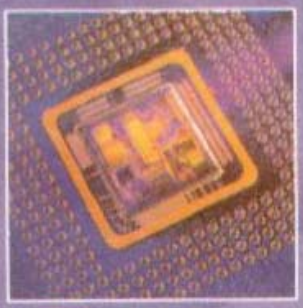
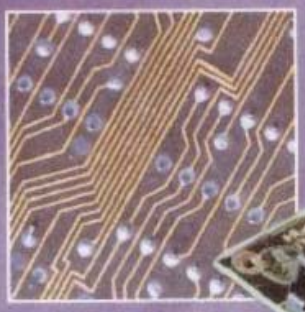
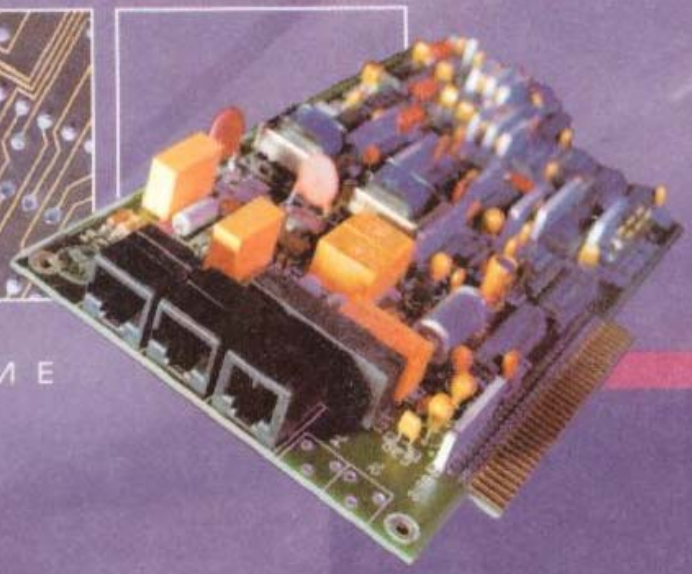
МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
УКРАИНЫ
ХАРЬКОВСКИЙ НАЦИОНАЛЬНЫЙ
ЭКОНОМИЧЕСКИЙ УНИВЕРСИТЕТ

Вдовёнков В. Ю. , Гоков А. М., Жидко Е. А.

ОСНОВЫ ЭЛЕКТРОТЕХНИКИ И ЭЛЕКТРОНИКИ. ИНТЕЛЛЕКТУАЛЬНЫЕ КОМПОНЕНТЫ НА ОСНОВЕ ИСКУССТВЕННЫХ НЕЙРОННЫХ СЕТЕЙ

5 ЧАСТЬ

УЧЕБНОЕ ПОСОБИЕ

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ УКРАИНЫ
ХАРКОВСКИЙ НАЦИОНАЛЬНЫЙ ЭКОНОМИЧЕСКИЙ УНИВЕРСИТЕТ

Вдовенков В.Ю.
Гоков А. М.
Жидко Е. А.

ОСНОВЫ ЭЛЕКТРОТЕХНИКИ И ЭЛЕКТРОНИКИ

ИНТЕЛЛЕКТУАЛЬНЫЕ КОМПОНЕНТЫ НА ОСНОВЕ ИСКУССТВЕННЫХ НЕЙРОННЫХ СЕТЕЙ

Учебное пособие
Часть 5

.

Харьков, Изд. ХНЭУ, 2008

УДК 621.3+004.89(075.8)
ББК 31.2я73
В25

Рецензенты: докт. техн. наук, профессор кафедры экспериментальной физики Харьковского национального университета им. В. Н. Каразина *Пойда В. П.*; канд. физ.-мат. наук, ст. научный сотрудник, зав. лабораторией космической радиофизики Харьковского национального университета им. В. Н. Каразина *Поднос В. А.*; канд. техн. наук, доцент кафедры автоматизации и компьютерно-интегрированных технологий Харьковского национального автомобильно-дорожного университета *Плахтеев А. П.*

Рекомендовано к изданию решением ученого совета Харьковского национального экономического университета.

Протокол №8 от 01.04.2008 г.

Вдовёнков В. Ю.

В25 Основы электротехники и электроники. Интеллектуальные компоненты на основе искусственных нейронных сетей. Учебное пособие. Ч. 5 / В. Ю. Вдовёнков, А. М. Гоков, Е. А. Жидко. – Харьков: Изд. ХНЭУ, 2008. – 268 с. (Русск. яз.)

Излагаются вопросы, связанные с искусственными нейронными сетями. Изучается функционирование нейронных сетей при решении задач аппроксимации, распознавания образов, кластеризации и "автоматической" классификации. Рассмотрены основы применения нейронных сетей и нейротехнологий. Приведен широкий круг характерных практических примеров, в том числе идей по нейроуправлению объектами, созданию многомодальных интерфейсов, сжатию сигналов.

Рекомендовано для студентов, обучающихся по профилю "Компьютеризованные технологии и системы издательско-полиграфических производств", а также для изучающих дисциплину по другим профилям подготовки.

ISBN 966-676-141-6
ISBN 966-676-142-4
ISBN 966-676-181-5
ISBN 966-676-186-6
ISBN 978-966-676-279-8

УДК 621.3+004.89(075.8)
ББК 31.2я73

Введение

Для человеческого общества является характерным стремление к познанию и преобразованию окружающего мира путем приобретения, хранения, передачи, обработки и использования информации. На протяжении своей истории человечество накапливало знания, полученные из наблюдения и опыта, предпринимало неудержимые попытки применить эти знания на практике. В настоящее время без привлечения информационных ресурсов не могут быть решены никакие задачи созидания и не удивительно, что весьма стремительным в последнее время стало развитие новых технологий в области информации.

Нельзя не обратить внимание и на то обстоятельство, что все возрастающие требования к качеству, стоимости, и доступности печатных средств информации привели к тому, что все более важную роль в общей производственной цепочке от подготовки до выпуска печатной продукции стала играть электронная техника. Она обладает уникальными возможностями хранения и обработки больших массивов информации, управления ими. Сегодня, благодаря использованию компьютеризированных систем и разнообразных инновационных технологий, стало возможным слияние традиционно обособленных в полиграфии процессов, – допечатного, печати, послепечатной обработки, – в один технологический поток производства печатной продукции. Составной частью производства печатной продукции стал цифровой информационный поток.

Нельзя не заметить и то, что процесс выпуска полиграфической продукции в настоящее время неразрывно связан с оборудованием, в различных функциональных узлах которого встроены микропроцессоры (микроконтроллеры). Компьютеризированное оборудование повсеместно применяется для обработки изображений на всех стадиях производства. Оно также, выполняя вычисления в реальном времени, преобразовывая информацию и формируя требуемые «траектории» управления обеспечивает, при изменяющихся условиях работы, наилучшее протекание технологического процесса в системах. Помимо этого, микроконтроллеры обеспечивают решение задач диагностирования, безопасности и надежности оборудования. Компьютеризация оборудования изменила существенным образом элементы интерфейса информационного канала, участвующего в процессе управления, и, в первую очередь, устройства «сбора и обработки» информации о состоянии оборудования и

окружающей среды. Использование микропроцессоров привело к развитию аппаратно-программных средств, которые осуществляют с недоступной для человека скоростью сбор, обработку, анализ и отображение информации. Вместе с тем, современные тенденции в развитии электронных человеко-машинных систем связаны, не только с устройствами, способными быстро выполнить огромное количество вычислений в секунду и сформировать точно рассчитанные в определенном направлении шаги, но и с компьютеризированными объектами, наделенными при обработке информации принципиально новыми качествами, свойственными только человеческому мышлению. Человек, как известно, способен воспринимать информацию об окружающей среде, ориентироваться в окружающей обстановке, и на основе своего интеллекта строить рассуждения, выявлять признаки, присущие чему-то, анализировать выполняемые действия, выделять из предоставляемой ему информации общее. Он способен запоминать, узнавать то, что когда-то встречал, обучаться на своих ошибках и, что очень важно, он способен самообучаться. Наиболее поразительным свойством человека является его способность принимать правильные решения в обстановке неполной и нечеткой информации. Такие облегчающие интеллектуальный труд человека системы, моделирующие его умственную деятельность, способные за счет системной обработки информации к получению «знаний», к «обучению», к выработке «рекомендаций», назвали системами с «искусственным» интеллектом. В отличие от традиционных вычислительных систем, работающих по строго определенным программам, они могут выполнять следующие специфические функции: 1. Создавать интеллектуальный интерфейс, который бы усиливал коммуникативные способности человека. Примером этому может служить использование голосового ввода команд в системах управления, выдача оператору рекомендаций по оптимальному управлению объектом, общение пользователя посредством графических образов, которые генерируются в соответствии с изменениями параметров моделируемых или наблюдаемых процессов. 2. Решать задачи, которые из-за неопределенности и динамичности исходных данных требуют построения каждый раз, в зависимости от конкретной ситуации, нового решения. Задачи подобного рода важны, когда требуется практически мгновенное реагирование на изменение внешних воздействий, быстрое принятие правильных решений в сложной обстановке, а также когда необходимо прогнозирование и предотвращение

экстремальных и непредвиденных ситуаций. 3. Автоматически извлекать «знания» из накопленного опыта, применять их для «самообучения» и дальнейшего решения задач. Самообучающиеся системы искусственного интеллекта могут «обучаться» на примерах, осуществлять автоматическую классификацию ситуаций на основе данных реальной практики.

Появление искусственного интеллекта было связано с пониманием того, что человеческий мозг это очень сложная система, которая работает иначе, чем многие технические средства и даже компьютер. Поэтому в сферу приложений искусственного интеллекта, вообще говоря, входят многие весьма сложные вопросы современных исследований в области биологии, информатики и других наук, из-за чего очень трудно «объять необъятное». Если некоторые публикации ориентированы на «сильный искусственный интеллект», на создание машин, которые копировали бы поведение человека, могли «думать» как люди, то наша книга преследует более практические цели и посвящена отдельным вопросам искусственного интеллекта, который можно назвать «слабым». В ней рассматриваются «интеллектуальные» функциональные узлы, которые действуют таким образом, что, при определенных допущениях, могут восприниматься человеком как «разумные». Такие узлы, при добавлении их в более сложные системы, могут придавать оборудованию различные «разумные свойства». Этим требованиям в первую очередь удовлетворяют искусственные нейронные сети, которые приобрели широкую популярность благодаря способности легко приспосабливаться к требованиям различных практических приложений. Поэтому в учебном пособии преследоваллись цели: объяснить базовые концепции теории нейронных сетей; изучить наиболее важные модели нейронных сетей.

Основой настоящего учебного пособия стал курс лекций по учебной дисциплине «Основы электротехники и электроники», который читается в Харьковском национальном экономическом университете студентам, обучающимся по специальностям 6.092704 «Комп’ютерізовані технології та системи видавничо-поліграфічних виробництв” и 6.092702 “Технологія мультимедійних видань”. Для учебного пособия характерно то, что в нем логически продолжается учебный материал, изложенный в первой - четвертой частях дисциплины. Это, в первую очередь, определило принцип обора материала и степень детальности освещения. При работе над содержанием пособия был сделан упор на фундаментальные знания, которые, как известно, являются основным достоинством

университетского образования. Вместе с тем, учитывая тот факт, что современная электроника является отраслью знаний чрезвычайно бурно развивающейся, как в научном, так и в техническом плане, авторы, при отборе материала, излагаемого в курсе, стремились и к тому, чтобы он отвечал современным требованиями и практическим задачам, решаемым в настоящее время. По этой причине определенная часть содержания пособия изложена нетрадиционно.

При написании пособия авторы ставили целью четко, строго и логично изложить материал учебной дисциплины в соответствии с современными стандартами высшего образования в Украине.

Учебный материал в пособии представлен таким образом, чтобы у студента не возникало необходимости обращаться к дополнительным литературным источникам. Вместе с тем, для самостоятельной работы, расширения и углубления знаний рекомендован широкий список литературы. Приведенные в каждом разделе простые и наглядные примеры и модели помогут студенту при усвоении учебного материала и приобретении умений самостоятельной работы.

Первая глава пособия посвящена изложению основ теории и общих вопросов нейронных сетей. Обсуждается то, что привело к такому важному направлению в теории интеллектуальных систем – к искусственным нейронным сетям, представляющим собой упрощенную модель человеческого биологического мозга. Затем мы рассматриваем некие абстрактные органы «машинного» зрения и основные понятия из этой области. Сюда включены необходимые сведения, касающиеся построения математических моделей и предварительной обработки информации. Рассматривается простейшая концептуальная модель, лежащая в основе искусственных нейронных сетей, которая состоит из одного обрабатывающего элемента – процессора (называемого также нейроном). Поскольку современная эра нейронных сетей начиналась с новаторской работы Мак-Каллока и Питца, в которой они описали логику вычислений в нейронных сетях на основе результатов нейрофизиологии и математической логики, то мы подробно изучим модель персептрона, названного в их честь. Для компьютерного моделирования как персептрона, так и других видов нейронных сетей, будем использовать MATLAB и графическую среду моделирования и визуального программирования Simulink. В данной главе будет продемонстрировано применение нейронной сети с персептроном для реализации логических функций И, ИЛИ, НЕ.

С инженерной точки зрения искусственные нейронные сети это параллельно распределенная система обработки информации, которая имеет свойство обучаться. В связи с этим вторая глава пособия посвящена вопросам обучения многослойных нейронных сетей. Один из важнейших алгоритмов обучения в нейронных сетях – алгоритм обратного распространения. Мы обсудим ряд задач, которые возникают при работе алгоритма обратного распространения. Затем мы рассмотрим, как нейронная сеть может накапливать экспериментальные знания, обобщать их, делать доступными для пользователей в форме, удобной для интерпретации и принятия решений. Будет продемонстрировано, как нейронная сеть, использующая алгоритм обратного распространения может быть применена для управления оборудованием.

В настоящее время существует множество вариантов нейронных сетей и обучающих алгоритмов, приспособленных к решению конкретных задач. В третьей главе мы проиллюстрируем как многослойная нейронная сеть может быть применена для решения задач аппроксимации и генерирования сигналов заданной формы. Будет показано, что если «обучение с учителем» проведено достаточно «хорошо», то нейронная сеть приобретает способность проводить «обобщение» и моделировать неизвестную функцию, связывающую значения входных и выходных переменных. Мы опишем эвристики, касающиеся того, как получить «хорошую сеть» для решения поставленной задачи, как улучшить работу алгоритма обратного распространения. Мы рассмотрим также другие разновидности алгоритмов обучения искусственных нейронных сетей.

В четвертой главе излагаются основы теории распознавания образов с обучением (с учителем). Будет показано, как нейронная сеть может относить неклассифицированные объекты (явления, ситуации, процессы) к заранее описанным классам (обучающим группам). Мы продемонстрируем, как осуществляется классификация линейно-разделимых и линейно-неразделимых множеств, распознавание буквенно-цифровых символов. Кроме того, мы покажем, как выполняется разделение множества образов с помощью нейронных сетей на основе радиальных базисных функций и классификация образов машиной опорных векторов.

Пятая глава посвящена решению задач распознавания образов без учителя и детальному изложению вопросов функционирования нейронных сетей, получивших в работе Т. Кохонена название самоорганизующихся. Мы продемонстрируем работу алгоритма SOM для двумер-

ных данных. Мы также рассмотрим нейронную сеть векторного квантования, которая может быть использована для «сжатия» данных.

В шестой главе мы рассмотрим вопросы нейродинамики и те понятия, которые свойственны динамическим системам. Мы изучим работу нейронной сети Хопфилда и продемонстрируем, как такая сеть может быть использована в качестве ассоциативной памяти. Будет показано, путем компьютерного моделирования, как нейронная сеть BSB может осуществлять кластеризацию без учителя.

В седьмой главе излагаются основы применения нейронных сетей и нейротехнологий. Эту главу мы начнем с освещения общих вопросов применения нейронных сетей. Затем мы обратимся к вопросам построения (архитектуры) «умных» датчиков, которые в литературе иногда относят к интеллектуальным устройствам. Наконец, завершая пособие, мы перейдем к вопросам электронной реализации нейронных сетей и методам аппаратной реализации нейрокомпьютеров.

Настоящее пособие, как и предыдущие книги по учебной дисциплине, может быть полезно студентам, обучающимся по другим направлениям и специальностям, а также для изучающих одноименную дисциплину по другим профилям подготовки.

Авторы благодарны за конструктивную критику и оказанную помощь в улучшении содержания учебного пособия сотрудникам кафедры физики и электроники Харьковского национального экономического университета (зав. кафедрой доктор технических наук, доцент С.И. Лапта).
Авторы выражают глубокую признательность рецензентам – доктору технических наук, профессору кафедры экспериментальной физики Харьковского национального университета имени В.Н. Каразина Владимиру Павловичу Пойде; лауреату Государственной премии УССР, кандидату физико-математических наук, заведующему лабораторией космической радиофизики Харьковского национального университета имени В.Н. Каразина Валентину Ароновичу Подносу; кандидату технических наук, доценту кафедры автоматизации и компьютерно-интегрированных технологий Харьковского национального автомобильно-дорожного университета Анатолию Павловичу Плахтееву за внимательное прочтение рукописи и высказанные полезные замечания.

1. ОСНОВЫ ИСКУСТВЕННЫХ НЕЙРОННЫХ СЕТЕЙ

1.1. Общие подходы к созданию средств искусственного интеллекта

Если следовать подходу к концептуальному представлению средств искусственного интеллекта, предложенному Н. М. Амосовым и его учениками, то во взгляде на суть (на парадигму) проблемы моделирования рассуждений и принцип ее решения в настоящее время можно говорить о следующем. Для построения механизма рассуждений в средствах искусственного интеллекта могут быть использованы два разных способа обработки информации: дедуктивный (deductive) и индуктивный (inductive). При дедуктивной обработке информации при решении задач опираются на известные аксиомы и уже доказанные теоремы, а для определения конкретных фактов используются дедуктивно-логические выводы, основанные на правилах (rule-based reasoning). В этом случае необходимо заранее сформулировать весь набор закономерностей, описывающих предметную область, сформировать, как говорят, «базу знаний (knowledge base)» и, затем, вложить ее в программную среду, осуществляющую формирование вывода на основе базы знаний. (В первом приближении, под термином «знания», используемом в системах искусственного интеллекта, можно понимать название «данные», набор фактов и «процедуры» манипулирования этими фактами). Эта парадигма лежит в основе применения экспертных систем (expert system) логического вывода. В интеллектуальных системах такого класса успех решаемой задачи зависит от хорошего представления знаний, то есть от того, насколько точно представлены информация или модели, используемые для интерпретации, предсказания и получения отклика на внешние воздействия. При этом каждый обрабатываемый объект должен быть явно указан в памяти и нужно иметь точное описание алгоритма.

При индуктивной обработке информации на основе потока экспериментальных данных и практического опыта создаются некие общие «шаблоны» переработки сведений, наборы «обучаемых элементов», которые используют полученную информацию для решения поставленной задачи. Поэтому такой, в корне отличающийся от известных методов, применяемых в цифровых компьютерах, альтернативный подход к соз-

данию интеллектуальных систем, основывается на концепции обучения на примерах (case-based reasoning), взятых из окружающего мира. В этом случае при построении интеллектуальной системы не требуется заранее знать о всех закономерностях в исследуемой области, достаточно лишь располагать достаточным количеством примеров для «настройки» системы, которая после обучения будет способна получать требуемые результаты с определенной степенью достоверности. В таких средствах искусственного интеллекта для формирования обучающей базы данных следует тщательно отбирать различные (как положительные, так и отрицательные) примеры, проводить обработку наблюдений и, затем, используя индуктивное обучение на основе обобщения сведений, накопленных в базе данных, превращать базу данных в базу знаний. При этом система, «пройдя курс обучения», должна сама, на основе новых данных, получать общие выводы из частных утверждений, выявлять степень общности той или иной зависимости между объектами наблюдения. Если о том же говорить другими словами, то следует сказать, что в системах такого класса создается так называемый «решебник задач», то есть набор уже решенных задач с заданными ответами. Причем каждая задача состоит из исходных данных (входного сигнала) и соответствующего ему желаемого отклика (desired response). Эти маркированные учебные примеры (training sample) решенных задач предъявляются системе, которая на них учится обрабатывать входные сигналы и, соответственно, решать задачи подобного класса. При обучении отклонение от правильного ответа «штрафуется», поэтому система стремится к минимизации штрафа. В конце концов, после обобщения (generalization) примеров, система «научится» получать обоснованный результат на основании данных, даже если они не встречались в процессе обучения. Идея перехода от обработки формализованных знаний к реализации более свойственных человеку приемов обработки информации, сходных с мыслительной деятельностью, привели к применению искусственных нейронных сетей (в дальнейшем, нейронных сетей). Устройства, основными элементами которых являются нейронные сети, в литературе часто называют нейрокомпьютерами, сетями связей.

Нейронные сети, в первом приближении, представляют устройства параллельных вычислений, состоящие из множества соединенных между собой некоторым образом и взаимодействующих обрабатывающих элементов (процессоров). Каждый процессор имеет дело только с сиг-

налами, которые он получает, и сигналами, которые он посылает другим процессорам. Как оказалось, будучи соединенными в достаточно большую сеть с управляемым взаимодействием, эти простые процессоры вместе способны решать довольно сложные задачи.

Нейронные сети обладают рядом преимуществ, основные из которых следующие: а) они способны адаптироваться (приспосабливаться) к изменениям окружающей среды; б) они устойчивы к шумам, поэтому искажения входных сигналов значительно слабее влияют на получаемый результат; в) они более отказоустойчивы; г) они могут работать со скрытой информацией, с объектами, которые ранее не встречались.

Нейронные сети различаются между собой моделями нейрона, топологией связей между нейронами и правилами обучения.

1.2. Основные понятия и специальные средства обработки сигналов искусственных нейронных сетей

1.2.1. Одноэлементный процессор и обеспечивающие его компоненты

Искусственные нейронные сети (neural network) имитируют работу мозга и представляют его упрощенную модель. Мозг состоит из нейронов, которые соединены между собой с помощью нервных окончаний. Подобно мозгу искусственная нейронная сеть является совокупностью индивидуальных элементов (компонентов), соединенных некоторым образом так, чтобы между ними образовывались сложные, нелинейные связи и обеспечивалось взаимодействие посредством обмена сигналами. Эти индивидуальные компоненты, называемые также нейронами или узлами, представляют собой некие обрабатывающие элементы (процессоры), вычислительные возможности которых обычно определяются некоторым набором правил комбинирования входных сигналов и «законом активизации», позволяющим вычислить выходной сигнал по совокупности входных сигналов. Структурная схема модели нейрона, лежащего в основе многих искусственных нейронных сетей, показана на рис. 1.1. Не вникая пока в детали, отметим, что это так называемый одноэлементный персептрон (perceptron) – модель, состоящая из одного процессора. Структурная схема одноэлементного процессора, помимо самого нейро-

на, содержит большое количество вспомогательных блоков, в которых, на первый взгляд, трудно ориентироваться. Поэтому данную схему мы начнем рассматривать последовательно, двигаясь слева направо.

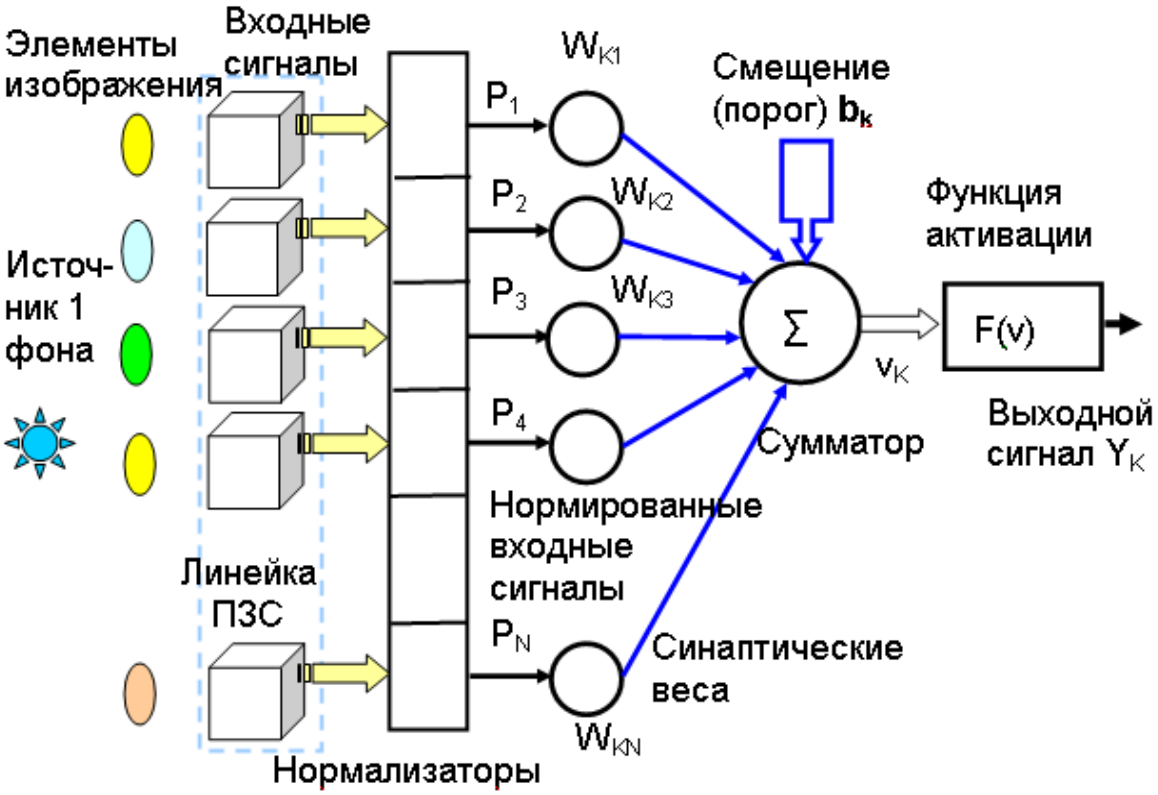


Рис. 1.1. **Структурная схема модели нейрона, получающего входные сигналы от линейки фотоприемников с зарядовой связью**

Во-первых, в левую часть схемы мы ввели несколько блоков, которые поясняют, какие сигналы поступают на вход нейрона. Известно, что человек получает информацию об окружающем мире почти на 90 % виде видеоинформации, имеющейся в спектральном диапазоне 380 – 800 нм. Поэтому мы сначала предположили, что некая информация об окружающей среде в виде изображения поступает на фоточувствительную линейку сенсоров (датчиков изображения) – приборов с зарядовой связью (ПЗС). В настоящее время сенсоры изображений (image sensors) широко используются в электронике. Они являются главным элементом цифровых фотоаппаратов, сканеров, широко применяются в функциональных узлах полиграфического оборудования.

Во-вторых, мы предположили, что изучаемое изображение в виде некой «сцены» состоит из конечного числа элементов, каждый из кото-

рых расположен в конкретном месте и имеет определенную интенсивность света. Если говорить о формировании изображений с общих позиций, то можно отметить, что регистрируемый сенсором (или совокупностью сенсоров одновременно) сигнал возникает в результате взаимодействия источника «освещения» с элементами изображаемой «сцены» в условиях эффектов отражения и поглощения энергии этого источника. Из курса физики известно, что если тело отражает солнечный свет приблизительно одинаково во всем видимом диапазоне длин волн, то оно представляется наблюдателю белым. С другой стороны, когда тело отражает цвет в каком-то ограниченном диапазоне длин волн, то оно воспринимается с некоторым цветовым оттенком. Например, предмет зеленого цвета, в основном, отражает свет с длинами волн 500 – 570 нм, поглощая большую часть энергии в других интервалах волн. Оптическое излучение, «лишенное» цветовой окраски, называется ахроматическим. Единственным параметром такого освещения является его интенсивность, или яркость. Для описания яркости в этом случае используется термин уровень серого, поскольку в таком случае яркость изменяется от черного до белого, с промежуточными серыми оттенками. Не вникая в вопросы того, как отражается свет от поверхностей элементов изображения, не описывая детали преобразования энергии «освещения» в изображение, будем полагать, что в нашем случае изображение «сцены» формируется за счет того, что создается тестовая картина из источников оптического излучения, управляемая по яркости. Наблюдаемая сцена формируется некими идеализированными маленькими лампочками или светодиодами, помещенными в металлический чехол с отверстиями разного диаметра и создающими черно-белое изображение «сцены». От величины этого отверстия зависит яркость (уровень серого) оптического излучения. При этом яркость элементов изменяется от черного до белого, с промежуточными серыми оттенками. В общем случае изображение представляет собой двумерную функция $f(x, y)$, где x и y - это пространственные координаты, а представленная в любой точке в виде некоторого численного значения скалярная величина $f(\cdot\cdot)$, с парой координат, – интенсивность (яркость) элемента изображения.

В-третьих, обработка картины изображения, как показано на рис. 1.1, начинается с регистрации (измерения) яркости элементов изображения с помощью ПЗС линейки, представляющей собой одномерный

массив сенсоров, располагаемых вдоль прямой. Эти элементы принято называть элементами изображения или пикселами (от английского словосочетания – PICture ELement). Энергия, излучаемая пикселами сцены, фиксируется с помощью каждого сенсора ПЗС и преобразуется в заряд (напряжение). В ответ на энергию внешнего оптического излучения элементы ПЗС выдают выходные электрические сигналы $\alpha_1, \alpha_2, ..., \alpha_N$. Следует при этом иметь в виду, что в процессе обработки «сцены» изображения с помощью ячеек ПЗС, имеющих конечные размеры, непрерывное пространственное распределение энергетической освещенности на плоскости, приобретает дискретный характер. Информация об изображении, размещенном на плоскости, преобразуется в кадр – двумерный массив чисел α_{MN}. При этом обрабатываемый элемент изображения уже не является мельчайшей точкой изображения – пикселем. Это уже некая прямоугольная область. По этой причине физический смысл чисел α_{MN} – это значение выходного сигнала ячейки ПЗС, которое характеризует уже среднюю энергетическую освещенность. Каждому значению пары индексов MN соответствует соотнесенная с изображением поверхность в пределах поля изображения, отображаемая на один сенсор ПЗС. Подобная дискретизация определяет пространственное разрешение – размер мельчайших различимых деталей на изображении.

