобождаем вспомогательные массивы}
        If Not EraseArray(DirectMap) Then;
        QName = "CloseSession";                                 {Задаем имя запроса}
        Map = NewArray(mRealArray, 1);                  {Создаем массив для аргументов запроса}
        If Map = Null Then GoTo Exit1;
        TPointer(Map^[1]) = @Handle;                            {Заносим адрес единственного аргумента}
        If Not GenerateQuest(@QName, Map) Then;{Открываем сеанс работы с задачником}
        If Not FreeArray(mRealArray, Map) Then;   {Освобождаем массив для аргументов}
```

```
Exit1:
    End
End Instructor
```

8.4 Стандарт второго уровня компонента учитель

Компонент учитель одновременно работает только с одним учителем. Запросы к компоненту учитель можно разбить на следующие группы.

1. Обучение сети.
2. Чтение/запись учителя.
3. Инициация редактора учителя.
4. Работа с параметрами учителя.

8.4.1 Обучение сети

К данной группе относятся три запроса – обучить сеть (InstructNet), провести N шагов обучения (NInstructSteps) и прервать обучение (CloseInstruction).

8.4.1.1 Обучить сеть (InstructNet)

Описание запроса:
Pascal:

```
Function InstructNet : Logic;
```

C:

```
Logic InstructNet()
```

Аргументов нет.

Назначение – производит обучение сети.
Описание исполнения.

1. Если Error <> 0, то выполнение запроса прекращается.
2. Если в момент получения запроса учитель не загружен, то возникает ошибка 601 – неверное имя компонента, управление передается обработчику ошибок, а обработка запроса прекращается.
3. Выполняется главная процедура загруженного учителя.
4. Если во время выполнения запроса возникает ошибка, а значение переменной Error равно нулю, то генерируется внутренняя ошибка 605 – ошибка исполнения учителя, управление передается обработчику ошибок, а обработка запроса прекращается.
5. Если во время выполнения запроса возникает ошибка, а значение переменной Error не равно нулю, то обработка запроса прекращается.

8.4.1.2 Провести N шагов обучения (NInstructSteps)

Описание запроса:
Pascal:

```
Function NInstructNet( N : Integer ) : Logic;
```

C:

```
Logic NInstructNet(Integer N)
```

Описание аргумента:
N – число выполнений блока одного шага обучения сети.

Назначение – производит обучение сети.
Описание исполнения.

1. Если Error <> 0, то выполнение запроса прекращается.
2. Если в момент получения запроса учитель не загружен, то возникает ошибка 601 – неверное имя компонента, управление передается обработчику ошибок, а обработка запроса прекращается.
3. Выполняется блок инициации главной процедуры загруженного учителя, N раз выполняется блок одного шага обучения, выполняется блок завершения обучения.
4. Если во время выполнения запроса возникает ошибка, а значение переменной Error равно нулю, то генерируется внутренняя ошибка 605 – ошибка исполнения учителя, управление передается обработчику ошибок, а обработка запроса прекращается.
5. Если во время выполнения запроса возникает ошибка, а значение переменной Error не равно нулю, то обработка запроса прекращается.

8.4.1.3 Прервать обучение (CloseInstruction)

Описание запроса:
Pascal:

Function CloseInstruction: Logic;

C:

Logic CloseInstruction()

Аргументов нет.

Назначение – прерывает обучение сети.
Описание исполнения.

1. Если Error <> 0, то выполнение запроса прекращается.
2. Если в момент получения запроса учитель не загружен, то возникает ошибка 601 – неверное имя компонента, управление передается обработчику ошибок, а обработка запроса прекращается.
3. Если в момент получения запроса не выполняется ни один из запросов обучить сеть (InstructNet) или провести N шагов обучения (NInstructSteps), то возникает ошибка 606 – неверное использование запроса на прерывание обучения, управление передается обработчику ошибок, а обработка запроса прекращается.
4. Завершается выполнение текущего шага обучения сети.
5. Выполняется блок завершения обучения сети.
6. Если во время выполнения запроса возникает ошибка, а значение переменной Error равно нулю, то генерируется внутренняя ошибка 605 – ошибка исполнения учителя, управление передается обработчику ошибок, а обработка запроса прекращается.
7. Если во время выполнения запроса возникает ошибка, а значение переменной Error не равно нулю, то обработка запроса прекращается.

8.4.2 Чтение/запись учителя

В данном разделе описаны запросы, позволяющие загрузить учителя с диска или из памяти, выгрузить учителя и сохранить текущего учителя на диске или в памяти.

8.4.2.1 Прочитать учителя (inAdd)

Описание запроса:
Pascal:

Function inAdd(CompName : PString) : Logic;

C:

Logic inAdd(PString CompName)

Описание аргумента:
CompName – указатель на строку символов, содержащую имя файла компонента или адрес описания компонента.

Назначение – читает учителя с диска или из памяти.

Описание исполнения.

1. Если в качестве аргумента CompName дана строка, первые четыре символа которой составляют слово File, то остальная часть строки содержит имя компонента и после пробела имя файла, содержащего компоненту. В противном случае считается, что аргумент CompName содержит указатель на область памяти, содержащую описание компонента в формате для записи на диск. Если описание не вмещается в одну область памяти, то допускается включение в текст описания компонента ключевого слова Continue, за которым следует четыре байта, содержащие адрес следующей области памяти.
2. Если в данный момент загружен другой учитель, то выполняется запрос inDelete. Учитель считывается из файла или из памяти.
3. Если считывание завершается по ошибке, то возникает ошибка 602 – ошибка считывания учителя, управление передается обработчику ошибок, а обработка запроса прекращается.

8.4.2.2 Удаление учителя (inDelete)

Описание запроса:
Pascal:

Function inDelete : Logic;

C:

Logic inDelete()

Аргументов нет.

Назначение – удаляет загруженного в память учителя.
Описание исполнения.

1. Если список в момент получения запроса учитель не загружен, то возникает ошибка 601 – неверное имя учителя, управление передается обработчику ошибок, а обработка запроса прекращается.

8.4.2.3 Запись компонента (inWrite)

Описание запроса:
Pascal:

Function inWrite(Var FileName : PString) : Logic;

C:

Logic inWrite(PString* FileName)

Описание аргументов:
CompName – указатель на строку символов, содержащую имя компонента.
FileName – имя файла или адрес памяти, куда надо записать компонент.

Назначение – сохраняет учителя в файле или в памяти.
Описание исполнения.

1. Если в момент получения запроса учитель не загружен, то возникает ошибка 601 – неверное имя компонента, управление передается обработчику ошибок, а обработка запроса прекращается.
2. Если в качестве аргумента FileName дана строка, первые четыре символа которой составляют слово File, то остальная часть строки содержит имя файла для записи компонента. В противном случае FileName должен содержать пустой указатель. В этом случае запрос вернет в нем указатель на область памяти, куда будет помещено описание компонента в формате для записи на диск. Если описание не вмещается в одну область памяти, то в текст будет включено ключевое слово Continue, за которым следует четыре байта, содержащие адрес следующей области памяти.
3. Если во время сохранения компонента возникнет ошибка, то возникает ошибка 603 – ошибка сохранения компонента, управление передается обработчику ошибок, а обработка запроса прекращается.

8.4.3 Инициация редактора учителя

К этой группе запросов относится запрос, который инициирует работу не рассматриваемого в данной работе компонента – редактора учителя.

8.4.3.1 Редактировать компонент (inEdit)

Описание запроса:
Pascal:

Procedure inEdit(CompName : PString);

C:

void inEdit(PString CompName)

Описание аргумента:
CompName – указатель на строку символов – имя файла или адрес памяти, содержащие описание учителя.

Если в качестве аргумента CompName дана строка, первые четыре символа которой составляют слово File, то остальная часть строки содержит имя учителя и после пробела имя файла, содержащего описание учителя. В противном случае считается, что аргумент CompName содержит указатель на область памяти, содержащую описание учителя в формате для записи на диск. Если описание не вмещается в одну область памяти, то допускается включение в текст описания ключевого слова Continue, за которым следует четыре байта, содержащие адрес следующей области памяти.

Если в качестве аргумента CompName передан пустой указатель или указатель на пустую строку, то редактор создает нового учителя.

8.4.4 Работа с параметрами учителя

В данном разделе описаны запросы, позволяющие изменять параметры учителя.

8.4.4.1 Получить параметры (inGetData)

Описание запроса:
Pascal:

```
Function inGetData(Var Param : PRealArray ) : Logic;
```

C:

```
Logic inGetData(PRealArray* Param)
```

Описание аргумента:
Param – адрес массива параметров.

Назначение – возвращает вектор параметров учителя.
Описание исполнения.

1. Если Error <> 0, то выполнение запроса прекращается.
2. Если в момент получения запроса учитель не загружен, то возникает ошибка 601 – неверное имя компонента, управление передается обработчику ошибок, а обработка запроса прекращается.
3. В массив, адрес которого передан в аргументе Param, заносятся значения параметров. Параметры заносятся в массив в порядке описания в разделе описания статических переменных.

8.4.4.2 Получить имена параметров (inGetName)

Описание запроса:
Pascal:

```
Function inGetName(Var Param : PRealArray ) : Logic;
```

C:

```
Logic inGetName(PRealArray* Param)
```

Описание аргумента:
Param – адрес массива указателей на названия параметров.

Назначение – возвращает вектор указателей на названия параметров учителя.
Описание исполнения.

1. Если Error <> 0, то выполнение запроса прекращается.
2. Если в момент получения запроса учитель не загружен, то возникает ошибка 601 – неверное имя компонента, управление передается обработчику ошибок, а обработка запроса прекращается.
3. В массив, адрес которого передан в аргументе Param, заносятся адреса символьных строк, содержащих названия параметров.

8.4.4.3 Установить параметры (inSetData)

Описание запроса:
Pascal:

```
Function inSetData(Param : PRealArray ) : Logic;
```

C:

```
Logic inSetData(PRealArray Param)
```

Описание аргументов:
Param – адрес массива параметров.

Назначение – заменяет значения параметров учителя на значения, переданные, в аргументе Param.
Описание исполнения.

1. Если Error <> 0, то выполнение запроса прекращается.
2. Если в момент получения запроса учитель не загружен, то возникает ошибка 601 – неверное имя компонента, управление передается обработчику ошибок, а обработка запроса прекращается.
3. Параметры, значения которых хранятся в массиве, адрес которого передан в аргументе Param, передаются учителю.

8.4.5 Обработка ошибок

В табл. 4 приведен полный список ошибок, которые могут возникать при выполнении запросов компонентом учитель, и действия стандартного обработчика ошибок.

Таблица 4.

Ошибки компонента учитель и действия стандартного обработчика ошибок.

№На	звание ошибкиСтандартная	обработка
601	Несовместимость сети и учителяЗанесение	номера в Error
602	Ошибка считывания учителяЗанесение	номера в Error
603	Ошибка сохранения учителяЗанесение	номера в Error
604	Некорректная работа с памятьюЗанесение	номера в Error
605	Ошибка исполнения учителяЗанесение	номера в Error
606	Неверное использование запроса на прерывание обученияЗанесение	номера в Error

Контрастер

Компонент контрастер предназначен для контрастирования нейронных сетей. Первые работы, посвященные контрастированию (скелетонизации) нейронных сетей появились в начале девяностых годов [64.]. Однако, задача контрастирования нейронных сетей не являлась центральной, поскольку упрощение сетей может принести реальную пользу только при реализации обученной нейронной сети в виде электронного (оптоэлектронного) устройства. Только в работе А.Н. Горбаня и Е.М. Миркеса «Логически прозрачные нейронные сети» [82] (более полный вариант работы см. [75]), опубликованной в 1995 году задаче контрастирования нейронных сетей был придан самостоятельный смысл – впервые появилась реальная возможность получать новые явные знания из данных. В связи с тем, что контрастирование нейронных сетей не является достаточно развитой ветвью нейроинформатики, стандарт, приведенный в данной главе, является очень общим.

Задачи для контрастера

Из анализа литературы и опыта работы группы НейроКомп можно сформулировать следующие задачи, решаемые с помощью контрастирования нейронных сетей.

1. Упрощение архитектуры нейронной сети.
2. Уменьшение числа входных сигналов.
3. Сведение параметров нейронной сети к небольшому набору выделенных значений.
4. Снижение требований к точности входных сигналов.
5. Получение явных знаний из данных.

Далее в этом разделе все перечисленные выше задачи рассмотрены более подробно.

Упрощение архитектуры нейронной сети

Стремление к упрощению архитектуры нейронных сетей возникло из попытки ответить на следующие вопрос: «Сколько нейронов нужно использовать и как они должны быть связаны друг с другом?» При ответе на этот вопрос существует две противоположные точки зрения. Одна из них утверждает, что чем больше нейронов использовать, тем более надежная сеть получится. Сторонники этой позиции ссылаются на пример человеческого мозга. Действительно, чем больше нейронов, тем больше число связей между ними, и тем более сложные задачи способна решить нейронная сеть. Кроме того, если использовать заведомо большее число нейронов, чем необходимо для решения задачи, то нейронная сеть точно обучится. Если же начинать с небольшого числа нейронов, то сеть может оказаться неспособной обучиться решению задачи, и весь процесс придется повторять сначала с большим числом нейронов. Эта точка зрения (чем больше – тем лучше) популярна среди разработчиков нейросетевого программного обеспечения. Так, многие из них как одно из основных достоинств своих программ называют возможность использования любого числа нейронов.

Вторая точка зрения опирается на такое «эмпирическое» правило: чем больше подгоночных параметров, тем хуже аппроксимация функции в тех областях, где ее значения были заранее неизвестны. С математической точки зрения задачи обучения нейронных сетей сводятся к продолжению функции заданной в конечном числе точек на всю область определения. При таком подходе входные данные сети считаются аргументами функции, а ответ сети – значением функции. На рис. 1 приведен пример аппроксимации табличной функции полиномами 3-й (рис. 1.а) и 7-й (рис. 1.б) степеней. Очевидно, что аппроксимация, полученная с помощью полинома 3-ей степени больше соответствует внутреннему представлению о «правильной» аппроксимации. Несмотря на свою простоту, этот пример достаточно наглядно демонстрирует суть проблемы.

Второй подход определяет нужное число нейронов как минимально необходимое. Основным недостатком является то, что это, минимально необходимое число, заранее неизвестно, а процедура его определения путем постепенного наращивания числа нейронов весьма трудоемка. Опираясь

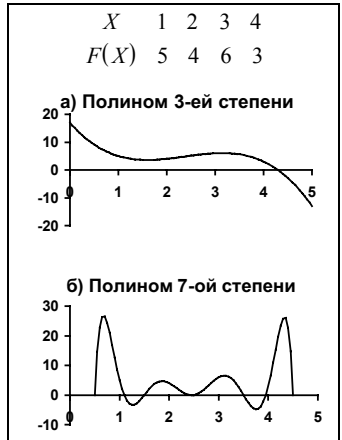


Рис. 1. Аппроксимация табличной функции

на опыт работы группы НейроКомп в области медицинской диагностики [[18, 49 – 52, 72, 90, 91, 160, 161, 165, 182 – 187, 190 – 208, 255, 295 – 298, 316, 317, 341 – 345, 351, 361]], космической навигации и психологии можно отметить, что во всех этих задачах ни разу не потребовалось более нескольких десятков нейронов.

Подводя итог анализу двух крайних позиций, можно сказать следующее: сеть с минимальным числом нейронов должна лучше («правильнее», более гладко) аппроксимировать функцию, но выяснение этого минимального числа нейронов требует больших интеллектуальных затрат и экспериментов по обучению сетей. Если число нейронов избыточно, то можно получить результат с первой попытки, но существует риск построить «плохую» аппроксимацию. Истина, как всегда бывает в таких случаях, лежит посередине: нужно выбирать число нейронов большим, чем необходимо, но не намного. Это можно осуществить путем удвоения числа нейронов в сети после каждой неудачной попытки обучения. Наиболее надежным способом оценки минимального числа нейронов является использование процедуры контрастирования. Кроме того, процедура контрастирования позволяет ответить и на второй вопрос: какова должна быть структура сети.

Уменьшение числа входных сигналов

При постановке задачи для нейронной сети не всегда удается точно определить сколько и каких входных данных нужно подавать на вход. В случае недостатка данных сеть не сможет обучиться решению задачи. Однако гораздо чаще на вход сети подается избыточный набор входных параметров. Например, при обучении сети постановке диагноза в задачах медицинской диагностики на вход сети подаются все данные, необходимые для постановки диагноза в соответствии с существующими методиками. Следует учесть, что стандартные методики постановки диагнозов разрабатываются для использования на большой территории (например, на территории России). Как правило, при диагностике заболеваний населения какого-нибудь небольшого региона (например города) можно обойтись меньшим набором исходных данных. Причем этот усеченный набор будет варьироваться от одного малого региона к другому. Требуется определить, какие данные необходимы для решения конкретной задачи, поставленной для нейронной сети. Кроме того, в ходе решения этой задачи определяются значимости входных сигналов. Следует заметить, что умение определять значимость входных сигналов представляет самостоятельную ценность.

Сведение параметров нейронной сети к выделенным значениям

При обучении нейронных сетей на универсальных компьютерах параметры сети являются действительными числами из заданного диапазона. При аппаратной реализации нейронной сети не всегда возможно реализовать веса связей с высокой точностью (в компьютерном представлении действительных чисел хранятся первые 6-7 цифр мантиссы). Опыт показывает, что в обученной сети веса многих синапсов можно изменять в довольно широком диапазоне (до полуширины интервала изменения веса) не изменяя качество решения сетью поставленной перед ней задачи. Исходя из этого, полезно уметь решать задачу замены значений параметров сети на значения из заданного набора.

Снижение требований к точности входных сигналов

При обработке экспериментальных данных полезно знать, что измерение с высокой точностью, как правило, дороже измерения с низкой точностью. Причем достаточно часто получение очередной значащей цифры измеряемого параметра стоит на несколько порядков дороже. В связи с этим задача снижения требований к точности измерения входных параметров сети приобретает смысл.

Получение явных знаний из данных

Одной из главных загадок мышления является то, как из совокупности данных об объекте, появляется знание о нем. До недавнего времени наибольшим достижением в области искусственного интеллекта являлось либо воспроизведение логики человека-эксперта (классические экспертные системы), либо построение регрессионных зависимостей и определение степени зависимости одних параметров от других.

С другой стороны, одним из основных недостатков нейронных сетей, с точки зрения многих пользователей, является то, что нейронная сеть решает задачу, но не может рассказать как. Иными словами из обученной нейронной сети нельзя извлечь алгоритм решения задачи. Таким образом нейронные сети позволяют получать неявные знания из данных.

В домашнем задании I Всесоюзной олимпиады по нейрокомпьютингу, проходившей в мае 1991 года в городе Омске, в исследовательской задаче участникам было предложено определить, как нейронная сеть решает задачу распознавания пяти первых букв латинского алфавита (полный текст задания и

наиболее интересные варианты решения приведены в [47]). Это была первая попытка извлечения алгоритма решения задачи из обученной нейронной сети.

В 1995 году была сформулирована идея логически прозрачных сетей, то есть сетей на основе структуры которых можно построить вербальное описание алгоритма получения ответа. Это достигается при помощи специальным образом построенной процедуры контрастирования.

Построение логически прозрачных сетей

Зададимся классом сетей, которые будем считать логически прозрачными (то есть такими, которые решают задачу понятным для нас способом, для которого легко сформулировать словесное описания в виде явного алгоритма). Например потребуем, чтобы все нейроны имели не более трех входных сигналов.

Зададимся нейронной сетью у которой все входные сигналы подаются на все нейроны входного слоя, а все нейроны каждого следующего слоя принимают выходные сигналы всех нейронов предыдущего слоя. Обучим сеть безошибочному решению задачи.

После этого будем производить контрастирование в несколько этапов. На первом этапе будем удалять только входные связи нейронов входного слоя. Если после этого у некоторых нейронов осталось больше трех входных сигналов, то увеличим число входных нейронов. Затем аналогичную процедуру выполним поочередно для всех остальных слоев. После завершения описанной процедуры будет получена логически прозрачная сеть. Можно произвести дополнительное контрастирование сети, чтобы получить минимальную сеть. На рис. 2 приведены восемь минимальных сетей. Если под логически прозрачными сетями понимать сети, у которых каждый нейрон имеет не более трех входов, то все сети кроме пятой и седьмой являются логически прозрачными. Пятая и седьмая сети демонстрируют тот факт, что минимальность сети не влечет за собой логической прозрачности.

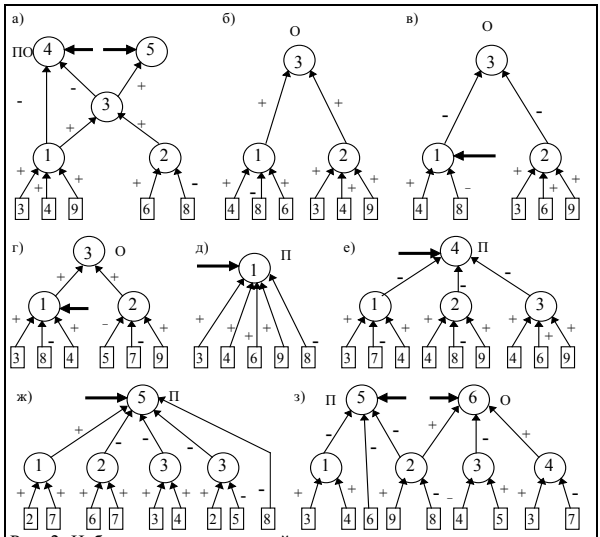


Рис. 2. Набор минимальных сетей для решения задачи о предсказании результатов выборов президента США. В рисунке использованы следующие обозначения: буквы «П» и «О» – обозначают вид ответа, выдаваемый нейроном: «П» – положительный сигнал означает победу правящей партии, а отрицательный – оппозиционной; «О» – положительный сигнал означает победу оппозиционной партии, а отрицательный – правящей;

Получение явных знаний

После получения логически прозрачной нейронной сети наступает этап построения вербального описания. Принцип построения вербального описания достаточно прост. Используемая терминология заимствована из медицины. Входные сигналы будем называть симптомами. Выходные сигналы нейронов первого слоя – синдромами первого уровня. Очевидно, что синдромы первого уровня строятся из симптомов. Выходные сигналы нейронов k-о слоя будем называть синдромами k-о уровня. Синдромы k-о первого уровня строятся из симптомов и синдромов более низких уровней. Синдром последнего уровня является ответом.

В качестве примера приведем интерпретацию алгоритма рассуждений, полученного по второй сети приведенной на рис. 2. Постановка задачи: по ответам на 12 вопросов необходимо предсказать по-

беду правящей или оппозиционной партии на выборах Президента США. Ниже приведен список вопросов.

1. Правящая партия была у власти более одного срока?
2. Правящая партия получила больше 50% голосов на прошлых выборах?
3. В год выборов была активна третья партия?
4. Была серьезная конкуренция при выдвижении от правящей партии?
5. Кандидат от правящей партии был президентом в год выборов?
6. Год выборов был временем спада или депрессии?
7. Был ли рост среднего национального валового продукта на душу населения больше 2.1%?
8. Произвел ли правящий президент существенные изменения в политике?
9. Во время правления были существенные социальные волнения?
10. Администрация правящей партии виновна в серьезной ошибке или скандале?
11. Кандидат от правящей партии – национальный герой?
12. Кандидат от оппозиционной партии – национальный герой?

Ответы на вопросы описывают ситуацию на момент, предшествующий выборам. Ответы кодировались следующим образом: «да» – единица, «нет» – минус единица. Отрицательный сигнал на выходе сети интерпретируется как предсказание победы правящей партии. В противном случае, ответом считается победа оппозиционной партии. Все нейроны реализовывали пороговую функцию, равную 1, если алгебраическая сумма входных сигналов нейрона больше либо равна 0, и -1 при сумме меньшей 0.

Проведем поэтапно построение вербального описания второй сети, приведенной на рис. 2. После автоматического построения вербального описания получим текст, приведенный на рис. 3. Заменим все симптомы на тексты соответствующих вопросов. Заменим формулировку восьмого вопроса на обратную. Подставим вместо Синдром1_Уровня2 название ответа сети при выходном сигнале 1. Текст, полученный в результате этих преобразований приведен на рис. 4.

Заметим, что все три вопроса, ответы на которые формируют Синдром1_Уровня1, относятся к оценке качества правления действующего президента. Поскольку положительный ответ на любой из этих вопросов характеризует недостатки правления, то этот синдром можно назвать синдромом плохой политики. Аналогично, три вопроса, ответы на которые формируют Синдром2_Уровня1, относятся к характеристике политической стабильности. Этот синдром назовем синдромом политической нестабильности.

Тот факт, что оба синдрома первого уровня принимают значение 1, если истинны ответы хотя бы на два из трех вопросов, позволяет избавиться от математических действий с ответами на вопросы. Окончательный ответ может быть истин-

Синдром1_Уровня1 равен 1, если выражение Симптом4 + Симптом6 – Симптом 8 больше либо равно нулю, и –1 – в противном случае.

Синдром2_Уровня1 равен 1, если выражение Симптом3 + Симптом4 + Симптом9 больше либо равно нулю, и –1 – в противном случае.

Синдром1_Уровня2 равен 1, если выражение Синдром1_Уровня1 + Синдром2_Уровня1 больше либо равно нулю, и –1 – в противном случае.

Рис. 3. Автоматически построенное вербальное описание

Синдром1_Уровня1 равен 1, если выражение «Была серьезная конкуренция при выдвижении от правящей партии?» + «Год выборов был временем спада или депрессии?» + «Правящий президент не произвел существенных изменений в политике?» больше либо равно нулю, и –1 – в противном случае.

Синдром2_Уровня1 равен 1, если выражение «В год выборов была активна третья партия?» + «Была серьезная конкуренция при выдвижении от правящей партии?» + «Во время правления были существенные социальные волнения?» больше либо равно нулю, и –1 – в противном случае.

Оппозиционная партия победит, если выражение Синдром1_Уровня1 + Синдром2_Уровня1 больше либо равно нулю.

Рис. 4. Вербальное описание после элементарных преобразований

Правление плохое, если верны хотя бы два из следующих высказываний: «Была серьезная конкуренция при выдвижении от правящей партии», «Год выборов был временем спада или депрессии», «Правящий президент не произвел существенных изменений в политике».

Ситуация политически нестабильна, если верны хотя бы два из следующих высказываний: «В год выборов была активна третья партия», «Была серьезная конкуренция при выдвижении от правящей партии», «Во время правления были существенные социальные волнения».

Оппозиционная партия победит, если правление плохое или ситуация политически нестабильна.

Рис. 5. Окончательный вариант вербального описания

ным только если оба синдрома имеют значение –1.

Используя приведенные соображения, получаем окончательный текст решения задачи о предсказании результатов выборов президента США, приведенный на рис. 5.

Таким образом, использовав идею логически прозрачных нейронных сетей и минимальные интеллектуальные затраты на этапе доводки вербального описания, был получен текст решения задачи. Причем процедура получения логически прозрачных нейронных сетей сама отобрала значимые признаки, сама привела сеть к нужному виду. Далее элементарная программа построила по структуре сети вербальное описание.

На рис. 2 приведены структуры шести логически прозрачных нейронных сетей, решающих задачу о предсказании результатов выборов президента США [299 – 301]. Все сети, приведенные на этом рисунке минимальны в том смысле, что из них нельзя удалить ни одной связи так, чтобы сеть могла обучиться правильно решать задачу. По числу нейронов минимальна пятая сеть.

Заметим, что все попытки авторов обучить нейронные сети со структурами, изображенными на рис. 2, и случайно сгенерированными начальными весами связей закончились провалом. Все сети, приведенные на рис. 2, были получены из существенно больших сетей с помощью процедуры контрастирования. Сети 1, 2, 3 и 4 были получены из трехслойных сетей с десятью нейронами во входном и скрытом слоях. Сети 5, 6, 7 и 8 были получены из двухслойных сетей с десятью нейронами во входном слое. Легко заметить, что в сетях 2, 3, 4 и 5 изменилось не только число нейронов в слоях, но и число слоев. Кроме того, почти все веса связей во всех восьми сетях равны либо 1, либо -1.

Процедура контрастирования

Существует два типа процедуры контрастирования – контрастирование по значимости параметров и не ухудшающее контрастирование. В данном разделе описаны оба типа процедуры контрастирования.

Контрастирование на основе показателей значимости

С помощью этой процедуры можно контрастировать, как входные сигналы, так и параметры сети. Далее в данном разделе будем предполагать, что контрастируются параметры сети. При контрастировании входных сигналов процедура остается той же, но вместо показателей значимости параметров сети используются показатели значимости входных сигналов. Обозначим через χ_p – показатель значимости p-о параметра; через w_p^0 – текущее значение p-о параметра; через w_p^* – ближайшее выделенное значение для p-о параметра.

Используя введенные обозначения процедуру контрастирования можно записать следующим образом:

1. Вычисляем показатели значимости.
2. Находим минимальный среди показателей значимости – χ_{p^*}.
3. Заменим соответствующий этому показателю значимости параметр $w_{p^*}^0$ на $w_{p^*}^*$, и исключаем его из процедуры обучения.
4. Предъявим сети все примеры обучающего множества. Если сеть не допустила ни одной ошибки, то переходим ко второму шагу процедуры.
5. Пытаемся обучить полученную сеть. Если сеть обучилась безошибочному решению задачи, то переходим к первому шагу процедуры, в противном случае переходим к шестому шагу.
6. Восстанавливаем сеть в состояние до последнего выполнения третьего шага. Если в ходе выполнения шагов со второго по пятый был отконтрастирован хотя бы один параметр, (число обучаемых параметров изменилось), то переходим к первому шагу. Если ни один параметр не был отконтрастирован, то получена минимальная сеть.

Возможно использование различных обобщений этой процедуры. Например, контрастировать за один шаг процедуры не один параметр, а заданное пользователем число параметров. Наиболее радикальная процедура состоит в контрастировании половины параметров связей. Если контрастирование половины параметров не удается, то пытаемся контрастировать четверть и т.д. Другие варианты обобщения процедуры контрастирования будут описаны при описании решения задач. Результаты первых работ по контрастированию нейронных сетей с помощью описанной процедуры опубликованы в [47, 302, 303].

Контрастирование без ухудшения

Пусть нам дана только обученная нейронная сеть и обучающее множество. Допустим, что вид функции оценки и процедура обучения нейронной сети неизвестны. В этом случае так же возможно контрастирование сети. Предположим, что данная сеть идеально решает задачу. В этом случае возможно контрастирование сети даже при отсутствии обучающей выборки, поскольку ее можно сгенерировать используя сеть для получения ответов. Задача не ухудшающего контрастирования ставится следующим образом: необходимо так провести контрастирование параметров, чтобы выходные сигналы сети при решении всех примеров изменились не более чем на заданную величину. Для решения задача редуцируется на отдельный адаптивный сумматор: необходимо так изменить параметры, чтобы выходной сигнал адаптивного сумматора при решении каждого примера изменился не более чем на заданную величину.

Обозначим через x_p^q p-й входной сигнал сумматора при решении q-о примера; через f^q – выходной сигнал сумматора при решении q-о примера; через w_p – вес p-о входного сигнала сумматора; через ε – требуемую точность; через n – число входных сигналов сумматора; через m – число примеров. Очевидно, что при решении примера выполняется равенство $f^q = \sum_{p=1}^{n} w_p x_p^q$. Требуется найти такой набор индексов $I = \{i_1,\ldots,i_k\}$, что $\left\| f - \sum_{p\in I} \alpha_p x_p \right\| < \varepsilon$, где α_p – новый вес p-о входного сигнала сумматора. Набор индексов будем строить по следующему алгоритму.

1. Положим $f^{(0)} = f$, $x_p^* = x_p$, $I^{(0)} = \varnothing$, $J^{(0)} = \{1,\ldots,n\}$, k=0.
2. Для всех векторов x_p^* таких, что $p \in J^{(k)}$, проделаем следующее преобразование: если $\|x_p^*\| << \varepsilon$, то исключаем p из множества обрабатываемых векторов – $J^{(k)} = J^{(k)} / \{p\}$, в противном случае нормируем вектор x_p^* на единичную длину – $x_p^{(k)} = x_p^* / \|x_p^*\|$.
3. Если $\|f^{(k)}\| < \varepsilon$ или $J^{(0)} = \varnothing$, то переходим к шагу 10.
4. Находим i_{k+1} – номер вектора, наиболее близкого к $f^{(k)}$ из условия
$$\left|\left(f^{(k)}, x_{i_{k+1}}^{(k)}\right)\right| = \min_{p\in J^{(k)}} \left|\left(f^{(k)}, x_p^{(k)}\right)\right|.$$
5. Исключаем i_{k+1} из множества индексов обрабатываемых векторов: $J^{(k+1)} = J^{(k)} / \{i_{k+1}\}$.
6. Добавляем i_{k+1} в множество индексов найденных векторов: $I^{(k+1)} = I^{(k)} \bigcup \{i_{k+1}\}$
7. Вычисляем не аппроксимированную часть (ошибку аппроксимации) вектора выходных сигналов: $f^{(k+1)} = f^{(k)} - \left(f^{(k)}, x_{i_{k+1}}^{(k)}\right) x_{i_{k+1}}^{(k)}$
8. Преобразуем обрабатываемые вектора к промежуточному представлению – ортогонализуем их к вектору $x_{i_{k+1}}^{(k)}$, для чего каждый вектор $x_p^{(k)}$, у которого $p \in J^{(k)}$ преобразуем по следующей формуле: $x_p^* = x_p^{(k)} - \left(x_p^{(k)}, x_{i_{k+1}}^{(k)}\right) x_{i_{k+1}}^{(k)}$.
9. Увеличиваем k на единицу и переходим к шагу 2.
10. Если $k = 0$, то весь сумматор удаляется из сети и работа алгоритма завершается.
11. Если $k = n+1$, то контрастирование невозможно и сумматор остается неизменным.

12. В противном случае полагаем $I = I^{(k)}$ и вычисляем новые веса связей α_p ($p \in I$) решая систему уравнений $f - f^{(k)} = \sum_{p \in I} \alpha_p x_p$.
13. Удаляем из сети связи с номерами $p \in J$, веса оставшихся связей полагаем равными α_p ($p \in I$).

Данная процедура позволяет производить контрастирование адаптивных сумматоров. Причем значения, вычисляемые каждым сумматором после контрастирования, отличаются от исходных не более чем на заданную величину. Однако, исходно была задана только максимально допустимая погрешность работы сети в целом. Способы получения допустимых погрешностей для отдельных сумматоров исходя из заданной допустимой погрешности для всей сети описаны в ряде работ [94 –96, 167, 209 – 213, 352].

Гибридная процедура контрастирования

Можно упростить процедуру контрастирования, описанную в разд. «Контрастирование без ухудшения». Предлагаемая процедура годится только для контрастирования весов связей адаптивного сумматора (см. разд. «Составные элементы»). Контрастирование весов связей производится отдельно для каждого сумматора. Адаптивный сумматор суммирует входные сигналы нейрона, умноженные на соответствующие веса связей. Для работы нейрона наименее значимым будем считать тот вес, который при решении примера даст наименьший вклад в сумму. Обозначим через x_p^q входные сигналы рассматриваемого адаптивного сумматора при решении q-го примера. Показателем значимости веса назовем следующую величину: $\chi_p^q = \left| \left(w_p - w_p^* \right) \cdot x_p^q \right|$. Усредненный по всем примерам обучающего множества показатель значимости имеет вид $\chi_p = \left| \left(w_p - w_p^* \right) \right| \cdot \max_q \left| x_p^q \right|$. Производим контрастирование по процедуре, приведенной в разд. «Контрастирование на основе показателей значимости». В самой процедуре контрастирования есть только одно отличие – вместо проверки на наличие ошибок при предъявлении всех примеров проверяется, что новые выходные сигналы сети отличаются от первоначальных не более чем на заданную величину.

Контрастирование при обучении

Существует еще один способ контрастирования нейронных сетей. Идея этого способа состоит в том, что функция оценки модернизируется таким способом, чтобы для снижения оценки было выгодно привести сеть к заданному виду. Рассмотрим решение задачи приведения параметров сети к выделенным значениям. Используя обозначения из предыдущих разделов требуемую добавку к функции оценки, являющуюся штрафом за отклонение значения параметра от ближайшего выделенного значения:, можно записать в виде $\sum_p \left(w_p - w_p^* \right)^2$.

Для решения других задач вид добавок к функции оценки много сложнее.

Определение показателей значимости

В данном разделе описан способ определения показателей значимости параметров и сигналов. . Далее будем говорить об определении значимости параметров. Показатели значимости сигналов сети определяются по тем же формулам с заменой параметров на сигналы.

Определение показателей значимости через градиент

Нейронная сеть двойственного функционирования может вычислять градиент функции оценки по входным сигналам и обучаемым параметрам сети

Показателем значимости параметра при решении q-о примера будем называть величину, которая показывает насколько изменится значение функции оценки решения сетью q-о примера если текущее значение параметра w_p заменить на выделенное значение w_p^*. Точно эту величину можно определить произведя замену и вычислив оценку сети. Однако учитывая большое число параметров сети вычисление

показателей значимости для всех параметров будет занимать много времени. Для ускорения процедуры оценки параметров значимости вместо точных значений используют различные оценки [33, 64, 90]. Рассмотрим простейшую и наиболее используемую линейную оценку показателей значимости. Разложим функцию оценки в ряд Тейлора с точностью до членов первого порядка: $H_q(w^*) = H_q^0 + \sum_p \frac{\partial H_q}{\partial w_p}(w_p - w_p^*)$,где H_q^0 – значение функции оценки решения q-о примера при $w^* = w$. Таким образом показатель значимости p-о параметра при решении q-о примера определяется по следующей формуле:

$$\chi_p^q = \left| \frac{\partial H_q}{\partial w_p}\left(w_p - w_p^*\right) \right| \tag{1}$$

Показатель значимости (1) может вычисляться для различных объектов. Наиболее часто его вычисляют для обучаемых параметров сети. Однако показатель значимости вида (1) применим и для сигналов. Как уже отмечалось в главе «Описание нейронных сетей» сеть при обратном функционировании всегда вычисляет два вектора градиента – градиент функции оценки по обучаемым параметрам сети и по всем сигналам сети. Если показатель значимости вычисляется для выявления наименее значимого нейрона, то следует вычислять показатель значимости выходного сигнала нейрона. Аналогично, в задаче определения наименее значимого входного сигнала нужно вычислять значимость этого сигнала, а не сумму значимостей весов связей, на которые этот сигнал подается.

Усреднение по обучающему множеству

Показатель значимости параметра χ_p^q зависит от точки в пространстве параметров, в которой он вычислен и от примера из обучающего множества. Существует два принципиально разных подхода для получения показателя значимости параметра, не зависящего от примера. При первом подходе считается, что в обучающей выборке заключена полная информация о всех возможных примерах. В этом случае, под показателем значимости понимают величину, которая показывает насколько изменится значение функции оценки по обучающему множеству, если текущее значение параметра w_p заменить на выделенное значение w_p^*. Эта величина вычисляется по следующей формуле:

$$\chi_p = \left| \frac{\partial H_{OM}}{\partial w_p}\left(w_p - w_p^*\right) \right|. \tag{2}$$

В рамках другого подхода обучающее множество рассматривают как случайную выборку в пространстве входных параметров. В этом случае показателем значимости по всему обучающему множеству будет служить результат некоторого усреднения по обучающей выборке.

Существует множество способов усреднения. Рассмотрим два из них. Если в результате усреднения показатель значимости должен давать среднюю значимость, то такой показатель вычисляется по следующей формуле:

$$\chi_p = \frac{1}{m}\sum_{q=1}^{m} \chi_p^q . \tag{3}$$

Если в результате усреднения показатель значимости должен давать величину, которую не превосходят показатели значимости по отдельным примерам (значимость этого параметра по отдельному примеру не больше чем χ_p), то такой показатель вычисляется по следующей формуле:

$$\chi_p = \max_q \chi_p^q . \tag{4}$$

Показатель значимости (4) хорошо зарекомендовал себя при использовании в работах группы НейроКомп.

Накопление показателей значимости

Все показатели значимости зависят от точки в пространстве параметров сети, в которой они вычислены, и могут сильно изменяться при переходе от одной точки к другой. Для показателей значимости,

вычисленных с использованием градиента эта зависимость еще сильнее, поскольку при обучении по методу наискорейшего спуска (см. раздел «Метод наискорейшего спуска») в двух соседних точках пространства параметров, в которых вычислялся градиент, градиенты ортогональны. Для снятия зависимости от точки пространства используются показатели значимости, вычисленные в нескольких точках. Далее они усредняются по формулам аналогичным (3) и (4). Вопрос о выборе точек в пространстве параметров в которых вычислять показатели значимости обычно решается просто. В ходе нескольких шагов обучения по любому из градиентных методов при каждом вычислении градиента вычисляются и показатели значимости. Число шагов обучения, в ходе которых накапливаются показатели значимости, должно быть не слишком большим, поскольку при большом числе шагов обучения первые вычисленные показатели значимости теряют смысл, особенно при использовании усреднения по формуле (4).

Стандарт первого уровня компонента контрастер

В этом разделе приводится стандарт языка описания компонента контрастер. Компонент контрастер во многом подобен компоненту учитель. Так в языке описания компонента контрастер допускается использование функций, описанных в разделе «Список стандартных функций» главы «Учитель».

Язык описания контрастера

В отличие от таких компонент как оценка, сеть и интерпретатор ответа, контрастер не является составным объектом. Однако, контрастер может состоять из множества функций, вызывающих друг друга. Собственно контрастер – это процедура, управляющая процессом контрастирования. Ключевые слова, специфические для языка описания контрастера приведены в табл. 3

Библиотеки функций контрастера

Библиотеки функций контрастера содержат описание функций, необходимых для работы одного или нескольких контрастеров. Использование библиотек позволяет избежать дублирования функций в различных контрастерах. Описание библиотеки функций аналогично описанию контрастера, но не содержит главной процедуры.

Таблица 3.

Ключевые слова специфические для языка описания контрастера

Ключевое слово	Краткое описание
1. Main	Начало главной процедуры
2. Contrast	Заголовок описания контрастера
3. ContrLib	Заголовок описания библиотеки функций
4. Used	Подключение библиотек функций
5. ContrastFunc	Глобальная переменная типа функция.

БНФ языка описания контрастера

Обозначения, принятые в данном расширении БНФ и описание ряда конструкций приведены в главе «Общий стандарт» в разделе «Описание языка описания компонент».

<Описание библиотеки> ::= <Заголовок библиотеки> <Описание глобальных переменных> <Описание функций> <Конец описания библиотеки>

<Заголовок библиотеки> ::= **ContrLib** <Имя библиотеки> [**Used** <Список имен библиотек>]

<Имя библиотеки> ::= <Идентификатор>

<Список имен библиотек> ::= <Имя используемой библиотеки> [,<Список имен библиотек>]

<Имя используемой библиотеки> ::= <Идентификатор>

<Конец описания библиотеки> ::= **End ContrLib**

<Описание контрастера> ::= <Заголовок контрастера> <Описание глобальных переменных> <Описание функций> <Главная процедура> <Конец описания контрастера>

<Заголовок контрастера> ::= **Contrast** <Имя библиотеки> [**Used** <Список имен библиотек>]

<Главная процедура> ::= **Main** <Описание статических переменных> <Описание переменных> <Тело функции>

<Конец описания контрастера> **End Contrast**

Описание языка описания контрастера

Язык описания контрастера является наиболее простым из всех языков описания компонент. Фактически все синтаксические конструкции этого языка описаны в главе «Общий стандарт». В теле функции, являющемся частью главной процедуры недопустим оператор возврата значения, поскольку главная процедура не является функцией.

Контрастер имеет одну глобальную предопределенную переменную ContrastFunc. Эта переменная должна обязательно быть определена – ей нужно присвоить адрес функции, которая будет вызывать-

ся каждый раз после того, как нейронная сеть вычислит градиент после решения одного примера. Функция, адрес которой присваивается переменной ContrastFunc должна быть объявлена следующим образом:

Function MyContrast(TheEnd : Logic) : Logic;

Значения аргумента TheEnd имеют следующий смысл: истина – обучение ведется позадачно или закончен просмотр обучающего множества; ложь – обработан еще один пример обучающего множества при обучении по всему задачнику в целом. Следует учесть, что при обучении по всему обучающему множеству в целом, нейронная сеть накапливает градиенты всех примеров, так что при первом вызове функции в сети хранится градиент функции оценки по результатам решения первого примера; при втором – результатам решения первых двух примеров и т.д. Функция возвращает значение ложь, если в ходе ее работы произошла ошибка. В противном случае она возвращает значение истина.

Значение переменной ContrastFunc присваивается оператором присваивания:

ContrastFunc = MyContrast

Если значение переменной ContrastFunc не задано, то она указывает на используемую по умолчанию функцию EmptyContrast, которая просто возвращает значение истина.

Стандарт второго уровня компонента контрастер

Компонента контрастер одновременно работает только с одним контрастером. Запросы к компоненте контрастер можно разбить на следующие группы.

1. Контрастирование сети.
2. Чтение/запись контрастера.
3. Инициация редактора контрастера.
4. Работа с параметрами контрастера.

Контрастирование сети

К данной группе относятся три запроса – контрастировать сеть (ContrastNet), прервать контрастирование (CloseContrast) и контрастировать пример (ContrastExample).

Контрастировать сеть(ContrastNet)

Описание запроса:
Pascal:

Function ContrastNet: Logic;

C:

Logic ContrastNet()

Аргументов нет.

Назначение – производит контрастирование сети.
Описание исполнения.

1. Если Error <> 0, то выполнение запроса прекращается.
2. Если в момент получения запроса контрастер не загружен, то возникает ошибка 701 – неверное имя компонента, управление передается обработчику ошибок, а обработка запроса прекращается.
3. Выполняется главная процедура загруженного контрастера.
4. Если во время выполнения запроса возникает ошибка, а значение переменной Error равно нулю, то генерируется внутренняя ошибка 705 – ошибка исполнения контрастера, управление передается обработчику ошибок, а обработка запроса прекращается.
5. Если во время выполнения запроса возникает ошибка, а значение переменной Error не равно нулю, то обработка запроса прекращается.

Прервать контрастирование (CloseContrast)

Описание запроса:
Pascal:

Function CloseContrast: Logic;

C:

Logic CloseContrast()

Аргументов нет.

Назначение – прерывает контрастирование сети.
Описание исполнения.

1. Если Error <> 0, то выполнение запроса прекращается.
2. Если в момент получения запроса контрастер не загружен, то возникает ошибка 701 – неверное имя компонента, управление передается обработчику ошибок, а обработка запроса прекращается.
3. Если в момент получения запроса не выполняется запрос ContrastNet, то возникает ошибка 706 – неверное использование запроса на прерывание контрастирования, управление передается обработчику ошибок, а обработка запроса прекращается.
4. Завершается выполнение текущего шага контрастирования сети.
5. Если во время выполнения запроса возникает ошибка, а значение переменной Error равно нулю, то генерируется внутренняя ошибка 705 – ошибка исполнения контрастера, управление передается обработчику ошибок, а обработка запроса прекращается.
6. Если во время выполнения запроса возникает ошибка, а значение переменной Error не равно нулю, то обработка запроса прекращается.

Контрастировать пример (ContrastExample)

Описание запроса:
Pascal:

```
Function ContrastExample( TheEnd : Logic) : Logic;
```

C:

```
Logic ContrastExample(Logic TheEnd )
```

Описание аргумента:
TheEnd – значение аргумента имеет следующий смысл: ложь – обработан еще один пример обучающего множества при обучении по всему задачнику в целом.

Назначение – извлекает из сети необходимые для вычисления показателей значимости параметры.

Описание исполнения.

1. Если Error <> 0, то выполнение запроса прекращается.
2. Вызывает функцию, адрес которой хранится в переменной ContrastFunc, передавая ей аргумент TheEnd в качестве аргумента.
3. Если функция ContrastFunc возвращает значение ложь, а значение переменной Error равно нулю, то генерируется внутренняя ошибка 705 – ошибка исполнения контрастера, управление передается обработчику ошибок, а обработка запроса прекращается.
4. Если функция ContrastFunc возвращает значение ложь, а значение переменной Error не равно нулю, то обработка запроса прекращается.
5. Запрос в качестве результата возвращает возвращенное функцией ContrastFunc значение.

Чтение/запись контрастера

В данном разделе описаны запросы позволяющие, загрузить контрастер с диска или из памяти, выгрузить контрастера и сохранить текущего контрастера на диске или в памяти.

Прочитать контрастера (cnAdd)

Описание запроса:
Pascal:

```
Function cnAdd( CompName : PString ) : Logic;
```

C:

```
Logic cnAdd(PString CompName)
```

Описание аргумента:
CompName – указатель на строку символов, содержащую имя файла компонента или адрес описания компонента.

Назначение – читает контрастера с диска или из памяти.

Описание исполнения.

1. Если в качестве аргумента CompName дана строка, первые четыре символа которой составляют слово File, то остальная часть строки содержит имя компонента и после пробела имя файла, содержащего компонент. В противном случае считается, что аргумент CompName содержит указатель на область памяти, содержащую описание компонента в формате для записи на диск. Если описание не вмещается в одну область памяти, то допускается включение в текст описания ком-

понента ключевого слова Continue, за которым следует четыре байта, содержащие адрес следующей области памяти.

2. Если в данный момент загружен другой контрастер, то выполняется запрос cnDelete. Контрастер считывается из файла или из памяти.
3. Если считывание завершается по ошибке, то возникает ошибка 702 – ошибка считывания контрастера, управление передается обработчику ошибок, а обработка запроса прекращается.

Удаление контрастера (cnDelete)

Описание запроса:
Pascal:

```
Function cnDelete : Logic;
```

C:

```
Logic cnDelete()
```

Аргументов нет.

Назначение – удаляет загруженного в память контрастера.
Описание исполнения.

1. Если список в момент получения запроса контрастер не загружен, то возникает ошибка 701 – неверное имя контрастера, управление передается обработчику ошибок, а обработка запроса прекращается.

Запись контрастера (cnWrite)

Описание запроса:
Pascal:

```
Function cnWrite(Var FileName : PString) : Logic;
```

C:

```
Logic cnWrite(PString* FileName)
```

Описание аргументов:
CompName – указатель на строку символов, содержащую имя контрастера.
FileName – имя файла или адрес памяти, куда надо записать контрастера.

Назначение – сохраняет контрастера в файле или в памяти.
Описание исполнения.

1. Если в момент получения запроса контрастер не загружен, то возникает ошибка 701 – неверное имя контрастера, управление передается обработчику ошибок, а обработка запроса прекращается.
2. Если в качестве аргумента FileName дана строка, первые четыре символа которой составляют слово File, то остальная часть строки содержит имя файла, для записи компонента. В противном случае FileName должен содержать пустой указатель. В этом случае запрос вернет в нем указатель на область памяти, куда будет помещено описание компонента в формате для записи на диск. Если описание не вмещается в одну область памяти, то в текст будет включено ключевое слово Continue, за которым следует четыре байта, содержащие адрес следующей области памяти.
3. Если во время сохранения компонента возникнет ошибки, то возникает ошибка 703 – ошибка сохранения контрастера, управление передается обработчику ошибок, а обработка запроса прекращается.

Инициация редактора контрастера

К этой группе запросов относится запрос, который инициирует работу не рассматриваемого в данной работе компонента – редактора контрастера.

Редактировать контрастера (cnEdit)

Описание запроса:
Pascal:

```
Procedure cnEdit(CompName : PString);
```

C:

```
void cnEdit(PString CompName)
```

Описание аргумента:

CompName – указатель на строку символов – имя файла или адрес памяти, содержащие описание контрастера.

Если в качестве аргумента CompName дана строка, первые четыре символа которой составляют слово File, то остальная часть строки содержит имя контрастера и после пробела имя файла, содержащего описание контрастера. В противном случае считается, что аргумент CompName содержит указатель на область памяти, содержащую описание контрастера в формате для записи на диск. Если описание не вмещается в одну область памяти, то допускается включение в текст описания ключевого слова Continue, за которым следует четыре байта, содержащие адрес следующей области памяти.

Если в качестве аргумента CompName передан пустой указатель или указатель на пустую строку, то редактор создает нового контрастера.

Работа с параметрами контрастера

В данном разделе описаны запросы, позволяющие изменять параметры контрастера.

Получить параметры (cnGetData)

Описание запроса:
Pascal:

```
Function cnGetData(Var Param : PRealArray ) : Logic;
```

C:

```
Logic cnGetData(PRealArray* Param)
```

Описание аргумента:
Param – адрес массива параметров.

Назначение – возвращает вектор параметров контрастера.
Описание исполнения.

1. Если Error <> 0, то выполнение запроса прекращается.
2. Если в момент получения запроса контрастер не загружен, то возникает ошибка 701 – неверное имя компонента, управление передается обработчику ошибок, а обработка запроса прекращается.
3. В массив, адрес которого передан в аргументе Param, заносятся значения параметров. Параметры заносятся в массив в порядке описания в разделе описания статических переменных.

Получить имена параметров (cnGetName)

Описание запроса:
Pascal:

```
Function cnGetName(Var Param : PRealArray ) : Logic;
```

C:

```
Logic cnGetName(PRealArray* Param)
```

Описание аргумента:
Param – адрес массива указателей на названия параметров.

Назначение – возвращает вектор указателей на названия параметров контрастера.
Описание исполнения.

1. Если Error <> 0, то выполнение запроса прекращается.
2. Если в момент получения запроса контрастер не загружен, то возникает ошибка 701 – неверное имя компонента, управление передается обработчику ошибок, а обработка запроса прекращается.
3. В массив, адрес которого передан в аргументе Param, заносятся адреса символьных строк, содержащих названия параметров.

Установить параметры (cnSetData)

Описание запроса:
Pascal:

```
Function cnSetData(Param : PRealArray ) : Logic;
```

C:

```
Logic cnSetData(PRealArray Param)
```

Описание аргументов:
Param – адрес массива параметров.

Назначение – заменяет значения параметров контрастера на значения, переданные, в аргументе Param.

Описание исполнения.

1. Если Error <> 0, то выполнение запроса прекращается.
2. Если в момент получения запроса контрастер не загружен, то возникает ошибка 701 – неверное имя компонента, управление передается обработчику ошибок, а обработка запроса прекращается.
3. Параметры, значения которых хранятся в массиве, адрес которого передан в аргументе Param, передаются контрастеру.

Обработка ошибок

В табл. 4 приведен полный список ошибок, которые могут возникать при выполнении запросов компонентом контрастер, и действия стандартного обработчика ошибок.

Таблица 4.

Ошибки компонента контрастер и действия стандартного обработчика ошибок.

№Название ошибкиСтандартная обработка		
701	Несовместимость сети и контрастераЗанесение	номера в Error
702	Ошибка считывания контрастераЗанесение	номера в Error
703	Ошибка сохранения контрастераЗанесение	номера в Error
704	Некорректная работа с памятьюЗанесение	номера в Error
705	Ошибка исполнения контрастераЗанесение	номера в Error
706	Неверное использование запроса на прерывание контрастированияЗанесение	номера в Error

Список литературы[1]

1. Айвазян С.А., Бежаева З.И., Староверов О.В. Классификация многомерных наблюдений.- М.: Статистика, 1974.- 240 с.
2. Айзерман М.А., Браверман Э.М., Розоноэр Л.И. Метод потенциальных функций в теории обучения машин. М.: Наука, 1970.- 383 с.
3. Анастази А. Психологическое тестирование. М. Педагогика, 1982. Книга 1. 320 с.; Книга 2. 360 с.
4. Андерсон Т. Введение в многомерный статистический анализ.- М.: Физматгиз, 1963. 500 с.
5. Андерсон Т. Статистический анализ временных рядов. М.: Мир, 1976. 755 с.
6. Ануфриев А.Ф. Психодиагностика как деятельность и научная дисциплина // Вопросы психологии, 1994, № 2. С. 123-130.
7. Аркадьев А.Г., Браверман Э.М. Обучение машины классификации объектов.- М.: Наука, 1971.- 172 с.
8. Ахапкин Ю.К., Всеволдов Н.И., Барцев С.И. и др. Биотехника - новое направление компьютеризации. Серия "Теоретическая и прикладная биофизика", М: изд. ВИНИИТИ, 1990. 144 с.
9. Барцев С.И. Некоторые свойства адаптивных сетей. Препринт ИФ СО АН СССР, Красноярск, 1987, №71Б, 17 с.
10. Барцев С.И., Гилев С.Е., Охонин В.А. Принцип двойственности в организации адаптивных систем обработки информации// Динамика химических и биологических систем, Новосибирск, Наука, 1989, с.6-55.
11. Барцев С.И., Ланкин Ю.П. Моделирование аналоговых адаптивных сетей. Препринт Института биофизики СО РАН, Красноярск, 1993, №203Б, 36 с.
12. Барцев С.И., Ланкин Ю.П. Сравнительные свойства адаптивных сетей с полярными и неполярными синапсами. Препринт Института биофизики СО РАН, Красноярск, 1993, №196Б, 26 с.
13. Барцев С.И., Машихина Н.Ю., Суров С.В. Нейронные сети: подходы к аппаратной реализации. Препринт ИФ СО АН СССР, Красноярск, 1990, №122Б, 14 с.
14. Барцев С.И., Охонин В.А. Адаптивные сети обработки информации. Препринт ИФ СО АН СССР, Красноярск, 1986, №59Б, 20 с.
15. Барцев С.И., Охонин В.А. Адаптивные сети, функционирующие в непрерывном времени // Эволюционное моделирование и кинетика, Новосибирск, Наука, 1992, с.24-30.
16. Беркинблит М.Б., Гельфанд И.М., Фельдман А.Г. Двигательные задачи и работа параллельных программ // Интеллектуальные процессы и их моделирование. Организация движения.- М.: Наука, 1991.- С. 37-54.
17. Боннер Р.Е. Некоторые методы классификации // Автоматический анализ изображений.- М.: Мир, 1969.- С. 205-234.
18. Борисов А.В., Гилев С.Е., Головенкин С.Е., Горбань А.Н., Догадин С.А., Коченов Д.А., Масленникова Е.В., Матюшин Г.В., Миркес Е.М., Ноздрачев К.Г., Россиев Д.А., Савченко А.А., Шульман В.А. Нейроимитатор "MultiNeuron" и его применения в медицине. // Математическое обеспечение и архитектура ЭВМ: Материалы научно-технической конференции "Проблемы техники и технологий XXI века", 22-25 марта 1994 г. / КГТУ. Красноярск, 1994. С. 14-18.
19. Браверман Э.М., Мучник И.Б. Структурные методы обработки эмпирических данных. - М.: Наука. Главная редакция физико-математической литературы. 1983. - 464 с.
20. Букатова И.Л. Эволюционное моделирование и его приложения.- М.: Наука, 1979.- 231 с.
21. Бурлачук Л.Ф., Коржова Е.Ю. К построению теории "измеренной индивидуальности" в психодиагностике // Вопросы психологии 1994, N5. С. 5-11.
22. Вавилов Е.И., Егоров Б.М., Ланцев В.С., Тоценко В.Г. Синтез схем на пороговых элементах. - М.: Сов. радио, 1970.
23. Вапник В.Н., Червоненкис А.Ф. Теория распознавания образов. - М.: Наука, 1974.
24. Веденов А.А. Моделирование элементов мышления. - М.: Наука, 1988.

[1] Списоклитературы заведомо неполон. В него включены классические работы, монографии и статьи обзорного характера, а также публикации, непосредственно использованные в книге. Кроме того, в список включены все сколько-нибудь существенные публикации членов красноярской группы НейроКомп, вышедшие к моменту сдачи книги в печать.

25. Гаврилова Т.А., Червинская К.Р., Яшин А.М. Формирование поля знаний на примере психодиагностики // Изв. АН СССР. Техн. кибернетика, 1988, № 5.- С. 72-85.
26. Галушкин А.И. Синтез многослойных схем распознавания образов. - М.: Энергия, 1974.
27. Галушкин А.И., Фомин Ю.И. Нейронные сети как линейные последовательные машины. - М.: Изд-во МАИ, 1991.
28. Гельфанд И.М., Цетлин М.Л. О математическом моделировании механизмов центральной нервной системы // Модели структурно-функциональной организации некоторых биологических систем.- М.: Наука, 1966.- С. 9-26.
29. Гилев С.Е. "Сравнение методов обучения нейронных сетей", // Тезисы докладов III Всеросийского семинара "Нейроинформатика и ее приложения", Красноярск: Изд-во КГТУ, сс. 80-81.
30. Гилев С.Е. "Сравнение характеристических функций нейронов", // Тезисы докладов III Всеросийского семинара "Нейроинформатика и ее приложения", Красноярск: Изд-во КГТУ, 1995, с. 82
31. Гилев С.Е. Forth-propagation - метод вычисления градиентов оценки // Тезисы докладов II Всероссийского рабочего семинара "Нейроинформатика и ее приложения" (Красноярск, 7-10 октября 1994 г.) / Красноярск: Изд-во КГТУ, 1994, с. 36-37.
32. Гилев С.Е. Автореф. дисс. канд.физ.-мат. наук // Красноярск, КГТУ, 1997.
33. Гилев С.Е. Алгоритм сокращения нейронных сетей, основанный на разностной оценке вторых производных целевой функции // Нейроинформатика и ее приложения. Тезисы докладов 5 Всероссийского семинара, 3-5 октября 1997 г. / Под ред. А.Н.Горбаня. Красноярск: изд. КГТУ, 1997. С. 45-46.
34. Гилев С.Е. Гибрид сети двойственности и линейной сети// Нейроинформатика и нейрокомпьютеры/ тезисы докладов рабочего семинара 8-11 октября 1993 г., Красноярск/ Институт биофизики СО РАН, 1993. С. 25.
35. Гилев С.Е. Метод получения градиентов оценки по подстроечным параметрам без использования back propagation // Нейроинформатика и ее приложения: Материалы III Всероссийского семинара, 6-8 октября 1995 г. Ч. 1/Под ред. А.Н.Горбаня; Красноярск: Изд-во КГТУ, 1995. С. 91-100.
36. Гилев С.Е. Нейросеть с квадратичными сумматорами// Нейроинформатика и нейрокомпьютеры/ тезисы докладов рабочего семинара 8-11 октября 1993 г., Красноярск/ Институт биофизики СО РАН, 1993. С. 11-12.
37. Гилев С.Е., Горбань А.Н. “Плотность полугрупп непрерывных функций”.- 4 Всероссийский рабочий семинар “Нейроинформатика и ее приложения”, 5-7 октября 1996 г., Тезисы докладов. Красноярск: изд. КГТУ, 1996, с. 7-9.
38. Гилев С.Е., Горбань А.Н. О полноте класса функций, вычислимых нейронными сетями. Второй Сибирский конгресс по Прикладной и Индустриальной Математике, посвященный памяти А.А.Ляпунова (1911-1973), А.П.Ершова (1931-1988) и И.А.Полетаева (1915-1983). Новосибирск, июнь 1996. Тезисы докладов, часть 1. Изд. Института математики СО РАН. С. 6.
39. Гилев С.Е., Горбань А.Н., Миркес Е.М. Малые эксперты и внутренние конфликты в обучаемых нейронных сетях // Доклады Академии Наук СССР.- 1991.- Т.320, N.1.- С.220-223.
40. Гилев С.Е., Горбань А.Н., Миркес Е.М., Коченов Д.А., Россиев Д.А. "Определение значимости обучающих параметров для принятия нейронной сетью решения об ответе"// Нейроинформатика и нейрокомпьютеры/ тезисы докладов рабочего семинара 8-11 октября 1993 г., Красноярск/ Институт биофизики СО РАН, 1993. С. 8.
41. Гилев С.Е., Горбань А.Н., Миркес Е.М., Коченов Д.А., Россиев Д.А. "Нейросетевая программа MultiNeuron"// Нейроинформатика и нейрокомпьютеры/ тезисы докладов рабочего семинара 8-11 октября 1993 г., Красноярск/ Институт биофизики СО РАН, 1993. С. 9.
42. Гилев С.Е., Горбань А.Н., Миркес Е.М., Новоходько А.Ю. "Пакет программ имитации различных нейронных сетей"// Нейроинформатика и нейрокомпьютеры/ тезисы докладов рабочего семинара 8-11 октября 1993 г., Красноярск/ Институт биофизики СО РАН, 1993. С. 7.
43. Гилев С.Е., Коченов Д.А., Миркес Е.М., Россиев Д.А. Контрастирование, оценка значимости параметров, оптимизация их значений и их интерпретация в нейронных сетях // Нейроинформатика и ее приложения: Материалы III Всероссийского семинара, 6-8 октября 1995 г. Ч. 1/Под ред. А.Н.Горбаня; Красноярск: Изд-во КГТУ, 1995. С. 66-78.

44. Гилев С.Е., Миркес Е.М. Обучение нейронных сетей // Эволюционное моделирование и кинетика. Новосибирск: Наука, 1992. С. 9-23.
45. Гилев С.Е., Миркес Е.М., Новоходько А.Ю., Царегородцев В.Г., Чертыков П.В. "Проект языка описания нейросетевых автоматов" // Тезисы докладов II Всероссийского рабочего семинара "Нейроинформатика и ее приложения" (Красноярск, 7-10 октября 1994 г.) / Красноярск: Изд-во КГТУ, 1994, с. 35.
46. Гилева Л.В., Гилев С.Е., Горбань А.Н. Нейросетевой бинарный классификатор "CLAB" (описание пакета программ). Красноярск: Ин-т биофизики СО РАН, 1992. 25 с. Препринт № 194 Б.
47. Гилева Л.В., Гилев С.Е., Горбань А.Н., Гордиенко П.В., Еремин Д.И., Коченов Д.А., Миркес Е.М., Россиев Д.А., Умнов Н.А. Нейропрограммы. Учебное пособие: В 2 ч. // Красноярск, Красноярский государственный технический университет, 1994.260 с.
48. Гилл Ф., Мюррей У., Райт М. Практическая оптимизация. М.: Мир,1985. 509 с.
49. Головенкин С.Е., Горбань А.Н., Шульман В.А., Россиев Д.А., Назаров Б.Н., Мосина В.А., Зинченко О.П., Миркес Е.М., Матюшин Г.В., Бугаенко Н.Н. База данных для апробации систем распознавания и прогноза: осложнения инфаркта миокарда // Нейроинформатика и ее приложения. Тезисы докладов 5 Всероссийского семинара, 3-5 октября 1997 г. / Под ред. А.Н.Горбаня. Красноярск: изд. КГТУ, 1997. С. 47.
50. Головенкин С.Е., Назаров Б.В., Матюшин Г.В., Россиев Д.А., Шевченко В.Ф., Зинченко О.П., Токарева И.М. Прогнозирование возникновения мерцательной аритмии в острый и подострый периоды инфаркта миокарда с помощью компьютерных нейронных сетей // Актуальные проблемы реабилитации больных с сердечно-сосудистыми заболеваниями. Тез. докл. симпозиума 18-20 мая 1994 г., "Красноярское Загорье", Красноярск.1994.- С.28.
51. Головенкин С.Е., Россиев Д.А., Назаров Б.В., Шульман В.А., Матюшин Г.В., Зинченко О.П. Прогнозирование возникновения фибрилляции предсердий как осложнения инфаркта миокарда с помощью нейронных сетей // Диагностика, информатика и метрология - 94.- Тез. научно-технической конференции (г. Санкт-Петербург, 28-30 июня 1994 г.).С.-Петербург.- 1994.- С.349.
52. Головенкин С.Е., Россиев Д.А., Шульман В.А., Матюшин Г.В., Шевченко В.Ф. Прогнозирование сердечной недостаточности у больных со сложными нарушениями сердечного ритма с помощью нейронных сетей // Диагностика, информатика и метрология - 94.- Тез. научно-технической конференции (г. Санкт-Петербург, 28-30 июня 1994 г.).- С.-Петербург.- 1994.С.350-351.
53. Голубь Д.Н., Горбань А.Н. Многочастичные сетчатки для ассоциативной памяти. Второй Сибирский конгресс по Прикладной и Индустриальной Математике, посвященный памяти А.А.Ляпунова (1911-1973), А.П.Ершова (1931-1988) и И.А.Полетаева (1915-1983). Новосибирск, июнь 1996. Тезисы докладов, часть 3. Изд. Института математики СО РАН. С. 271.
54. Горбань А.Н, НейроКомп или 9 лет нейрокомпьютерных исследований в Красноярске // Актуальные проблемы информатики, прикладной математики и механики, ч. 3, Новосибирск - Красноярск: из-во СО РАН, 1996. - с. 13 - 37.
55. Горбань А.Н. Алгоритмы и программы быстрого обучения нейронных сетей. // Эволюционное моделирование и кинетика. Новосибирск: Наука, 1992. С.36-39.
56. Горбань А.Н. Быстрое дифференцирование сложных функций и обратное распространение ошибки // Нейроинформатика и ее приложения. Тезисы докладов 5 Всероссийского семинара, 3-5 октября 1997 г. / Под ред. А.Н.Горбаня. Красноярск: изд. КГТУ, 1997. С. 54-56.
57. Горбань А.Н. Быстрое дифференцирование, двойственность и обратное распространение ошибки / Нейроинформатика Новосибирск: Наука, Сибирская издательская фирма РАН, 1998.
58. Горбань А.Н. Возможности нейронных сетей / Нейроинформатика Новосибирск: Наука, Сибирская издательская фирма РАН, 1998.
59. Горбань А.Н. Двойственность в сетях автоматов // Нейроинформатика и ее приложения: Материалы III Всероссийского семинара, 6-8 октября 1995 г. Ч. 1/Под ред. А.Н.Горбаня; Красноярск: Изд-во КГТУ, 1995. С. 32-66.
60. Горбань А.Н. Мы предлагаем для контроля качества использовать нейрокомпьютеры // Стандарты и качество, 1994, № 10, с. 52.

61. Горбань А.Н. Нейрокомп // Нейроинформатика и ее приложения: Материалы III Всероссийского семинара, 6-8 октября 1995 г. Ч. 1/Под ред. А.Н.Горбаня; Красноярск: Изд-во КГТУ, 1995. С. 3-31.
62. Горбань А.Н. Нейрокомпьютер, или Аналоговый ренессанс. Мир ПК, 1994. № 10. С. 126-130.
63. Горбань А.Н. Обобщение аппроксимационной теоремы Стоуна // Нейроинформатика и ее приложения. Тезисы докладов 5 Всероссийского семинара, 3-5 октября 1997 г. / Под ред. А.Н.Горбаня. Красноярск: изд. КГТУ, 1997. С. 59-62.
64. Горбань А.Н. Обучение нейронных сетей. М.: изд. СССР-США СП "ParaGraph", 1990. 160 с. (English Translation: AMSE Transaction, Scientific Siberian, A, 1993. Vol. 6. Neurocomputing. PP. 1-134).
65. Горбань А.Н. Проблема скрытых параметров и задачи транспонированной регрессии // Нейроинформатика и ее приложения. Тезисы докладов 5 Всероссийского семинара, 3-5 октября 1997 г. / Под ред. А.Н.Горбаня. Красноярск: изд. КГТУ, 1997. С. 57-58.
66. Горбань А.Н. Проекционные сетчатки для обработки бинарных изображений. // Математическое обеспечение и архитектура ЭВМ / Материалы научно-технической конференции "Проблемы техники и технологий XXI века", 22-25 марта 1994 г., Красноярск: изд. КГТУ, 1994. С. 50-54.
67. Горбань А.Н. Размытые эталоны в обучении нейронных сетей // Тезисы докладов II Всероссийского рабочего семинара "Нейроинформатика и ее приложения" (Красноярск, 7-10 октября 1994 г.) / Красноярск: Изд-во КГТУ, 1994, с. 6-9.
68. Горбань А.Н. Решение задач нейронными сетями / Нейроинформатика Новосибирск: Наука, Сибирская издательская фирма РАН, 1998.
69. Горбань А.Н. Системы с наследованием и эффекты отбора. // Эволюционное моделирование и кинетика. Новосибирск: Наука, 1992. С. 40-71.
70. Горбань А.Н. Точное представление многочленов от нескольких переменных с помощью линейных функций, операции суперпозиции и произвольного нелинейного многочлена от одного переменного // Нейроинформатика и ее приложения. Тезисы докладов 5 Всероссийского семинара, 3-5 октября 1997 г. / Под ред. А.Н.Горбаня. Красноярск: изд. КГТУ, 1997. С. 63-65.
71. Горбань А.Н. Этот дивный новый компьютерный мир. Заметки о нейрокомпьютерах и новой технической революции // Математическое обеспечение и архитектура ЭВМ / Материалы научно-технической конференции "Проблемы техники и технологий XXI века", 22-25 марта 1994 г., Красноярск: изд. КГТУ, 1994. С.42-49.
72. Горбань А.Н., Дружинина Н.В., Россиев Д.А. “Нейросетевая интерпретация спектрофотометрического способа исследования содержания меланина в ресницах и подсчет значимости обучающих параметров неросети с целью диагностики увеальных меланом”.- 4 Всероссийский рабочий семинар “Нейроинформатика и ее приложения”, 5-7 октября 1996 г., Тезисы докладов. Красноярск: изд. КГТУ, 1996, с. 94.
73. Горбань А.Н., Кошур В.Д. Нейросетевые модели и методы решения задач динамики сплошных сред и физики взаимодействующих частиц // 10 Зимняя школа по механике сплошных сред (тезисы докладов) / Екатеринбург: УрО РАН, 1995. С. 75-77.
74. Горбань А.Н., Миркес Е.М. "Компоненты нейропрограмм", тезисы докладов III Всеросийского семинара "Нейроинформатика и ее приложения", Красноярск: Изд-во КГТУ, с. 17
75. Горбань А.Н., Миркес Е.М. Логически прозрачные нейронные сети для производства знаний из данных. Вычислительный центр СО РАН в г. Красноярске. Красноярск, 1997. 12 с., библиогр. 12 назв. (Рукопись деп. в ВИНИТИ 17.07.97, № 2434-В97)
76. Горбань А.Н., Миркес Е.М. Нейронные сети ассоциативной памяти, функционирующие в дискретном времени. Вычислительный центр СО РАН в г. Красноярске. Красноярск, 1997. 23 с., библиогр. 8 назв. (Рукопись деп. в ВИНИТИ 17.07.97, № 2436-В97)
77. Горбань А.Н., Миркес Е.М. “Информационная емкость тензорных сетей”.- 4 Всероссийский рабочий семинар “Нейроинформатика и ее приложения”, 5-7 октября 1996 г., Тезисы докладов. Красноярск: изд. КГТУ, 1996, с. 22-23.
78. Горбань А.Н., Миркес Е.М. “Помехоустойчивость тензорных сетей”.- 4 Всероссийский рабочий семинар “Нейроинформатика и ее приложения”, 5-7 октября 1996 г., Тезисы докладов. Красноярск: изд. КГТУ, 1996, с. 24-25.

79. Горбань А.Н., Миркес Е.М. “Тензорные сети ассоциативной памяти”.- 4 Всероссийский рабочий семинар “Нейроинформатика и ее приложения”, 5-7 октября 1996 г., Тезисы докладов. Красноярск: изд. КГТУ, 1996, с. 20-21.
80. Горбань А.Н., Миркес Е.М. Кодирование качественных признаков для нейросетей // Тезисы докладов II Всероссийского рабочего семинара "Нейроинформатика и ее приложения" (Красноярск, 7-10 октября 1994 г.) / Красноярск: Изд-во КГТУ, 1994, с. 29.