Линейка ПЗС, состоящая из конечного числа соотнесенных с элементами изображения сенсоров, обеспечивает одновременную регистрацию дискретных элементов изображения в одном направлении (условно говоря, по столбцу матрицы, как показано на рис. 1.1). Получить все столбцы изображения, то есть массив элементов на плоскости, позволяет перемещение всей линейки в перпендикулярном направлении – вдоль строки (подобная конструкция применяется в большинстве планшетных сканеров). В общем случае, изображение может сразу проецироваться на набор сенсоров в виде двумерного массива (ПЗС-матрицы). В таком случае отдельные сенсоры изображения расположены в дискретном виде на прямоугольной сетке, подобно матрице, у которой имеется M строк и N столбцов. Несложно догадаться, что изображенная на рис. 1.1 ПЗС линейка, состоящая из чувствительных к оптическому излучению сенсоров, упорядоченных в виде столбца матрицы, представляет частный случай матрицы изображения размером *1x*N. По этой причине такое расположение сенсоров можно назвать вектором-столбцом.

В-четвертых, если полагать, что во время регистрации «сцена» изображения не меняется, то на выходе линейки ПЗС возникнет «моментальный снимок изображения». Иными словами, возникнет сигнал в виде набора данных о яркости, элементы которого, порождены изображением в различных точках пространства. Этот дискретный сигнал, в соответствии с рис. 1.1, образует столбец значений и представляет собой упорядоченную последовательность чисел. Элементы сигнала можно также записать в виде строки $\{\alpha_1, \alpha_2, \alpha_3, ..., \alpha_N\}^T$ (буква T обозначает транспонирование – замену столбца на строку с сохранением порядка следования). С математических позиций дискретный сигнал подобного вида, сформированный на выходе линейки ПЗС, можно характеризовать общим и широко используемым понятием «вектора» конечномерного пространства, имеющего конечное число «координат». Как известно, в математике вектором u_N размерности N (или N-мерным вектором) в конечномерном пространстве называется упорядоченная совокупность из N чисел, расположенных в определенном порядке, при которой допускается повторение одного и того же числа, но не допускается изменения положения числа. При этом, если u_N – вектор, определяемый числами $\alpha_1, \alpha_2, ..., \alpha_N$, то его обозначают $u_N = \alpha_1, \alpha_2, ..., \alpha_N$. Числа $\alpha_1, \alpha_2, ..., \alpha_N$ называются координатами (компонентами) вектора. Здесь N – количество координат вектора. Таким образом, сигналы, формируемые на выходе линейки ПЗС, являются неким N-мерным вектором, координатами которого являются упорядоченные совокупности из N чисел значений сигналов $u = \{\alpha_1, \alpha_2, ..., \alpha_N\}$, отображающих яркости элементов изображения. При этом количество коэффициентов $\alpha_1, \alpha_2, ..., \alpha_N$, определяющих размерность конечномерного пространства над полем действительных чисел R, соответствует числу ячеек ПЗС.

В-пятых. Функционирование линейки ПЗС или ПЗС-матрицы, преобразующей изображение в электрический сигнал, вообще говоря, может происходить в условиях, когда яркости отдельных пикселей могут существенно различаться. Так, в ясный день солнце создает на земной поверхности освещенность 90000 лм/м^2, а в пасмурную погоду эта величина падает до 10000 лм/м^2. В типичных служебных помещениях поддерживается уровень освещенности 1000 лм/м^2. Типичные значения коэффициента отражения составляют: для черного бархата – 0,01; для

нержавеющей стали – 0,65; для поверхности стены, окрашенной в ровный белый цвет – 0,8. В типичной реальной ситуации при формировании «сцены» изображения диапазон яркостей отдельных пикселей может быть очень большим. Зрительная система человека способна адаптироваться к этому огромному, порядка 10^{10}, диапазону значений яркости – от порога чувствительности зрения до предела ослепляющего блеска. Фотоприборы, реагирующие на оптическое излучение, за счет принятия определенных технических мер, также способны «сжимать» динамический диапазон яркостей изображений, чтобы наиболее яркие пиксели не доминировали над слабыми (зачастую очень важными при анализе информации, особенно при обработке мелкомасштабных изображений). Нейронная сеть, как будет следовать из дальнейшего, также может работать только с числовыми данными, лежащими в определенном ограниченном диапазоне. Учитывая сказанное на структурной схеме модели нейрона (рис. 1.1), получающего входные сигналы от линейки фотоприемников с зарядовой связью, введены функциональные узлы – нормализаторы, осуществляющие предварительную обработку исходной информации (часто говорят, предобработку данных), поступающей с линейки ПЗС. Предварительная обработка данных в этих функциональных узлах не только отражает возможность работы с данными, диапазоны изменения которых могут существенно различаться, но и, как мы увидим в дальнейшем, направлена на то, чтобы обеспечить достоверность вычислений в нейроне, повышение точности его работы и, в конечном счете, на повышение эффективности функционирования нейронной сети.

Функциональные узлы, именуемые на рис. 1.1 как нормализаторы, обычно выполняют две основные функции: 1) масштабирование данных, 2) статистическую обработку данных. Числовые значения сигналов, поступающие с линейки ПЗС, должны быть приведены в масштаб, подходящий для работы сети. Чаще всего входные данные должны лежать в диапазоне либо между 0 и 1, либо между -1 и 1. Поэтому цель масштабирования (иногда говорят шкалирования) состоит в том, чтобы каждая компонента вектора сигнала $\alpha_1, \alpha_2, ..., \alpha_N$, поступающего с линейки ПЗС, лежала в отрезке $[-1,1]$ ($[0,1]$), или, по крайней мере, не слишком далеко выходила из этого отрезка. Простейшим методом масштабирования является деление значения каждой компоненты сигнала

$u(n)=\{\alpha_1,\alpha_2,...,\alpha_N\}$, $n=\overline{1,N}$, на норму сигнала. Наиболее общей в n-мерном координатном пространстве $l^P(N)$ для сигналов $u(n)=\{\alpha_1,\alpha_2,...,\alpha_N\}$, представляющих кодовую комбинацию из N упорядоченных чисел, является, как известно из математики, норма: $\|u(n)\|_{l^P}=\left[\sum_{k=0}^{N-1}|u(n)|^P\right]^{1/P}$ ·На практике чаще всего используют варианты p = 1, p = 2 и $p=\infty$, для которых нормы конечномерного сигнального пространства принимают, соответственно, следующий вид: $\|u(n)\|_{l^1}=\left[\sum_{k=0}^{N-1}|u(n)|\right]$, $\|u(n)\|_{l^2}=\sqrt{\sum_{k=0}^{N-1}|u(n)|^2}$ и $\|u(n)\|_{l^\infty}=\max|u(n)|$. При масштабировании широко используется норма третьего типа $\|u(n)\|_{l^\infty}=\max|u(n)|$, которая определяет максимальное (пиковое) значение сигнала. В этом случае для масштабирования вектора сигнала $\alpha_1,\alpha_2,...,\alpha_N$, поступающего с линейки ПЗС, осуществляется деление значения каждой координаты сигнала $u(n)=\{\alpha_1,\alpha_2,...\alpha_N\}$ на максимальное значение α_{MAX}, содержащееся в векторе сигнала $\alpha_1,\alpha_2,...\alpha_N$.

$P_N=\frac{u(n)}{\|u(n)\|}=\frac{u(n)}{\max|\alpha_n|}=\frac{u(n)}{\alpha_{MAX}}, n=\overline{1,N}$, где P_N – масштабированное значение компоненты сигнала (после нормализатора). Каждая компонента сигнала может быть масштабирована и к другому диапазону, например, [0.1–0.9], с помощью следующей формулы: $P_N=\frac{0.9-0.1}{\alpha_{MAX}-\alpha_{MIN}}\alpha_N+\left(0.9-\frac{0.9-0.1}{\alpha_{MAX}-\alpha_{MIN}}\alpha_{MAX}\right)$, где P_N – масштабированное значение координаты сигнала (после нормализатора); α_N – первоначальное значение компоненты сигнала (на входе нормализатора). Например, предположим, что две компоненты сигнала имеют значение 2 и 20. Тогда в результате значение 2 перейдет в значение $P_N=[(0.9-0.1)/(20-2)]2+\left(0.9-[(0.9-0.1)/(20-2)]20\right)$ =0,1. Значение 20 отобразится в значение $P_N=\frac{0.9-0.1}{20-2}20+\left(0.9-\frac{0.9-0.1}{20-2}20\right)$=0,9.

В MATLAB в составе пакета «Нейронные сети» (Neural Networks Toolbox) имеются специальные средства (функции), которые осуществляют масштабирование данных, приводя их к промежутку $[-1,1]$. Так, например, если на выходе ПЗС имеются компоненты двух сигналов $u1=\{\alpha_1,\alpha_2,...,\alpha_{10}\}$ и $u2=\{\alpha_1,\alpha_2,...,\alpha_{10}\}$, то, используя оператор premnmx(u1,u2), можно получить их отображение в промежуток$[-1,1]$.

Пусть векторы входных сигналов имеют вид:

u1 = [0 7 5 4 0.5 2.5 5 7.5 10 8];

u2 = [0.1 8 10 5 0.4 4 20 16 20 12].

Вводим в командное окно вместе с векторами оператор:

[p1n,minu1,maxu1,p2n,minu2,maxu2] = premnmx(u1,u2).

После проведения вычислений значения компонент первого масштабированного вектора сигнала будут:

p1n = -1.0 0.4 0 -0.2 -0.9 -0.5 0 0.5 1.0 0.6 .

При этом для сигнала $u1$ максимальные и минимальные значения равны: minu1 = 0 maxu1 = 10.

Аналогично для компонент второго масштабированного вектора сигнала:

p2n = -1.0 -0.206 -0.005 -0.5075 -0.9698 -0.6080 1.0 0.598 1.0 0.196

minu2 = 0.1 maxu2 = 20.

Обычно изображение содержит постоянную составляющую (среднее значение яркости), которая не несет полезной информации об изображении, а отражает лишь общий фон освещения, создаваемый на все элементы сенсора изображения часто посторонним, побочным источником освещения (на рис. 1.1 это показано в виде источника фона – лампочки, которая создает одновременную «засветку» всех элементов ПЗС). Постоянная составляющая может также возникать из-за особенностей конструкции ПЗС. Нормализатор, как правило, удаляет содержащуюся в векторе сигнала $\alpha_1,\alpha_2,...,\alpha_N$ постоянную составляющую. Удаление среднего значения из каждой компоненты с индексом k, принимающим значение от 1 до N, может быть осуществлено с помощью формулы: $\alpha_{k0}=\alpha_k-\text{¢},k=\overline{1.N}$, где ¢- среднее значение, которое определяется формулой: $\text{¢}=\frac{1}{N}\sum_{k=1}^{N}\alpha_k$.

1.2.2. Статистическая обработка сигналов нейрона

При сканировании изображения могут иметь место вибрации, акустические шумы. Из-за этого, пусть даже очень малого изменения расстояний между элементами изображения и сенсорами, возникнут случайные флуктуации в формируемом сигнале. Ячейка ПЗС, исходя из принципа своего действия, также создает физические «шумы» аппаратуры. По этим причинам сигналы, формируемые ПЗС в ответ на изображение, наряду с полезной информацией содержат шумовую составляющую. «Шумы» приводят к тому, что измеренная величина будет иметь случайную погрешность, которую трудно предсказать. Поэтому нормализатор, показанный на структурной схеме, наряду с масштабированием, осуществляет вторую функцию – статистическую обработку всех подобных случайных погрешностей с целью уменьшения их влияния.

Знакомство с методами статистической обработки начнем с простого примера. Для этого предположим, как и ранее, что изображение не меняется. Однако при этом будем исходить из того, что изображение в силу ряда причин флуктуирует, причем отклонения яркости будут больше и меньше среднего значения с одинаковой вероятностью. Если при этом мы многократно проведем «считывание» яркости изображения, то каждая I-я компонента α_I сигнала, многократно измеренная через равномерные промежутки времени, на выходе элемента ПЗС, образует строку, называемую выборкой $\widetilde{\alpha}_{I1},\widetilde{\alpha}_{I2},...,\widetilde{\alpha}_{IL}$. Индекс L соответствует числу проведенных измерений (наблюдений) яркости одного и того же элемента изображения во временной области. По этим данным мы хотим определить наилучшее значение α^O, которое давало бы возможно меньшую по абсолютной величине разность $\alpha^O - \alpha$ с истинным значением α. Было доказано, в предположении, что отсутствуют «грубые» ошибки, что наилучшей оценкой истинного значения α является среднее выборочное или просто среднее α^O. Простейший метод определения приближенного значения α^O состоит в том, что в качестве α^O выбирают среднее арифметическое значение α^O_{CP} (выборочную оценку математического ожидания $M(\widetilde{\alpha}_{IL})$) чисел $\alpha^O_{CP} = M(\widetilde{\alpha}_{IL}) = \dfrac{\widetilde{\alpha}_{I1} + \widetilde{\alpha}_{I2} + \cdots + \widetilde{\alpha}_{IL}}{L}$.

Формулу, используемую для вычисления приближенного значения α^O, называют оценкой этой величины.

В качестве показателя воспроизводимости (стабильности) измерений каждой компоненты полезным оказывается параметр – разброс значений параметра вокруг среднего или отклонение величины от среднего – это $\alpha_{IL} - \alpha_{CP}^O$. Так как отклонение может быть как положительным, так и отрицательным, то целесообразно использовать квадрат отклонения. Среднее значение квадратов отклонений называют дисперсией. Для получения значения дисперсии той же размерности, что и исходные измерения, требуется взять квадратный корень. Полученная величина называется среднеквадратическим отклонением. Выборочные оценки дисперсии и среднего квадратичного отклонения для сигнала, измеренного во временной области, могут быть получены, соответственно, по формулам: $\sigma^2 = \frac{1}{L-1}\sum_{k=1}^{L}(\alpha_{IK} - \alpha_{CP}^O)^2$, $d = \sqrt{\frac{1}{L-1}\sum_{k=1}^{L}(\alpha_{IK} - \alpha_{CP}^O)^2}$. Следует обратить внимание на то, что выборочные оценки могут быть улучшены путем увеличения числа сделанных измерений L. В теории вероятностей доказывается, в частности, что при неограниченном увеличении L оценка α_{CP}^O стремится к некоторому пределу $M(\tilde{\alpha}_{IL})$ – математическому ожиданию случайной величины. Стандартные преобразования исходной I-й выборки α_{IK}, $k = 1, 2, \ldots, L$, при статистической обработке данных состоят в том, что имеющиеся значения сигнала преобразуются с помощью вычитания среднего и деления результата на значение среднеквадратичного отклонения для данной выборки $P_{IK} = \frac{\alpha_{IK} - \alpha_{CP}^O}{\sigma(\alpha_{IK})}$. Из этой формулы несложно заметить, что любое изменение выборки сопровождается изменением компоненты вектора сигнала.

В MATLAB в составе пакета «Нейронные сети» имеется специальная функция, которая осуществляет статистическую обработку векторов входов, приводит их к нормальному закону распределения. При этом расчеты случайных векторов осуществляются таким образом, чтобы они имели среднее значение (математическое ожидание) стремящееся к нулю, а среднеквадратичное отклонение для данной выборки, приближающееся к единице.

Так, например, если на выходе ПЗС имеются два сигнала, $u1 = \{\alpha_{11}, \alpha_{12}, ..., \alpha_{15}; \alpha_{21}, \alpha_{22}, ..., \alpha_{25};\}$ и $u2 = \{\alpha_1, \alpha_2, ..., \alpha_{10}\}$, причем первый сигнал получен в результате двух выборок то, используя оператор prestd(u1,u2), можно получить их образы, имеющие почти нормальное распределение с заданными параметрами. Пусть векторы входных сигналов имеют вид:

u1 = [-0.3 -1.28 0.244 1.27 1.198; 1.73 -2.18 -0.234 1.09 -1.09];

u2 = [0.61 -1.4 0.76 0.82 1.1].

Вводим в командное окно.

[pn1,meanp1,stdp1,pn2,meanp2,stdp2] = prestd(u1,u2).

Тогда первый статистически обработанный с помощью функции prestd(u1,u2) вектор, состоящий из двух выборок, уже распределенный почти по нормальному закону, на выходе нормализатора будет иметь значения:

pn1 = -0.4920 -1.4078 0.0164 0.9753 0.9080
1.1758 -1.2869 -0.0612 0.7727 -0.6004.

При этом его выборочные оценки средних значений (математических ожиданий) будут равны meanp1 = 0.2264 -0.1368, а выборочные оценки среднеквадратичного отклонения, соответственно, stdp1 = 1.0700 1.5877.

Аналогично для второго сигнала после нормализатора получим:

pn2 = 0.2298 -1.7610 0.3783 0.4378 0.7151

meanp2 = 0.3780 stdp2 = 1.0097.

Таким образом, резюмируя сказанное, отметим, что линейка ПЗС и нормализаторы создают математический «образ» изображения, который представляет собой, с точки знания математики, полученный по определенным правилам вектор в сигнальном пространстве.

Рассматривая блоки, представленные на рис. 1.1 слева, следует также учитывать два обстоятельства. Первое: мы добавили к нейрону и рассмотрели в качестве примера блоки преобразования изображения в вектор в сигнальном пространстве лишь потому, что, как мы полагаем, эти блоки являются типичными в полиграфическом оборудовании. Но аналогично могут быть построены и другие подобные системы. Например, подобным образом функционируют измерительные системы, которые с помощью плат сбора информации преобразуют сигналы датчиков в векторы сигнального пространства. Конечно, построение и функциони-

рование таких измерительных систем могут отличаться в деталях от изученного, но, и это важно, в данный момент времени рассматриваемый объект будет также характеризоваться набором величин («моментальным снимком данных») и, в конце концов, на выходе измерительной системы появятся компоненты, представляющие вектор в сигнальном пространстве, обработанный соответствующим образом.

Второе: для цельности рассмотрения вопроса отметим также, что частным случаем в системах искусственного интеллекта могут быть данные, которые изначально представляют собой, как говорят, бинарные векторы с координатами равными либо 0 либо 1. Ситуации, когда приходится работать с такими сигналами, довольно часты. В таких случаях проводить масштабирование, статистическую обработку таких сигналов обычно не целесообразно.

Наконец, после того, как мы предпослали основному материалу подробное изложение ряда важных вопросов получения, преобразования, масштабирования и статистической обработки сигналов $\{P_K\}$, поступающих к нейрону, перейдем к более подробному изучению самого нейрона. С этой целью рассмотрим детальнее устройство самого нейрона, представленного в правой части рис. 1.1, и принцип его действия.

1.3. Устройство нейрона

1.3.1. Простейшая модель искусственного нейрона

Искусственный нейрон (neuron) представляет собой некую «единицу» обработки информации в нейронной сети. Его простейшая модель (рис. 1.2), лежащая в основе искусственных нейронных сетей, состоит из одного обрабатывающего элемента – процессора, (называемого также узлом), задача которого состоит в том, чтобы осуществить, по определенным правилам, комбинирование посылаемых к нему входных сигналов $\{P_N\}$ и, в соответствии с правилами активации, вычислить выходной сигнал Y_K. На вход нейрона поступает множество сигналов (входной вектор сигналов $\{P_K\}$, $k = \overline{1, N}$). Эти сигналы перераспределяются в соответствии с набором связей (connecting link). Далее, каждая компонента вектора $\{P_K\}$ умножается на число $\{W_K\}$, называемое весом (weight). В

отличие от мозга, вес искусственного нейрона может иметь как положительные, так и отрицательные значения. Затем, все полученные путем «взвешивания» компонент входного сигнала произведения складываются в линейном сумматоре (adder), определяя, тем самым, потенциал активации (уровень активации) нейрона v_K. В модель нейрона включен пороговый элемент (bias), который обозначен символом b_k. Он увеличивает или уменьшает полученный в сумматоре сигнал.

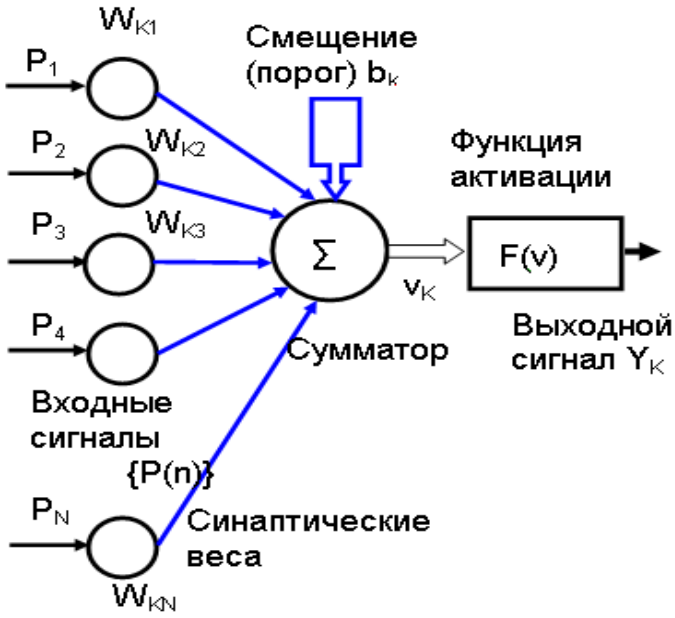


Рис. 1.2. **Формальная модель искусственного нейрона**

1.3.2. Функционирование искусственного нейрона

Комбинирование входящих сигналов, путем суммирования их взвешенных значений, с математической точки зрения, эквивалентно скалярному произведению векторов входа $\{P_K\}$ и весовых коэффициентов нейрона $\{W_K\}$. Если учесть и так называемый порог b_k («нейронное смещение»), вводимый при создании сети, то уровень (потенциал) активации нейрона (часто используют также термин – индуцированное локальное поле) можно представить в виде линейной комбинации входных воздействий $u_k = \sum_{k=1}^{N} P_k W_k$ и порога $v_K = \sum_{k=1}^{N} P_k W_k + b_k = u_k + b_k$. Например, если порог b_k = 0, вектор сигнала P = {0.7 0.1 0.3}, а весовые коэффициенты нейрона W = {-0.3 3.1 0.5}, то результат расчета уровня ак-

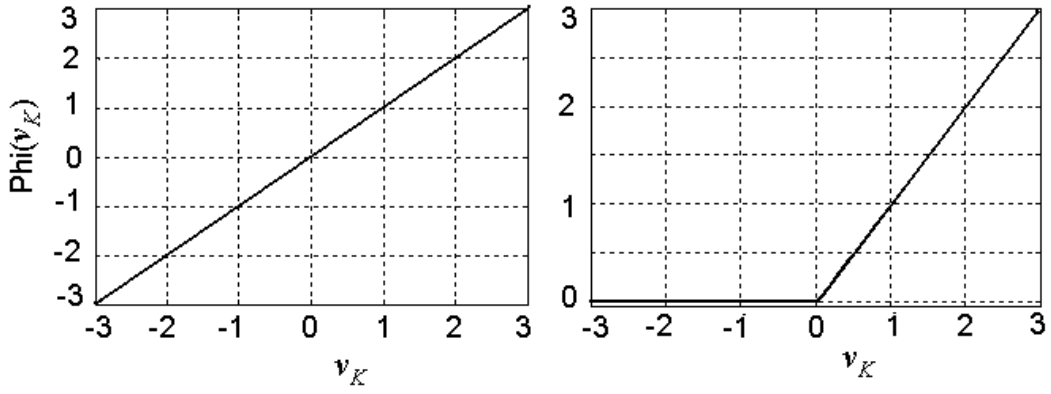
тивации нейрона: v_K=(0.7x-0.3)+(0.1x3.1)+0.3x0.5)=0.25. В векторном представлении это будет иметь следующий вид: $v_K = PW^T$,

$$v_K = \begin{bmatrix} 0.7 & 0.1 & 0.3 \end{bmatrix} \begin{bmatrix} -0.3 \\ 3.1 \\ 0.5 \end{bmatrix} = \begin{bmatrix} 0.7 & 0.1 & 0.3 \end{bmatrix} \cdot \begin{bmatrix} -0.3 & 3.1 & 0.5 \end{bmatrix}^T = 0{,}25.$$

Использование порога b_k обеспечивает эффект, как говорят математики, «аффинного преобразования» выхода линейного сумматора. В зависимости от того, какое значение принимает порог b_k, положительное или отрицательное, потенциал активации нейрона v_K (индуцированное локальное поле) увеличивается или уменьшается и график зависимости $v_k = \varphi(u_k)$, который при $b_k = 0$ имеет вид линейной зависимости, параллельно смещается вверх или вниз на величину порога. Порог b_k является внешним параметром нейрона κ. По этой причине его можно «трансформировать» таким образом, будто бы к нейрону добавился еще один входной сигнал P_0 = 1 с весом равным порогу W = b_k. В таком случае уровень (потенциал) активации нейрона v_K (индуцированное локальное поле) будет определяться выражением: $v_K = \sum_{k=0}^{N} P_k W_k$.

У искусственного нейрона имеется правило вычисления выходного значения Y_K, которое предполагается передать во внешнюю (по отношению к нейрону) среду. Это правило называют функцией активации (activation function) нейрона $F(v_K)$, а соответствующее выходное значение называют активностью нейрона. На вход функции активации поступает значение $v_K = \sum_{k=0}^{N} P_k W_k$. Функция активации, представленная математически как $F(v_K)$, в зависимости от потенциала его активации (от индуцированного локального поля) v_K, определяет выходной сигнал нейрона. В моделях нейронных сетей используются различные активационные функции $F(v_K)$. Наиболее простой является линейная функция активации нейрона, имеющая вид прямой, проходящей через начало координат. Она определяется формулой: $Y_K = F(v_K) \Leftrightarrow Y_K = Kv_K$,

где K – угловой коэффициент в уравнении прямой. При K =1 функция становится тождественной. Это означает, что посылаемый другим элементам сигнал активности нейрона оказывается равным уровню (потенциалу) активации нейрона v_K, или по-другому, индуцированному локальному полю (линейной комбинации входных воздействий $u_k = \sum_{k=1}^{N} P_k W_k$ и порога). В MATLAB линейная функция активации рассчитывается с помощью функции a = purelin(v). График линейной функции активации, построенный с помощью функции a = purelin(v), показан на рис. 1.3а.



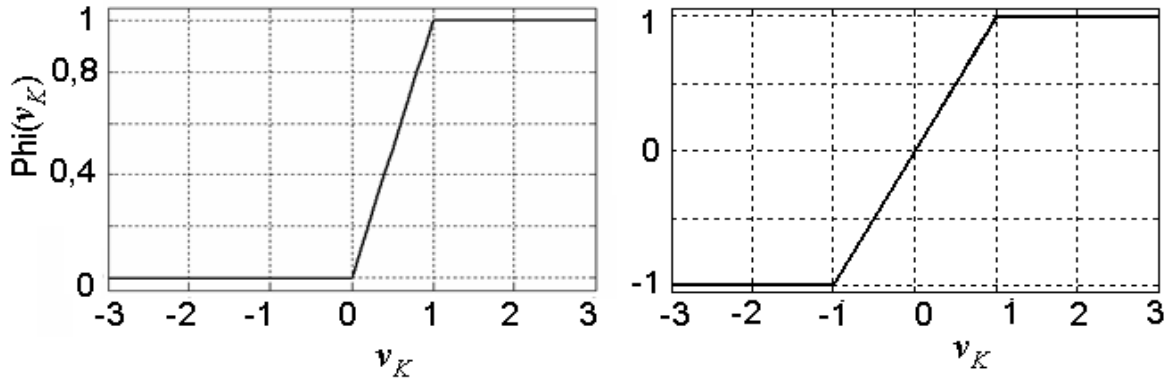
а) б)

Рис. 1.3. **Линейная и полулинейная функций активации нейрона**

В моделях нейронных сетей используется положительная линейная (полулинейная) функция активации нейрона. Она определяется формулой: $Y_K = F(v_K) \Leftrightarrow Y_K = \begin{cases} 0, & v_K < 0 \\ v_{K,} & v_K \geq 0 \end{cases}$. В MATLAB полулинейная функция активации рассчитывается с помощью функции a = poslin(v). График полулинейной функции активации нейрона, построенный с помощью функции a = = poslin(v), показан на рис. 1.3б.

В искусственных нейронных сетях используется также кусочно-линейная или, чаще именуемая как линейная с ограничениями функция активации нейрона. Она определяется следующей формулой: $Y_K = F(v_K) \Leftrightarrow Y_K = \begin{cases} 0, & v_K < 0; \\ v_K & 1 \geq v_K \geq 0; \\ 1 & v_K > 0. \end{cases}$. В MATLAB кусочно-линейная функция активации рассчитывается с помощью функции a = satlin(v). График

линейной с ограничениями функции активации нейрона, построенный с помощью функции a=satlin(v);показан на рис. 1.4а.



а) б)

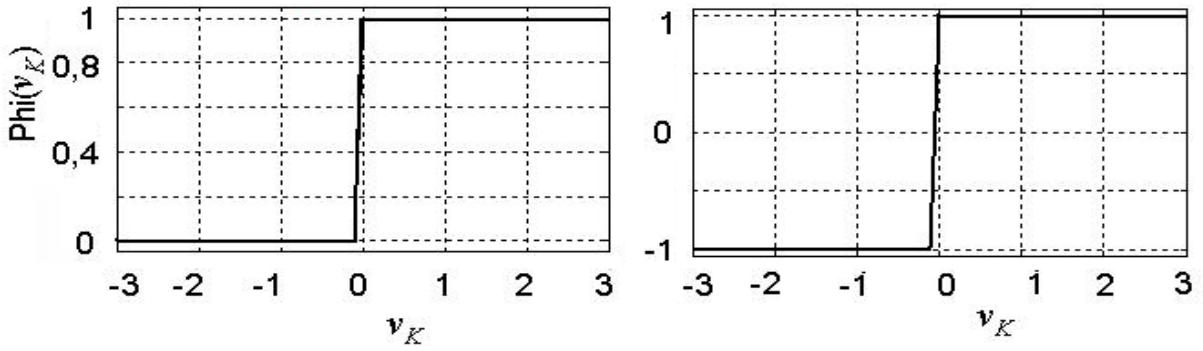
Рис. 1.4. **Графики линейной с ограничениями (а) и симметричной линейной с ограничениями (б) функций активации нейрона**

Область значений линейной с ограничениями функции активации, когда она подобна тождественной, представляет собой отрезок от 0 до 1. Иногда требуется такая функция активации, чтобы она была симметричной относительно начала координат, а также имела область значений функции активации, при которых функция подобна тождественной, в диапазоне от -1 до +1. Функцией активации нейрона, которая является нечетной функцией индуцированного локального поля и удовлетворяет названным требованиям, является симметричная линейная с ограничениями. Она определяется формулой: $Y_K = F(v_K) \Leftrightarrow Y_K = \begin{cases} -1, & v_K < 0; \\ v_K, & 1 \geq v_K \geq 0; \\ 1, & v_K > 0. \end{cases}$ В MATLAB симметричная линейная с ограничениями функция активации рассчитывается с помощью функции a = satlins(v). График симметричной линейной с ограничениями функции активации нейрона, построенный с помощью функции a = satlins(v), показан на рис. 1.4б.

Заметим, что симметричная линейная с ограничениями функция активации по виду напоминает амплитудную характеристику электронного нелинейного усилителя, работающего с сигналами как малой, так и большой амплитуды. Пока потенциал активации (индуцированное локальное поле) v_K нейрона не достигнет порога насыщения, функция активации является тождественной, что означает, что посылаемый другим

элементам сигнал активности нейрона оказывается равным индуцированному локальному полю (линейной комбинации входных воздействий $u_k = \sum_{k=1}^{N} P_k W_k$ и порога) нейрона. В этом случае нейрон работает как линейный сумматор (сумматор на операционном усилителе). После достижения потенциалом активации пороговой величины ±1 линейные комбинации входных сигналов будут создавать одно и то же значение активности нейрона из-за «насыщения» нейрона.

Во многих моделях нейронных сетей используются существенно нелинейные функции активации нейрона. Одной из таких функций является униполярная пороговая функция (threshold function). Ее также называют функцией единичного скачка. В технической литературе эту функцию именуют функцией Хэвисайда (Heaviside function). Униполярная пороговая функция определяется формулой $Y_K = F(v_K) \Leftrightarrow Y_K = \begin{cases} 0, & v_K < 0 \\ 1, & v_K \geq 0 \end{cases}$, вследствие чего выходной сигнал нейрона может принимать два значения – 0, 1. В MATLAB пороговая функция активации рассчитывается с помощью функции a = hardlim(v).



а) б)

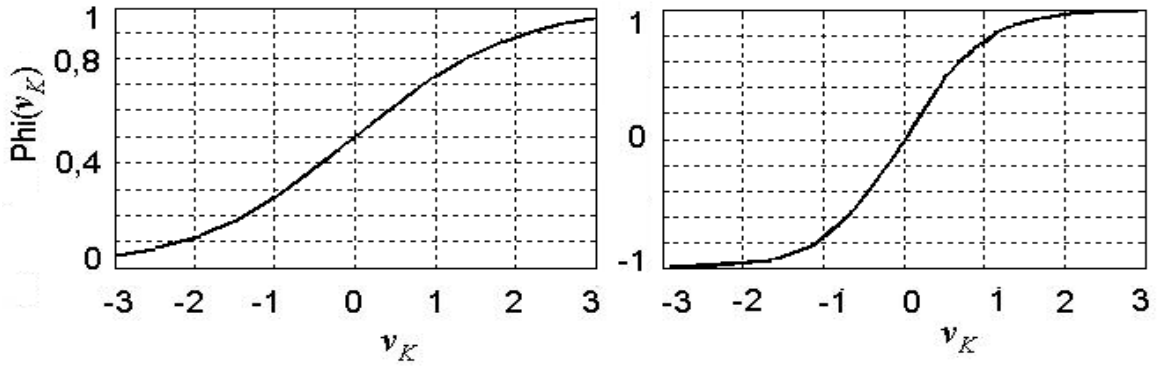
Рис. 1.5. **Графики униполярной и биполярной пороговых функций активации нейрона**

График такой функции активации нейрона показан на рис. 1.5а. Модель персептрона, в которой используется униполярная пороговая функция активации нейрона, в литературе называют моделью Мак-Каллока – Питца (McCalloch-Pitts). В этой модели выходной сигнал (сигнал активности) нейрона принимает значение 1, если индуцированное локальное поле этого нейрона (потенциал активации) не отрицательно,

и 0 – в противном случае. В моделях нейронных сетей используется биполярная пороговая функция активации нейрона, которая является разновидностью предыдущей функции. Она определяется формулой: $Y_K = F(v_K) \Leftrightarrow Y_K = \begin{cases} -1, & v_K < 0 \\ 1, & v_K \geq 0 \end{cases}$. В MATLAB биполярная пороговая функция активации нейрона рассчитывается с помощью функции a = hardlims(v). График этой функции активации нейрона, построенный с помощью функции a = hardlims(v) показан на рис. 1.5б.

При построении искусственных нейронных сетей часто приходится работать не только с самими функциями активации нейрона, но и с ее производными. Для таких ситуаций необходимо иметь монотонные функции, которые бы можно было бы дифференцировать в классическом смысле, а не с помощью обобщенных производных. Для таких случаев в качестве функции активации нейрона используют сигмоидальную функцию. Сигмоидальная функция (sigmoid function), график которой напоминает букву *S*, является, пожалуй, самой распространенной функцией, используемой для создания искусственных нейронных сетей. В частности, примером униполярной сигмоидальной функции может служить логистическая функция (logistic function), или, более точно, функция логистического распределения. В общем виде она определяется формулой: $Y_K = F(v_K) \Leftrightarrow Y_K = \frac{1}{1 + \ell^{(-av)}}$, где a – параметр наклона (slope parameter) униполярной сигмоидальной функции. Изменяя параметр a, можно построить функции активации нейрона с различной крутизной. В пределе, когда параметр наклона достигает бесконечности, сигмоидальная функция вырождается в пороговую. Тем самым, униполярная сигмоидальная (логистическая) функция служит как бы неким переходом между линейным и нелинейным поведением нейрона.

В MATLAB в составе пакета «Нейронные сети» униполярная сигмоидальная (логистическая) функция активации (для $a = 1$) рассчитывается с помощью функции y = logsig(v). График униполярной сигмоидальной функции активации, построенный с помощью функции y = logsig(v), показан на рис. 1.6а.



а) б)

Рис. 1.6. **Графики униполярной (а) и биполярной (гиперболического тангенса) (б) сигмоидальных функций активации нейрона**

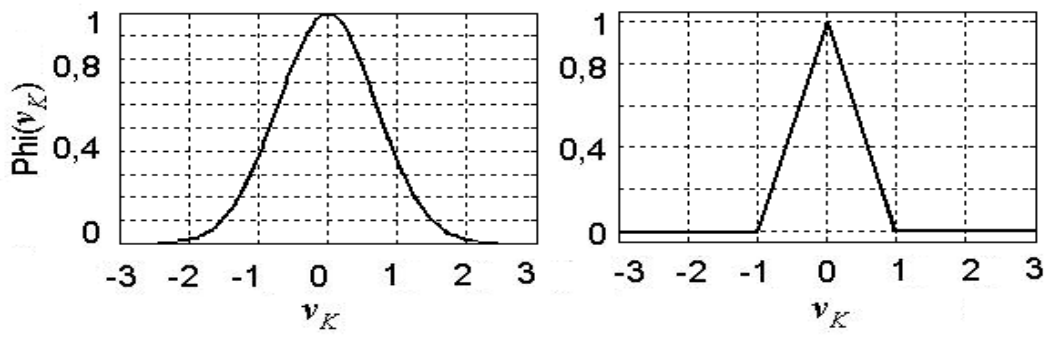
Первая производная логистической функции описывается выражением: $Y_K' = F'(v_K) \Leftrightarrow Y_K' = \frac{dF}{dv} = aF(v)[1 - F(v)] = \frac{a\ell^{-av}}{[1+\ell^{-av}]^2}$. Производная при $v = 0$ достигает своего максимального значения равного $Y_K' = 0.25$, а минимального, равного нулю, – на краях индуцированного локального поля этого нейрона. Область значений сигмоидальной функции активации нейрона представляет собой отрезок от 0 до 1.

Часто требуется, чтобы функция активации была симметричной относительно начала координат, то есть имела область значений от -1 до 1 и была при этом нечетной функцией индуцированного локального поля. Часто это существенно облегчает вычисления. В таких случаях используют биполярную сигмоидальную функцию активации нейрона, которую называют также гиперболической тангенциальной. Функция гиперболического тангенса в общем виде является масштабированной и смещенной логистической функцией и описывается выражением:

$Y_K = F(v_K) \Leftrightarrow Y_K = a\tanh(bv) = a\frac{\ell^{(bv)} - \ell^{(-bv)}}{\ell^{(bv)} + \ell^{(-bv)}}$, где a, b – константы биполярной сигмоидальной функции. В MATLAB биполярная сигмоидальная (гиперболического тангенса) функция активации (для $a = 1, b = 1$) рассчитывается с помощью функции y = tansig(v) ($Y_K = \frac{2}{1+\ell^{(-2v)}}$). График биполярной сигмоидальной функции активации, построенный с помощью функции y = tansig(v) показан на рис. 1.6б.

Несложно определить, что это нечетная, монотонно возрастающая функция. Начало координат является точкой перегиба. Прямые Y= ± 1 служат горизонтальными асимптотами. Первая производная функции гиперболического тангенса описывается следующим выражением: $Y'_K = \frac{dF}{dv} = ab[1 - F(v)]^2 = ab\sec h(bv) == ab[1 - \tanh^2(bv)]$. Производная при $v = 0$ достигает своего максимального значения равного $Y'_K = 1$, а минимального, равного нулю, – на краях индуцированного локального поля этого нейрона.

Наряду с рассмотренными функциями активации используются и ряд других функций. Одной из таких функций активации нейрона, которая применяется искусственных нейронных сетях, является радиальная базисная функция RBS (radial-basis function). В MATLAB радиальная базисная функция RBS активации нейрона, называемая также экспоненциальной, функцией Гаусса, рассчитывается с помощью функции a = radbas(v).График радиальной базисной функции активации, построенный с помощью функции a = radbas(v)показан на рис. 1.7а.



а) б)

Рис. 1.7. **Графики радиальной (а) и треугольной (б) базисных функций активации нейрона**

В качестве функции активации нейрона в искусственных нейронных сетях применяется также треугольная функция. В MATLAB треугольная функция активации нейрона рассчитывается с помощью функции a = tribas(v). График треугольной функции активации, построенный с помощью функции a = tribas(v), показан на рис. 1.7б. В MATLAB графики рассмотренных функций активации нейрона (а также их первых производных) могут быть построены с использованием следующего фрагмен-

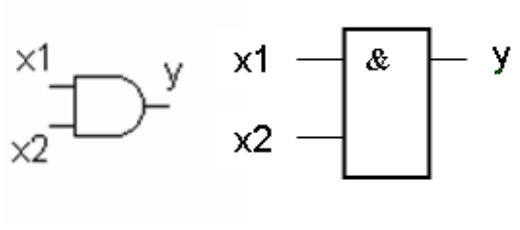
та программы, вводимого в командное окно (вид требуемого графика задается функцией, указанной во второй строке):

```
v = -3:0.01:3;
a = tribas(v);
figure(1),
clf,
plot(v,a,'k','linewidth',2),
xlabel('Vk');
ylabel('Phi(Vk)');
grid on.
```

1.4. Элементарные устройства цифровой электроники на основе одноэлементного персептрона

Состоящая из одного обрабатывающего элемента – процессора, модель искусственного нейрона (рис. 1.2), которая выполняет комбинирование посылаемых к нему входных сигналов $\{P_N\}$ и вычисляет, в соответствии с правилами активации, выходной сигнал Y_K, несмотря на свою простоту, обладает рядом важных возможностей. В частности, с помощью одноэлементного персептрона можно создать устройства, которые работали подобно некоторым логическим элементам (ЛЭ).

Из курса электроники известно, что логический элемент «И» *(*AND) реализует операцию логического умножения (конъюнкции) $Y = X_1 \cdot X_2 = X_1 \wedge X_2$. Символическое изображение двухвходового ЛЭ «И», и его таблица истинности приведены на рис. 1.8.



X_1	X_2	Y
0	0	0
0	1	0
1	0	0
1	1	1

Рис. 1.8. **Представление на схемах логического элемента «И» (AND) и его таблица истинности**

Поскольку $v_K = \sum_{k=1}^{N} P_k W_k + b_k = \sum_{k=1}^{2} P_k W_k + b_k = P_1 W_1 + P_2 W_2 + b = u_k + b_k = n,$

$Y_K = F(v_K) \Leftrightarrow Y_K = \begin{cases} 0, & v_K \le 0 \\ 1, & v_K > 0 \end{cases}$, то, полагая $W_1 = 1, W_2 = 1, b = -1{,}05$ для раз-

личных комбинаций входных сигналов из таблицы истинности получаем следующее: 1) P_1 = 0, P_2 = 0, v_K = n =0x1+0x1-1,05 = -1.05, Y_K = a = 0. Следовательно, сигнал на выходе одноэлементного персептрона Мак-Каллока – Питца равен нулю. Это соответствует таблице истинности.

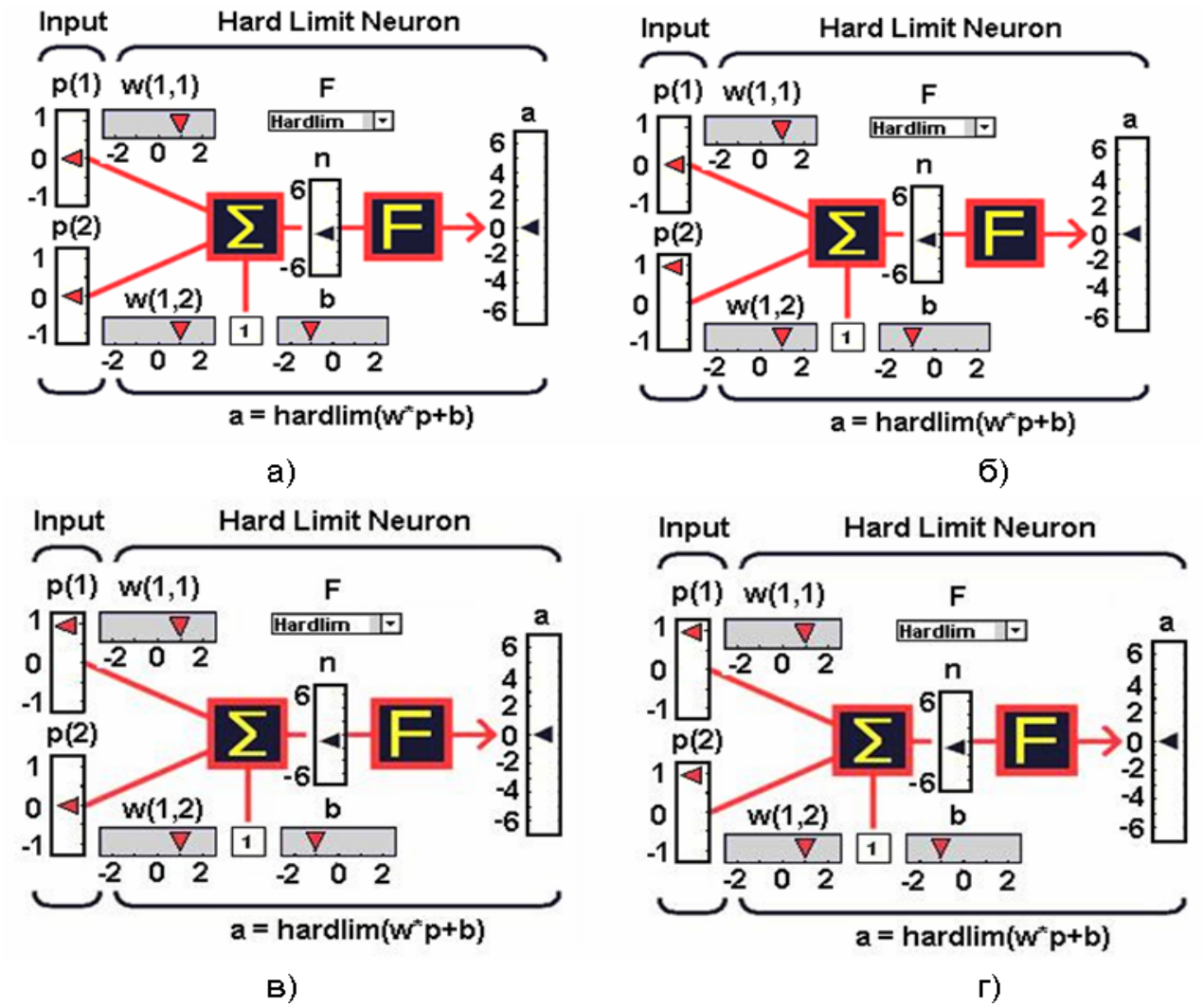
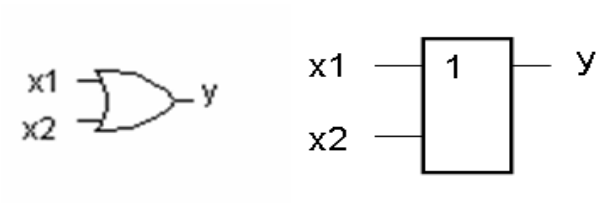


Рис. 1.9. **Представление работы персептрона при осуществлении функции логического элемента «И» (AND) и входных сигналах P_1=0, P_2=0 (а), P_1=0, P_2=1 (б), P_1=1, P_2=0 (в) и P_1=1, P_2=1 (г)**

Процесс работы одноэлементного персептрона можно наглядно представить, используя рис. 1.9а. Малыми треугольниками на рисунке показаны значения входного вектора и порога, синаптических весов, линейной комбинации входных воздействий $u_k = \sum_{k=1}^{N} P_k W_k$, а также сигнал активности нейрона на выходе, посылаемый другим элементам сети. 2)

P_1 = 0, P_2 = 1, v = n =0x1+1x1-1,05 = -0.05, Y_K = a = 0. Это также соответствует таблице истинности и рис. 1.9б. 3) P_1 = 1, P_2 = 0, v = n =1x1+0x1-1,05= 0.05, Y_K = a = 0. Это также соответствует таблице истинности и рис. 1.9в. 4) P_1=1, P_2 =1, v = n =1x1+1x1-1,05 = 0.95, Y_K = a = 1. Что также соответствует таблице истинности и рис. 1.9г.

Известный из курса электроники логический элемент (ЛЭ) «ИЛИ» (OR) реализует функцию логического сложения (дизъюнкцию) $Y = X_1 + X_2 = X_1 v X_2$. Символическое изображение двухвходового вентиля «ИЛИ» и его таблица работы показаны на рис. 1.10.



X_1	X_2	Y
0	0	0
0	1	1
1	0	1
1	1	1

Рис. 1.10. **Представление на схемах логического элемента «ИЛИ» (OR) и его таблица истинности**

Выход вентиля «ИЛИ» имеет высокий уровень, если высокий уровень присутствует хотя бы на одном из его входов (говорят, «что-нибудь или все», то есть на выходе будет сигнал, если воздействует активный уровень или на один, или на второй входы, или на оба сразу). Ноль на выходе появляется только тогда, когда на его оба входа подаются нули.

Полагая $W_1 = 1, W_2 = 1, b = -0.05$, для различных комбинаций входных сигналов из таблицы истинности для устройства осуществляющего функцию логического элемента «ИЛИ» получаем следующее: 1) P_1 = 0, P_2= 0, v = n =0x1+0x1+(-0.05) = -0.05, Y_K = a = 0. Это соответствует таблице истинности и рис. 1.11а. 2) P_1= 0, P_2 = 1, v = n =0x1+1x1+(-0.05) =~ 1, Y_K = a= 1. Это соответствует таблице истинности и рис. 1.11б. 3) P_1=1, P_2 = 0, v = n =1x1+0x1+(-0.05) =~ 1, Y_K = a= 1. Это соответствует таблице истинности и рис. 1.11в. 4) P_1 =1, P_2 = 1, v = n = =1x1+1x1+(-0.05) =~ 2, Y_K = a= 1. Что также соответствует таблице истинности и рис. 1.11г.

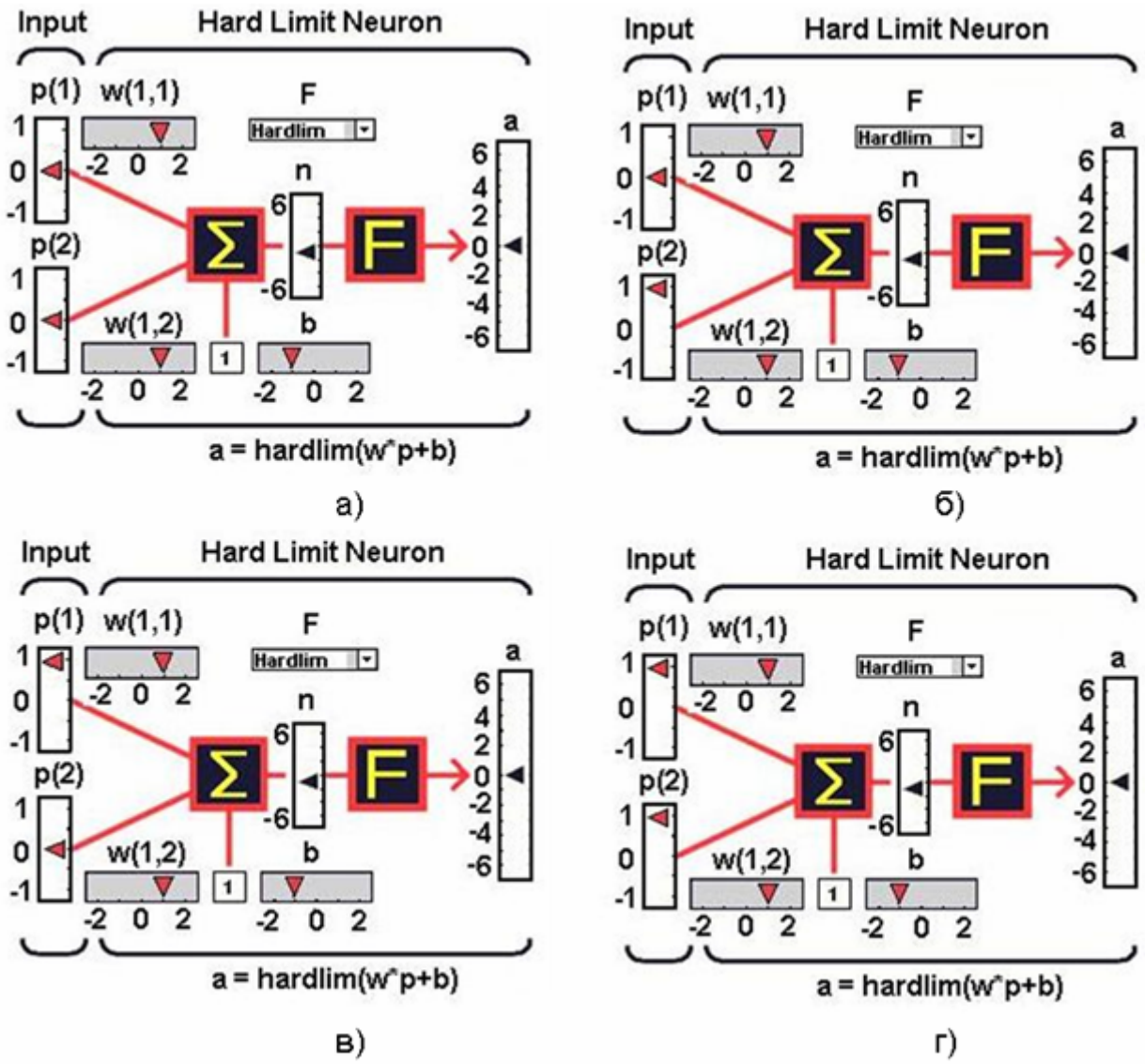
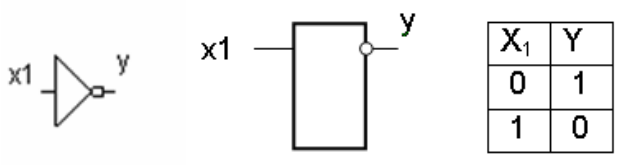


Рис. 1.11. **Представление работы персептрона при осуществлении функции логического элемента «ИЛИ» (OR) и входных сигналах P_1=0, P_2=0 (а), P_1=0, P_2=1 (б), P_1=1, P_2=0 (в) и P_1=1, P_2=1 (г)**

Полагая $W_1 = -1, b = 0.5$, для различных комбинаций входных сигналов из таблицы истинности, можно создать устройство осуществляющего функцию инвертора (элемент «НЕ» (NOT)**).** Проходя через инвертор, сигнал меняет свой активный уровень, поэтому элемент «НЕ» выполняет логическую функцию инверсия: $Y = \overline{x}$.

На схемах ЛЭ «НЕ» обозначают, как показано на рис. 1.12. Там же представлена таблица истинности, характеризующая логику работы инвертора.



X_1	Y
0	1
1	0

Рис. 1.12. **Представление на схемах логического элемента «НЕ» (NOT) и его таблица истинности**

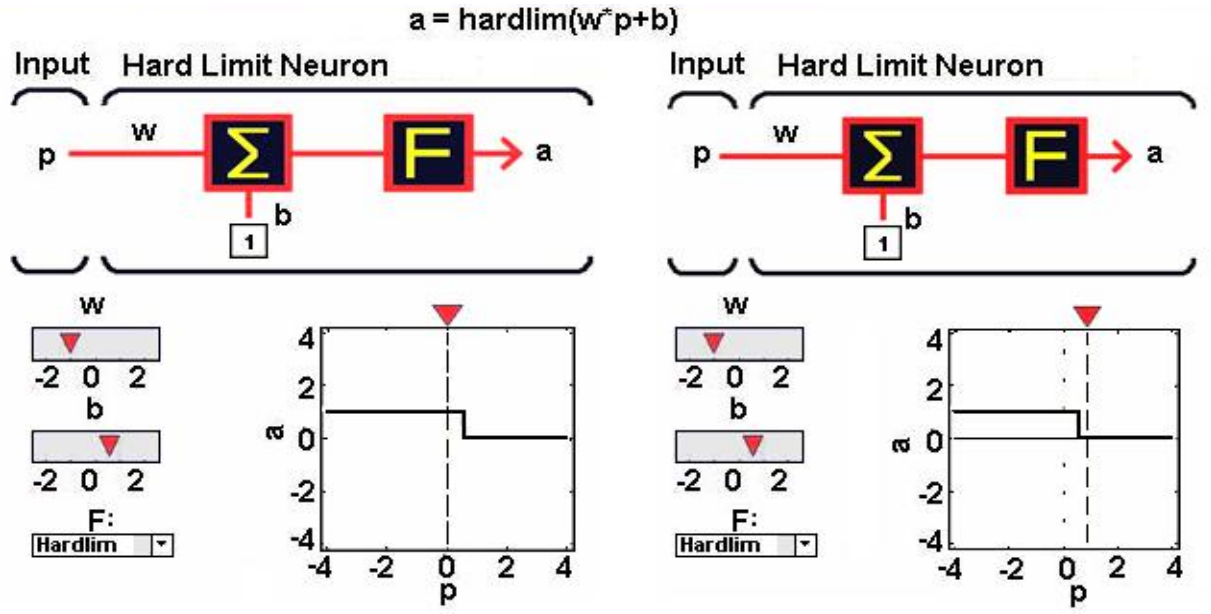


Рис. 1.13. **Представление работы персептрона при входном сигнале P_1=0 и осуществлении функции логического элемента «НЕ» (NOT). Сигнал на выходе равен 1 (а) и 0 (б) при входном сигнале P_1=1**

На рис. 1.13 показано, как, используя одноэлементный персептрон, в котором используется униполярная пороговая функция активации нейрона, можно создать устройство, работающее подобно ЛЭ «НЕ».

1.5. Структура сложной нейронной сети и ее основные понятия

Рассмотренные простые модели искусственных нейронов можно группировать различным образом в сеть (в совокупность взаимосвязанных элементарных обрабатывающих элементов). Можно предположить, что даже если каждый элемент будет иметь ограниченные вычислительные возможности, то вся сеть в целом, объединяя большое число таких элементов, окажется способной выполнять сложные задачи. При

этом сетевые структуры будут различаться способом объединения нейронов и тем, какая функция активности положена в их основу. Детали конструкции сети отражает структура связей, показывающая, какие элементы соединены, в каком направлении работают соединения, как по ним осуществляется передача сигналов и каков уровень значимости (вес) каждого из соединений. В простейшем случае элементы сети, соединенные между собой, могут быть организованы в некую иерархию, обычно называемую термином «слой». Слой состоит из множества нейронов и множества связей. В нейронном слое (single-layer network) с прямой передачей сигнала множество искусственных нейронов имеют общий вектор входа. При этом каждая компонента (координата) входного вектора сигнала $P_r = \{p_1, p_2, ..., p_R\}$, $r = \overline{1, R}$, поступает на входы всех нейронов. Нейронный слой, состоящий, в общем случае, из S нейронов, с прямой передачей сигнала и структура возникающих при этом связей, показаны на рис. 1.14.