81. Горбань А.Н., Миркес Е.М. Контрастирование нейронных сетей, тезисы докладов III Всеросийского семинара "Нейроинформатика и ее приложения", Красноярск: Изд-во КГТУ, сс. 78-79
82. Горбань А.Н., Миркес Е.М. Логически прозрачные нейронные сети, Нейроинформатика и ее приложения: тезисы докладов III Всеросийского семинара , Красноярск: Изд-во КГТУ, с. 32
83. Горбань А.Н., Миркес Е.М. Нейросетевое распознавание визуальных образов "EYE" (описание пакета программ) Красноярск: Ин-т биофизики СО РАН, 1992. 36 с. Препринт № 193 Б.
84. Горбань А.Н., Миркес Е.М. Оценки и интерпретаторы ответа для сетей двойственного функционирования. Вычислительный центр СО РАН в г. Красноярске. Красноярск, 1997. 24 с., библиогр. 8 назв. (Рукопись деп. в ВИНИТИ 25.07.97, № 2511-В97)
85. Горбань А.Н., Миркес Е.М. Функциональные компоненты нейрокомпьютера // Нейроинформатика и ее приложения: Материалы III Всероссийского семинара, 6-8 октября 1995 г. Ч. 1/Под ред. А.Н.Горбаня; Красноярск: Изд-во КГТУ, 1995. С. 79-90.
86. Горбань А.Н., Миркес Е.М., Свитин А.П. Метод мультиплетных покрытий и его использование для предсказания свойств атомов и молекул. - Журнал физ. химии, 1992. Т. 66, № 6. С. 1503-1510.
87. Горбань А.Н., Миркес Е.М., Свитин А.П. Полуэмпирический метод классификации атомов и интерполяции их свойств. // Математическое моделирование в химии и биологии. Новые подходы. Новосибирск : Наука, 1992. С. 204-220.
88. Горбань А.Н., Новоходько А.Ю. Нейронные сети в задаче транспонированной регрессии. Второй Сибирский конгресс по Прикладной и Индустриальной Математике, посвященный памяти А.А.Ляпунова (1911-1973), А.П.Ершова (1931-1988) и И.А.Полетаева (1915-1983). Новосибирск, июнь 1996. Тезисы докладов, часть 2. Изд. Института математики СО РАН. С. 160-161.
89. Горбань А.Н., Новоходько А.Ю., Царегородцев В.Г. “Нейросетевая реализация транспонированной задачи линейной регрессии” .- 4 Всероссийский рабочий семинар “Нейроинформатика и ее приложения”, 5-7 октября 1996 г., Тезисы докладов. Красноярск: изд. КГТУ, 1996, с. 37-39.
90. Горбань А.Н., Россиев Д.А. Нейронные сети на персональном компьютере // Новосибирск: Наука, 1996.- 276 с.
91. Горбань А.Н., Россиев Д.А., Бутакова Е.В., Гилев С.Е., Головенкин С.Е., Догадин С.А., Коченов Д.А., Масленикова Е.В., Матюшин Г.В., Миркес Е.М., Назаров Б.В., Ноздрачев К.Г., Савченко А.А., Смирнова С.В., Чертыков П.А., Шульман В.А. Медицинские и физиологические применения нейроимитатора "MultiNeuron" // Нейроинформатика и ее приложения: Материалы III Всероссийского семинара, 6-8 октября 1995 г. Ч. 1/Под ред. А.Н.Горбаня; Красноярск: Изд-во КГТУ, 1995. С. 101-113.
92. Горбань А.Н., Россиев Д.А., Головенкин С.Е., Шульман В.А., Матюшин Г.В. Нейросистема прогнозирования осложнений инфаркта миокарда. Второй Сибирский конгресс по Прикладной и Индустриальной Математике, посвященный памяти А.А.Ляпунова (1911-1973), А.П.Ершова (1931-1988) и И.А.Полетаева (1915-1983). Новосибирск, июнь 1996. Тезисы докладов, часть 1. Изд. Института математики СО РАН. С. 40.
93. Горбань А.Н., Россиев Д.А., Коченов Д.А. Применение самообучающихся нейросетевых программ. Раздел 1. Введение в нейропрограммы: Учебно-методическое пособие для студентов специальностей 22.04 и 55.28.00 всех форм обучения. Красноярск: Изд-во СТИ, 1994. 24 с.
94. Горбань А.Н., Сенашова М.Ю. Метод обратного распространения точности. Препринт ВЦ СО РАН, Красноярск, 1996, №17, 8 с.
95. Горбань А.Н., Сенашова М.Ю. Погрешности в нейронных сетях. Вычислительный центр СО РАН в г. Красноярске. Красноярск, 1997. 38 с., библиогр. 8 назв. (Рукопись деп. в ВИНИТИ 25.07.97, № 2509-В97)

96. Горбань А.Н., Сенашова М.Ю. Погрешности в нейронных сетях. Рукопись деп. в ВИНИТИ, 25.07.97, №2509-В97, 1997, 38 с.
97. Горбань А.Н., Фриденберг В.И. Новая игрушка человечества. МИР ПК, 1993, № 9. С. 111-113.
98. Горбань А.Н., Хлебопрос Р.Г. Демон Дарвина. Идея оптимальности и естественный отбор. М.: Наука, 1988. 208 с.
99. Гордиенко П.В. Стратегии контрастирования // Нейроинформатика и ее приложения. Тезисы докладов 5 Всероссийского семинара, 3-5 октября 1997 г. / Под ред. А.Н.Горбаня. Красноярск: изд. КГТУ, 1997. С. 69.
100. Грановская Р.М., Березная И.Я. Интуиция и искусственный интеллект.- Л.: ЛГУ, 1991.- 272 с.
101. Гутчин И.Б., Кузичев А.С. Бионика и надежность. М.: Наука, 1967.
102. Демиденко Е.З. Линейная и нелинейная регрессия.- М.: Финансы и статистика, 1981.- 302 с.
103. Деннис Дж. мл., Шнабель Р. Численные методы безусловной оптимизации и решения нелинейных уравнений. М.: Мир, 1988. 440 с.
104. Дертоузос М. Пороговая логика. - М.: Мир, 1967.
105. Джордж Ф. Мозг как вычислительная машина. М. Изд-во иностр. лит., 1963. 528 с.
106. Дианкова Е.В., Квичанский А.В., Мухамадиев Р.Ф., Мухамадиева Т.А., Терехов С.А. Некоторые свойства нелинейных нейронных сетей. Исследование трехнейронной модели// Тезисы докладов III Всеросийского семинара "Нейроинформатика и ее приложения", Красноярск: Изд-во КГТУ, 1995, с. 86
107. Дискусия о нейрокомпьютерах / Под ред. В.И.Крюкова. Пущино, 1988.
108. Дорофеюк А.А. Алгоритмы автоматической классификации (обзор). - Автоматика и телемеханика, 1971, № 12. С. 78-113.
109. Доррер М.Г., Горбань А.Н., Копытов А.Г., Зенкин В.И. Психологическая интуиция нейронных сетей // Нейроинформатика и ее приложения: Материалы III Всероссийского семинара, 6-8 октября 1995 г. Ч. 1/Под ред. А.Н.Горбаня; Красноярск: Изд-во КГТУ, 1995. С. 114-127.
110. Дуда Р., Харт П. Распознавание образов и анализ сцен.М.: Мир, 1976.- 512 с.
111. Дунин-Барковский В.Л. Информационные процессы в нейронных структурах. - М.: Наука, 1978.
112. Дунин-Барковский В.Л. Нейрокибернетика, нейроинформатика, нейрокомпьютеры / Нейроинформатика Новосибирск: Наука, Сибирская издательская фирма РАН, 1998.
113. Дунин-Барковский В.Л. Нейронные схемы ассоциативной памяти // Моделирование возбудимых структур. Пущино: изд.ЦБИ, 1975. С.90-141.
114. Дюк В.А. Компьютерная психодиагностика.Санкт-Петербург: Братство, 1994.- 364 с.
115. Ермаков С.В., Мышов К.Д., Охонин В.А. К вопросу о математическом моделировании базового принципа мышления человека. Препринт ИБФ СО РАН, Красноярск, 1992, №173Б, 36 с.
116. Журавлев Ю.И. Об алгебраическом подходе к решению задач распознавания и классификации // Проблемы кибернетики.- М.: Наука, 1978, вып. 33.- С. 5-68.
117. Загоруйко Н.Г. Методы распознавания и их применение.М.: Сов. радио, 1972.- 206 с.
118. Загоруйко Н.Г., Елкина В.Н., Лбов Г.С. Алгоритмы обнаружения эмпирических закономерностей.- Новосибирск: Наука, 1985.- 110 с.
119. Захарова Л.Б., Полонская М.Г., Савченко А.А. и др. Оценка антропологического напряжения пришлого населения промышленной зоны Заполярья (биологический аспект) // Препринт ИБФ СО РАН. Красноярск, 1989. № 110Б. 52 с.
120. Захарова Л.М., Киселева Н.Е. Мучник И.Б., Петровский А.М., Сверчинская Р.Б. Анализ развития гипертонической болезни по эмпирическим данным. - Автоматика и телемеханика, 1977, № 9. С. 114-122.
121. Ивахненко А.Г. "Персептроны". - Киев: Наукова думка, 1974.
122. Ивахненко А.Г. Самообучающиеся системы распознавания и автоматического регулирования.- Киев: Техника, 1969.- 392 с.
123. Искусственный интеллект: В 3-х кн. Кн. 1. Системы общения и экспертные системы: Справочник / под ред. Э.В.Попова.- М.: Радио и связь, 1990.- 464 с.
124. Искусственный интеллект: В 3-х кн. Кн. 2. Модели и методы: Справочник / под ред. Д.А. Поспелова.- М.: Радио и связь, 1990.- 304 с.
125. Итоги науки и техники. Сер. "Физ. и Матем. модели нейронных сетей" /Под ред. А.А.Веденова. - М.: Изд-во ВИНИТИ, 1990-92 - Т. 1-5.

126. Квичанский А.В., Дианкова Е.В., Мухамадиев Р.Ф., Мухамадиева Т.А. Программный продукт NNN для исследования свойств нелинейных сетей в компьютерном эксперименте // Тезисы докладов III Всероссийского семинара "Нейроинформатика и ее приложения", Красноярск: Изд-во КГТУ, 1995, с. 10
127. Кендалл М., Стьюарт А. Статистические выводы и связи. 11 М.: Наука, 1973.- 900 с.
128. Кирдин А.Н., Новоходько А.Ю., Царегородцев В.Г. Скрытые параметры и транспонированная регрессия/ Нейроинформатика Новосибирск: Наука, Сибирская издательская фирма РАН, 1998.
129. Костюк Ф.Ф. Инфаркт миокарда. Красноярск: Офсет, 1993. 224 с.
130. Кохонен Т. Ассоциативная память. - М.: Мир, 1980.
131. Кохонен Т. Ассоциативные запоминающие устройства. - М.: Мир, 1982.
132. Коченов Д.А., Миркес Е.М. "Определение чувствительности нейросети к изменению входных сигналов", тезисы докладов III Всеросийского семинара "Нейроинформатика и ее приложения", Красноярск: Изд-во КГТУ, с. 61
133. Коченов Д.А., Миркес Е.М. "Синтез управляющих воздействий", тезисы докладов III Всеросийского семинара "Нейроинформатика и ее приложения", Красноярск: Изд-во КГТУ, с. 31
134. Коченов Д.А., Миркес Е.М., Россиев Д.А. Автоматическая подстройка входных данных для получения требуемого ответа нейросети // Проблемы информатизации региона. Труды межрегиональной конференции (Красноярск, 27-29 ноября 1995 г.). Красноярск, 1995.- С.156.
135. Коченов Д.А., Миркес Е.М., Россиев Д.А. Метод подстройки параметров примера для получения требуемого ответа нейросети // Нейроинформатика и ее приложения. Тез. докл. Всероссийского рабочего семинара 7 - 10 октября 1994 г. Красноярск.- 1994.- С.39.
136. Коченов Д.А., Россиев Д.А. Аппроксимация функций класса C[a,b] нейросетевыми предикторами // Тезисы докладов рабочего семинара "Нейроинформатика и нейрокомпьютеры", Красноярск, 8-11 октября 1993 г., Красноярск.- 1993.- С.13.
137. Крайзмер Л.П., Матюхин С.А., Майоркин С.Г. Память кибернетических систем (Основы мнемологии). - М.: Сов. радио, 1971.
138. Куссуль Э.М., Байдык Т.Н. Разработка архитектуры нейроподобной сети для распознавания формы объектов на изображении // Автоматика.- 1990.- № 5.- С. 56-61.
139. Кушаковский М.С. Аритмии сердца. Санкт-Петербург: Гиппократ, 1992. 544 с.
140. Лбов Г.С. Методы обработки разнотипных экспериментальных данных.- Новосибирск: Наука, 1981.- 157 с.
141. Лоули Д., Максвелл А. Факторный анализ как статистический метод.- М.: Мир, 1967.- 144 с.
142. Мазуров В.Д. Метод комитетов в задачах оптимизации и классификации.- М.: Наука, Гл. ред. физ.-мат. лит., 1990. 248 с.
143. МакКаллок У.С., Питтс В. Логическое исчисление идей, относящихся к нервной активности // Нейрокомпьютер, 1992. №|3, 4. С. 40-53.
144. Масалович А.И. От нейрона к нейрокомпьютеру // Журнал доктора Добба.- 1992.- N.1.- С.20-24.
145. Минский М., Пайперт С. Персептроны. - М.: Мир, 1971.
146. Миркес Е.М. Глобальные и локальные оценки для сетей двойственного функционирования. Тезисы докладов III Всеросийского семинара "Нейроинформатика и ее приложения", Красноярск: Изд-во КГТУ, сс. 76-77
147. Миркес Е.М. Использование весов примеров при обучении нейронных сетей. тезисы докладов III Всероссийского семинара "Нейроинформатика и ее приложения", Красноярск: Изд-во КГТУ, с. 75
148. Миркес Е.М. Логически прозрачные нейронные сети и производство явных знаний из данных / Нейроинформатика Новосибирск: Наука, Сибирская издательская фирма РАН, 1998.
149. Миркес Е.М. Нейроинформатика и другие науки // Вестник КГТУ, 1996, вып. 6, с.5-33.
150. Миркес Е.М. Нейронные сети ассоциативной памяти / Нейроинформатика Новосибирск: Наука, Сибирская издательская фирма РАН, 1998.
151. Миркес Е.М. Обучение сетей с пороговыми нейронами. Тезисы докладов III Всеросийского семинара "Нейроинформатика и ее приложения", Красноярск: Изд-во КГТУ, с. 72
152. Миркес Е.М. Оценки и интерпретаторы ответа для сетей двойственного функционирования. Тезисы докладов III Всероссийского семинара "Нейроинформатика и ее приложения", Красноярск: Изд-во КГТУ, сс. 73-74

153. Миркес Е.М., Свитин А.П. Применение метода ассоциативных сетей для прогнозирования переносов заряда при адсорбции молекул. // Эволюционное моделирование и кинетика. Новосибирск: Наука, 1992. С.30-35.
154. Миркес Е.М., Свитин А.П., Фет А.И. Массовые формулы для атомов. // Математическое моделирование в химии и биологии. Новые подходы. Новосибирск : Наука, 1992. С. 199-204.
155. Миркин Б.Г. Анализ качественных признаков и структур.М.: Статистика, 1980.- 319 с.
156. Мкртчян С.О. Проектирование логических устройств ЭВМ на нейронных элементах. - М.: Энергия, 1977.
157. Мостеллер Ф., Тьюки Дж. Анализ данных и регрессия.- М.: Финансы и статистика, 1982.- 239 с.
158. Муллат И.Э. Экстремальные подсистемы монотонных систем. I, II, III. - Автоматика и телемеханика, 1976, № 5. С. 130-139; 1976, № 8. С. 169-178; 1977, № 1. С. 143-152.
159. Мучник И.Б. Анализ структуры экспериментальных графов. - Автоматика и телемеханика, 1974, № 9. С. 62-80.
160. Мызников А.В., Россиев Д.А., Лохман В.Ф. Нейросетевая экспертная система для оптимизации лечения облитерирующего тромбангиита и прогнозирования его непосредственных исходов // Ангиология и сосудистая хирургия.- 1995.- N 2.- С.100.
161. Мызников А.В., Россиев Д.А., Лохман В.Ф. Прогнозирование непосредственных результатов лечения облитерирующего тромбангиита с помощью нейронных сетей // В сб.: Молодые ученые - практическому здравоохранению.Красноярск, 1994.- С. 42.
162. Назаров Б.В. Прогностические аспекты некоторых нарушений ритма и проводимости при остром инфаркте миокарда: Автореф. дис.... канд. мед. наук. Новосибирск, 1982. 22 с.
163. Назимова Д.И., Новоходько А.Ю., Царегородцев В.Г. Нейросетевые методы обработки информации в задаче прогнозирования климатических параметров. // Математические модели и методы их исследования: Междунар. конференция. Тезисы докладов. Красноярск: изд. Краснояр. гос. ун-та. С. 135.
164. Назимова Д.И., Новоходько А.Ю., Царегородцев В.Г. Нейросетевые методы обработки информации в задаче восстановления климатических данных // Нейроинформатика и ее приложения. Тезисы докладов 5 Всероссийского семинара, 3-5 октября 1997 г. / Под ред. А.Н.Горбаня. Красноярск: изд. КГТУ, 1997. С. 124.
165. Народов А.А., Россиев Д.А., Захматов И.Г. Оценка компенсаторных возможностей головного мозга при его органических поражениях с помощью искусственных нейронных сетей // В сб.: Молодые ученые - практическому здравоохранению.- Красноярск, 1994.- С.30.
166. Научное открытие в России... . - Красноярский комсомолец (газета), Красноярск, 1992, 11 августа, № 86.
167. Нейроинформатика / А.Н.Горбань, В.Л.Дунин-Барковский, А.Н.Кирдин, Е.М.Миркес, А.Ю.Новоходько, Д.А.Россиев, С.А.Терехов, М.Ю.Сенашова, В.Г.Царегородцев. Новосибирск: Наука, Сибирская издательская фирма РАН, 1998.
168. Николаев П.П. Методы представления формы объектов в задаче константного зрительного восприятия / Интеллектуальные процессы и их моделирование. Пространственно-временная организация.- М.: Наука, 1991.- С. 146-173.
169. Николаев П.П. Трихроматическая модель констант восприятия окраски объектов // Сенсорные системы. 1990. Т.4 Вып. 4. С. 421-442.
170. Нильсен Н. Обучающиеся машины. - М.: Мир, 1967.
171. Новоходько А.Ю., Царегородцев В.Г. Нейросетевое решение транспонированной задачи линейной регрессии // Тезисы докладов четвертой международной конференции "Математика, компьютер, образование". - Москва, 1997.
172. Охонин В.А. Вариационный принцип в теории адаптивных сетей. Препринт ИФ СО АН СССР, Красноярск, 1987, №61Б, 18 с.
173. Парин В.В., Баевский Р.М. Медицина и техника.- М.: Знание, 1968.- С.36-49.
174. Переверзев-Орлов В.С. Советчик специалиста. Опыт разработки партнерской системы // М.: Наука, 1990.- 133 с.
175. Петров А.П. Аксиоматика игры "в прятки" и генезис зрительного пространства / Интеллектуальные процессы и их моделирование. Пространственно-временная организация.- М.: Наука, 1991.- С. 174-182.

176. Питенко А.А. Нейросетевое восполнение пробелов данных в ГИС // Нейроинформатика и ее приложения. Тезисы докладов 5 Всероссийского семинара, 3-5 октября 1997 г. / Под ред. А.Н.Горбаня. Красноярск: изд. КГТУ, 1997. С. 140.
177. Позин И.В. Моделирование нейронных структур. - М.: Наука, 1970.
178. Пшеничный Б.Н., Данилин Ю.М. Численные методы в экстремальных задачах. М.:Наука, 1975. 319 с.
179. Распознавание образов и медицинская диагностика / под ред. Ю.М. Неймарка.- М.: Наука, 1972.- 328 с.
180. Розенблатт Ф. Принципы нейродинамики. Перцептрон и теория механизмов мозга. М.: Мир, 1965. 480 с.
181. Россиев А.А. Генератор 0-таблиц в среде WINDOWS-95 // Нейроинформатика и ее приложения. Тезисы докладов 5 Всероссийского семинара, 3-5 октября 1997 г. / Под ред. А.Н.Горбаня. Красноярск: изд. КГТУ, 1997. С. 151.
182. Россиев Д.А, Захматов И.Г. Оценка компенсаторных возможностей головного мозга при его органических поражениях (опыт применения нейросетевого векторного предиктора) // Нейроинформатика и ее приложения. Тез. докл. Всероссийского рабочего семинара 7 - 10 октября 1994 г. Красноярск.- 1994. 17 С.42.
183. Россиев Д.А. Медицинская нейроинформатика / Нейроинформатика Новосибирск: Наука, Сибирская издательская фирма РАН, 1998.
184. Россиев Д.А. Нейросетевые самообучающиеся экспертные системы в медицине // Молодые ученые - практическому здравоохранению.- Красноярск, 1994.- С.17.
185. Россиев Д.А., Бутакова Е.В. Нейросетевая диагностика и дифференциальная диагностика злокачественных опухолей сосудистой оболочки глаза // Нейроинформатика и ее приложения: Материалы III Всероссийского семинара, 6-8 октября 1995 г. Ч. 1/Под ред. А.Н.Горбаня; Красноярск: Изд-во КГТУ, 1995. С. 167-194.
186. Россиев Д.А., Бутакова Е.В. Ранняя диагностика злокачественных опухолей сосудистой оболочки глаза с использованием нейронных сетей // Нейроинформатика и ее приложения. Тез. докл. Всероссийского рабочего семинара 7 10 октября 1994 г. Красноярск.- 1994.- С.44.
187. Россиев Д.А., Винник Н.Г. Предсказание "удачности" предстоящего брака нейросетевыми экспертами // Нейроинформатика и ее приложения. Тез. докл. Всероссийского рабочего семинара 7 - 10 октября 1994 г. Красноярск.- 1994.С.45.