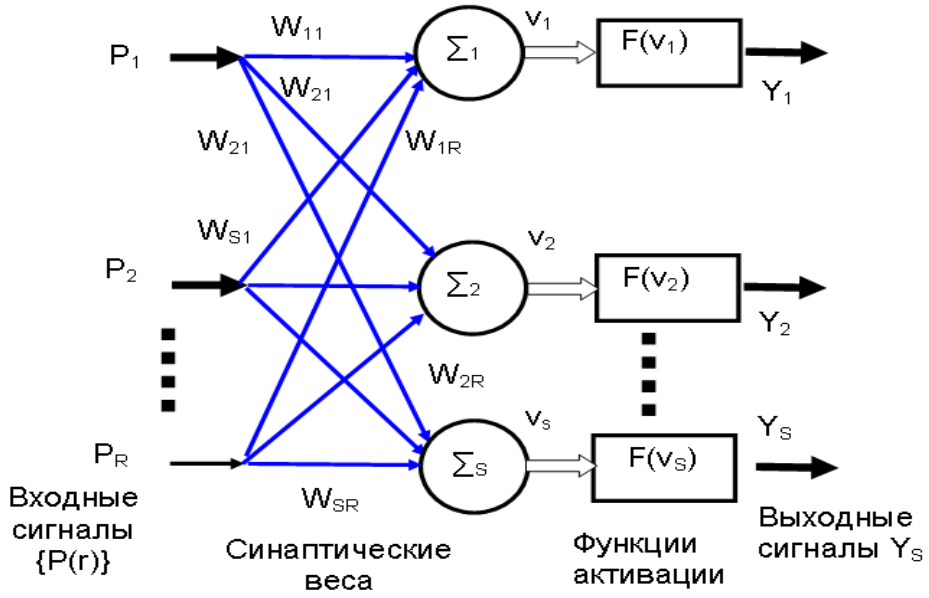


Рис. 1.14. **Нейронный слой, состоящий из S нейронов с прямой передачей сигналов и его структура связей при поступлении на вход сигнала в виде вектора из R компонент (координат)**

Каждая компонента вектора входного сигнала связана с простым нейроном стрелкой, указывающей, откуда исходит данная связь и к какому нейрону она направлена. Рядом со стрелкой указывается весовой коэффициент (вес связи) $W_{SR.}$ Следует обратить внимание на то, что

первый нижний индекс S этого коэффициента показывает, к какому нейрону направлена связь, а второй R – от какой компоненты входного вектора поступает сигнал.

Образованная в нейронном слое система связей обычно представляется весовой матрицей $W_{SR.}$ размера SxR, элементами которой являются синаптические веса w_{IJ}: $W = (w_{IJ})_{SxR} = \begin{bmatrix} w_{11} & w_{12} & \dots & w_{1R} \\ w_{21} & w_{22} & \dots & w_{2R} \\ \dots & \dots & \dots & \dots \\ w_{S1} & w_{S2} & \dots & w_{SR} \end{bmatrix}$. Поскольку каждый простой нейрон порождает, как описано ранее, свой выходной сигнал Y_s, то слой нейронов формирует на своем выходе вектор-столбец выходного сигнала $Y_s = \{y_1, y_2, ..., y_S\}^T, s = \overline{1, S}$. Этот вектор часто называют вектором выхода нейронного слоя.

Структура нейронной сети может быть задана в виде графа, в котором вершинами являются нейроны, а ребра графа представляют связи. В общем случае искусственная нейронная сеть может состоять из нескольких слоев, то есть, как говорят, быть многослойной. В многослойной нейронной сети нейроны располагаются по нескольким слоям, каждый из которых обрабатывает вектор сигналов от предыдущего слоя (или входной вектор). В такой сети обязательно существует входной слой (input layer), получающий внешние (входные) сигналы, выходной слой (output layer), отражающий реакцию (отклик) нейронов на комбинацию входных сигналов. Могут также существовать один или несколько скрытых слоев (hidden layer). Функция последних заключается в посредничестве между входным сигналом и выходом нейронной сети.

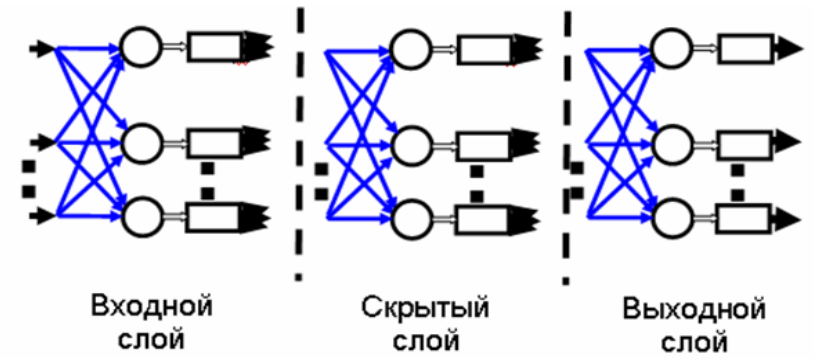


Рис. 1.15. **Упрощенная структура многослойной искусственной нейронной сети прямого распространения с одним скрытым слоем**

Упрощенная структура многослойной искусственной нейронной сети прямого распространения (feed forward), имеющая несколько слоев со связями между нейронами, показана на рис. 1.15. Данная многослойная сеть имеет R входов (входной вектор состоит из R координат), M выходов и K слоев, из которых K_h = (K-2) – являются скрытыми. Минимальной реализацией многослойной нейронной сети является двухслойная нейронная сеть, состоящая из входного, промежуточного (скрытого) и выходного слоя. При подсчете числа слоев входной слой обычно не учитывается, так как служит лишь для распределения входных сигналов по нейронам последующего слоя. В многослойной сети каждый нейрон представляется множеством синаптических связей, внешним порогом и функцией активации. Такой структурный элемент сети получает на входе свой вектор сигналов, вычисляет его скалярное произведение на вектор весовых коэффициентов нейрона и посредством функции активации образует выходной сигнал. В общем случае каждый слой нейронов имеет свой тип функции активации нейронов. В нейронной сети, показанной на рис. 1.15, все нейроны каждого конкретного слоя соединены с нейронами смежных слоев.

Кратко отметим следующее. Многослойная сеть имеет входной, выходной и, в общем случае, скрытые слои. Входные нейроны составляют входной слой сети. На них подается информация, поступающая в сеть («стимул» для сети). Входной сигнал, поступающий в сеть, распространяется по сети в прямом направлении, слева направо, от слоя к слою. Выходные нейроны составляют выходной слой сети. С выходного слоя «снимается» результат, полученный в результате работы сети (общий «отклик» сети). Скрытые нейроны получили свое название по причине того, что они, обрабатывая информацию, вместе с тем не являются ни частью входа, ни частью выхода сети. Нейроны каждого слоя используют в качестве входных сигналов выходные сигналы предыдущего слоя. Считается, когда сигнал проходит слева направо, что нейрон j находится на один слой правее нейрона i, а нейрон k – еще на один слой правее нейрона j, если последний принадлежит скрытому слою. Сигнал достигает конца сети в виде выходного сигнала. В простейшем случае полагают, что связи существуют только между соседними слоями. Связи между нейронами слоев задаются в виде векторов и матриц. Веса удобно представлять элементами матрицы $W = \left\lfloor w_{\xi\psi} \right\rfloor$ размерности

AxB, где A – количество входов, B – количество нейронов. Элемент матрицы весов $\lfloor w_{\xi\psi} \rfloor$ отражает связь между ξ-м и ψ-м нейронами. Во время прямого прохода синаптические веса слоев фиксированы.

Заметим, что многослойная сеть характеризуется тремя отличительными признаками: 1) каждый нейрон сети (или слой нейронов) имеет, как правило, нелинейную функцию активации. При этом стремятся, чтобы данная нелинейная функция была гладкой (то есть всюду дифференцируемой). Наличие нелинейности играет очень важную роль. 2) Сеть содержит один или несколько скрытых слоев нейронов, не являющихся частью входа или выхода сети. Это позволяет сети решать сложные задачи, последовательно извлекая наиболее важные признаки из входного вектора. 3) Сеть обладает высокой степенью связности.

1.6. Средства моделирования работы искусственных нейронных сетей в MATLAB

1.6.1. Компьютерное моделирование нейронной сети с персептроном

Пакет «Нейронные сети» MATLAB содержит средства, позволяющие относительно просто осуществлять создание и моделирование работы искусственных нейронных сетей. При этом, используя специальные функции нейронных сетей, можно выбирать тип и детали конструкции сети, легко изменять структуру связей (архитектуру сети), задавать различные функции активации нейрона. В пакете содержатся модели более 15 наиболее распространенных типов сетей. Использование графической среды моделирования и визуального программирования Simulink, в качестве расширения MATLAB, позволяет наглядно, в виде блок-схемы представить архитектуру и иерархическую структуру нейронной сети, ее основные блоки, наблюдать генерирование и формирование сигналов в сети, графики процессов. Рассмотрим, как, используя MATLAB, можно осуществить построение нейронной сети, проанализировать ее работу и выполнить решение поставленной задачи. При этом, несмотря на ограниченную сферу практических применений, в начале возьмем для изучения одноэлементный персептрон, довольно простой и подробно рассмотренный нами ранее. Модель персептрона позволит

понимать архитектуры и принципы функционирования более сложных искусственных нейронных сетей. Для создания нейронной сети (по имени net) с персептроном используется функция net = newp(PR, S, TF, LF). Первый аргумент функции PR представляет собой матрицу размера Rx2, составленную из минимальных и максимальных значений R компонент (координат) входного вектора. К примеру, если искусственный нейрон выполняет комбинирование двух посылаемых к нему входных сигналов $\{P_2\}$, и при этом каждый входной вектор $P_r = \{p_1, p_2, ..., p_R\}$, $r = \overline{1, R}$ состоит из четырех компонент (координат) $P_1 = \{-2, +2, -3, +4\}$, $P_2 = \{+5, -6, -3, +4\}$, $r = \overline{1,4}$, то матрица минимальных и максимальных значений будет иметь вид [-3 4;-6 5]. Число S определяет число нейронов, используемых в сети. Понятно, что для одноэлементного персептрона оно будет равно 1. TF определяет выбранный тип функции активации нейрона. Если данное значение не вводить, то, по умолчанию, будет использоваться униполярная пороговая функция активации a = hardlim(v). LF определяет функцию обучения нейрона.

Известно, что при обработке информации широко используется объектно-ориентированный подход, который предполагает применение классов объектов. При работе с нейронными сетями также используется класс объектов network. Поэтому при доступе к некоторым сведениям о нейронной сети, в ряде случаев приходится получать данные, используя ссылки на конкретные свойства и методы объекта network. Так, при определении весов и смещений используют следующую программную конструкцию объектно-ориентированного программирования. Например, весовая матрица может задаваться следующим выражением net.IW{1,1} = [2, 3], а значение порога (смещения) - net.b{1} = -1. Следует обратить внимание, что число 2 определяет «вес» компоненты, поступающей на сумматор от первого входного вектора, а 3 – от второго. Порог (смещение) равный -1, как следует из net.b{1} = -1, поступает на нейрон первого слоя. Комплекс графических средств генерирует функция gensim(net, st). Аргумент net определяет имя сети, а параметр st – интервал дискретизации. Если нейронная сеть не имеет задержек, касающихся ее входов и слоев, значение этого параметра устанавливается равным 1 или опускается вообще. Запуск этой функции приведет к открытию ряда блок-диаграмм, о которых мы поговорим позже. После этих замечаний созда-

дим нейронную сеть с одноэлементным персептроном Мак-Каллока – Питца, в котором используется униполярная пороговая функция активации нейрона. Для конкретного выбора значений весов и смещений в сети создадим устройство, работающее подобно ЛЭ «И». Для этого случая -$W_1 = 1, W_2 = 1, b = -1.05$. Программа модели одноэлементного персептрона, которую следует ввести в командное окно, будет иметь вид:

```
clear,
%Создаем одноэлементный персептрон, который работает
%подобно ЛЭ "И".
%Считаем, что нейрон для сети это один слой (1).
%Полагаем, что имеется 2 вектора входа P и каждый
%из них состоит из четырех компонент P1=0 0 1 1 и P2=0 1 0 1
%Координаты вектора  меняются в диапазонах [0 1;0 1].
net = newp([0 1;0 1],1);
%Задаем  синаптические веса равными W{1,1}=[1, 1]
net.IW{1,1} = [1, 1];
%Вводим значение смещения
b{1} = -1
net.b{1} = -1.05;
%Проверяем правильность введенных
%синаптических весов и смещения
disp(  '    Значения весов')
wts = net.IW{1,1},
disp(  '    Величина смещения (порога)')
bias = net.b{1}
%Осуществляем также контроль созданной  сети с помощью
%графических средств, генерирующих нейросетевые блоки
gensim(net)
```

После запуска программы получаем: значения весов: wts = 1 1. Величина смещения: (порога) bias = -1.0500. Запуск программы приведет к открытию функциональной схемы, характеризующей в общем виде нейросетевую модель (рис. 1.16).

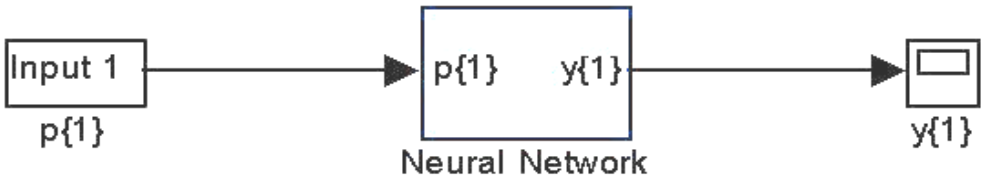


Рис. 1.16. **Упрощенная структура искусственной нейронной сети**

В левом прямоугольнике блок-схемы использовано общее обозначение входного вектора сети. С ним связана представленная в общем виде нейронная сеть. Правый прямоугольник, в виде осциллографа, отражает вектор выходных сигналов. Если дважды щелкать левой кнопкой мыши по расположенному в центре блоку Neural Network, то получим

детальную информацию о составе слоев сети и их структуре (рис. 1.17). Если затем, опять же дважды щелкать левой кнопкой мыши по расположенному в центре блоку Layer1, то можно увидеть архитектуру слоя сети, состав его элементов и связей (рис. 1.18).

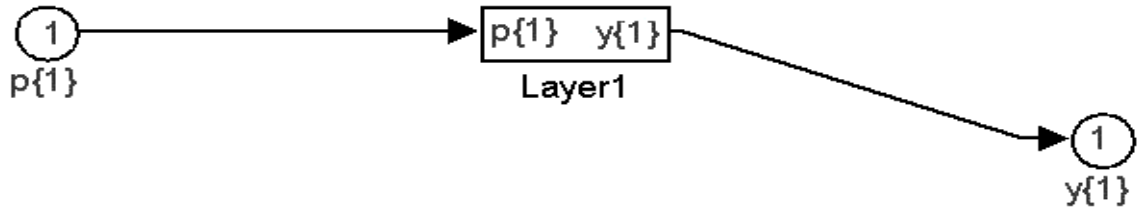


Рис. 1.17. **Блок-схема модели искусственной нейронной сети, показывающая сколько слоев в нее входит и связи слоя**

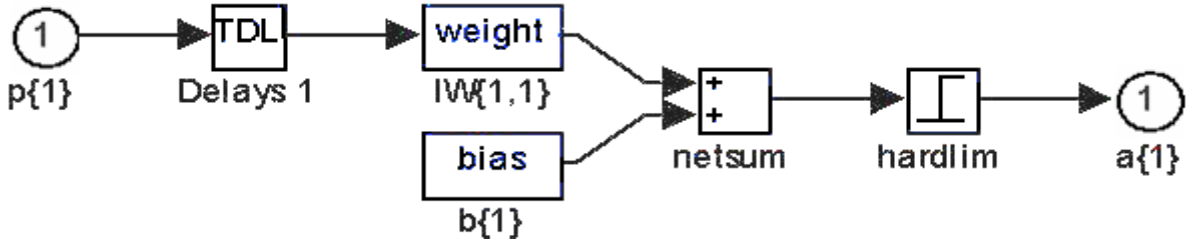


Рис. 1.18. **Блок-схема слоя искусственной нейронной сети с персептроном**

Назначение блоков, входящих в состав слоя созданной сети, становится ясным, если сравнить полученную схему с моделью нейрона, показанную на рис. 1.2. Заметим по поводу блока TDL, не входившего ранее в описание сети. Рассмотренная ранее простая модель искусственного нейрона игнорирует многие свойства своего биологического прототипа, важные для работы человеческого мозга. Например, искусственная нейронная сеть не принимает во внимание возникающие в процессе работы нейрона задержки по времени. Обычно считается, что входные сигналы мгновенно порождают выходной сигнал. Вместе с тем известно, что задержки повсеместно встречаются при работе мозга и играют важную роль в нейробиологической обработке информации. Также не учитываются такие, свойственные биологическим системам особенности, как влияние функции частотной модуляции или синхронизирующие функции биологического нейрона. Тем не менее, часто для обработки сигналов, поступающих на вход сети и изменяющихся во времени, понятие «времени» используют. При таком подходе «статическая» ней-

ронная сеть начинает обладать свойством «динамичности» и кратковременной «памяти». Одним из простейших способов встраивания кратковременной памяти в структуру нейронной сети является использование задержки сигналов во времени (time delay). Данный процесс можно реализовать на синаптическом уровне нейрона. Для этой цели и предназначен блок TDL. Поскольку в нашем случае мы имеем дело с так называемой статической нейронной сетью, которая не учитывает задержки сигналов, то можно полагать, что блок TDL представляет собой мгновенно работающую короткозамкнутую перемычку. Заметим, что подробную информацию о созданной сети можно получить, используя также функцию display(net). Покажем далее, как функционирует созданная сеть, используя для примеров логические элементы. Для определенности сначала реализуем сеть, работающую подобно ЛЭ «И».

Из таблицы истинности, показанной на рис. 1.8. следует, что на вход сети должна подаваться последовательность бинарных значений $\begin{matrix} P_1 \\ P_2 \end{matrix}\left\{\begin{bmatrix}0\\0\end{bmatrix},\begin{bmatrix}0\\1\end{bmatrix},\begin{bmatrix}1\\0\end{bmatrix},\begin{bmatrix}1\\1\end{bmatrix}\right\}$.Эту последовательность можно перегруппировать в следующий равнозначный числовой массив P = [0 0 1 1; 0 1 0 1], поскольку после его введения в командное окно получим ответ: P =

0 0 1 1

0 1 0 1.

Если ввести в командное окно:

```
P = [0 0 1 1; 0 1 0 1];%Входная бинарная последовательность
disp( '    Сигналы на входе сети')
Px1x2=P
disp('    Ответ сети И')
AnsAND = sim(net,P)
```

то получим на выходе «ответ AnsAND» сети, который соответствует таблице истинности ЛЭ «И». Сигналы на входе сети:

Px1x2 = 0 0 1 1

0 1 0 1

Ответ сети «И»: AnsAND = 0 0 0 1.

Рассмотрим, как функционирует сеть, работающая подобно ЛЭ «ИЛИ». Поскольку структуру (архитектуру) сети менять не требуется, то в ранее созданной сети переустановим значения весов и порога (смещения). Для сети, моделирующей работу ЛЭ «ИЛИ» они равны $W_1 = 1, W_2 = 1, b = -0.05$. Вводим в командное окно:

%Задаем синаптические веса равными W{1,1}=[1, 1]

```
net.IW{1,1} = [1, 1];
%Вводим значение смещения b{1} = -0.05
net.b{1} = -0.05;
```

Если опять же в командное окно ввести соответствующий входным сигналам числовой массив P = [0 0 1 1; 0 1 0 1], то есть:

```
P = [0 0 1 1; 0 1 0 1];%Входная бинарная последовательность
disp(' Сигналы на входе сети')
Px1x2=P
disp(' Ответ сети ИЛИ')
AnsOR = sim(net,P)
```

то получим на выходе «ответ AnsOR» сети, который соответствует таблице истинности ЛЭ «ИЛИ». Сигналы на входе сети:

```
Px1x2 =     0     0     1     1
            0     1     0     1
```

Ответ сети «ИЛИ»: AnsOR = 0 1 1 1.

1.6.2. Компьютерное моделирование многослойной нейронной сети

Рассмотрим теперь, как, используя MATLAB, можно осуществить построение многослойной нейронной сети. Для создания многослойной нейронной сети прямого распространения (feed forward) (по имени netff) используется функция netff = newff (PR, [S1, S2, …, SN], {[TF1, TF2, …, TFi]}, BTF, BLF, PF). Первый аргумент функции PR представляет собой матрицу размера Rx2, составленную из минимальных и максимальных значений R компонент (координат) входного вектора. Числа S1, S2, …, SN определяют число нейронов в N -м скрытом слое. При этом следует иметь в виду, что если, к примеру, сеть прямого распространения имеет R входов, H1 нейронов первого скрытого слоя, H2 нейронов второго скрытого слоя и Q нейронов выходного слоя, то такая сеть называется сетью типа «R - H1 - H2 - Q». Так тип сети, имеющей R =10 координат входного вектора или, что то же самое, R =10 входных (распределительных) нейронов, H1=S1=4 скрытых первого слоя, H2=S2=3 скрытых второго слоя и Q =2 выходных нейрона, будет определяться формулой 10-4-3-2. TFi – выбранный тип функции активации нейронов i -го слоя. BTF, BLF, PF – параметры, которые определяют тип функций обучения сети, настройки весов и смещений и определения параметров. Об этих параметрах мы поговорим позже. Предположим, что архитектура ней-

ронной сети такова, что она содержит два скрытых слоя (это двухслойная сеть). Уже отмечалось, что при подсчете числа слоев входной слой не учитывается, так как он служит лишь для распределения входных сигналов. При подсчете слоев, по аналогии, «не замечают» и выходной слой, поскольку он лишь распределяет выходные сигналы, поступившие от последнего скрытого слоя. Пусть при этом первый скрытый слой содержит 5 нейронов с функцией активации tansig; второй слой – 3 нейрона с функцией активации purelin; диапазон изменения координат (компонент) входного вектора – [0 10]. Программа, моделирующая работу многослойной сети прямого распространения, которую следует ввести в командное окно, будет иметь вид:

```
clear,
%Создаем многослойную сеть прямого распространения
%На вход сети подается вектор входа
P = [0 1 2 3 4 5 6 7 8 9 10];
%Координаты вектора  меняются в диапазонах [0 10].
%Считаем, что первый слой содержит 5 нейронов
%Полагаем, что второй слой содержит 3 нейрона
% Функция активации первого слоя нейронов
% гиперболическая тангенциальная
tansig;.
% Функция активации второго слоя нейронов линейная purelin.
P = [0 1 2 3 4 5 6 7 8 9 10];
netff = newff([0 10],[5 3],{'tansig' 'purelin'});
%Осуществляем контроль созданной  сети с помощью
%графических средств, генерирующих нейросетевые блоки
gensim(netff)
```

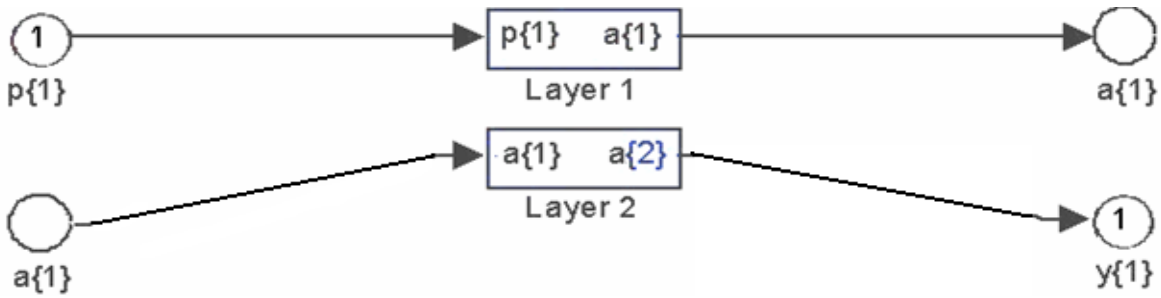


Рис. 1.19. **Блок-схема модели многослойной нейронной сети**

Как уже отмечалось, запуск программы приведет к открытию функциональной схемы, характеризующей в общем виде нейросетевую модель. Если дважды щелкать левой кнопкой мыши по блоку Neural Net-

work, то получим детальную информацию о составе слоев сети прямого распространения и их структуре (рис. 1.19).

Несмотря на то, что, в отличие от рис. 1.15, слои расположены не слева направо, а сверху вниз, из рис. 1.19, легко усмотреть следующее. Первый скрытый слой нейронов получает данные из входного (распределительного) слоя, составленного из узлов источника сигнала (компонент входного сигнала, поступающего в сеть). Результирующий сигнал первого скрытого слоя (вектор $a\{1\}$), в свою очередь, поступает на следующий, второй скрытый слой. Этот слой формирует вектор второго слоя $a\{2\}$. В конце концов, на выходе сети формируется выходной сигнал $y\{1\}$. Если затем дважды щелкать левой кнопкой мыши по блокам Layer1 и Layer2, то можно увидеть архитектуру первого (рис. 1.20а) и второго слоя (рис. 1.20б) сети, состав их элементов и связей.

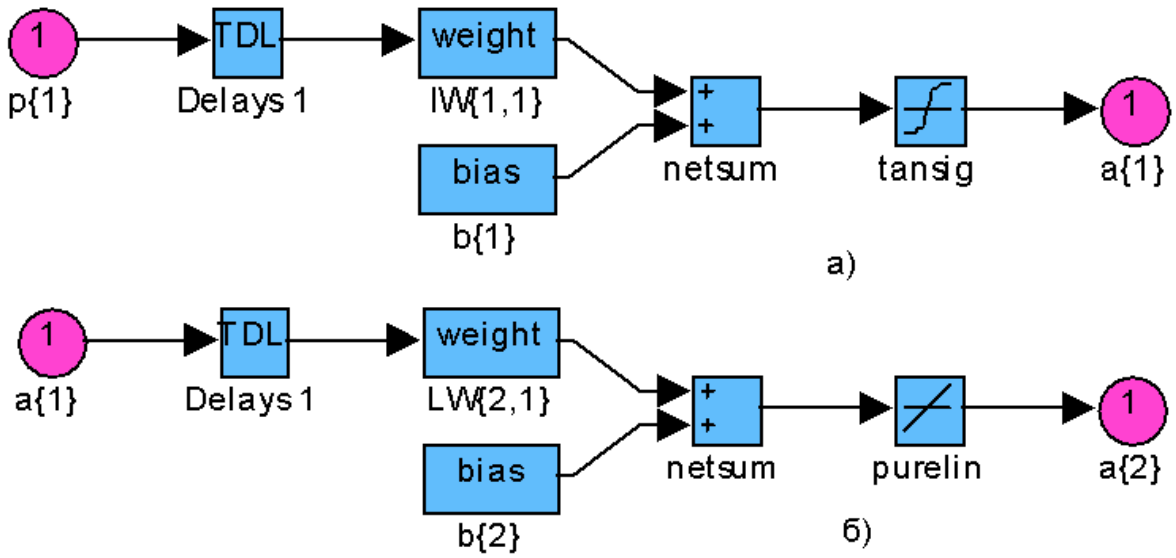


Рис. 1.20. **Блок-схема первого (а) и второго (б) слоя многослойной нейронной сети**

Из рис. 2.20 видно, что: 1) В модели многослойной нейронной имеется общий вектор входа $P\{1\} = P_r$. Входной слой мгновенно, без задержки, «распределяет» входной сигнал по нейронам первого скрытого слоя, из-за чего каждая компонента (координата) входного вектора сигнала, $P_r = \{p_1, p_2, ..., p_R\}$, $r = \overline{1, R}$, поступает на входы всех нейронов первого скрытого слоя. Каждый нейрон скрытого слоя представляется

множеством синаптических связей, внешних порогов и единой для слоя функцией активации.

В модели сети для обозначения синаптических весов первого скрытого слоя используется символ $IW\{1,1\}$. Элементы этой матрицы весов характеризуют синаптические веса, связывающие координаты входного вектора и нейроны первого слоя. Нейроны первого скрытого слоя, получив на входе общий вектор входных сигналов, вычисляют скалярное произведение, формируют индуцированное локальное поле этих нейронов (совокупность потенциалов активации) и, посредством функции активации нейрона, образуют выходной сигнал $a\{1\}$. Аналогично, символом $LW\{2,1\}$ обозначают синаптические веса, связывающие нейроны первого и второго скрытого слоя. Соответственно, нейроны второго скрытого слоя, получив сигналы $a\{1\}$, вычислив скалярное произведение, сформировав индуцированное локальное поле этих нейронов, посредством функции активации нейрона, образуют выходной сигнал $a\{2\}$.

2) Пороги, применяемые к нейронам J, обозначаются символами $b\{1\}, b\{2\}$. Их влияние на слой, как уже отмечалось, может быть представлено синапсом с весом $w_{j0} = b_j$, соединенным с фиксированным входным сигналом, равным +1. После того, как задана архитектура сети, необходимо провести инициализацию сети – задать ей весь набор значений весов и смещений (порогов). Это важный шаг при создании сети, поскольку, как увидим далее, от правильной установки начальных значений весов и смещений существенным образом зависит эффективность сети, а зачастую и вообще ее работоспособность. В MATLAB, для установки начальных значений весов и смещений (инициализации многослойной нейронной сети прямого распространения) используется несколько функций, отражающих различные подходы к выбору значений элементов матриц весов и смещений. В простейшем случае можно задать нулевые значения элементов матрицы весов и векторов смещений. Для этой цели в MATLAB может быть использована, к примеру, следующая функция инициализации весов и смещений initzero(s, PR), где s – количество нейронов, а PR – представляет собой матрицу, составленную из минимальных и максимальных значений R компонент (координат) входного вектора. Например, можно установить нулевые значения вектора смещения первого слоя, используя следующую программу:

```
clear,
%Создаем многослойную сеть прямого распространения
%Считаем, что первый слой содержит 5 нейронов
%Полагаем, что второй слой содержит 3 нейрона
P = [0 1 2 3 4 5 6 7 8 9 10];
netff1 = newff([0 10],[5 3],{'tansig' 'purelin'});
netff1.layers{1}.initFcn = 'initwb';
b11=initzero(5,[4 6]);
netff1.b{1}=b11;
netff2 = init(netff1);
b1=netff2.b{1}
```

После решения задачи получим значения смешений (порогов) первого слоя: b1 = 0 0 0 0 0.

Можно задать элементы матрицы весов и векторов смещений таким образом, чтобы они были равны среднему арифметическому наибольшего и наименьшего значений из диапазона изменения компонент входного вектора. Для этой цели в MATLAB может быть использована, к примеру, следующая функция инициализации весов и смещений b=midpoint(s, PR), где s- количество нейронов, а PR – матрица, составленная из минимальных и максимальных значений R компонент (координат) входного вектора. Если устанавливать для вектора смещения первого слоя средние арифметические значения из диапазона изменения компонент входного вектора, используя следующую программу:

```
clear,
%Создаем многослойную сеть прямого распространения
%Считаем, что первый слой содержит 5 нейронов
%Полагаем, что второй слой содержит 3 нейрона
P = [0 1 2 3 4 5 6 7 8 9 10];
netff1 = newff([0 10],[5 3],{'tansig' 'purelin'});
netff1.layers{1}.initFcn = 'initwb';
b11=midpoint(5,[0 10]);
netff1.b{1}=b11;
netff2 = init(netff1);
b1=netff2.b{1}
```

то получим: b1 = 5 5 5 5 5.

Наиболее часто используются случайные веса и смещения. Для этого используется функция задания весов и смещений «rands», которая формирует матрицу случайных весов и смещений из диапазона [-1, 1]. Для инициализации нейронной сети с новым именем netff2 используется функция netff2=init(netff1), гдеnetff1 – имя ранее созданной сети. Установки весов и смещений производятся в соответствии с объектно-ориентированными установками «netff1.initFcn» и «netff1.initParam». Программа установки случайных весов и смещений из диапазона [-1, 1]

имеет вид (следует обратить внимание, что в целях экономии места в пособии уменьшено количество нейронов слоев):

```
clear,
%Создаем многослойную сеть прямого распространения
%Считаем, что первый слой содержит 3 нейрона
%Полагаем, что второй слой содержит 2 нейрона
P = [0 1 2 3 4 5 6 7 8 9 10];
netff1 = newff([0 10],[3 2],{'tansig' 'purelin'});
%Устанавливаеми веса и смещения в слоях,
% используя функцию rands
netff1.layers{1}.initFcn = 'initwb';
netff1.inputWeights{1,1}.initFcn = 'rands';
netff1.biases{1,1}.initFcn = 'rands';
netff1.layers{2}.initFcn = 'initwb';
netff1.inputWeights{2,1}.initFcn = 'rands';
netff1.biases{2,1}.initFcn = 'rands';
netff2 = init(netff1);
%Определяем матрицы весов и смещений
disp(  'Смещения (пороги) первого слоя')
b1=netff2.b{1}
disp(  'Матрица весов первого слоя')
w1=netff2.iw{1,1}
disp(  'Смещения (пороги) второго слоя')
b2=netff2.b{2}
disp(  'Матрица весов второго слоя')
w2=netff2.lw{2,1}
```

После запуска программы получаем:

Смещения (пороги) первого слоя: b1 = -0.5362 -0.0427 0.0530.

Матрица весов первого слоя: w1 = 0.5854 -0.6140 0.8192.

Смещения (пороги) второго слоя: b2 = 0.8444 -0.9735.

Матрица весов второго слоя: w2 = -0.3137 -0.7622 -0.4422
0.1259 -0.6620 0.1136.

2. ОБУЧЕНИЕ НЕЙРОННОЙ СЕТИ

2.1. Понятия и основные способы обучения сети

Самым важным свойством искусственных нейронных сетей является их способность обучаться (learn) на основе данных, поступающих из окружающей среды. На основе такого «обучения» сеть становится способной вырабатывать верные «отклики» на предъявляемые ей различные «стимулы». Поскольку с понятием обучения ассоциируется много видов деятельности, уточним, что в нейронной сети обучение – это процесс, в котором параметры нейронной сети настраиваются посредством моделирования той среды, в которую эта сеть встроена. В зависимости от способа подстройки параметров определяется тип обучения сети. В настоящее время наиболее широкое распространение получило «учение» сети, когда, при априорно заданной архитектуре сети, обучение происходит посредством интерактивного процесса корректировки синаптических весов и порогов. В идеале сеть получает «знания» об окружающей среде на каждом шаге (итерации) процесса обучения. При этом предполагается следующее. В сеть из внешней среды поступают стимулы. Сеть на них определенным образом, в соответствии с некими правилами, «реагирует», изменяя, настраивая матрицу весов и смещений, в результате чего сеть вырабатывает требуемые выходные сигналы и «отклики» на возбуждения, которые все более приближаются к желаемым. Указанный список четких правил решения задачи приближения к желанной цели называется алгоритмом обучения. Понятно, что алгоритм зависит от того, какой выбран способ настройки синаптических весов нейронов. Кроме того, не существует универсальной парадигмы обучения сети и универсального алгоритма обучения, подходящего для всех архитектур сетей. Принято говорить об обучении с учителем (supervised learning) и без учителя (learning without a teacher).

Первый тип обучения предполагает наличие некого «учителя», обладающего «знаниями» об окружающей среде. Исходя из этого, он может формировать обучающие пары «стимул»-«отклик», то есть для каждого входного вектора задавать требуемый (желаемый) выход сети, соответствующий данному входному вектору. Сеть, чтобы приблизиться к желаемому, выполняет такие действия: для каждого входного вектора

вычисляется выход сети; далее полученное значение сравнивается с требуемым откликом; определяется ошибка выхода, на основании которой корректируется вес. Обучающие пары предъявляются сети последовательно (пошагово) и веса подстраиваются (корректируются) до тех пор, пока ошибка по предъявляемым парам не достигнет требуемого уровня точности. Таким образом, сначала, на этапе обучения знания «учителя» передаются в сеть в максимально полном объеме. Затем, после окончания обучения, «учителя» можно «отключить» и позволить нейронной сети работать со средой самостоятельно. Описанная форма обучения с «учителем» является ни чем иным, как обучением на основе коррекции ошибок (error-correction learning). Чтобы проиллюстрировать такой процесс обучения, рассмотрим простейший случай. Пусть анализируется один обрабатывающий элемент – «$к$»-й нейрон из выходного слоя нейронной сети прямого распространения. На вход такого нейрона подается вектор сигнала $x(n)$, производимого одним или несколькими скрытыми слоями нейронов. Под n подразумевается номер шага итеративного процесса настройки синаптических весов нейрона $к$. Выходной сигнал нейрона $к$ обозначим как $y_k(n)$. Он является единственным выходом сети. Он сравнивается с желаемым выходом (desired response), обозначенным $d_k(n)$. В результате сравнения получается абсолютное значение сигнала ошибки (error signal) $e_k(n)$. По определению $e_k(n)=d_k(n)-y_k(n)$. Сигнал ошибки приводит в действие «механизм управления», основная задача которого состоит в том, чтобы осуществить последовательную корректировку синаптических весов нейрона $к$. Эти изменения нацелены на пошаговое приближение выходного сигнала $y_k(n)$ к желаемому $d_k(n)$. Учитывая, что сигнал ошибки в общем случае может иметь как положительные, так и отрицательные значения, несложно догадаться, что цель может быть достигнута при получении минимума функции, дающей текущее значение «энергии» ошибки $E(n)=\frac{1}{2}e_k^2(n)$.

Пошаговая корректировка синаптических весов нейрона $к$ продолжается до тех пор, пока система не достигнет устойчивого состояния, при котором синаптические веса практически стабилизируются. Минимиз-

зацию энергии ошибки выполняют по так называемому дельта-правилу, или правилу Видроу-Хоффа, названному так в честь его создателей.

Обозначим $w_{kj}(n)$ текущее значение синаптического веса нейрона k на шаге дискретизации n. Согласно дельта-правилу изменение $\Delta w_{kj}(n)$, применяемое к синаптическому весу $w_{kj}(n)$ на шаге дискретизации n задается выражением $\Delta w_{kj}(n) = \eta e_k(n) x_j(n) = \eta[d_k(n) - y_k(n)]x_j(n)$, где η – некоторая положительная константа, определяющая скорость обучения (rate of learning). Ее часто называют параметром скорости обучения; $x_j(n)$ – вектор сигнала, формируемый j -м нейроном скрытого слоя. Иными словами, корректировка, применяемая к синаптическому весу нейрона, пропорциональна произведению сигнала ошибки на входной сигнал, его вызвавший. Следует иметь в виду, что определенное таким образом дельта-правило предполагает возможность прямого измерения сигнала ошибки. Это требует корректировки синаптических весов во всех локализованных отдельных нейронах, для которых поступление желаемого отклика от некоторого внешнего источника, непосредственно доступного. Другими словами нейрон k должен быть «видимым» для внешнего мира. По этой причине обучение на основе коррекции ошибок может быть осуществлено лишь в однослойном персептроне.

Рассмотрим обучение нейронной сети с персептроном, в котором используется биполярная пороговая функция активации нейрона. Пусть однослойный персептрон получает вектор входа $p = \{p_1 p_2 p_3\}^T$ и имеет вектор весов $w(0) = \{0.5 \quad 0.1 \quad 1\}^T$. Смещение (порог) примем равным нулю. Следует изменить весовой вектор так, чтобы при появлении входного сигнала $p = \{-1 \quad -1 \quad 1\}^T$ на выходе персептрона появлялся сигнал -1.

Обучение нейронной сети посредством интерактивного процесса корректировки синаптических весов предполагает следующую последовательность событий: 1) Сначала нейрон получает «стимул» в виде входного вектора и, на основе имеющихся весовых коэффициентов, формирует выходной сигнал. Так как:

$v_K = \sum_{k=1}^{N} P_k W_k + b_k = \sum_{k=1}^{3} P_k W_k + b_k = P_1 W_1 + P_2 W_2 + P_3 W_3 + 0$, и

$Y_K = F(v_K) \Leftrightarrow Y_K = \begin{cases} -1, & v_K \le 0 \\ 1, & v_K > 0 \end{cases}$, то, для начальной комбинации входных сигналов и весов получаем выходной сигнал в виде: $y(0) = F\left[p(0)w(0)\right] = F\left[\{-1 \quad -1 \quad 1\}\{0.5 \quad 0.1 \quad 1\}^T\right] = F[0.4] = 1$. Выходной сигнал не соответствует требуемому, поэтому произведем, в соответствии с дельта-правилом коррекцию весов, положив $\eta = 0.01$:

$w(1) = w(0) + \Delta w(1) = w(0) + \eta e(1)x(1) = w(0) + \eta[d(1) - y(1)]x(1) =$

$\{0.5 \quad 0.1 \quad 1\}^T + 0.01(-1-1)\left[\{-1 \quad -1 \quad 1\}^T\right] = \{0.52 \quad 0.12 \quad 0.98\}^T$.

Для этой матрицы весов получаем выходной сигнал:

$y(1) = F\left[p(0)w(1)\right] = F\left[\{-1 \quad -1 \quad 1\}\{0.52 \quad 0.12 \quad 0.98\}^T\right] = F[0.34] = 1$.

Неверный «отклик» персептрона свидетельствует о необходимости дальнейшей коррекции весов. Поступая по аналогии, получаем следующие матрицы весов и выходного сигнала:

$$w(2) = \{0.54 \quad 0.14 \quad 0.96\}^T, y(2) = 1.$$

$$\cdots \quad \cdots$$

$$\cdots \quad \cdots$$

$$w(6) = \{0.62 \quad 0.22 \quad 0.88\}^T, y(6) = 1.$$

На седьмом шаге разница между выходным $y_k(n)$ и желаемым сигналом $d_k(n)$ станет равной нулю, поэтому отклик персептрона на заданный входной сигнал станет правильным и обучение закончится

$$w(7) = \{0.64 \quad 0.24 \quad 0.86\}^T, y(7) = -1.$$

Если бы мы взяли параметр скорости обучения $\eta = 0.02$, то:

$$w(1) = \{0.6 \quad 0.2 \quad 0.9\}^T, y(1) = 1$$

$$w(2) = \{0.7 \quad 0.2 \quad 0.9\}^T, y(2) = -1.$$

Следовательно, обучение закончилось бы за два шага. При выборе $\eta = 0.05$ сразу же получаем необходимое решение:

$$w(1) = \{0.9 \quad 0.5 \quad 0.6\}^T, y(1) = -1.$$

Выбор параметра η влияет на то, за сколько шагов заканчивается процесс обучения (на длительность обучения). Он, как оказалось, также

влияет на точность и другие характеристики процесса обучения. По этой причине, учитывая его ключевую роль в процессе обучения, требуется тщательно подбирать параметр скорости обучения.

От «ручных» расчетов перейдем к вычислениям с помощью пакета программ «Нейронные сети» MATLAB. Вводим в командное окно:

```
clear,
% Создаем однослойную сеть.
% На вход сети будет подан вектор из трех компонент.
% Полагаем, что каждая компонента вектора может
% изменяться в диапазоне[-2 2] .
% Скрытый слой содержит один нейрон.
% Функция активации нейрона -
% биполярная пороговая 'hardlims'
net = newp([-2 2;-2 2;-2 2],1,'hardlims');
% Определяем объектно-ориентированные параметры сети,
% инициализируемые по умолчанию.
disp('  Информация о характеристиках матрицы весов')
inputweights = net.inputweights{1,1}
disp('   Информация о характеристиках матрицы порогов')
biases = net.biases{1}
%  Определим веса и пороги персептрона,
%  инициализируемые по умолчанию.
disp(' Значения весов по умолчанию')
wts0 = net.IW{1,1},
disp(' Величина смещения (порога)по умолчанию')
bias0 = net.b{1}
```

Запуская программу получим следующее:

Информация о характеристиках матрицы весов:

inputweights = delays: 0 initFcn: 'initzero' learn: 1 learnFcn: 'learnp' learnParam: [] size: [1 3] userdata: [1x1 struct] weightFcn: 'dotprod'.

Информация о характеристиках матрицы порогов:

biases = initFcn: 'initzero' learn: 1 learnFcn: 'learnp' learnParam: [] size: 1 userdata: [1x1 struct].

Значения весов по умолчанию: wts0 = 0 0 0.

Величина смещения (порога) по умолчанию: bias0 = 0.

Полученные параметры матрицы весов свидетельствуют о следующем: 1) Задержки в используемой сети равны нулю. 2) В качестве функции начальной установки весов initFcn: используется функция 'initzero'. Как уже отмечалось, она задает (инициализирует) весам нулевые начальные значения. 3) В качестве функции начальной настройки параметров обучения нейронов learnFcn используется функция 'learnp'. С ее помощью реализуется требуемый алгоритм обучения персептрона. При начальной инициации параметр скорости обучения learnParam не задан.

4) Веса представляют собой матрицу размером 1x3 size: [1 3]. 5) Вход функции активации (функция придания входам P весов W) weightFcn вычисляется с помощью скалярного произведения 'dotprod' $v_K = \sum_{k=1}^{N} P_k W_k$.

Аналогичные пояснения имеют параметры матрицы смещений (порогов). При этом следует иметь в виду, что в рассматриваемом примере в скрытом слое имеется лишь один нейрон, поэтому матрица состоит из одного элемента. В соответствии с функцией 'initzero' инициализированные значения матриц весов и порога равны нулю. По умолчанию веса и смещение равны нулю, поэтому, чтобы установить необходимые по условию решения задачи значения, необходимо применить операторы задания начальных значений w= [0.5, 0.1, 1]; net.IW{1,1} = w; net.b{1} = 0. Запуская (моделируя) персептрон посредством y0 = sim(net,p) в MATLAB для входного вектора P можно рассчитать выход и ошибку на первом шаге решения задачи. Вводим в командное окно:

```
clear,
% Создаем однослойную сеть
% На вход сети будет подан вектор из трех компонент
% Полагаем, что каждая компонента вектора может
% изменяться в диапазоне[-2 2]
% Скрытый слой содержит один нейрон
% Функция активации нейрона -
% биполярная пороговая 'hardlims'
net = newp([-2 2;-2 2;-2 2],1,'hardlims');
%  Определяем объектно-ориентированные параметры сети,
%  инициализируемые по умолчанию.
disp(' Информация о характеристиках матрицы весов')
inputweights = net.inputweights{1,1}
disp(' Информация о характеристиках матрицы порогов')
biases = net.biases{1}
% Определим веса и пороги персептрона,
%  инициализируемые по умолчанию.
%  в соответствии с задачей.
w= [0.5, 0.1, 1];
net.IW{1,1} = w;
net.b{1} = 0;
%  Проверяем полученные веса и пороги персептрона.
%  инициализируемые по умолчанию.
disp(' Начальные значения весов и порогов')
wts1 = net.IW{1,1},
bias1 = net.b{1}
%  Задаем  значения компонент входного вектора
disp(  '   Входной вектор')
p= [-1, -1, 1]'
%  Моделируя персептрон, рассчитаем выход
%  и ошибку на первом шаге настройки
disp('  Выходной сигнал для начальной ')
disp(' комбинации входных сигналов и весов ')
y0 = sim(net,p),
%  Задаем желаемое значение сигнала
```

```
disp('  Значения весов по умолчанию')
wts0 = net.IW{1,1},
disp('  Величина смещения (порога)по умолчанию')
bias0 = net.b{1}
%  Задаем начальные значения весов и порогов персептрона,
выхода
disp(' Требуемый выход  персептрона')
test=-1
%  Определяем на этом шаге разницу между выходным,
%  и желаемым сигналом.
disp(' Начальная ошибка')
error0 = test-y0
```

После запуска программы получаем:

Начальные значения весов и порогов: wts1 = 0.5000 0.1000 1.0000. bias1 = 0.

Входной вектор: p = -1 -1 1.

Выходной сигнал для начальной комбинации входных сигналов и весов: y0 = 1.

Требуемый выход персептрона: test = -1.

Начальная ошибка: error0 = -2.

Как и при ручном расчете выходной сигнал не соответствует требуемому, поэтому произведем, в соответствии с дельта-правилом коррекцию весов, положив $\eta = 0.01$. Для этого воспользуемся функцией настройки весов (функцией обучения) нейрона, которая реализует дельта-правило, или алгоритм Видроу-Хоффа learnwh(w,p,[ ],[ ],[ ],[ ],error0,[ ],[ ],[ ],learnParam,[ ]). Вводим в командное окно:

```
clear,
% Создаем однослойную сеть
net = newp([-2 2;-2 2;-2 2],1,'hardlims');
%  Задаем начальные значения весов и порогов персептрона,
%  в соответствии с задачей.
w= [0.5, 0.1, 1];
net.IW{1,1} = w;
net.b{1} = 0;
%  Проверяем полученные веса и пороги персептрона.
%  инициализируемые по умолчанию.
disp(' Начальные значения весов и порогов')
wts1 = net.IW{1,1},
bias1 = net.b{1}
%  Задаем желаемое значение сигнала выхода
disp(' Требуемый выход  персептрона')
test=-1
% Определяем на этом шаге разницу между выходным
%  и желаемым сигналом.
disp(' Начальная ошибка')
error0 = test-y0
learnParam.lr=0.01;
%  Используя М-функцию настройки параметров learnwh,
%  найдем требуемое изменение весов
disp(' Необходимые изменения весов')
dw = learnwh(w,p,[ ],[ ],[ ],[ ],error0,[ ],[ ],[ ],learnParam,[ ])
```

```
% Задаем значения компонент входного вектора
disp('  Входной вектор')
p= [-1, -1, 1]'
% Моделируя персептрон, рассчитаем выход
% и ошибку на первом шаге настройки
disp(' Выходной сигнал для начальной ')
disp('  комбинации входных сигналов и весов ')
y0 = sim(net,p),
% Выполним один цикл настройки:
disp('  Новые веса после  цикла настройки')
w1=w+dw
disp('  Выходной сигнал для комбинации входных сигналов')
disp('  и весов после одного цикла настройки')
y1 = sim(net,p),
disp(' Ошибка после  цикла настройки')
error1 = test-y1
```

После запуска программы получаем:

Необходимые изменения весов: dw = 0.0200 0.0200 -0.0200.
Новые веса после цикла настройки: w1 = 0.5200 0.1200 0.9800.
Выходной сигнал для комбинации входных сигналов и весов после одного цикла настройки: y1 = 1.
Ошибка после цикла настройки: error1 = -2.

2.2. Алгоритм обратного распространения ошибки

Расширением рассмотренного дельта-правила является алгоритм обратного распространения ошибки (error back-propagation algorithm), кратко называемый алгоритмом обратного распространения (back propagation). Он реализует вычислительно эффективный метод обучения с учителем многослойного персептрона. Обучение с учителем многослойной сети по данному алгоритму основывается на коррекции ошибок и предполагает, в простейшем случае, два прохода по всем слоям сети: прямого и обратного. При прямом проходе (forward pass) входной вектор подается на вход сети, после чего он распространяется по сети слева направо от слоя к слою. В результате такого потока сигналов формируется набор выходных сигналов, который является фактическим «откликом» сети на данный входной «стимул». Во время прямого прохода все синаптические веса сети фиксированы. Во время обратного прохода (backward pass) все синаптические веса настраиваются в соответствии с действующим в сети правилом коррекции ошибок, а именно: фактический выход сети вычитается из желаемого (целевого) отклика, в результате чего формируется сигнал ошибки, то есть величины, в соот-

ветствии с которыми следует корректировать весовые коэффициенты в процессе обучения. Этот сигнал ошибок впоследствии распространяется по сети (по системе взвешенных связей) справа налево от слоя к слою в направлении, обратном направлению потока сигналов (отсюда и название). Вычисления ошибок проводятся для всех слоев. В обратном потоке нейроны последнего скрытого слоя будут получать сигналы ошибок от каждого обрабатывающего элемента выходного слоя, предыдущего слоя от последующего и так далее. При этом синаптические веса настраиваются с целью максимального приближения выходного сигнала сети к желаемому. Итогом и целью обучения является нахождение такого набора матрицы весов и смещений, который бы обеспечивал приближения отклика сети к желаемому с заданной степенью точности.

При обучении каждый входной вектор (стимул) должен иметь соответствующий желаемый выходной сигнал (отклик). Процесс обучения, в частном случае, состоит в предъявлении сети предопределенных обучающих пар (входного стимула и желаемого отклика), изучения того, как отреагировала на них сеть и коррекции элементов весовой матрицы. Один полный цикл рассмотрения всех имеющихся образцов (предъявления полного набора примеров обучения) называется эпохой. Процесс обучения, в общем случае, состоит из нескольких повторяющихся проходов (из циклов) по всем слоям сети. Другими словами, процесс обучения проводится от эпохи к эпохе, пока синаптические веса и уровни порога не стабилизируются, а среднеквадратическая ошибка на всем обучающем множестве не сойдется к некоторому минимальному значению.

Для реализации алгоритма обратного распространения, если говорить кратко, необходимо выполнить следующее: 1) Инициализировать сеть. При этом имеет смысл для сети, из-за отсутствия априорной (начальной, исходной) информации, задать синаптические веса и пороги в виде системы отличающихся друг от друга случайных чисел (с нулевым средним значением и дисперсией, выбранной таким образом, чтобы стандартное отклонение индуцированного локального поля нейрона приходилось на линейную часть сигмоидальной функции и не достигало области насыщения). Почему именно так надо делать станет ясно несколько позже. Пока лишь обратим внимание на следующее обстоятельство. Если задать все значения синаптических весов и порогов одинаковыми, то может возникнуть ситуация, при которой сеть не станет обучаться. Итак, запомним: для осуществления обучения первоначально

необходимы случайные и разные синаптические веса и пороги. 2) Предъявить для процесса обучения все примеры обучения (все образы) $\{P(n),\ \ T(n)\}$ из обучающего множества. 3) Выбрать из заданного обучающего множества первую (или очередную) пару (образ) и подать на вход сети входной сигнал. 4). Для поданного на вход сети сигнала выполнить прямой проход и вычислить отклик сети. В многослойной сети входной вектор поступает на элементы входного слоя, а желаемый отклик предъявляется выходному слою нейронов (для формирования сигнала ошибки). Проходя по сети послойно в прямом направлении, для каждого слоя нейронов вычисляются индуцированные локальные поля и функции активности нейронов. 5) Сравнить полученный отклик (выходной сигнал) с требуемым (желаемым) и определить ошибку (error signal). 6) Осуществить обратный проход. Т.е., для данного входного сигнала нужно обновить (скорректировать) веса и пороги так, чтобы ошибка уменьшилась. С этой целью для каждого образа последовательно выполнить вычисления новых матриц весов и смещений. 7) Выполнить требуемое количество итераций. Предъявляя сети все примеры обучения из эпохи, последовательно выполнить прямой и обратный проходы. То есть надо повторить шаги 3-5 для всего множества обучающих пар. Поскольку в результате проведения коррекции при каждой эпохе ошибка в идеале должна уменьшаться, вычисления (прямой и обратные проходы) следует выполнять до тех пор (столько эпох), пока на заданном множестве образов ошибка обучения не достигнет требуемой величины.

Для оценки способности многослойной сети решать стоящие перед ней задачи важным является то, как вычисляются ошибки сети и как осуществляется коррекция весов нейронов при обратном проходе. Поэтому имеет смысл остановиться на этих вопросах более подробно.

Сначала, чтобы было понятно, рассмотрим один нейрон j выходного слоя сети. Выход данного нейрона является выходом сети, поэтому такой нейрон для сети «видим». Это означает, что для него на итерации n известен желаемый отклик $d(n)$ и сигнал $y(n)$, формируемый им на выходе при поступлении на его вход потока взвешенных сигналов от предыдущего слоя. Сигнал абсолютной ошибки выходного нейрона j на итерации n (соответствующий n-му примеру обучения) определяется соотношением $e_J(n) = d_J(n) - y_J(n)$. Уже отмечалась, что на практике часто важна энергия ошибки. Поэтому текущее значение энергии ошибки

нейрона j выходного слоя будет $1/2[e_J(n)]^2 = [d_J(n) - y_J(n)]^2$. Соответственно текущее значение общей энергии ошибки сети $E(n)$, которое определяется путем сложения величин $1/2[e_J(n)]^2$ по всем «видимым» нейронам выходного слоя, можно записать $E(n) = \frac{1}{2}\sum_{J \in C} e_J^2(n)$, где множество C включает все нейроны выходного слоя сети.

Пусть N – общее число образов в обучающем множестве. Энергия среднеквадратической ошибки в таком случае вычисляется как нормированная по N сумма всех значений энергии ошибки $E(n)$: $E_{MSE}(n) = \frac{1}{N}\sum_{n=1}^{N} E(n)$. Текущая энергия шибки $E(n)$, а значит, и средняя энергия ошибки $E_{MSE}(n)$ зависят от величин синаптических весов и значений порогов сети. Поэтому для данного обучающего множества $E_{MSE}(n)$ является мерой, определяющей эффективность обучения, позволяющей судить о том, насколько хорошо настроена сеть на решение поставленной задачи или, как еще говорят по иному, $E_{MSE}(n)$ представляет собой функцию стоимости (полагают, что ошибки это величина обратная «цене» покупаемого товара, и хотят стоимость покупки минимизировать). Отсюда вытекает, что целью процесса обучения является подбор такого набора весов и смещений, чтобы для элементов обучающего множества средняя энергия ошибки была минимальной и, в идеале, стремилась к нулю. Средняя энергия ошибки $E_{MSE}(n)$ (среднеквадратическая ошибка или сумма квадратов ошибок на обучающей выборке) зависит от величин синаптических весов и значений порогов сети (представляет функцию многих переменных). Для такой функции можно построить многомерную поверхность ошибок (error surface) от параметров весов и порогов. Любое действие сети при обучении с учителем фактически представляется точкой на «поверхности» ошибок, а сам процесс обучения это не что иное, как перемещение в сторону минимума на «поверхности» ошибок. В простейшем случае, когда один нейрон получает сигнал посредством взвешивания с помощью двух весовых коэффициентов, средняя энергия ошибки графически представляет собой поверхность в трехмерном пространстве, напоминающую чашу. Мини-

мум ошибки нейрона в таком случае – «на дне чаши». Известно, что минимум функции всегда связан с производной. Поэтому хорошо известная в математике стратегия поиска минимума функции нескольких переменных связана с определенным вычислением производных по координатам (метод покоординатного спуска). Разумным является также метод, при котором поиск точки минимума функции нескольких переменных (спуск на дно «чаши», которая теперь называется «гиперповерхностью») производится таким образом, чтобы «движение к минимуму» осуществлялся вдоль «наилучшего» направления. Производная векторного поля по направлению называется градиентом (оператор градиента $\nabla = \left[\frac{\partial}{\partial w_1}, \quad \frac{\partial}{\partial w_2}, \quad ..., \quad \frac{\partial}{\partial w_m} \right]^T$). Из математики известно, что направление градиента (вектор градиента функции $E(w_1, w_2, ..., w_m)$ $\nabla E(w) = \left[\frac{\partial E}{\partial w_1}, \quad \frac{\partial E}{\partial w_2}, \quad ..., \quad \frac{\partial E}{\partial w_m} \right]^T$) является направлением наискорейшего возрастания функции. Следовательно, противоположное направление является направлением наискорейшего убывания функции. Из сказанного становится понятным, что минимизация функции ошибок в методе обратного распространения ошибок в первую очередь должна быть либо как то связана, либо являться специфической реализацией градиентного спуска (gradient descent) в пространстве весов многослойных сетей. Так и оказалось в действительности. Основная идея метода обратного распространения состоит в вычислении частных производных от функции средней энергии ошибки $E_{MSE}(n)$ по всем элементам настраиваемого вектора весов для данного входного вектора и проведении коррекции, определяемой на основании дельта-правила. Частные производные $\partial E(n) / \partial w_{ji}(n)$ представляют собой «фактор чувствительности к направлению наискорейшего возрастания функции», определяющий направление поиска в пространстве весов. Если использовать знак «минус», то получим реализацию градиентного спуска в пространстве весов, то есть стратегию поиска направления изменения весов, уменьшающих значение энергии ошибки. Коррекция весов Δw_{ji}, применяемая к w_{ji}, в соответствии с дельта-правилом, будет определяться формулой

$\Delta w_{ji}(n) = -\eta \dfrac{\partial E(n)}{w_{ji}(n)} = \eta \delta_j(n)$, где η – параметр скорости обучения алгоритма обратного распространения; $-\partial E(n)/\partial v_j(n)$ – локальный градиент, указывающий на требуемое изменение синаптического веса.