188. Россиев Д.А., Гилев С.Е., Коченов Д.А. "MultiNeuron, Версии 2.0 и 3.0", тезисы докладов III Всеросийского семинара "Нейроинформатика и ее приложения", Красноярск: Изд-во КГТУ, с. 14
189. Россиев Д.А., Гилев С.Е., Коченов Д.А.. Нейроэмулятор "MultiNeuron".Второй Сибирский конгресс по Прикладной и Индустриальной Математике, посвященный памяти А.А.Ляпунова (1911-1973), А.П.Ершова (1931-1988) и И.А.Полетаева (1915-1983). Новосибирск, июнь 1996. Тезисы докладов, часть 1. Изд. Института математики СО РАН. С. 45.
190. Россиев Д.А., Головенкин С.Е. Прогнозирование осложнений инфаркта миокарда с помощью нейронных сетей // Нейроинформатика и ее приложения. Тез. докл. Всероссийского рабочего семинара 7 - 10 октября 1994 г. Красноярск.- 1994.С.40.
191. Россиев Д.А., Головенкин С.Е., Назаров Б.В., Шульман В.А., Матюшин Г.В. Определение информативности медицинских параметров с помощью нейронной сети // Диагностика, информатика и метрология - 94.- Тез. научно-технической конференции (г. Санкт-Петербург, 28-30 июня 1994 г.).С.-Петербург.- 1994.- С.348.
192. Россиев Д.А., Головенкин С.Е., Шульман В.А., Матюшин Г.В. Использование нейронных сетей для прогнозирования возникновения или усугубления застойной сердечной недостаточности у больных с нарушениями ритма сердца // Тезисы докладов рабочего семинара "Нейроинформатика и нейрокомпьютеры", Красноярск, 8-11 октября 1993 г., Красноярск.- 1993.- С.16.
193. Россиев Д.А., Головенкин С.Е., Шульман В.А., Матюшин Г.В. Прогнозирование осложнений инфаркта миокарда нейронными сетями // Нейроинформатика и ее приложения: Материалы III Всероссийского семинара, 6-8 октября 1995 г. Ч. 1/Под ред. А.Н.Горбаня; Красноярск: Изд-во КГТУ, 1995. С. 128-166.
194. Россиев Д.А., Догадин С.А., Масленикова Е.В., Ноздрачев К.Г., Борисов А.Г. Обучение нейросетей выявлению накопленной дозы радиоактивного облучения // Тезисы докладов рабочего семинара "Нейроинформатика и нейрокомпьютеры", Красноярск, 8-11 октября 1993 г., Красноярск.- 1993.- С.15.

195. Россиев Д.А., Догадин С.А., Масленникова Е.В., Ноздрачев К.Г., Борисов А.Г. Выявление накопленной дозы радиоактивного облучения с помощью нейросетевого классификатора // Современные проблемы и методологические подходы к изучению влияния факторов производственной и окружающей среды на здоровье человека (Тез. докл. республиканской конф.).- Ангарск-Иркутск: изд. "Лисна".1993.- С.111-112.
196. Россиев Д.А., Коченов Д.А. Пакет программ "MultiNeuron" - "Configurator" - "Tester" для конструирования нейросетевых приложений // Нейроинформатика и ее приложения. Тез. докл. Всероссийского рабочего семинара 7 - 10 октября 1994 г. Красноярск.- 1994.- С.30.
197. Россиев Д.А., Мызников А.А. Нейросетевое моделирование лечения и прогнозирование его непосредственных результатов у больных облитерирующим тромбангиитом // Нейроинформатика и ее приложения: Материалы III Всероссийского семинара, 6-8 октября 1995 г. Ч. 1/Под ред. А.Н.Горбаня; Красноярск: Изд-во КГТУ, 1995. С. 194-228.
198. Россиев Д.А., Мызников А.В. Прогнозирование непосредственных результатов лечения облитерирующего тромбангиита с помощью нейронных сетей // Нейроинформатика и ее приложения. Тез. докл. Всероссийского рабочего семинара 7 - 10 октября 1994 г. Красноярск.- 1994.- С.41.
199. Россиев Д.А., Савченко А.А., Гилев С.Е., Коченов Д.А. Применение нейросетей для изучения и диагностики иммунодефицитных состояний// Нейроинформатика и нейрокомпьютеры/ Тезисы докладов рабочего семинара 8-11 октября 1993 г., Красноярск/ Институт биофизики СО РАН, 1993. С. 32.
200. Россиев Д.А., Суханова Н.В., Швецкий А.Г. Нейросетевая система дифференциальной диагностики заболеваний, проявляющихся синдромом "острого живота" // Нейроинформатика и ее приложения. Тез. докл. Всероссийского рабочего семинара 7 - 10 октября 1994 г. Красноярск.- 1994.- С.43.
201. Савченко А.А., Догадин С.А., Ткачев А.В., Бойко Е.Р., Россиев Д.А. Обследование людей в районе возможного радиоактивного загрязнения с помощью нейросетевого классификатора // Нейроинформатика и ее приложения. Тез. докл. Всероссийского рабочего семинара 7 - 10 октября 1994 г. Красноярск.- 1994.- С.46.
202. Савченко А.А., Митрошина Л.В., Россиев Д.А., Догадин С.А. Моделирование с помощью нейросетевого предиктора реальных чисел иммуноэндокринного взаимодействия при заболеваниях щитовидной железы // Тезисы докладов рабочего семинара "Нейроинформатика и нейрокомпьютеры", Красноярск, 8-11 октября 1993 г., Красноярск.- 1993.- С.18.
203. Савченко А.А., Россиев Д.А., Догадин С.А., Горбань А.Н. Нейротехнология для обследования людей в районе возможного радиоактивного загрязнения. Второй Сибирский конгресс по Прикладной и Индустриальной Математике, посвященный памяти А.А.Ляпунова (1911-1973), А.П.Ершова (1931-1988) и И.А.Полетаева (1915-1983). Новосибирск, июнь 1996. Тезисы докладов, часть 1. Изд. Института математики СО РАН. С. 46-47.
204. Савченко А.А., Россиев Д.А., Захарова Л.Б. Применение нейросетевого классификатора для изучения и диагностики вилюйского энцефалита // Тезисы докладов рабочего семинара "Нейроинформатика и нейрокомпьютеры", Красноярск, 8-11 октября 1993 г., Красноярск.- 1993.- С.17.
205. Савченко А.А., Россиев Д.А., Ноздрачев К.Г., Догадин С.А. Обучение нейросетевого классификатора дифференцировать пол человека по метаболическим и гормональным показателям // Нейроинформатика и ее приложения. Тез. докл. Всероссийского рабочего семинара 7 - 10 октября 1994 г. Красноярск.- 1994.С.47.
206. Савченко А.А., Россиев Д.А., Ноздрачев К.Г., Догадин С.А. Подтверждение с помощью нейросетевого классификатора существования гомеостатических уровней в группе практически здоровых людей // Нейроинформатика и ее приложения. Тез. докл. Всероссийского рабочего семинара 7 - 10 октября 1994 г. Красноярск.- 1994.- С.49.
207. Савченко А.А., Россиев Д.А., Ноздрачев К.Г., Догадин С.А., Гилев С.Е. Нейроклассификатор, дифференцирующий пол человека по метаболическим и гормональным показателям. Второй Сибирский конгресс по Прикладной и Индустриальной Математике, посвященный памяти А.А.Ляпунова (1911-1973), А.П.Ершова (1931-1988) и И.А.Полетаева (1915-1983). Новосибирск, июнь 1996. Тезисы докладов, часть 1. Изд. Института математики СО РАН. С. 47.
208. Савченко А.А., Смирнова С.В., Россиев Д.А. Применение нейросетевого классификатора для изучения и диагностики аллергических и псевдоаллергических реакций //

Нейроинформатика и ее приложения. Тез. докл. Всероссийского рабочего семинара 7 - 10 октября 1994 г. Красноярск.- 1994.С. 48.
209. Сенашова М.Ю. Метод обратного распространения точности с учетом независимости погрешностей сигналов сети // Тез.конф.молодых ученых Красноярского научного центра. – Красноярск, Президиум КНЦ СО РАН, 1996, сс.96-97.
210. Сенашова М.Ю. Метод обратного распространения точности. // Нейроинформатика и ее приложения. Тез. докл. IV Всероссийского семинара, 5 - 7 октября, 1996 г. Красноярск; КГТУ. 1996, с.47
211. Сенашова М.Ю. Погрешности в нейронных сетях / Нейроинформатика Новосибирск: Наука, Сибирская издательская фирма РАН, 1998.
212. Сенашова М.Ю. Упрощение нейронных сетей: приближение значений весов синапсов при помощи цепных дробей. Вычислительный центр СО РАН в г. Красноярске. Красноярск, 1997. 11 с., библиогр. 6 назв. (Рукопись деп. в ВИНИТИ 25.07.97, № 2510-В97)
213. Сенашова. М.Ю. Упрощение нейронных сетей. Использование цепных дробей для приближения весов синапсов. // Нейроинформатика и ее приложения: Тезисы докладов V Всероссийского семинара, 3-5 октября, 1997 г., Красноярск; КГТУ. 1997, с. 165-166.
214. Соколов Е.Н., Вайткявичус Г.Г. Нейроинтеллект: от нейрона к нейрокомпьютеру. М.: Наука, 1989. 238 с.
215. Степанян А.А., Архангельский С.В. Построение логических схем на пороговых элементах. Куйбышевское книжное изд-во, 1967.
216. Судариков В.А. Исследование адаптивных нейросетевых алгоритмов решения задач линейной алгебры // Нейрокомпьютер, 1992. № 3,4. С. 13-20.
217. Тарасов К.Е., Великов В.К., Фролова А.И. Логика и семиотика диагноза (методологические проблемы).- М.: Медицина, 1989.- 272 с.
218. Терехов С.А. Нейросетевые информационные модели сложных инженерных систем / Нейроинформатика Новосибирск: Наука, Сибирская издательская фирма РАН, 1998.
219. Транспьютерные и нейронные ЭВМ. /Под ред. В.К.Левина и А.И.Галушкина - М.: Российский Дом знаний, 1992.
220. Уидроу Б., Стирнз С. Адаптивная обработка сигналов. М.: Мир, 1989. 440 с.
221. Уоссермен Ф. Нейрокомпьютерная техника.- М.: Мир, 1992.
222. Федотов Н.Г. Методы стохастической геометрии в распознавании образов. - М.: Радио и связь, 1990.- 144 с.
223. Фор А. Восприятие и распознавание образов.- М.: Машиностроение, 1989.- 272 с.
224. Фролов А.А., Муравьев И.П. Информационные характеристики нейронных сетей. - М.: Наука, 1988.
225. Фролов А.А., Муравьев И.П. Нейронные модели ассоциативной памяти.- М.: Наука, 1987.- 160 с.
226. Фу К. Структурные методы в распознавании образов.- М.: Мир, 1977.- 320 с.
227. Фукунга К. Введение в статистическую теорию распознавания образов.- М.: Наука, 1979.- 367 с.
228. Хартман Г. Современный факторный анализ.- М.: Статистика, 1972.- 486 с.
229. Химмельблау Д. Прикладное нелинейное программирование. М.: Мир, 1975. 534 с.
230. Хинтон Дж.Е. Обучение в параллельных сетях / Реальность и прогнозы искусственного интеллекта.- М.: Мир, 1987.- С. 124-136.
231. Царегородцев В.Г. Транспонированная линейная регрессия для интерполяции свойств химических элементов // Нейроинформатика и ее приложения. Тезисы докладов 5 Всероссийского семинара, 3-5 октября 1997 г. / Под ред. А.Н.Горбаня. Красноярск: изд. КГТУ, 1997. С. 177-178.
232. Цыганков В.Д. Нейрокопьютер и его применение.- М.: "Сол Систем", 1993.
233. Цыпкин Я.З. Основы теории обучающихся систем. М.: Наука, 1970. 252 с.
234. Шайдуров В.В. Многосеточные методы конечных элементов. - М.: Наука, 1989.
235. Шварц Э., Трис Д. Программы, умеющие думать // Бизнес Уик.- 1992.- N.6.- С.15-18.
236. Шенк Р., Хантер Л. Познать механизмы мышления / Реальность и прогнозы искусственного интеллекта.- М.: Мир, 1987.- С. 15-26.
237. Щербаков П.С. Библиографическая база данных по методам настройки нейронных сетей // Нейрокомпьютер, 1993. № 3,4. С.5-8.
238. Aleksander I., Morton H. The logic of neural cognition // Adv. Neural Comput.- Amsterdam etc., 1990.- PP. 97-102.
239. Alexander S. Th. Adaptive Signal Processing. Theory and Applications. Springer. 1986. 179 p.

240. Allen J., Murray A.. Development of a neural network screening aid for diagnosing lower limb peripheral vascular disease from photoelectric plethysmography pulse waveforms // Physiol. Meas.- 1993.- V.14, N.1.- P.13-22.
241. Amari Sh., Maginu K. Statistical Neurodynamics of Associative Memory // Neural Networks, 1988. V.1. N1. P. 63-74.
242. Arbib M.A. Brains, Machines, and Mathematics. Springer, 1987. 202 p.
243. Astion M.L., Wener M.H., Thomas R.G., Hunder G.G., Bloch D.A. Application of neural networks to the classification of giant cell arteritis // Arthritis Reum.- 1994.- V.37, N.5.- P.760-770.
244. Aynsley M., Hofland A., Morris A.J. et al. Artificial intelligence and the supervision of bioprocesses (real-time knowledge-based systems and neural networks) // Adv. Biochem. Eng. Biotechnol.- 1993.- N.48.- P.1-27.
245. Baba N. New Topics in Learning Automate Theory and Applications. Springer, 1985. 131 p. (Lec. Not. Control and Information, N71).
246. Barschdorff D., Ester S., Dorsel T et al. Phonographic diagnostic aid in heart defects using neural networks // Biomed. Tech. Berlin.- 1990.- V.35, N.11.- P.271-279.
247. Bartsev S.I., Okhonin V.A. Optimization and Monitoring Needs: Possible Mechanisms of Control of Ecological Systems. Nanobiology, 1993, v.2, p.165-172.
248. Bartsev S.I., Okhonin V.A. Self-learning neural networks playing "Two coins"// Proc. of International Workshop "Neurocomputers and attention II", Manchester Univ.Press, 1991, p.453-458.
249. Bartsev S.I., Okhonin V.A. The algorithm of dual functioning (back-propagation): general approuch, versions and applications. Preprint of Biophysics Institute SB AS USSR, Krasnoyarsk, 1989, №107B, 16 p.
250. Bartsev S.I., Okhonin V.A. Variation principle and algorithm of dual functioning: examples and applications// Proc. of International Workshop "Neurocomputers and attention II", Manchester Univ.Press, 1991, p.445-452.
251. Baxt W.G. A neural network trained to identify the presence of myocardial infarction bases some decisions on clinical associations that differ from accepted clinical teaching // Med. Decis. Making.- 1994.- V.14, N.3.- P.217-222.
252. Baxt W.G. Analysis of the clinical variables driving decision in an artificial neural network trained to identify the presence of myocardial infarction // Ann. Emerg. Med.- 1992.- V.21, N.12.- P.1439-1444.
253. Baxt W.G. Complexity, chaos and human physiology: the justification for non-linear neural computational analysis // Cancer Lett.- 1994.- V.77, N.2-3.- P.85-93.
254. Baxt W.G. Use of an artificial neural network for the diagnosis of myocardial infarction // Ann. Intern. Med.- 1991.- V.115, N.11.- P.843-848.
255. Borisov A.G., Gilev S.E., Golovenkin S.E., Gorban A.N., Dogadin S.A., Kochenov D.A., Maslennikova E.V., Matyushin G.V., Mirkes Ye.M., Nozdrachev K.G., Rossiyev D.A., Savchenko A.A., Shulman V.A. "MultiNeuron" neural simulator and its medical applications // Modelling, Measurement & Control, C.- 1996.- V.55, N.1.- P.1-5.
256. Bruck J., Goodman J. W. On the power of neural networks for solving hard problems // J. Complex.- 1990.- 6, № 2.PP. 129-135.
257. Budilova E.V., Teriokhin A.T. Endocrine networks // The RNNS/IEEE Symposium on Neuroinformatics and Neurocomputers, Rostov-on-Don, Russia, October 7-10, 1992.- Rostov/Don, 1992.- V.2.- P.729-737.
258. Carpenter G.A., Grossberg S. A Massivly Parallel Architecture for a Self-Organizing Neural Pattern Recognition Machine. - Computer Vision, Graphics, and Image Processing, 1987. Vol. 37. PP. 54-115.
259. Connectionism in Perspective/Ed. by R. Pfeifer, Z. Schreter, F.Fogelman-Soulie and L. Steels. North-Holland, 1989. 518 p.
260. Cybenko G. Approximation by superposition of a sigmoidal function. - Mathematics of Control, Signals, and Systems, 1989. Vol. 2. PP. 303 - 314.
261. Diday E., Simon J.C. Clustering analysis, (dans Digital Pattern Recognition), Redacteur: K.S.F.U., Springer Verlag, Berlin, 1980, P. 47-93.
262. Disordered Systems and biological Organization/Ed. by Bienenstock F., Fogelman-Soulie G. Weisbuch. Springer, 1986. 405 p.

263. Dorrer M.G., Gorban A.N., Kopytov A.G., Zenkin V.I. Psychological intuition of neural networks. Proceedings of the WCNN'95 (World Congress on Neural Networks'95, Washington DC, July 1995). PP. 193-196.
264. Dorrer M.G., Gorban A.N., Zenkin V.I. Neural networks in psychology: classical explicit diagnoses // Neuroinformatics and Neurocomputers, Proceedings of the second RNNS-IEEE Simposium, Rostov-na-Donu, September 1995. PP. 281-284.
265. Draper N. R. Applied regression analysis bibliographi update 1988-89 // Commun. Statist. Theory and Meth.- 1990.1990.- 19, № 4.- PP. 1205-1229.
266. Ercal F., Chawla A., Stoeker W.V. et al. Neural network diagnosis of malignant melanoma from color images // IEEE Trans. Biomed. Eng.- 1994.- V.41, N.9.- P.837-845.
267. Ferretti C., Mauri G. NNET: some tools for neural Networks simulation // 9th Annu. Int. Phoenix Conf. Comput. and Commun., Scottsdate, Ariz., March 21-23, 1990.- Los Alamitos (Calif.) etc., 1990.- PP. 38-43.
268. Filho E.C.D.B.C., Fairhurst M.C., Bisset D.L. Adaptive pattern recognition using goal seeking neurons // Pattern Recogn. Lett.- 1991.- 12, № 3.- PP. 131-138.
269. Floyd C.E.Jr., Lo J.Y., Yun A.J. et al. Prediction of breast cancer malignancy using an artificial neural network // Cancer.- 1994.- V.74, N.11.- P.2944-2948.
270. Forbes A.B., Mansfield A.J. Neural implementation of a method for solving systems of linear algebraic equations // Nat. Phys. Lab. Div. Inf. Technol. and Comput. Rept.- 1989.№ 155.- PP. 1-14.
271. Fu H.C., Shann J.J. A fuzzy neural network for knowledge learning // Int. J. Neural Syst.- 1994.- V.5, N.1.- P.13-22.
272. Fukushima K. Neocognitron: A self-organizing Neural Network model for a Mechanism of Pattern Recognition uneffected by shift in position // Biological Cybernetics.1980. V. 36, № 4. PP. 193-202.
273. Fulcher J. Neural networks: promise for the future? // Future Generat. Comput. Syst.- 1990-1991.- 6, № 4.- PP. 351-354.
274. Gallant A.R., White H. There exist a neural network that does not make avoidable mistakes. - IEEE Second International Coferense on Neural Networks, San Diego, CA, New York: IEEE Press, vol. 1, 1988. PP. 657 - 664.
275. Gecseg F. Products of Automata. Springer, 1986. 107 p.
276. Gemignani M. C. Liability for malfunction of an expert system // IEEE Conf. Manag. Expert Syst. Program and Proj., Bethesda, Md. Sept. 10-12, 1990: Proc.- Los Alamitos (Calif.) etc., 1990.- PP. 8-15.
277. Genis C. T. Relaxation and neural learning: points of convergence and divergence // J. Parallel and Distrib. Comput.- 1989.- 6, № 2.- PP. 217-244.
278. George N., Wang hen-ge, Venable D.L. Pattern recognition using the ring-wedge detector and neural-network software: [Pap.] Opt. Pattern Recogn. II: Proc. Meet., Paris, 26-27 Apr., 1989 // Proc. Soc. Photo-Opt. Instrum. Eng.- 1989.- PP. 96-106.
279. Gilev S.E. A non-back-propagation method for obtaining the gradients of estimate function // Advances in Modelling & Analysis, A, AMSE Press, 1995. Vol. 29, № 1. PP. 51-57.
280. Gilev S.E., Gorban A.N. On Completeness of the Class of Functions Computable by Neural Networks, Proc. of the World Congress on Neural Networks, Sept. 15-18, 1996, San Diego, CA, Lawrence Erlbaum Associates, 1996, pp. 984-991.