Из выражений видно, что основным фактором, оказывающим ключевую роль на требуемое изменение синаптического веса, на величину коррекции Δw_{ji} является сигнал ошибки $e_J(n)$ нейрона j на итерации n. Поэтому, исходя из положения нейрона в многослойной сети, следует рассматривать два случая. В первом случае нейрон является выходным узлом, для каждого выходного узла сети известен соответствующий желаемый отклик, поэтому вычислить сигнал ошибки можно с помощью простой арифметической операции. Вес должен изменяться в направлении, противоположном тому, которое указывает производная функции энергии ошибок и с учетом параметра скорости обучения. Во втором случае нейрон j может находиться в скрытом слое. Он «не видим», непосредственно недоступен, однако он тоже несет ответственность за ошибку получаемую на выходе сети, и вносит свой вклад в общую ошибку. В этих случаях коррекция весов происходит сложнее. Весовые коэффициенты скрытого нейрона следует корректировать пропорционально его «вкладу» в величину ошибки следующего слоя. Этот «вклад» для нейрона скрытого слоя зависит от величины ошибки выходного элемента и весового коэффициента, соединяющего нейроны. Другими словами, нейрон с большей ошибкой делает больший «вклад» в ошибку того нейрона скрытого слоя, который связан с данным выходным элементом.

Не приводя доказательств, определим соотношения, свойственные алгоритму обратного распространения. Если формулировать в общем виде, то коррекция Δw_{ji}, применяемая к синаптическому весу, соединяющему нейроны i и j, определяется следующим правилом:

$$\begin{pmatrix} Коррекция \\ веса \\ \Delta w_{JI}(n) \end{pmatrix} = \begin{pmatrix} Скорость \\ обучения \\ \eta \end{pmatrix} \cdot \begin{pmatrix} Локальный \\ градиент \\ \delta_J(n) \end{pmatrix} \cdot \begin{pmatrix} Выходной \\ сигнал \quad нейрона \\ y_i(n) \end{pmatrix}.$$

Следует иметь при этом в виду следующее: 1) Если нейрон j – выходной, то локальный градиент $\delta_J(n)$ равен произведению производной

функции активации $F'_J[v_J(n)]$ на сигнал ошибки $e_J(n)$, и коррекция Δw_{ji}, применяемая к синаптическому весу определяется выражениями: $w_{JI}(n+1) = w_{JI}(n) + \eta\delta_J(n)y_i(n)$ и $\delta_J(n) = e_J(n)F'_J[v_J(n)]$. 2) Если нейрон j – скрытый, то локальный градиент $\delta_J(n)$ равен произведению производной функции активации $F'_J[v_J(n)]$ на взвешенную сумму градиентов, вычисленных для нейрона следующего скрытого или выходного, которые непосредственно связаны с данным нейроном j. Коррекция Δw_{ji}, применяемая к синаптическому весу определяется выражениями: $w_{JI}(n+1) = w_{JI}(n) + \eta\delta_J(n)y_i(n)$ и $\delta_J(n) = F'_J[v_J(n)]\sum_k \delta_k(n)w_{kj}(n)$. Заметим, что во втором случае локальный градиент зависит как от функции активации, так и от двух множеств параметров. Первое из них, – $\delta_k(n)$,– требует знания сигналов ошибок $e_k(n)$ для всех нейронов слоя, находящегося правее скрытого нейрона j. Второе множество, $-w_{kj}(n)$, – состоит из синаптических весов этих связей.

Изучим работу алгоритма обратного распространения, взяв в качестве примера нейронную сеть, показанную на рис. 2.1.

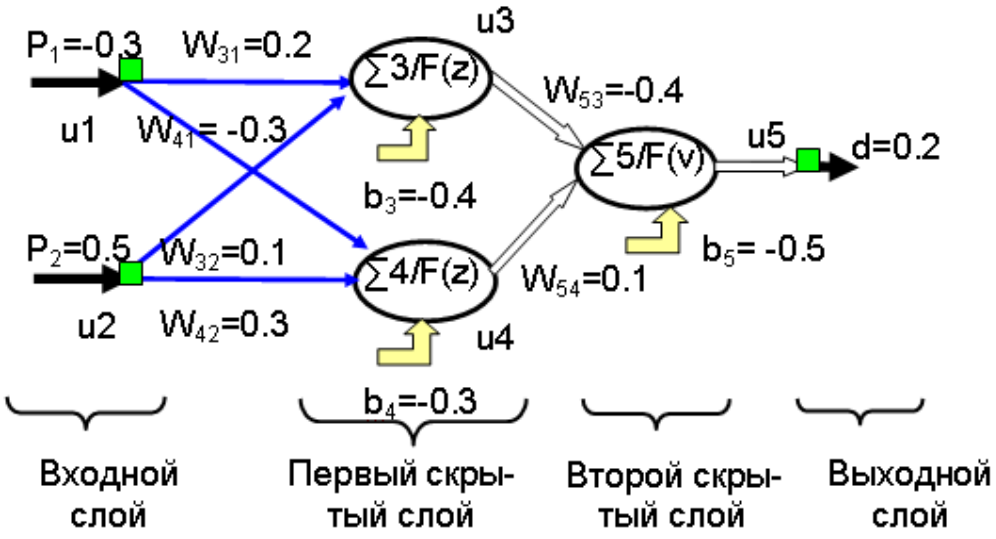


Рис. 2.1. **Пример сети обратного распространения, определяемой формулой 2 - 2 - 1 - 1, с начальными значениями весов и порогов**

Так сеть имеет 2 координаты входного вектора (2 входных узла), 2 скрытых нейрона первого слоя, 1 скрытый второго слоя и 1 выходной

нейрон (1 выходной узел), то сеть имеет архитектуру, определяемую формулой 2 - 2 - 1 - 1. Это двухслойная сеть. Первые два входных узла формируют индуцированное поле для двух нейронов первого скрытого слоя. Уже отмечалось, что при подсчете числа слоев входной слой нейронов не учитывается, так как он служит лишь для распределения входных сигналов. При подсчете слоев, по аналогии, «не замечают» и выходной слой, поскольку он лишь распределяет выходные сигналы, поступившие от последнего скрытого слоя. Каждый нейрон сети имеет свой номер, указанный после символа ∑. На вход нейрона поступает множество своих сигналов, которые перераспределяются в соответствии с набором связей (connecting link). Каждый нейрон осуществляет по определенным правилам комбинирование посылаемых к нему входных сигналов и, в соответствии с правилами активации, вычисляет выходной сигнал Y_K (соответствующее выходное значение называют активностью нейрона). Выходной сигнал каждого нейрона определяет функция активации (activation function) нейрона $F(v_K)$. На вход функции активации поступает значение – уровень активации нейрона (или индуцированное локальное поле) – $v_K = \sum_{k=0}^{N} P_k W_k$. Уровень активации нейрона определяется линейной комбинацией входных воздействий $u_k = \sum_{k=1}^{N} P_k W_k$ и порога $v_K = \sum_{k=1}^{N} P_k W_k + b_k = u_k + b_k$. В качестве функции активации всех нейронов $F(v_K)$ использована униполярная сигмоидальная функция (логистическая), которая определяется формулой $Y_K = F(v_K) \Leftrightarrow Y_K = \frac{1}{1+\ell^{(-v)}}$.

В методе обратного распространения ошибок сначала выполняется расчет движения входного сигнала по сети. Проведем вычисления: для нейрона 3 первого скрытого слоя уровень активации нейрона $z_3 = \Sigma_3 + b_3 = w_{31}u_1 + w_{32}u_2 + b_3$= -0,3x0,2+0,5x0,1+(-0,4) = -0,41. Выходной сигнал этого нейрона определяет функция активации $u_3 = F(z_3)$. Он равен $u_3 = F(-0.41) = 0.39$. Для нейрона 4 первого скрытого слоя уровень активации нейрона $z_4 = \Sigma_4 + b_4 = w_{41}u_1 + w_{42}u_2 + b_4$= -0,3x(-0,3)+ 0,5x0,3+(-0,3) = -0,06. Выходной сигнал этого нейрона определяет функ-

ция активации $u_4 = F(z_4)$. Он равен $u_4 = F(-0.06) = 0.485$. На этом этапе сигнал дошел до второго скрытого слоя. Поэтому следующий шаг состоит в том, чтобы переместить сигнал в выходной слой. Для нейрона 5 второго скрытого слоя уровень активации нейрона $v_5 = \Sigma_5 + b_5 = w_{53}u_3 + w_{54}u_4 + b_5$ = 0,399x(-0,4)+0,485x0,1+(-0,5) = -0,17. Выходной сигнал этого нейрона определяет функция активации $u_5 = F(v_5)$. Он равен $u_5 = F(-0.17) = 0.352$. Этот же сигнал является выходом сети. Правильным (желаемым) откликом сети на тестовый входной сигнал является $d_5(1)$=0,2. Расчитанный же сетью $y_5(1)$=0,352. Следовательно, на первом шаге возникла ошибка $e_5(1) = d_5(1) - y_5(1)$=(0,2-0,352)=-0,152. Чаще ошибку характеризуют величиной среднеквадратичной ошибки $E(n) = \frac{1}{2}\sum_{J \in C} e_J^2(n) = 0.5[d_J(n) - y_J(n)]^2$= 0,5x0,152x0,152 = 0,076. Применим обратное распространение. Этот процесс предполагает изменение синаптических весов в соответствии с дельта-правилом. Используем при расчетах коэффициент обучения, равный 0,5. Начнем расчеты с того, что обновим веса, участвующие в связях между нейронами первого и второго скрытого слоя. Коррекция Δw_{ji}, применяемая к синаптическому весу, определяется общими формулами коррекции $\Delta w_{JI}(n+1) = -\eta\delta_J(n)y_i(n)$ и локального градиента $\delta_J(n) = e_J(n)F_J'[v_J(n)]$. В принятых ранее обозначениях нейроны j являются нейронами второго скрытого слоя. Нейрон k является нейроном выходного слоя. Поэтому локальный градиент на этом шаге $\delta_k(1) = e_k(1)F_k'[v_k(1)]$. Поскольку производная униполярной сигмоидальной функции определяется как $dF/dv = F(v)[1 - F(v)]$, получаем формулу коррекции синаптического веса Δw_{ji} для рассматриваемого случая, $\Delta w_{kj}(n+1) = -2\eta\delta_k(n)y_k(n) =$ $= -2\eta e_k(n)F_k'[v_k(n)]y_j(n) = -2\eta e_k(n)F(v_k)[1 - F(v_k)]y_j(n)$. Коррекция Δw_{53}, применяемая к синаптическому весу между нейронами 5 и 3, рассчитывается так: $\Delta w_{53}(2)$ =(-2)x0,5x(0,2-0,352)x0,352x(1-0,352)x0,399 =-0,0138. Коррекция $\Delta w_{54}(2)$=(-2)x0,5x(0,2-0,352)x0,352x(1-0,352)x0,485 =-0,0168. Коррекция Δw_{B5}, применяемая к порогу нейрона 5, рассчиты-

вается следующим образом $\Delta w_{B5} = -2\eta e_k(n)F(v_k)[1-F(v_k)]y_j(n) =$ (-2)x0,5x(0,2-0,352)x0,352x(1-0,352)x1=-0,03462. Теперь нужно рассчитать обновление весов, участвующих в связях между входными нейронами и первого скрытого слоя. Нейроны i это нейроны входного слоя. Нейроны j являются нейронами первого скрытого слоя. Коррекция Δw_{ji}, применяемая к синаптическому весу, в этом случае определяется общей формулой, которая требует знания сигналов ошибок $e_k(n)$ для всех нейронов слоя, находящегося правее скрытого нейрона j – $\delta_k(n)$ и множества синаптических весов этих связей $w_{kj}(n)$.

$$\Delta w_{JI}(n+1) = -2\eta\delta_J(n)y_i(n) = -\eta\left\{F_J'[z_J(n)]\sum_k \delta_k(n)w_{kj}(n)\right\}y_i(n) -$$

$$-2\eta\left\{F(z)[1-F(z)]\cdot\sum_k e_k(n)F_k'(v_k(n))w_{kj}(n)\right\}y_i(n)$$

Подставляя численные значения, получаем:

$\Delta w_{31}(2) =$(-2)x0,5x0,352x(1-0,352)x(0,2-0,352)x0,399x(1-0,399)x0,3=-0,001.

Аналогично можно посчитать другие требуемые коррекции:

$\Delta w_{32}(2) = 0{,}00166$, $\Delta w_{41}(2) =$-0,001, $\Delta w_{42}(2) =$-0,00043.

Далее, требуется провести коррекцию порогов нейронов скрытых слоев. Коррекция Δw_B, применяемая к порогу нейронов 3 и 4, определяется как Δw_{B3}=0,00332, Δw_{B4}=-0,00086. Последний шаг проведения расчетов на этом цикле – это обновление, то есть получение новых значений весов и порогов с помощью формулы $w_{JI}(n+1) = w_{JI}(n) + \Delta w_{JI}(n+1))$. Подставляя численные значения получаем $w_{53}(2) = w_{53}(1) + \Delta w_{53}(2) =$ -0,4-0,01381= -0,414, $w_{54}(2) = 0{,}083$, $w_{31}(2) = 0{,}199, w_{41}(2) =$-0,3, $w_{32}(2) = 0{,}102, w_{42}(2) =$-0,30, $b_3 =$-0,397, $b_4 =$-0,301, $b_5 =$-0,535. Обновление весов на данном цикле (на данной эпохе) для обучающего сигнала завершено. Если ввести тот же обучающий сигнал, провести расчет движения входного сигнала по сети, но уже с новыми весами, то можно убедиться, что алгоритм обратного распространения действительно уменьшит ошибку на выходе сети.

Как вы убедились алгоритм обратного распространения ошибок при проведении «ручных» расчетов, важных для понимания «техноло-

гии» метода, является громоздким и трудоемким. Ситуация меняется коренным образом если для этой цели использовать пакет программ «Нейронные сети» в составе MATLAB. Вводим в командное окно:

```
clear,
% Создаем многослойную сеть обратного распространения ошибок
% На вход сети будет подан вектор из двух компонент
% Полагаем, что каждая компонента вектора может
% изменяться в диапазоне[-1 1] [0 1]
% Первый скрытый слой содержит два нейрона
% Второй скрытый слой содержит один нейрон
% Функция активации всех нейронов
% униполярная сигмоидальная функция 'logsig'
net = newff([-1 1; 0 1],[2,1],{'logsig','logsig'},'traingd');
%  Задаем начальные значения весов и порогов
%  сети, в соответствии с задачей.
net.IW{1,1}=[0.2 0.1;-0.3 0.3];
net.b{1,1}=[-0.4; 0.3];
net.LW{2,1}=[-0.4 0.1];
net.b{2,1}=-0.5;
%  Проверяем установленные веса и пороги
disp(' Начальные значения весов и порогов')
disp(' первого скрытого слоя')
w1=net.IW{1,1}
b1=net.b{1,1}
disp(' второго скрытого слоя')
w2=net.LW{2,1}
b2=net.b{2,1}
disp(' Входной вектор')
P = [-0.3;0.5]
disp('Абсолютная ошибка при прямом')
disp('движении входного сигнала по ети')
E=Test-Yk
disp('Среднеквадратическая ошибка при прямом движении')
disp(' входного сигнала по сети')
err1=E*E
% Определяем объектно-ориентированные параметры сети,
%  определяющие процесс обучения.
%  Параметр скорости обучения
net.trainParam.lr = 0.5;
%  Используемое число эпох
net.trainParam.epochs = 1;
%  Проводим обучение сети
net1 = train(net,P,Test);
%  Определяем новые веса и пороги,
%  полученные в результате коррекции
%  при обратном движении сигналов
disp(' Обновленные на протяжении одной эпохи')
disp(' значения весов и порогов')
disp(' первого скрытого слоя')
w1=net1.IW{1,1}
b1=net1.b{1,1}
disp(' второго скрытого слоя')
w2=net1.LW{2,1}
b2=net1.b{2,1}
%  Рассчитываем обновленный отклик
%  сети на входной тестовый сигнал
disp('Отклик сети при скорректирован -')
disp('ных значениях весов и порогов')
Yk2 = sim(net1,P)
Test=0.2;
disp('Абсолютная ошибка при прямом движении входного ')
```

```
%  Рассчитываем отклик сети
%  на входной тестовый сигнал
disp('Отклик сети при начальном значе-
нии весов и порогов")
Yk = sim(net,P)
Test=0.2;
disp(' сигнала по обновленной сети')
E2=Test-Yk2
disp('Среднеквадратическая ошибка при
прямом движении входного сигнала по
обновленной сети')
err2=E2*E2
```

После расчетов по программе получаем:

Начальные значения весов и порогов:
первого скрытого слоя: w1= 0.2000 0.1000 -0.3000 0.3000 b1= -0.4000
0.3000 ; второго скрытого слоя: w2 = -0.4000   0.1000 b2 = -0.5000.
Входной вектор: P =   -0.3000   0.5000.
Отклик сети при начальном значении весов и порогов Yk =   0.3552.
Абсолютная ошибка при прямом   движении входного сигнала по сети:
E =   -0.1552.
Среднеквадратическая ошибка при прямом   движении входного сигнала
по сети:err1 =   0.0241.
TRAINGD, Epoch 0/1, MSE 0.0240782/0, Gradient 0.0890971/1e-010
TRAINGD, Epoch 1/1, MSE 0.0203021/0, Gradient 0.080473/1e-010
TRAINGD, Maximum epoch reached, performance goal was not met.
Обновленные на протяжении одной эпохи   значения весов и порогов:
первого скрытого слоя: w1 =   0.1990   0.1017   -0.2998   0.2996
b1 =   -0.3966   0.2992.
второго скрытого слоя: w2 =   -0.4142   0.0775 b2 =   -0.5355.
Отклик сети при скорректированных значениях весов и порогов:
Yk2 =   0.3425.
Абсолютная ошибка при прямом движении входного сигнала по обновленной сети: E2 =   -0.1425.
Среднеквадратическая ошибка при прямом движении входного сигнала по обновленной сети: err2 =   0.0203.

Проведенные расчеты свидетельствуют о том, что одна итерация (одна эпоха) алгоритма обратного распространения ошибки позволила уменьшить среднеквадратическое значение ошибки с величины err1 = 0.0241 до величины err2 = 0.0203. Это также отражает выводимая запись отчета о вычислениях (Epoch 0/1, MSE 0.0240782/0, Epoch 1/1, MSE 0.0203021/0) и график, характеризующий процесс обучения. Однако такая продолжительность обучения явна недостаточна. Можно предположить, исходя из рассчитанных значений градиентов, что за такое число

итераций (эпох) вряд ли можно достигнуть минимума на поверхности ошибок. Более ценным для практики являются ситуации, когда при одном и том же множестве примеров процесс обучения длится в течение нескольких сотен эпох. Полагают, что при таком количестве итераций удается достигнуть некого локального минимума на поверхности ошибок. Вводим в командное окно:

```
clear,
% Создаем многослойную сеть обратного распространения ошибок
net = newff([-1 1; 0 1],[2,1],{'logsig','logsig'},'traingd');
%  Задаем начальные значения весов и порогов
%  сети, в соответствии с задачей.
net.IW{1,1}=[0.2 0.1;-0.3 0.3];
net.b{1,1}=[-0.4; 0.3];
net.LW{2,1}=[-0.4 0.1];
net.b{2,1}=-0.5;
figure(1);
w3=net.IW{1,1};
hintonw(w3)
disp(' Входной вектор')
P = [-0.3;0.5]
%  Рассчитываем отклик сети
%  на входной тестовый сигнал
disp(  'Отклик сети при начальном ')
disp(  ' значении весов и порогов')
Yk = sim(net,P)
Test=0.2;
disp('Абсолютная ошибка при прямом')
disp('движении входного сигнала по сети')
E=Test-Yk
disp('Среднеквадратическая ошибка при прямом')
disp('движении входного сигнала по сети')
err1=E*E
%  Определяем объектно-
net.trainParam.lr = 0.5;
%  Используемое число эпох
net.trainParam.epochs = 150;
% Количество итераций (эпох), по истечении которых будут
%  представляться (выводиться) промежуточные результаты
net.trainParam.show = 50;
%  Проводим обучение сети
net1 = train(net,P,Test);
%  Определяем новые веса и пороги,
%  полученные в результате коррекции
%  при обратном движении сигналов
disp('Обновленные на протяжении всех эпох')
disp(' значения весов и порогов')
disp(' первого скрытого слоя')
w1=net1.IW{1,1}
b1=net1.b{1,1}
disp(' второго скрытого слоя')
w2=net1.LW{2,1}
b2=net1.b{2,1}
w4=w1;
figure(2);
hintonw(w4)
%  Рассчитываем обновленный отклик
%  сети на входной тестовый сигнал
disp('Отклик сети при скорректированных значениях весов и порогов')
Yk2 = sim(net1,P)
EN=Test-Yk2
disp(' Среднеквадратическая ошибка в итоге ')
```

ориентированные параметры сети,
% определяющие процесс обучения.
% Параметр скорости обучения

disp(' работы нейронной сети')
errN=EN*EN

После запуска программы по истечении 150 эпох получаем следующие результаты:

Отклик сети при начальном значении весов и порогов: Yk = 0.3552.
Абсолютная ошибка при прямом движении по сети: E = -0.1552.
Среднеквадратическая ошибка при прямом движении err1 = 0.0241.
TRAINGD, Epoch 0/150, MSE 0.0240782/0, Gradient 0.0890971/1e-010
TRAINGD,Epoch 50/150, MSE 8.95693e-005/0, Gradient 0.00397012/1e-010
TRAINGD,Epoch100/150,MSE1.18457e-006/0,Gradient 0.000443471/1e-010
TRAINGD,Epoch150/150,MSE1.74174e-008/0,Gradient5.35904e-005/1e-010
TRAINGD, Maximum epoch reached, performance goal was not met.
Обновленные на протяжении всех эпох значения весов и порогов первого скрытого слоя: w1 = 0.1821 0.1299 -0.3018 0.3030.
b1 = -0.3402 0.3060; второго скрытого слоя: w2 = -0.6028 -0.2143
b2 = -0.9975.
Отклик сети при скорректированных значениях весов и порогов: Yk2 = 0.2001.
Среднеквадратическая ошибка в итоге работы нейронной сети:
errN = 1.7417e-008.

График на рис. 2.2 наглядно показывает, как происходит изменение ошибки в сети в процессе ее обучения.

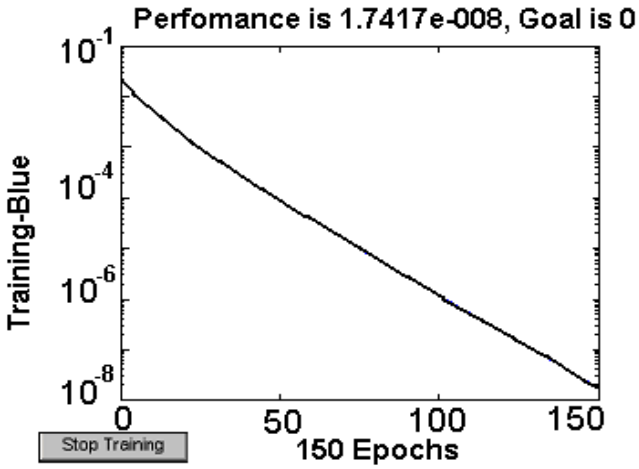
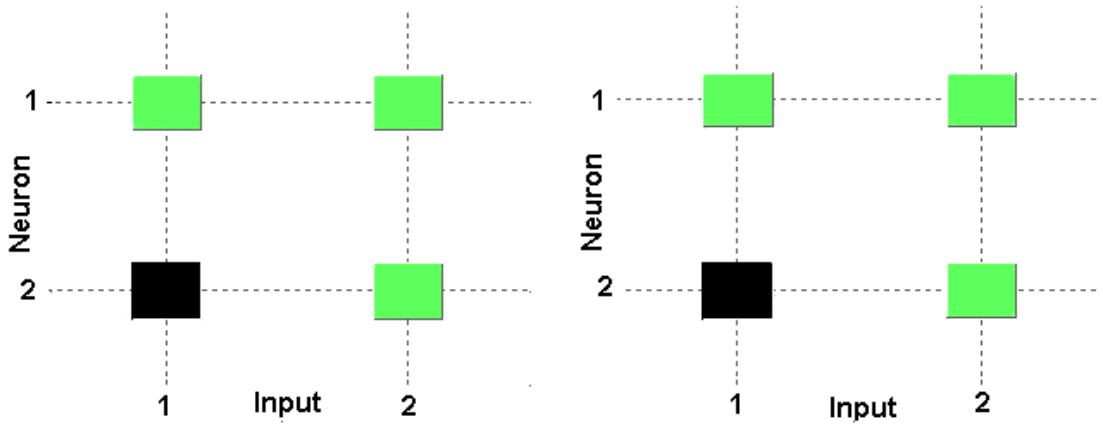


Рис. 2.2. **График изменения среднеквадратичной ошибки нейронной сети в процессе ее обучения за 150 эпох**

В процессе обучения происходит коррекция весов соединений и порогов. Чему равны, к примеру, веса между входными узлами и первым скрытым слоем в начале и в конце обучения наглядно можно посмотреть, используя так называемые хинтоновские графики матрицы весов, которые каждый весовой коэффициент отображают квадратом с площадью, пропорциональной величине данного коэффициента. Знак на таких графиках отображается цветом квадрата (рис. 2.3).



а) б)

Рис. 2.3. **Хинтоновские графики матрицы весов между входными узлами и первым скрытым слоем в начале (а) и конце (б) обучения**

2.3. Понятие нейроконтроллера

Если рассмотренную нейронную сеть создать в виде электронного изделия, например, полупроводниковой микросхемы, то такая аппаратная реализация для своего функционирования потребует очень небольшой памяти. Действительно, чтобы прошедшее обучение изделие работало, обрабатывало поступающую новую информацию и формировало выходной «отклик» на «стимул», не выходящий за определенные пределы, в «памяти обученного устройства» следует сохранять лишь информацию о структуре сети (о бинарной матрице, соответствующей графу сети) и о матрицах весовых и пороговых коэффициентов. Кроме того, несложно предположить, что такая аппаратная реализация будет обладать большим быстродействием, так как в ней осуществляется «распараллеливание» процесса вычислений. Таким образом, развивая мысль, можно предположить, что, используя достижения в области современной микроэлектроники, вполне вероятно создать вычислители

нового класса, обеспечивающие более быстрое, более дешевое и качественное решение конкретных задач. Подобные вычислительные средства («нейрокомпьютеры») могут, к примеру, применяться для создания нейроконтроллеров (neurocontroller), используемых для трудно формализуемых задач управления сложным оборудованием. Из электроники известно, что создание эффективной системы управления представляет собой довольно сложную задачу, так как реальные процессы в электронном оборудовании характеризуются, как правило, нелинейными зависимостями, высоким уровнем шумов, меняющимися условиями функционирования, изменением передаточных характеристик объектов при вариации параметров окружающей среды. Необходимость решения задач управления в реальном масштабе времени выдвигает серьезные требования, как к самим алгоритмам управления, так и к техническим средствам их реализующим.

Чтобы понять разницу между нейроконтроллером и традиционным контроллером, используемым для решения задач управления, рассмотрим так называемое ситуационное управление. Оно заключается в том, что для каждого значения вектора, характеризующего сложившуюся ситуацию, надо найти вектор, описывающий то решение, которое следует предпринять. При этом типично, что все ситуации отразить невозможно. По этому поводу отметим следующее. В традиционных ЭВМ программы выполняют точно установленные инструкции в определенный момент времени, формируя последовательность действий, пока не будет получен результат. Напротив, в нейрокомпьютерах прохождение данных по нейронной сети и их преобразования не могут быть заранее определены. Кроме того, входные данные могут быть недоопределены, что в традиционных программных продуктах не представляется возможным. Поставленная задача может быть решена нейрокомпьютером, даже если входная информация не рассматривалась ранее при обучении, при условии, что новые обрабатываемые данные не выходят за предъявляемые к ним ограничения.

Как известно, наряду с управляющими нейроконтроллерами для решения таких прикладных «интеллектуальных» задач, как выдача рекомендаций на основе анализа данных (решения задач ситуационного управления, «расчета поведения», выбора действия из доступного списка на основании того, в какой окружающей среде функционирует оборудование), могут быть применены использующие традиционные ЭВМ

экспертные системы. Они используют фиксированные «деревья поведения» или «конечные автоматы», что формирует предсказуемый «отклик» в данной окружающей среде. Основное преимущество нейроконтроллеров заключается в возможности избежать традиционной процедуры сбора информации (получения «знаний») при помощи «экспертов». При создании экспертных систем требуется проводить «интервью» с экспертами для получения «правил» поведения исследуемого оборудования при определенных условиях, что требует значительных временных и материальных затрат. Более того, процедура получения экспертных оценок может не дать желаемого эффекта, так как нет гарантии, что все необходимые правила будут получены и экспертная система будет работать в слегка изменившихся условиях окружающей среды.

2.4. Устройства «интеллектуального» управления на основе нейронной сети и их особенности

После этих общих замечаний, возникших на основе ассоциаций о работе алгоритма обратного распространения в многослойной нейронной сети и касающихся возможностей использования нейронных сетей в целях непосредственного управления оборудованием, повышения эффективности управления путем коррекции управляемых параметров на основе «обучения», перейдем к дальнейшему изучению нейронных сетей и конкретных примеров их реализации. Рассмотрим, в частности, как нейронная сеть, работающая с использованием алгоритма обратного распространения и обученная на ограниченном количестве примеров (образцов поведения в зависимости от обстановки), приобретет способностью генерировать правильный отклик на различные ситуации, не входящие в набор обучающих.

Чтобы было понятнее, о чем в дальнейшем пойдет речь, предположим, что мы держим в руках некий цифровой фотоаппарат, обладающий широким спектром функциональных возможностей. Как известно, для проведения качественной съемки всегда требуется правильно установить значение экспозиции (exposure value) – количество света, попадающего на ПЗС матрицу во время процесса съемки одного кадра. Экспозиция (или, как часто говорят, выдержка) определяется временем, в течение которого затвор фотоаппарата остается открытым и изображение попадает на матрицу. Измерение экспозиции производится в долях

секунды. Чтобы правильно определить экспозицию цифровые камеры производят измерение яркости изображения. Значение экспозиции находится в зависимости от размера диафрагмы и чувствительности ISO (ISO sensitivity) ячеек ПЗС матрицы (чем ниже значение чувствительности, тем большее количество времени потребуется для попадания нужного количества света внутрь фотоаппарата). Цифровые фотоаппараты имеют автоматическую установку режима ISO. Это значит, что при уменьшении освещения или при переходе к съемке в инфракрасном диапазоне чувствительность автоматически изменяется. В современных цифровых аппаратах имеются электронные устройства ситуационного управления, которые в зависимости от чувствительности ISO автоматически управляют экспозицией с помощью комбинации диаграммы и выдержки. При этом такие устройства могут работать, используя различные режимы. В режиме приоритета выдержки задается нужное значение выдержки, а размер диафрагмы устанавливается автоматически. В другом режиме можно задать значение диафрагмы, а выдержка будет установлена автоматически. При этом на управляющее электронное устройство может быть также возложена функция осуществления контроля состояния всех механизмов, отвечающих за получение качественной фотографии. В итоге получается, что в руках мы держим фотоаппарат, у которого есть электронное устройство, которое в зависимости от сюжета (портрет, спорт, пейзаж, ночная сцена) автоматически должно на основе сигналов датчиков рассчитать и произвести настройку всех нужных параметров.

Пусть управляющее электронное устройство содержит входы (куда поступают сигналы с датчиков с информацией о чувствительности ISO, об установленном значение экспозиции) и выходы для формирования управляющих воздействий (установки требуемого значения диафрагмы). В результате своего функционирования оно должно рассчитать «отклик» на сигналы входов и сформировать сигнал, посредством которого можно будет выполнить действия, необходимые в соответствии со сценарием съемки. Предположим, что требуется провести расчет «поведения» управляющего устройства для ситуации, представленной в таблице 2.1. Пусть вас не смущают численные значения в таблице, которые явно не соответствуют величинам, имеющимся в техническом описании фотоаппарата. Мы изучаем работу устройства, поэтому не будем пока придерживаться точных фактических данных, а используем простые цифры,

удобные для анализа. Чтобы упредить возможно уже возникший вопрос, отметим следующее. Казалось бы, приведенная таблица имеет некое сходство с известными таблицами истинности.

Таблица 2.1

№	Установленное значение выдержки	Чувствительность ISO	Нужное поведение (значение диафрагмы)
	P_1	P_2	T
1	-1	0	-1
2	-1	5	-1
3	2	0	1
4	2	5	1

Поэтому это наводит на мысль, что под данную «задачу», используя схемотехнический подход, применяя систему логических функций, можно в принципе создать электронную схему. Такая электронная схема на логических элементах могла бы верно служить, способствуя определению отклика на сложившуюся ситуацию. Такое предположение было бы верным, если бы мы предусмотрели все возможные ситуации и точно бы знали, какое решение соответствует каждой из них, и всегда обладали полной и точной информацией о происходящих событиях. Но наше устройство должно функционировать в условиях неопределенности и помех. Действительно, если детально знакомиться с тем, как производятся измерения интенсивности света (яркости) изображения, то можно увидеть, что существуют несколько подходов к построению экспозамера. В одном случае, в качестве данных могут быть использованы измерения во всем пространстве кадра, которые «принимают во внимание» все части изображения. В другом случае могут проводиться более «грубые» измерения яркости, отдавая предпочтение центральной части изображения. Наконец, зона замера может ограничиваться лишь небольшим фрагментом от общей части кадра и, даже, несколькими пикселями. В условиях «бликов» и съемки сюжетов с различной яркостью участков изображения можно получить совершенно разные результаты и, соответственно, разное качество снимков. Следует отметить и то, что при увеличении чувствительности элементов ПЗС усиливается электронный «шум», также оказывающий влияние на качество снимка. Все это свиде-

тельствует о том, что в изучаемой ситуации мы вынуждены от точного, детерминированного представления перейти в область приблизительного, неточного, как часто говорят, стохастического (статистического). И здесь, в решении такой задачи, не свойственной электронной технике, булевы переменные и логические функции нам не помогут.

Осознав невозможность использования традиционного схемотехнического подхода, приступим к рассмотрению «интеллектуального» устройства управления на основе нейронной сети, выходом которого являются «действия» на основе «способности к пониманию поведения» управляемого объекта и к работе в изменяющихся условиях. Другими словами, рассмотрим самонастраивающуюся систему с «учителем», включенную в некую «среду».

Пусть исходная ситуация характеризуется тем, что из технических параметров фотоаппарата известно множество тех значений выдержек $\mathbf{P_1}$ и чувствительностей ISO $\mathbf{P_2}$, которые были заложены при его конструировании. Набор цифр, представленный в таблице 2.1, в принципе можно рассматривать с двух диаметрально противоположных точек зрения. С одной стороны, каждую строку таблицы, состоящую из компонент сигнала **P**, можно воспринимать как некий «изменяющийся во времени» сигнал (uniformly spaced in time), у которого значения «измерены» через равномерные промежутки времени (шаги с номером). А, с другой стороны, множество входных сигналов можно рассматривать как некие элементы (точки) абстрактного пространства, поэтому таблицу, состоящую из компонент сигнала **P**, можно считать моментальным снимком данных в пространстве (snapshot of data). Из этого следует, что и предопределенное множество обучающих примеров может быть предъявлено обучающейся сети двумя различными способами – последовательно («во времени») и сразу все (в виде группы).

При последовательном режиме (sequential mode) обучения по методу обратного распространения, называемом также интерактивным (on-line), корректировка весов производится после подачи каждого примера. Это похоже на то, как мы делали ранее, изучая алгоритм обратного распространения. Мы выполняли следующее: если имеется N обучающих примеров, упорядоченных следующим образом $\{(p(1),d(1)),\ \ ...,\ \ (p(N),d(N))\}$, то мы предъявляли сети сначала первый пример $(p(1),d(1))$, после чего осуществляли описанные выше

«прямые» и «обратные» вычисления или, по-другому, проходы (passes). Результатом этого являлась корректировка синаптических весов и уровней порогов сети. После этого сети предъявляется вторая пара $(p(2), d(2))$, повторяются прямой и обратный проходы, приводящие к следующей коррекции весов. Этот процесс повторяется, пока сеть не завершит обработку последнего примера (пары) $(p(N), d(N))$. Исходя из таблицы 2.1, при последовательном режиме обучения на вход сети подается последовательность из четырех векторов входа:

$$p_1 = \begin{pmatrix} p_{11} \\ p_{21} \end{pmatrix} = \begin{pmatrix} -1 \\ 0 \end{pmatrix},\ p_2 = \begin{pmatrix} p_{12} \\ p_{22} \end{pmatrix} = \begin{pmatrix} -1 \\ 5 \end{pmatrix},$$

$$p_1 = \begin{pmatrix} p_{13} \\ p_{23} \end{pmatrix} = \begin{pmatrix} 2 \\ 0 \end{pmatrix},\quad p_1 = \begin{pmatrix} p_{14} \\ p_{24} \end{pmatrix} = \begin{pmatrix} 2 \\ 5 \end{pmatrix}.$$

Номера индексов вектора таковы: первое число – это номер компоненты (координаты) вектора, второе – номер вектора (номер шага абстрактного времени).

При создании и моделировании работы искусственных нейронных сетей в пакете «Нейронные сети» MATLAB важно помнить, что при последовательном режиме обучения для ввода исходных данных, состоящих из нескольких последовательностей сигналов на входе, необходимо формировать массив ячеек, размер каждой из которых совпадает с числом таких последовательностей.

В учебных пособиях по MATLAB вы найдете информацию о том, что массив ячеек – это такая конструкция, в которой элементами являются ячейки-«контейнеры» (cell), позволяющие хранить «элементы» разных типов и размерностей. В частности, одна из ячеек может быть числом, вторая – вектором комплексных чисел, третья может содержать действительную матрицу. Массивы ячеек имеют определенные преимущества в тех случаях, когда данные могут содержать разнородные данные, различное количество полей, когда требуется применить одну инструкцию обработки ко многим полям и в ряде других важных для практики случаях. Указателем (конструктором) массива ячеек являются фигурные скобки { }. В эти скобки помещены индексы отдельных ячеек. Иначе говоря, чтобы знать, из каких ячеек состоит массив ячеек, получать доступ к содержимому ячейки, используют индексацию – присвоение каждой ячейке индексов, подобно тому, как это делается для мат-

риц. Например, массив ячеек с индексами в фигурных скобках A {2,2} означает, что в массиве имеется массив, состоящий из четырех ячеек, размером 2x2. Исходя из того, что в нашем случае на вход сети подается последовательность из четырех векторов входа, и при этом вход **P** должен быть массивом ячеек, каждая из которых содержит два элемента по числу последовательностей, оператор присваивания численных значений ячейкам массива [2x1 double] (2 строки, 1 столбец) будет иметь вид P = {[-1;0],[-1;5],[2;0],[2;5]}. Если необходимо убедиться в том, что правильно сформировался входной сигнал в виде массива формата cell, следует провести преобразование массива ячеек в числовой массив P1 = [P{:}] P1 = -1 -1 2 2 0 5 0 5. По этим результатам можно заключить, что имеем дело с четырьмя векторами, состоящими их двух компонент.

В пакетном режиме (batch mode) обучения по методу обратного распространения корректировка весов и порогов производится после подачи прохождения по сети всех примеров обучения. В этом случае входной сигнал состоит из двух векторов (вектор-столбцов), каждый из которых имеет по четыре координаты (компоненты). При моделировании работы искусственных нейронных сетей в MATLAB в пакетном режиме входной сигнал должен быть представлен в виде массивов формата double array: p = [-1 -1 2 2;0 5 0 5] (двух вектор-столбцов). Следует иметь в виду, что исходные данные о входном сигнале, представленные для обучения пакетном режиме могут быть преобразованы в массивы ячеек. Для этой цели используется оператор p1 = num2cell(p,1).

Используя оба рассмотренные режима, обучим управляющее устройство «поведению» для ситуаций, представленных в таблице 2.1. Будем полагать, что при проведении натурных испытаний фотоаппарата при съемках всевозможных пейзажей с разным уровнем освещенности удалось найти желательные значения для формирования управляющих воздействий (заводские установки требуемого значения диафрагмы). Эти значения представлены в столбце T таблицы 2.1. Будем также считать, что в устройстве так называемого ситуационного управления в воплощена двухслойная нейронная сеть с сигмоидальным и линейным слоями, способная для обучения ее на основе метода обратного распространения ошибки. Начнем с последовательного режима обучения, когда для ввода исходных данных на входе используется последовательность сигналов в виде четырех векторов. Чтобы подготовить модель

сети к процедуре последовательной адаптации (последовательного обучения) на основе алгоритма GD, необходимо указать ряд параметров. В первую очередь, это имя функции настройки learnFcn, соответствующее алгоритму градиентного спуска (в данном случае – это M-функция learngd). С функцией learngd связан лишь один параметр скорости настройки lr. Текущие приращения весов и смещений сети определяются умножением этого параметра на вектор градиента. Чем больше значение параметра, тем больше приращение на текущей итерации. Если параметр скорости настройки выбран слишком большим, алгоритм может стать неустойчивым; если параметр слишком мал, то алгоритм может потребовать длительного счета. При выборе функции learngd по умолчанию устанавливается значение параметра скорости настройки lr: 0.0100 . Мы будем использовать net.layerWeights{2,1}.learnParam.lr = 0.2, то есть увеличим значение этого параметра до 0.2.

Для обучения сети требуется задать обучающие пары (обучающее множество). Проще множество входов и целей задать по столбцам таблицы 2.1, то есть определить следующим образом p = [-1 -1 2 2;0 5 0 5]; t = [-1 -1 1 1]. Однако, поскольку используется последовательный способ обучения, необходимо преобразовать массивы входов и целей в массивы ячеек p = num2cell(p,1); t = num2cell(t,1). Последний параметр, который требуется установить при последовательном обучении, это число проходов. Выберем net.adaptParam.passes = 100. После этого можно выполнить настройку параметров, используя процедуру адаптации [net,a,e] = adapt(net,p,t). Чтобы проверить качество обучения, после окончания обучения проверим, как работает сеть. Для этого вычислим «отклик» на входной сигнал a = sim(net,p). Вводим в командное окно:

```
clear
% Создаем многослойную сеть обратного распространения ошибок
% На вход сети будет подан входной сигнал из четырех векторов
% Полагаем, что каждая компонента вектора может
% изменяться в диапазоне[-1 2] [0 5]
% Первый скрытый слой содержит три нейрона
% Второй скрытый слой содержит один
netsq.inputWeights{1,1}.learnFcn = 'learngd';
netsq.layerWeights{2,1}.learnFcn = 'learngd';
% Устанавливаем параметр скорости обучения
netsq.layerWeights{2,1}.learnParam.lr = 0.2;
% Устанавливаем число проходов
netsq.adaptParam.passes = 100;
%  Рассчитываем отклик сети
```

```
нейрон
% Функция активации нейронов первого слоя нейронов -
% биполярная сигмоидальная функция 'tansig'
% Функция активации нейронов второго слоя нейронов -
% линейная (тождественная)  'purelin'
% Функция обучения нейронной сети 'traingd'-
% реализует алгоритм обратного распространения ошибки
netsq = newff([-1 2; 0 5],[3,1],{'tansig','purelin'},'traingd');
% Устанавливаем объектно-ориентированные функции адаптации сети
netsq.biases{1,1}.learnFcn = 'learngd';
netsq.biases{2,1}.learnFcn = 'learngd';
%  на входной тестовый сигнал
disp(' Входной сигнал')
p = [-1 -1 2 2;0 5 0 5]
disp(' Желательный отклик')
test = [-1 -1 1 1]
p = num2cell(p,1);
t = num2cell(test,1);
%  Адаптируем(обучаем в последовательном режиме ) сеть
[netsq1,a,e] = adapt(netsq,p,t);
%  Рассчитываем отклик сети
%  на входной  сигнал
disp( 'Полученный отклик сети на входной  сигнал')
y = sim(netsq1,p)
disp('Среднеквадратическая ошибка в итоге ')
disp('работы нейронной сети')
err =mse(e)
```

После решения задачи получаем:

Входной сигнал: p = -1 -1 2 2 0 5 0 5.

Желательный отклик: test = -1 -1 1 1.

Полученный отклик сети на входной сигнал:

y = [-1.0000] [-1.0000] [1] [1]

Среднеквадратическая ошибка в итоге работы нейронной сети: err = 6.1630e-033.

Посмотрим теперь, как изменятся установки диафрагмы, если выдержка или чувствительность будут несколько отличаться от тех, которые были определены заводскими установками. Сразу после проведения предыдущих вычислений вводим в командное окно:

```
disp(' Входной сигнал 1')
p1 = [-1 -1 2 2;1 4.5 0 4]
disp(' Желательный отклик')
test = [-1 -1 1 1]
p1 = num2cell(p1,1);
%  Рассчитываем отклик сети
%  на входной  сигнал 1
disp('Полученный отклик сети на входной  сигнал 1')
disp(' Входной сигнал 2')
p2 = [-1 -1 2 1.8;1 5 0.5 5]
disp(' Желательный отклик')
test = [-1 -1 1 1]
p2 = num2cell(p2,1);
%  Рассчитываем отклик сети
%  на входной  сигнал
disp('Полученный отклик сети на входной  сигнал')
```

```
y1 = sim(netsq1,p1)                          y2 = sim(netsq1,p2)
disp('Среднеквадратическая ошибка в ')       disp('Среднеквадратическая ошибка в ')
disp(' итоге работы нейронной сети')         disp(' итоге работы нейронной сети')
err1 =mse(e)                                 err2 =mse(e)
```

После запуска программы получаем:

```
Входной сигнал: 1 p1 =   -1.0000   -1.0000    2.0000    2.0000
                          1.0000    4.5000         0    4.0000
  Желательный отклик: test =    -1    -1     1     1
Полученный отклик сети на входной  сигнал 1:
y1 =     [-1.1198]    [-1.0132]    [1]    [0.9813]
Входной сигнал 2:
p2 =   -1.0000   -1.0000    2.0000    1.8000
          1.0000    5.0000    0.5000    5.0000
 Желательный отклик: test =    -1    -1     1     1
Полученный отклик сети на входной  сигнал 2:
y2 =     [-1.1198]    [-1.0000]    [0.9994]    [0.5734].
```

Полученные результаты свидетельствуют о том, что неточности, связанные с определением выдержки при различных способах измерения яркости или погрешности установки чувствительности, оказывают слабое влияние на электронное устройство ситуационного управления и в фотоаппарате устанавливаются практически те же значения диафрагмы. Другими словами можно сказать, что вариации компонентов вектора **P**, характеризующего сложившуюся ситуацию, оказывают слабое влияние на значения координат вектора **Y**, описывающего решение, которое нужно предпринять. Отметим, что компоненты векторов не обязательно должны быть булевыми; они могут целыми и вещественными числами.

Осуществим теперь пакетный режим или, как часто говорят, групповое обучение сети. В этом случае для обучения сети на основе «классического» алгоритма GD обратного распространения ошибки взамен функции настройки learngd необходимо использовать M-функцию traingd. При этом нет необходимости, как это было в предыдущем случае, задавать индивидуальные функции обучения слоев для весов и смещений, а достаточно указать одну эту обучающую функцию для всей сети. Считаем, как и ранее, что в устройстве ситуационного управления реализована та же двухслойная нейронная сеть прямой передачи сигнала с сигмоидальным и линейным слоями, которая предназначена для обучения по методу обратного распространения ошибки. В

функции traingd по умолчанию заданны следующие параметры (net.trainParam): количество циклов обучения (epochs) – 100; целевая ошибка обучения (goal) – 0; максимально допустимая кратность превышения ошибки проверочной выборки по сравнению с достигнутым минимальным значением (max_fail) – 5; минимальное значение градиента (min_grad) – 1.0000e-010; количество циклов для показа промежуточных результатов (show) – 25; максимальное время обучения в секундах (time) – Inf, то есть бесконечность (∞). Наконец, по умолчанию параметр lr скорости обучения равен 0.01. Установим новые значения параметров обучения и, задав обучающую последовательность p = [-1 -1 2 2;0 5 0 5], t = [-1 -1 1 1] в виде массива double, выполним процедуру обучения net = train(net,p,t). Вводим в командное окно:

```
clear
% Создаем многослойную сеть обратного распространения ошибок
% На вход сети будет подан входной сигнал из двух
% четырехкомпонентных векторов
% Полагаем, что каждая компонента вектора может
% изменяться в диапазоне[-1 2] [0 5]
% Первый скрытый слой содержит три нейрона
% Второй скрытый слой содержит один нейрон
% Функция активации нейронов первого слоя нейронов - биполярная сигмоидальная функция 'tansig'
% Функция активации нейронов второго слоя нейронов -
% линейная (тождественная) 'purelin'
% Функция обучения нейронной сети 'traingd'- реализует алгоритм обратного распространения ошибки
netbm = newff([-1 2; 0 5],[3,1],{'tansig','purelin'},'traingd');
% Определяем объектно-ориентированные параметры сети,
% определяющие процесс обучения.
% Параметр скорости обучения
netbm.trainParam.lr = 0.05;
% Используемое число эпох
netbm.trainParam.epochs = 300;
% Количество итераций (эпох),по истечении которых будут
% представляться (выводиться) промежуточные результаты
netbm.trainParam.show = 50;
% Значение целевой ошибки обучения
netbm.trainParam.goal = 1e-5;
% Определяем вх-й и тестовый сигнал
disp(' Входной сигнал')
p = [-1 -1 2 2;0 5 0 5]
disp(' Желательный отклик')
test = [-1 -1 1 1]
% Проводим обучение сети
netbm1 = train(netbm,p,test);
% Рассчитываем отклик сети
% на входной сигнал
disp( 'Полученный отклик сети на входной сигнал')
y1 = sim(netbm1,p)
EN=test-y1;
disp('Среднеквадратическая ошибка в итоге работы нейронной сети')
err1 =mse(EN)
```

После запуска программы по истечении 237 эпох получаем следующие результаты:

Входной сигнал: p = -1 -1 2 2 0 5 0 5.

Желательный отклик: test = -1 -1 1 1.

TRAINGD, Epoch 0/300, MSE 0.665136/1e-005, Gradient 1.61966/1e-010

TRAINGD, Epoch 50/300,MSE 0.0154742/1e-005, Gradient 0.114185/1e-010

TRAINGD, Epoch 100/300, MSE 0.00204518/1e-005, Gradient 0.0399121/1e-010

TRAINGD, Epoch 150/300, MSE 0.000290677/1e-005, Gradient 0.0149554/1e-010

TRAINGD, Epoch 200/300, MSE 4.17785e-005/1e-005, Gradient 0.00566393/1e-010

TRAINGD, Epoch 237/300, MSE 9.95764e-006/1e-005, Gradient 0.00276464/1e-010

TRAINGD, Performance goal met.

(Последняя запись свидетельствует о том, что обучение было остановлено, поскольку ошибка обучения стала меньше заданной).

Полученный отклик сети на входной сигнал:

y1 = -0.9975 -1.0001 1.0043 0.9961.

Среднеквадратическая ошибка в итоге работы нейронной сети:

err1 = 9.9576e-006.

График на рис. 2.4 наглядно демонстрирует, как происходит изменение ошибки в сети в процессе ее обучения.

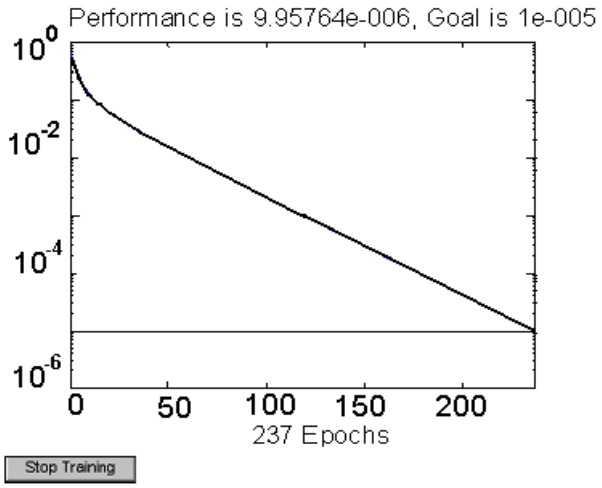


Рис. 2.4. **График изменения среднеквадратичной ошибки нейронной сети в процессе ее обучения за 237 эпох при значении целевой ошибки обучения goal = 1e-5**

Посмотрим и теперь, как изменятся установки диафрагмы, если выдержка или чувствительность будут несколько отличаться от тех, которые были определены заводскими установками. После проведения предыдущих вычислений вводим в командное окно:

```
disp(' Входной сигнал 1')
p1 = [-1 -1 2 2;1 4.5 0 4]
disp(' Желательный отклик')
test = [-1 -1 1 1]
%  Рассчитываем отклик сети
%  на входной  сигнал 1
disp('Полученный отклик сети на вход-
ной  сигнал 1')
y1 = sim(netbm1,p1)
EN1=test-y1;
disp('Среднеквадратическая ошибка в
итоге ')
disp(' работы нейронной сети')
err1 =mse(EN1)
```

```
disp(' Входной сигнал 2')
p2 = [-1 -1 2 1.8;1 5 0.5 5]
disp(' Желательный отклик')
test = [-1 -1 1 1]
%  Рассчитываем отклик сети
%  на входной  сигнал
disp('Полученный отклик сети на вход-
ной  сигнал')
y2 = sim(netbm1,p2)
EN2=test-y2;
disp(' Среднеквадратическая ошибка в
итоге ')
disp(' работы нейронной сети')
err2 =mse(EN2)
```

После запуска программы получаем:

```
Входной сигнал: 1 p1 =      -1.0000   -1.0000    2.0000    2.0000
                             1.0000    4.5000         0    4.0000
Желательный отклик: test =    -1    -1     1     1.
Полученный отклик сети на входной  сигнал 1:
y1 =   -1.1001   -1.0341    1.0043    1.0034.
Среднеквадратическая ошибка в итоге  работы нейронной сети:
err1 =    0.0028.
Входной сигнал: 2 p2 =      -1.0000   -1.0000    2.0000    1.8000.
                             1.0000    5.0000    0.5000    5.0000
Желательный отклик: test =    -1    -1     1     1.
Полученный отклик сети на входной  сигнал:
y2 =   -1.1001   -1.0001    1.0049    0.9834.
Среднеквадратическая ошибка в итоге  работы нейронной сети:
err2 =    0.0026.
```

Полученные результаты и на этот раз свидетельствуют о том, что вариации компонентов вектора **P**, характеризующего сложившуюся ситуацию, оказывают слабое влияние на значения координат вектора **Y**, описывающего решение, которое нужно предпринять.

Подмеченная нами при рассмотрении конкретных примеров закономерность имеет общий смысл и является весьма важной в современной технике и электронике. Мы увидели, что электронная система управления, несмотря на изменение своих параметров, осталась работоспособной. Это означает, что возмущения с малой энергией в нейронной сети незначительно влияют на результат, сеть «ошибкоустойчива» или, по-другому, «груба», «обладает робастной устойчивостью» к изменению своих параметров. Наличие свойства робастности (от англ. robust – грубый, сильный), то есть способности сохранения системой некоторых свойств при достаточно малых вариациях ее параметров, в настоящее время весьма необходимо, так как позволяет решать задачи управления в условиях, когда невозможно или затруднено получение полной предварительной информации о режиме функционирования системы. Робастные системы управления весьма полезны в тех случаях, когда требуется, чтобы качество ее работы не опускалось ниже допустимого уровня при действии ряда непредсказуемых факторов. Известно, к примеру, что в силу огромного числа причин параметры реальных технических устройств всегда отличаются от тех, которые использовались при проектировании (например, из-за неточности параметров компонентов при их изготовлении (из-за допусков), из-за явления «старения» элементов, из-за влияния температуры окружающей среды и т. п.). Использование для описания свойств системы более грубых, но и более достоверных числовых характеристик приводит к тому, что всевозможные изменения параметров в робастной системе не могут повлечь за собой снижение требований к качеству управления.

Итак, концепция обработки информации в нейронных сетях, проистекающая из принципа параллелизма (за счет взаимодействия большого количества нейронов), придает нейронным сетям «робастность». Обучение сети, даже на множестве «зашумленных» входных сигналов, если вычисления распределены между множеством нейронов, является почти совершенным и сеть выполняет свои функции на удовлетворительном уровне.

3. НЕЙРОННАЯ СЕТЬ ПРИ РЕШЕНИИ ЗАДАЧ АППРОКСИМАЦИИ И ГЕНЕРИРОВАНИИ СИГНАЛОВ СЛОЖНОЙ ФОРМЫ

3.1. Общие сведения

Созданное на основе нейронной сети «интеллектуальное» устройство управления параметрами съемки цифрового фотоаппарата, изученное нами в качестве примера, это частный случай решения задачи по автоматизации процесса проведения фотографирования. В настоящее время в качестве подобных, но универсальных средств, предназначенных для комплексной автоматизации производственных процессов, используют, образующие определенную управляемую целостность, в общем случае, наделенную искусственным интеллектом и имеющую возможность самообучаться, электромеханические и мехатронные системы в виде упорядоченного сочетания взаимосвязанных и взаимодействующих механических, электротехнических, электронных и микропроцессорных компонентов. Такие системы называют роботами третьего поколения. Данные роботы строятся как самообучающиеся, адаптирующиеся (то есть способные изменять, приспосабливать свое поведение в зависимости от условий внешней среды), самонастраивающиеся (то есть способные обеспечить наиболее подходящий для данной ситуации отклик, «настроиться» на данную ситуацию) системы с гибкими процедурами принятия решений об управлении. Возможности этих роботов таковы, что они способны вырабатывать «планы» решения поставленных перед ними задач и контролировать выполнение последних.