281. Gilev S.E., Gorban A.N., Kochenov D.A., Mirkes Ye.M., Golovenkin S.E., Dogadin S.A., Nozdrachev K.G., Maslennikova E.V., Matyushin G.V., Rossiev D.A., Shulman V.A., Savchenko A.A. "MultiNeuron" neural simulator and its medical applications // Proceedings of International Conference on Neural Information Processing, Oct. 17-20, 1994, Seoul, Korea.- V.2.- P.1261-1264.
282. Gilev S.E., Gorban A.N., Mirkes E.M. Internal Conflicts in Neural Networks // Transactions of IEEE-RNNS Simposium (Rostov-on-Don, September 1992). V.1. PP. 219-226.
283. Gilev S.E., Gorban A.N., Mirkes E.M. Several Methods for Accelerating the Traning Process of Neural Networks in Pattern Recognition. - Advances in Modelling & Analysis, A, AMSE Press, V. 12, No. 4, 1992, pp. 29-53.
284. Gilev S.E., Gorban A.N., Mirkes E.M. Small Experts and Internal Conflicts in Leanable Neural Networks // Advances in Modelling & Analysis.- AMSE Press.- 1992.- V.24, No. 1.P.45-50.
285. Gileva L.V., Gilev S.E. Neural Networks for binary classification// AMSE Transaction, Scientific Siberian, A, 1993, Vol. 6. Neurocomputing, pp. 135-167.

286. Gindi G.R., Darken C.J., O'Brien K.M. et al. Neural network and conventional classifiers for fluorescence-guided laser angioplasty // IEEE Trans. Biomed. Eng.- 1991.- V.38, N.3.- P.246-252.
287. Gluck M.A., Parker D.B., Reifsnider E.S. Some Biological Implications of a Differential-Hebbian Learning Rule. - Psychobiology, 1988. Vol. 16. No. 3. PP. 298-302.
288. Golub D.N. and Gorban A.N. Multi-Particle Networks for Associative Memory, Proc. of the World Congress on Neural Networks, Sept. 15-18, 1996, San Diego, CA, Lawrence Erlbaum Associates, 1996, pp. 772-775.
289. Gorban A.N., Novokhodko A.Yu.. Neural Networks In Transposed Regression Problem, Proc. of the World Congress on Neural Networks, Sept.15-18, 1996, San Diego, CA, Lawrence Erlbaum Associates, 1996, pp. 515-522.
290. Gorban A.N. Neurocomputing in Siberia // Advances in Modelling & Analysis.- AMSE Press.- 1992.- V.34(2).P.21-28.
291. Gorban A.N. Systems with inheritance and effects of selection.- Scientific Siberian A, Volume 1. Ecology, AMSE Press, ISBN: 2-909214-04-4, 1992, pp. 82-126
292. Gorban A.N., Mirkes Ye.M. and Wunsch D.C. II High order ortogonal tensor networks: Information capacity and reliability // ICNN97 (The 1997 IEEE International Conference on Neural Networks), Houston, IEEE, 1997. PP. 1311-1314.
293. Gorban A.N., Mirkes Ye.M. Functional Components of Neurocomputer. Труды третьей международной конференции "Математика, компьютер, образование". - Москва, 1996. с.352-359.
294. Gorban A.N., Mirkes Ye.M. Functional components of neurocomputer. 3-d International conference "Mathematics, computer, education", Dubna, Jan. 1996. Abstracts, p. 160.
295. Gorban A.N., Rossiev D.A., Butakova E.V., Gilev S.E., Golovenkin S.E., Dogadin S.A., Dorrer M.G., Kochenov D.A., Kopytov A.G., Maslennikova E.V., Matyushin G.V., Mirkes Ye.M., Nazarov B.V., Nozdrachev K.G., Savchenko A.A., Smirnova S.V., Shulman V.A., Zenkin V.I. Medical, psychological and physiological applications of MultiNeuron neural simulator. Neuroinformatics and Neurocomputers, Proceedings of the second RNNS-IEEE Simposium, Rostov-na-Donu, September 1995.- PP. 7-14.
296. Gorban A.N., Rossiev D.A., Gilev S.E. et al. "NeuroComp" group: neural-networks software and its application // Russian Academy of Sciences, Krasnoyarsk Computing Center, Preprint N 8.- Krasnoyarsk, 1995.- 38 p.
297. Gorban A.N., Rossiev D.A., Gilev S.E. et al. Medical and physiological applications of MultiNeuron neural simulator // Proceedings of World Congress on Neural Networks-1995 (WCNN'95).- Washington, 1995.- V.1, P.170-175.
298. Gorban A.N., Rossiev D.A., Gilev S.E., Dorrer M.A., Kochenov D.A., Mirkes Ye.M., Golovenkin S.E., Dogadin S.A., Nozdrachev K.G., Matyushin G.V., Shulman V.A., Savchenko A.A. Medical and physiological applications of MultiNeuron neural simulator // Proceedings of World Congress on Neural Networks - 1995 (WCNN'95).- PP. 170-175.
299. Gorban A.N., Waxman C. How many neurons are sufficient to elect the U.S.A. President?// AMSE Transaction, Scientific Siberian, A, 1993, Vol. 6. Neurocomputing, pp. 168-188.
300. Gorban A.N., Waxman C. How many Neurons are Sufficient to Elect the U.S.A. President? TWO! (Siberian neurocomputer forecasts results of U.S.A. Presidential elections) Krasnoyarsk, Institute of Biophysics, Russian Academy of Sciences Siberian Branch, 1992 - 29 pp. (preprint Russian Academy of Sciences Siberian Branch, Institute of Biophysics N 191 Б).
301. Gorban A.N., Waxman C. Neural networks for political forecast. Proceedings of the WCNN'95 (World Congress on Neural Networks'95, Washington DC, July 1995). PP. 179-184.
302. Gordienko P. Construction of efficient neural networks // Proceedings of the International Conference on Neural Information Processing (Oct. 17-20, 1994, Seoul, Korea) V.1. PP. 366-371.
303. Gordienko P. How to obtain a maximum of skills with minimum numbers of connections// AMSE Transaction, Scientific Siberian, A, 1993, Vol. 6. Neurocomputing, pp.204-208.
304. Gross G.W., Boone J.M., Greco-Hunt V. et al. Neural networks in radiologic diagnosis. II. Interpretation of neonatal chest radiographs // Invest. Radiol.- 1990.- V.25, N.9.- P.1017-1023.
305. Grossberg S. Nonlinear Neural Networks: Principles, Mechanism and Architectures// Neural Networks, 1988. V.1. N1. P. 17-62.
306. Guo Z., Durand L.G., Lee H.C. et al. Artificial neural networks in computer-assisted classification of heart sounds in patients with porcine bioprosthetic valves // Med. Biol. Eng. Comput.- 1994.- V.32, N.3.- P.311-316.
307. Hecht-Nielsen R. Neurocomputing: Picking the Human Brain/IEEE Spectrum, 1988. March. P. 36-41.

308. Heht-Nielsen R. Theory of the backpropagation neural network. - Neural Networks for Human and Mashine Perception. H.Wechsler (Ed.). Vol. 2. Boston, MA: Academic Press, 1992. PP. 65 - 93.
309. Hod H., Lew A.S., Keltai M. et al. Early atrial fibrilla tion during evolving myocardial infarction: a consequence of impaired left atrial perfusion // Circulation, 1987. V.75, N.1. PP. 146-150.
310. Hoher M., Kestler H.A., Palm G. et al. Neural network based QRS classification of the signal averaged electrocardiogram // Eur. Heart J.- 1994.- V.15.- Abstr. Supplement XII-th World Congress Cardiology (734).- P.114.
311. Hopfield J.J. Neural Networks and physical systems with emergent collective computational abilities//Proc. Nat. Sci. USA. 1982. V.79. P. 2554-2558.
312. Hornik K., Stinchcombe M., White H. Multilayer Feedforward Networks are Universal Approximators. - Neural Networks. 1989. Vol. 2,. PP. 359 - 366.
313. Jeffries C. Code recognition with neural network dynamical systems // SIAM Rev.- 1990.- 32, № 4.- PP. 636-651.
314. Kalman R.E. A theory for the identification of linear relations // Frontiers Pure and Appl. Math.: Collect. Pap. Dedicat. Jacques-Louis Lions Occas. His 60th Birthday: Sci. Meet., Paris, 6-10 June, 1988.- Amsterdam etc., 1991.- PP. 117-132.
315. Keller J.M., Yager R.R., Tahani H. Neural network implementation of fuzzy logic // Fuzzy Sets and Syst.1992.- 45, № 1. PP. 1-12.
316. Kirdin A.N., Rossiev D.A., Dorrer M.G. Neural Networks Simulator for Medical, Physiological and Psychological Applications. Труды третьей международной конференции "Математика, компьютер, образование". - Москва, 1996. с.360-367.
317. Kirdin A.N., Rossiev D.A.. Neural-networks simulator for medical and physiological applications.3-d International conference "Mathematics, computer, education", Dubna, Jan. 1996. Abstracts, p. 162.
318. Kochenov D.A., Rossiev D.A. Approximations of functions of C[A,B] class by neural-net predictors (architectures and results)// AMSE Transaction, Scientific Siberian, A, 1993, Vol. 6. Neurocomputing. PP. 189-203.
319. Koopmans T. Serial correlation and quadratic forms in normal variates // Ann. Math. Statist, 1942. V. 13. PP. 14-33.
320. Korver M., Lucas P.J. Converting a rule-based expert system into a belief network // Med. Inf. Lond.- 1993.- V.18, N.3.- P.219-241.
321. Kosko B. Bidirectional Associative Memories. - IEEE Transactions on Systems, Man, and Cybernetics, Jan. 1988. Vol. SMC-18. PP.49-60.
322. Lee H.-L., Suzuki S., Adachi Y. et al. Fuzzy Theory in Traditional Chinese Pulse Diagnosis // Proceedings of 1993 International Joint Conference on Neural Networks, Nagoya, Japan, October 25-29, 1993.- Nagoya, 1993.- V.1.- P.774-777.
323. Levine D.S., Parks R.W., Prueitt P.S. Methodological and theoretical issues in neural network models of frontal cognitive functions // Int. J. Neurosci.- 1993.- V.72, N.3-4.- P.209-233.
324. Lichtman A.J., Keilis-Borok V.I., Pattern Recognition as Applied to Presidential Elections in U.S.A., 1860-1980; Role of Integral Social, Economic and Political Traits, Contribution No. 3760. 1981, Division of Geological and Planetary Sciences, California Institute of Technology.
325. Maclin P.S., Dempsey J. Using an artificial neural network to diagnose hepatic masses // J. Med. Syst.- 1992.- V.16, N.5.- P.215-225.
326. Macukow B. Robot control with neural networks // Artif. Intell. and Inf.-Contr. Syst. Rob.-89: Proc. 5th Int. Conf., Strbske Pleso, 6-10 Nov., 1989.- Amsterdam etc., 1989.- PP. 373-376.
327. Mirkes E.M., Svitin A.P. The usage of adaptive neural networks for catalytic activity predictions // CHISA - 10th Int. Congr. of chem. eng., chem. equipment design and automation. Praha, 1990. Prepr. B3.80 [1418]. 7 pp.
328. Modai I., Stoler M., Inbar-Saban N. et al. Clinical decisions for psychiatric inpatients and their evaluation by a trained neural network // Methods Inf. Med.- 1993.- V.32, N.5.- P.396-399.
329. Modha D.S., Heht-Nielsen R. Multilayer Functionals. Mathematical Approaches to Neural Networks. J.G.Taylor (Ed.). Elsevier, 1993. PP. 235 - 260.
330. Nakajima H., Anbe J., Egoh Y. et al. Evaluation of neural network rate regulation system in dual activity sensor rate adaptive pacer // European Journal of Cardiac Pacing and Electrophysiology.- Abstracts of 9th International Congress, Nice Acropolis - French, Rivera, June 15-18, (228), 1994.- Rivera, 1994.- P.54.
331. Narendra K.S., Amnasway A.M. A stable Adaptive Systems. Prentice-Hall, 1988. 350 p.
332. Neural Computers/Ed. by R. Eckmiller, Ch. Malsburg. Springer, 1989. 556 p.

333. Okamoto Y., Nakano H., Yoshikawa M. et al. Study on decision support system for the interpretation of laboratory data by an artificial neural network // Rinsho. Byori.- 1994.- V.42, N.2.- P.195-199.
334. Pedrycz W. Neurocomputations in relational systems // IEEE Trans. Pattern Anal. and Mach. Intell.- 1991.- 13, № 3.- PP. 289-297.
335. Pham D.T., Liu X. Statespace identification of dynamic systems using neural networks // Eng. Appl. Artif. Intell.1990.- 3, № 3.- PP. 198-203.
336. Pineda F.J. Recurrent bakpropagation and the dynamical approach to adaptive neural computation. - Neural Comput., 1989. Vol. 1. PP.161 - 172.
337. Poli R., Cagnoni S., Livi R. et al. A Neural Network Expert System for Diagnosing and Treating Hypertension // Computer.- 1991.- N.3.- P.64-71.
338. Real Brains, Artificial Minds/Ed. by J.L. Casti, A. Karlqvist. Norton-Holland, 1987. 226 p.
339. Reinbnerger G., Weiss G., Werner-Felmayer G. et al. Neural networks as a tool for utilizing laboratory information: comparison with linear discriminant analysis and with classification and regression trees // Proc. Natl. Acad. Sci., USA.- 1991.- V.88, N.24.- P.11426-11430.
340. Rinast E., Linder R., Weiss H.D. Neural network approach for computer-assisted interpretation of ultrasound images of the gallbladder // Eur. J. Radiol.- 1993.- V.17, N.3.- P.175-178.
341. Rossiev D.A., Golovenkin S.E., Shulman V.A., Matyushin G.V. Forecasting of myocardial infarction complications with the help of neural networks // Proceedings of the WCNN'95 (World Congress on Neural Networks'95, Washington DC, July 1995). PP. 185-188.
342. Rossiev D.A., Golovenkin S.E., Shulman V.A., Matyushin G.V. Neural networks for forecasting of myocardial infarction complications // Proceedings of the Second IEEE RNNS International Symposium on Neuroinformatics and Neurocomputers, September 20-23, 1995, Rostov-on-Don. - PP 292-298.
343. Rossiev D.A., Golovenkin S.E., Shulman V.A., Matyushin G.V. The employment of neural networks to model implantation of pacemaker in patients with arrhythmias and heart blocks // Modelling, Measurument & Control, C, 1995. Vol. 48, № 2. PP. 39-46.
344. Rossiev D.A., Golovenkin S.E., Shulman V.A., Matyushin G.V. The employment of neural networks to model implantation of pacemaker in patients with arrhythmias and heart blocks // Proceedings of International Conference on Neural Information Processing, Oct. 17-20, 1994, Seoul, Korea.V.1.- PP.537-542.
345. Rossiev D.A., Savchenko A.A., Borisov A.G., Kochenov D.A. The employment of neural-network classifier for diagnostics of different phases of immunodeficiency // Modelling, Measurement & Control.- 1994.- V.42.- N.2. P.55-63.
346. Rozenbojm J., Palladino E., Azevedo A.C. An expert clinical diagnosis system for the support of the primary consultation // Salud. Publica Mex.- 1993.- V.35, N.3.- P.321-325.
347. Rumelhart D.E., Hinton G.E., Williams R.J. Learning internal representations by error propagation. - Parallel Distributed Processing: Exploration in the Microstructure of Cognition, D.E.Rumelhart and J.L.McClelland (Eds.), vol. 1, Cambridge, MA: MIT Press, 1986. PP. 318 - 362.
348. Rummelhart D.E., Hinton G.E., Williams R.J. Learning representations by back-propagating errors // Nature, 1986. V. 323. P. 533-536.
349. Saaf L. A., Morris G. M. Filter synthesis using neural networks: [Pap.] Opt. Pattern Recogn. II: Proc. Meet., Paris, 26-27 Apr., 1989 // Proc. Soc. Photo-Opt. Instrum. Eng.- 1989.- 1134.- PP. 12-16.
350. Sandberg I.W. Approximation for Nonlinear Functionals. - IEEE Transactions on Circuits and Systems - 1: Fundamental Theory and Applications, Jan. 1992. Vol.39, No 1. PP.65 67.
351. Savchenko A.A., Zakharova L.B., Rossiev D.A. The employment of neural networks for investigation & diagnostics of Viliuisk encephalomyelitis // Modelling, Measurement & Control, C.- 1995.- V.48, N.4.- P.1-15.
352. Senashova M.Yu., Gorban A.N. and. Wunsch D.C. II. Back-propagation of accuracy // ICNN97 (The 1997 IEEE International Conference on Neural Networks), Houston, IEEE, 1997. PP. 1998-2001.
353. Senna A.L., Junior W.M., Carvallo M.L.B., Siqueira A.M. Neural Networks in Biological Taxonomy // Proceedings of 1993 International Joint Conference on Neural Networks, Nagoya, Japan, October 25-29, 1993.- Nagoya, 1993.- V.1.- P.33-36.
354. Stefanuk V.L. Expert systems and its applications // The lectures of Union's workshop on the main problems of artificial intillegence and intellectual systems. Part 2, Minsk, 1990.- P.36-55.

355. Sussman H.J. Uniqueness of the weigts for minimal feedforward nets wits a given input - output map. Neural Networks, 1992, No. 5. PP. 589 - 593.
356. Sweeney J.W.P., Musavi M.T., Guidi J.N. Probabilistic Neural Network as Chromosome Classifier // Proceedings of 1993 International Joint Conference on Neural Networks, Nagoya, Japan, October 25-29, 1993.- Nagoya, 1993.-V.1.- P.935-938.
357. Tabatabai A., Troudet T. P. A neural net based architecture for the segmentation of mixed gray-level and binary pictures // IEEE Trans. Circuits and Syst.- 1991. 31 38, № 1.- PP. 66-77.
358. Tao K.M., Morf M. A lattice filter type of neuron model for faster nonlinear processing // 23th Asilomar Conf. Signals, Syst. and Comput., Pasific Grove, Calif. Oct. 30-Nov. 1, 1989: Conf. Rec. Vol. 1.- San Jose (Calif.), 1989.- PP. 123-127.
359. The Adaptive Brain/ S. Grossberg (Ed.). North-Holland, 1987. V.1. Cognition, Learning, Reforcement, and Rhythm. 498 p. V.2. Vision, Speech, Language, and Motor Control. 514 p.
360. The Computer and the Brain. Perspectives of Human and Artificial Intelligence/Ed. by J.R. Brinc, C.R. Haden, C. Burava. North-Holland, 1989. 300 p.
361. Vakhrushev S.G., Rossiev D.A., Burenkov G.I., Toropova L.A. Neural network forecasting of optimal parameters of laserotherapy in patients after tonsillectomy // Proceedings of World Congress on Neural Networks - 1995 (WCNN'95).- P. 176-178.
362. Van Leeuwen J.L. Neural network simulations of the nervous system // Eur. J. Morphol.- 1990.- V.28, N.2-4.- P.139-147.
363. Varela F.J., Coutinho A., Dupire B. et al. Cognitive networks: immune, neural and otherwise // Teoretical immunology. Ed. by Perelson A.- Addison Wesley, 1988.- Part 2.- P.359-375.
364. Waxman C. Neurocomputers in the human sciences: program: predictions of US presidential elections// Modelling, Measurement & Control, D, Vol.5, No.1, 1992, pp.41-53
365. Weckert J. How expert can expert systems really be? // Libr. and Expert Syst.: Proc. Conf. and Workshop [Centre Inf. Stud.], Riverina, July, 1990.- London, 1991.- PP. 99-114.
366. Wiedermann J. On the computation efficiency of symmetric neural networks // Theor. Comput. Sci.- 1991.- 80, № 2.- PP. 337-345.
367. Wong K.Y.M., Kahn P.E., Sherrington D. A neural network model of working memory exhibiting primacy and recency // J. Phys. A.- 1991.- 24, № 5.- PP. 1119-1133.
368. Yang T.-F., Devine B., Macfarlane P.W. Combination of artificial neural networks and deterministic logic in the electrocardiogram diagnosis of inferior myocardial infarction // Eur. Heart J.- 1994.- V.15.- Abstr. Supplement XII-th World Congress Cardiology (2408).- P.449.

СОДЕРЖАНИЕ