Искусственный интеллект базируется на использовании информации, «воспринимаемой» устройствами автоматики. Эту информацию часто «поставляют» датчики (регистрирующие приборы), как преобразователи контролируемых параметров в электрический сигнал. Известно, что опрос датчиков информации может проводиться последовательно, в определенные промежутки времени. Кроме того, при «транспортировке» непрерывного сигнала от точки его измерения к цифровому управляющему устройству происходит процесс «дискретизации» сигнала, в результате чего он «табулируется», то есть превращается в некоторую таблицу чисел. Также известно, что, вследствие сложной кинематики

механизмов и возросших требований к динамике, управление в реальном масштабе времени требует «точечных» траекторий управления. Все это свидетельствует о том, что с программно-аппаратной реализацией искусственного интеллекта тесно связан еще ряд задач обработки векторной и матричной дискретной информации. Во-первых, во многих задачах практики возникает необходимость приближенной замены одних величин другими. Например, при работе с измерительными приборами, предназначенными для косвенных измерений, часто бывает необходимо по результатам измерения одной величины «а» определить значение другой величины «у», недоступной непосредственному наблюдению. Косвенный характер эксперимента обусловливает то, что величины «у» связаны с искомыми параметрами «а» функциональной зависимостью, вытекающей из теории изучаемого процесса. Поэтому, среди вопросов обработки данных, полученных экспериментальным путем, изучения поведения технических объектов, выявления существующих в них закономерностей, широко распространены задачи интерполяции и аппроксимации. Во-вторых, при обработке дискретной информации часто возникает задача генерирования (имитации) временного ряда (сигнала сложной формы) с известными свойствами (параметрами).

Мы в дисциплине «Основы электротехники и электроники» уже широко использовали процедуры, обеспечивающие формирование числовых последовательностей (в виде вектора или матрицы), представляющих импульсы различной формы (прямоугольной, треугольной, трапециидальной). Естественно предположить, что нейронная сеть, преобразующая входные сигналы в выходные, может успешно решать названные классы задач по обработке дискретной информации. При этом нейронная сеть может работать в условиях, когда присутствует элемент случайности и не известен точный вид связей между входами и выходами, когда эта связь искажена «шумом» и нельзя моделировать непосредственно вид функциональной зависимости. Другая существенная особенность нейронных сетей при решении задач подобного класса состоит в том, что функциональная зависимость между выходом и входом может выявляться или формироваться с наперед заданной точностью в ходе обучения сети.

Задача приближения, в упрощенном виде, состоит в нахождении оценки неизвестной функции $y = F(x)$. В математике известны два подхода к приближению табличных данных. В одном из них необходимо,

чтобы приближающая кривая проходила через все точки, заданные таблицей. Это удается сделать при использовании методов интерполяции. В другом подходе данные аппроксимируют простой функцией, применимой во всем диапазоне табличных данных, но не обязательно проходящих через все точки. Такой подход называется подгонкой кривой, которую стремятся провести так, чтобы ее отклонения от табличных данных были минимальны. Часто при этом пользуются методом наименьших квадратов, который сводит к минимуму сумму квадратов разностей между значениями функции, определяемыми выбранной кривой и таблицей.

Нейронные сети вычисляют суперпозиции функций одной переменной и их линейные комбинации, поэтому можно предположить, что они годятся для решения задач аппроксимации. Такое утверждение имеет теоретическое обоснование. В работах А. Н. Колмогорова и В. И. Арнольда показано, что любую непрерывную функцию n переменных можно получить с помощью операций сложения, умножения, и суперпозиции непрерывных функций одной переменной. Это в теоретическом плане открывает возможность осуществления аппроксимации непрерывных функций многих переменных нейронными сетями с использованием функции одной переменной. Однако при этом остаются открытыми вопросы о том, какой тип функции активации надо выбрать, сколько скрытых слоев надо включить в сеть и какое количество нейронов должно быть включено в каждый слой.

Представление выходного сигнала в виде вложенной нелинейной функции не является традиционным для классической теории аппроксимации. Однако, если рассматривать многослойную сеть, обучаемую согласно алгоритму обратного распространения ошибки, как некий, общего вида, «механизм» осуществления нелинейного отображения «вход-выход», то в таком случае сеть – универсальный аппроксиматор (universal approximator). Аппроксимация будет заключаться в том, чтобы по обучающей выборке (возможно искаженной «шумом»), состоящей N обучающих примеров, – парам данных вход-выход, – упорядоченных как $\{(p(1),d(1)),\ ...,\ (p(N),d(N))\}$, для всех $p_1, p_2,..., p_{m0}$, принадлежащих входному пространству, найти функцию $F(p_1, p_2,..., p_{m0})$, которая является наилучшей реализацией аппроксимации функции $f(p_1, p_2,..., p_{m0})$, то есть $\left|F(p_1, p_2,..., p_{m0}) - f(p_1, p_2,..., p_{m0})\right| < \varepsilon$. Заметим при этом, что

если m_0 – количество входных узлов многослойной сети, а M – количество нейронов выходного слоя, то отношение «вход- выход» для такой сети определяет отображение m_0-мерного Евклидова пространства входных данных в M -мерное Евклидово пространство выходных сигналов. Для такого отображения была доказана теорема об универсальной аппроксимации, суть которой состоит в том, что вообще возможна такая аппроксимация для любой непрерывной функции и что для построения равномерной аппроксимации с точностью ε для любого обучающего множества, представленного набором входов $p_1, p_2, ..., p_{m0}$ желаемых откликов $f(p_1, p_2, ..., p_{m0})$ достаточно многослойной сети с одним скрытым слоем. Из теоремы, тем не менее, не следует, что один скрытый слой является оптимальным в смысле времени обучения, простоты реализации и, что более важно, качества обобщения. Из теоремы об универсальной аппроксимации можно получить оценку размера обучающего множества $N \approx m_0 m_1 / \varepsilon_0$, где m_1 - число нейронов первого скрытого слоя, ε_0 - среднеквадратическое значение ошибки оценивания.

Для хорошего качества аппроксимации размер обучающего множества должен превышать отношение общего количества свободных параметров сети (произведения числа входных узлов на количество нейронов первого слоя) к среднеквадратическому значению ошибки оценивания. Эта теорема является важной с теоретической точки зрения. В большинстве практических случаев условия, для которых она выполняется, нарушаются. По этой причине на практике чаще применяют сети, у которых имеются два скрытых слоя. При этом процесс аппроксимации становится более управляемым и по своей «философии» аналогичен подходу к аппроксимации кривых, использующему сплайны (splines).

Говоря об аппроксимации функций надо иметь в виду, что многослойные нейронные сети с гладкой функцией активности нейронов в приложениях, требующих аппроксимации функций и их производных, «учитывают» производные реализуемого гладкого отображения «вход-выход». Известно также, что многослойная сеть может аппроксимировать функции, не дифференцируемые в классическом смысле. При этом в случае кусочно-дифференцируемых функций сеть обеспечивает «учет» обобщенных производных.

3.2. Функционирование нейронной сети при решении задач аппроксимации и генерации сигналов сложной формы

После этих вводных замечаний изучим, как работает нейронная сеть при решении классов задач, связанных с аппроксимацией и генерированием сигналов сложной формы.

В электротехнике большое практическое значение имеет моделирование нелинейных свойств ферромагнитных материалов. Известно, что магнитная индукция в ферромагнитных веществах зависит от истории процесса намагничивания и что в катушках индуктивности с ферромагнитным сердечником имеет место явление магнитного гистерезиса. В ряде практических ситуаций в цепях переменного тока необходимо знать вебер-амперную характеристику катушки с ферромагнитным сердечником $\psi(i)$, то есть зависимость потокосцепления от тока, протекающего по виткам катушки. Эта зависимость является вполне определенной для данного материала и определяется большим числом факторов, которые при теоретическом рассмотрении сложно учесть. С помощью датчиков тока и напряженности магнитного поля можно относительно просто измерить отдельные точки вебер-амперной характеристики. Если по этим, полученным с ошибками и зависящим от многих факторов окружающей среды, экспериментальным данным провести аппроксимацию кривой $\psi(i)$ и создать устройство, воспроизводящее данную зависимость в реальной ситуации, то можно существенно улучшить качественные показатели электротехнического оборудования. При проведении аппроксимации весьма полезными оказываются искусственные нейронные сети.

Пусть экспериментальным путем получен набор из κ пар данных, характеризующих вебер-амперную характеристику катушки с ферромагнитным сердечником $\psi(i)$ в одном квадранте системы координат («положительную ветвь»). При токах i =[0 0.05 0.1 0.15 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.1 1.3 1.5] значения потокосцепления равны $\psi(i)$ =[0 0.01 0.03 0.051 0.066 0.088 0.102 0.113 0.120 0.125 0.129 0.132 0.137 0.14 0.141]. Чтобы иметь для цепи переменного тока полную информацию о характеристике катушки необходимо массив «положительных» экспериментальных данных дополнить отрицательной ветвью характеристики (выполнить «зеркальное» отражение массивов). Это можно легко выполнить с помощью команды «fliplr». Аппроксимируем полученную вебер-

амперную характеристику катушки с ферромагнитным сердечником $\psi(i)$ используя многослойную искусственную нейронную сеть, использующую алгоритм обратного распространения ошибок. Поскольку экспериментальные данные меняются в большом диапазоне значений, выполним масштабирование обучающих пар к диапазону [-1 1], то есть осуществим нормировку данных так, чтобы значения тока и потокосцепления попали в интервал [-1 1]. После осуществления моделирования зависимости $\psi(i)$ восстановим исходные значения. Введем в командное окно:

```
clear
% Определяем входной и тестовый сигнал
Iexper=[0 0.05 0.1 0.15 0.2 0.3 0.4 0.5 ...
0.6 0.7 0.8 0.9 1.1 1.3 1.5];
Irassch=[-fliplr(Iexper(2:end)) Iexper];
Psiexper= [0 0.01 0.03 0.051 0.066 0.088
... 0.102 0.113 0.120 0.125 0.129 0.132
0.137 0.14 0.141];
Psirassch=[-fliplr(Psiexper(2:end)) Psiex-
per];
% Выполняем нормировку, чтобы значения тока и потокосцепления
% попали в интервал [-1 1]
[pn,minp,maxp,tn,mint,maxt] =
premnmx(Irassch,Psirassch);
Pnn = pn;
test =tn;
% Создаем многослойную сеть обратного распространения ошибок
netapr = newff([-1
1],[45,1],{'tansig','tansig'},'traingd');
% Определяем объектно-ориентированные параметры сети,
% определяющие процесс обучения.
% Параметр скорости обучения
netapr.trainParam.lr = 0.01;
% Используемое число эпох
netapr.trainParam.epochs = 3000;
% Количество итераций (эпох),по истечении которых будут представляться
% промежуточные результаты
netapr.trainParam.show = 300;
% Значение целевой ошибки обучения
netapr.trainParam.goal = 1e-3;
% Проводим обучение сети
netapr1 = train(netapr,Pnn,test); grid on
Ynn = sim(netapr1,Pnn);
% Выполняем восстановление прежнего
% диапазона значений (изменения) потокосцепления
Y=postmnmx(Ynn,mint,maxt);
test=Psirassch;
P=Irassch;
% Строим графики зависимости потокосцепления от тока
figure(2), clf,
h1=plot(P,Y,'LineWidth',2);
xlabel('Tok, I');ylabel('Psi');
hold on,
set(h1,'Color',[1/4,1/4,0])
plot(P,test,'+b','MarkerSize',6,'LineWidth',2),
grid on
legend('output', 'input')
```

После запуска программы получаем график, показывающий, как осуществляется аппроксимация характеристики катушки (рис. 3.1). График свидетельствует о том, что нейронной сети удается относительно

хорошо аппроксимировать зависимость $\psi(i)$, несмотря на то, что полученная кривая не всегда имеет монотонный характер.

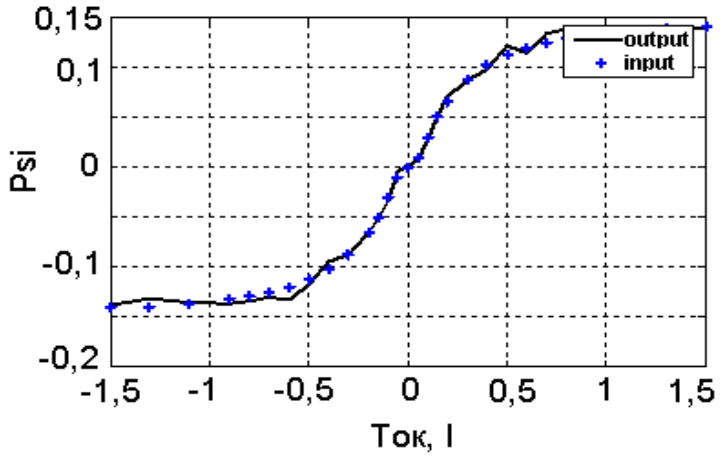


Рис. 3.1. **График аппроксимации вебер-амперной характеристики катушки с ферромагнитным сердечником $\psi(i)$ многослойной нейронной сетью по полученным экспериментальным данным**

На практике часто встречается задача аппроксимации гармонической функции, на которую накладывается «шум». При этом, если известно достаточно полное математическое описание физического явления, пытаются частично скомпенсировать искажающее влияние шума, преобразовав исходный сигнал должным образом. С «приборной» точки зрения такое преобразование называют редукцией (от лат reducere – приводить обратно, возвращать, ослаблять в каком либо отношении). Редукция (одного прибора к другому, более качественному) уменьшает искажения при проведении измерений, приближая их к значениям, свойственным изучаемому объекту. Выполнить аппроксимацию с редукцией возможно при использовании нейронной сети. Введем в командное окно:

```
clear
% Определяем входной  и тестовый
сигнал
P = [-1:.05:1];
test = sin(2*pi*P) + 0.12*randn(size(P));
% Создаем многослойную сеть обрат-
ного распространения ошибок
netapr= newff([-1.2
1.2],[35,1],{'tansig','tansig'},'traingd')
% Определяем объектно-
%  представляться (выводиться) проме-
жуточные результаты
netapr.trainParam.show = 300;
%  Значение целеваой ошибки обучения
netapr.trainParam.goal = 1e-3;
%  Проводим обучение сети
netapr1 = train(netapr,P,test);  grid on
% Строим графики зависимости сигнала
от времени
Y = sim(netapr1,P);
```

```
ориентированные параметры сети,
%  определяющие процесс обучения.
%  Параметр скорости обучения
netapr.trainParam.lr = 0.02;
%  Используемое число эпох
netapr.trainParam.epochs = 3000;
% Количество итераций (эпох), по ис-
течении которых будут
figure(2), clf,
h1=plot(P,Y,'LineWidth',2);
xlabel('Time');ylabel('Signal');
hold on,
set(h1,'Color',[1/2,1/2,0])
plot(P,test,'+b','MarkerSize',6,'LineWidth',2),
grid on
legend('output', 'input')
```

После запуска программы получаем график сигнала, полученный при аппроксимации нейронной сетью сигнала и набор данных, которые имелись в результате измерения (рис. 3.2).

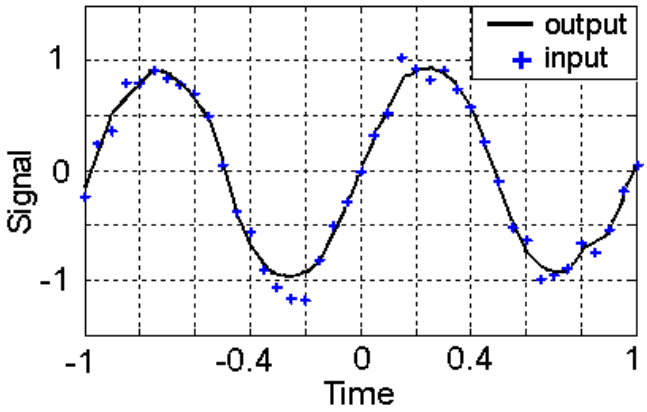


Рис. 3.2. **График аппроксимации сигнала многослойной нейронной сетью по экспериментальным данным**

График свидетельствует о том, что нейронная сеть аппроксимировала сигнал с «шумом» (результаты эксперимента, помеченные крестиком) таким образом, что он стал с точки зрения специалиста удобен для анализа и интерпретации. Исследователь, основываясь на своем научном опыте и располагая информацией о возможном физическом процессе, может уверенно предположить, что в данном случае он имеет дело с синусоидальным сигналом. Из графика сигнала можно высказать предположение о «коридоре ошибок», отражающем степень неуверенности в измеренных данных. Искусственная нейронная сеть подобным образом может быть использована для табличной аппроксимации (нахождения приближенного значения) функции многих переменных.

Искусственная нейронная сеть может быть применена для генерирования (имитации) сигнала сложной формы с известными свойствами (параметрами), вид которого может быть задан точками в табличной

форме. Покажем, как с помощью нейронной семи обеспечивается формирование импульсов различной формы. Введем в командное окно (для получения графика сигнала треугольной формы, рис. 3.3а):

```
clear
% Определяем таблицу данных для
сигнала
%  Время при котором определено
значение сигнала
Texp=[0 1 2 3 4 5 6 7 ...
8 9 10];
Trassch=Texp;
%  Мгновенные значения сигнала
при указанном времени
Sigexper= [0 1 2 3 4 5 ...
4 3 2 1 0];
Sigrassch=Sigexper;
%   Выполняем нормировку, чтобы
значения времени и мгновенного
%  значения попали в интервал [-1 1]
[pn,minp,maxp,tn,mint,maxt] =
premnmx(Trassch,Sigrassch);
Pnn = pn;
test =tn;
% Создаем многослойную сеть об-
ратного распространения ошибок
netapr = newff([-1
1],[45,1],{'tansig','tansig'},'traingd');
% Определяем объектно-
ориентированные параметры сети,
%  определяющие процесс обучения.
%  Параметр скорости обучения
netapr.trainParam.lr = 0.01;
% Используемое число эпох
netapr.trainParam.epochs = 3000;
% Количество итераций (эпох),по истечении
которых будут
% представляться (выводиться) промежу-
точные результаты
netapr.trainParam.show = 300;
%   Значение целевой ошибки обучения
netapr.trainParam.goal = 5e-4;
%  Проводим обучение сети
netapr1 = train(netapr,Pnn,test);  grid on
Ynn = sim(netapr1,Pnn);
%  Выполняем восстановление прежнего
%   диапазона значений (изменения) сигнала
Y=postmnmx(Ynn,mint,maxt);
test=Sigrassch;
P=Trassch;
% Строим графики зависимости сигнала от
времени
figure(2), clf,
h1=plot(P,Y,'LineWidth',2);
xlabel('Time');ylabel('Signal');
hold on,
set(h1,'Color',[1/4,1/4,0])
plot(P,test,'+b','MarkerSize',6,'LineWidth',2),
grid on
legend('output', 'input')
```

Введем в командное окно (для получения графика сигнала трапециидальной формы, рис. 3.3 б):

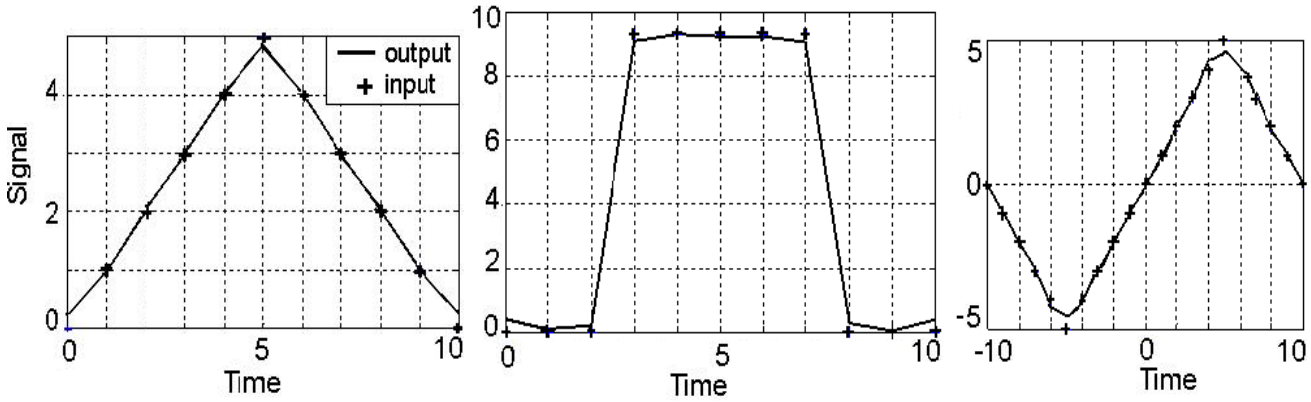
```
clear
% Определяем таблицу данных для
сигнала
% Время, при котором определено зна-
чение сигнала
Texp=[0 1 2 3 4 5 6 7 ...
% Параметр скорости обучения
netapr.trainParam.lr = 0.01;
%  Используемое число эпох
netapr.trainParam.epochs = 3000;
% Количество итераций (эпох), по исте-
чении которых будут
```

```
8 9 10];
Trassch=Texp;
%  Мгновенные значения сигнала  при
%  указанном времени
Um=9.3;
Sigexper= Um.*[0 0 0 1 1 1 ...
1 1 0 0 0];
Sigrassch=Sigexper;
%   Выполняем нормировку, чтобы зна-
чения времени и мгновенного
% значения попали в интервал [-1 1]
[pn,minp,maxp,tn,mint,maxt] =
premnmx(Trassch,Sigrassch);
Pnn = pn;
test =tn;
% Создаем многослойную сеть обрат-
ного распространения ошибок
netapr = newff([-1
1],[45,1],{'tansig','tansig'},'traingd');
% Определяем объектно-
ориентированные параметры сети,
% определяющие процесс обучения.
%  представляться (выводиться) проме-
жуточные результаты
netapr.trainParam.show = 300;
% Значение целевой ошибки обучения
netapr.trainParam.goal = 5e-4;
%  Проводим обучение сети
netapr1 = train(netapr,Pnn,test);  grid on
Ynn = sim(netapr1,Pnn);
%  Выполняем восстановление прежнего
% диапазона значений сигнала
Y=postmnmx(Ynn,mint,maxt);
test=Sigrassch;
P=Trassch;
% Строим графики зависимости сигнала
от времени
figure(2), clf,
h1=plot(P,Y,'LineWidth',2);
xlabel('Time');ylabel('Signal');
hold on,
set(h1,'Color',[1/4,1/4,0])
plot(P,test,'+b','MarkerSize',6,'LineWidth',2),
grid on
legend('output', 'input')
```

Введем в командное окно (для получения графика сигнала пилообразной формы, рис. 3.3 в):

```
clear
% Определяем таблицу данных для
сигнала
% Время при котором определено зна-
чение сигнала
Texp=[0 1 2 3 4 5 6 7 ... 8 9 10];
Trassch=[-fliplr(Texp(2:end)) Texp];
% Мгновенные значения сигнала  при
% указанном времени
Sigexper= [0 1 2 3 4 5 ...
4 3 2 1 0];
Sigrassch=[-fliplr(Sigexper(2:end)) Sigex-
per];
%  Выполняем нормировку, чтобы зна-
netapr.trainParam.lr = 0.01;
%  Используемое число эпох
netapr.trainParam.epochs = 3000;
% Количество итераций (эпох), по исте-
чении которых будут
%  представляться (выводиться) проме-
жуточные результаты
netapr.trainParam.show = 300;
%   Значение целевой ошибки обучения
netapr.trainParam.goal = 5e-4;
%  Проводим обучение сети
netapr1 = train(netapr,Pnn,test);  grid on
Ynn = sim(netapr1,Pnn);
%  Выполняем восстановление прежнего
```

```
чения времени и мгновенного
% значения попали в интервал [-1 1]
[pn,minp,maxp,tn,mint,maxt] =
premnmx(Trassch,Sigrassch);
Pnn = pn;
test =tn;
% Создаем многослойную сеть обрат-
ного распространения ошибок
netapr=newff([-1
1],[45,1],{'tansig','tansig'},'traingd');
% Определяем объектно-
ориентированные параметры сети,
% определяющие процесс обучения.
% Параметр скорости обучения
%   диапазона значений сигнала
Y=postmnmx(Ynn,mint,maxt);
test=Sigrassch;
P=Trassch;
% Строим графики зависимости сигнала
от времени
figure(2), clf,
h1=plot(P,Y,'LineWidth',2);
xlabel('Time');ylabel('Signal');
hold on,
set(h1,'Color',[1/4,1/4,0])
plot(P,test,'+b','MarkerSize',6,'LineWidth',2),
grid on
legend('output', 'input')
```



а) б) в)

Рис. 3.3. **Графики сигналов треугольной (а), трапециидальной (б) и пилообразной (в) формы, сформированные многослойной нейронной сетью по данным, заданным набором точек в виде таблицы**

Результаты аппроксимации и формирования (генерирования) сигналов различной формы подтверждают, что нейронные сети, обучаемой на основе коррекции ошибок, при минимальной их реализации двумя скрытыми слоями и логистическими функциями активации для нейронов в скрытом слое, обладают аппроксимирующими свойствами. Это позволяет сделать вывод о возможности их использования для моделирования сложных взаимосвязей во многих технических приложениях. Дополнительное преимущество нейронных сетей состоит в способности осуществлять редукцию, подсказывать некие общие принципы (обобщать) при предъявлении некоторого набора экспериментальных данных. При

этом способность нейронных сетей выявлять взаимосвязи в экспериментальных данных, к сожалению, связана с невозможностью проследить, каким образом этот результат достигнут. То есть нейронная сеть представляет собой, по сути «черный ящик», хотя при сравнении входных данных и откликов нейронной сети некоторые тенденции могут быть прослежены и получено объяснение того или иного поведения сети.

3.3 Эвристические средства при создании нейронных сетей

Внимательного читателя уже видимо насторожило то, что в предшествующем изложении мы не упоминали о том, как следует выбирать функции активации нейронов, сколько должно быть скрытых слоев в сети, каковы должны быть начальные веса и смещения в сети. Это было сделано не случайно. Дело в том, что следующей очень важной особенностью рассмотренных выше многослойных сетей, которая требует подробного разъяснения, является то, что работа с сетями, использующими алгоритм обратного распространения ошибок, является, скорее всего, искусством, чем наукой. Такое утверждение основывается на том, что, из-за отсутствия во многих случаях добротной теории («математического базиса»), нейронные сети, как объект изучения, неисчерпаем и он не может быть проанализирован только рутинными алгоритмическими процедурами. Обучение нейронной сети – это задача с неполно заданной информацией, в которой в общем случае не всегда однозначен ответ на вопрос о том, что именно представляют собой «подразумеваемые» в формулировке задачи или в способе ее решения данные. Для решения такого рода «нерутинных» задач, помимо алгоритмов ее решения, необходимы так называемые эвристические средства (эвристические сведения, эвристические предписания и рекомендации) или, кратко, эвристики. В контексте того, что мы рассматриваем, термин «эвристический» имеет примерно такой смысл: «способствующий получению новых правдоподобных суждений, но не обеспечивающий доказательства их истинности». По этой причине многочисленные параметры процесса обучения сети часто определяются на основе каких-то существующих «эвристических» соображений и личного опыта того, кто работает с сетью. Поэтому мы далее подробно остановимся на эвристиках, касающихся того, как получить «хорошую сеть» для решения поставленной задачи.

Первое из эвристических соображений является почти очевидным: в первую очередь при создании нейронной сети, использующей алгоритм обратного распространения ошибок, целесообразно использовать логистические функции активации. Вычисление локального градиента для каждого нейрона многослойной сети требует знания производной функции активности F(v), связанной с этим нейроном. Для существования такой производной функция активации должна быть непрерывной, т.е. удовлетворять требованиям дифференцируемости. Одним из примеров непрерывно дифференцируемой нелинейной функции является униполярная сигмоидальная (логистическая) функция активации. Как отмечалось, производная такой функции достигает своего максимального значения $Y_K' = 0.25$ при $v = 0$, а минимального, равного нулю, – на краях индуцированного локального поля нейрона. Так как величина коррекции синаптических весов в сети пропорциональна производной, то для максимального изменения синаптических весов нейрона его сигналы должны находиться в середине диапазона. Это свойство алгоритма обратного распространения, как показали исследования, вносит наибольший вклад в устойчивость алгоритма обучения.

В ряде случаев многослойная нейронная сеть, использующая алгоритм обратного распространения, может обучаться быстрее, если в качестве непрерывно дифференцируемой нелинейной функции используется функция активации, которую мы называли биполярной сигмоидальной функцией активации нейрона , или также гиперболической тангенциальной. Если вы помните, функция гиперболического тангенса также является логистической функцией, только масштабированной и смещенной. Как отмечалось, ее производная при $v = 0$ достигает своего максимального значения равного $Y_K' = 1$, а минимального, равного нулю, – на краях индуцированного локального поля этого нейрона. Так как величина коррекции синаптических весов в сети пропорциональна производной, то скорость обучения повышается (максимальная производная выросла в четыре раза). Важно и то, что такая функция активации является антисимметричной $F(-v) = -F(v)$ (то есть четной функцией своего аргумента). Униполярная сигмоидальная функция активации не удовлетворяет последнему условию. Область значений антисимметричной четной функции лежит в диапазоне [-1,1], в отличие от униполярной сигмоидальной функции, у которой все выходные значения положитель-

ны. Для нейронов следующего слоя использование униполярной функции может привести к экстремальной ситуации, когда синаптические веса могут либо одновременно увеличиваться, либо одновременно уменьшаться. Это равносильно зигзагообразному движению по поверхности ошибки и существенно замедляет процесс обучения. Таким образом, использование в качестве непрерывно дифференцируемой нелинейной функции биполярной сигмоидальной функцией активации нейрона с изложенной точки зрения несколько предпочтительнее.

Чтобы понять целесообразность использования логистических функции активации, следует отметить и то обстоятельство, что это «гладкие» функции. Если процесс обучения сети рассматривать как задачу аппроксимации кривой (curve-fitting), при которой сама сеть выступает как один нелинейный оператор, осуществляющий аппроксимацию входных данных, то непрерывность общей выходной функции обеспечивается за счет непрерывности отдельных функций активации многослойной сети. Чтобы не терять «гладкость» отображения входа на выход целесообразно, в соответствии с критерием, названным «бритвой Оккама»» (Occam's razor), при отсутствии каких-либо дополнительных априорных сведений, брать «простейшие» гладкие функции. Логистические функции активации именно такими функциями и являются. Упомянем еще о том, что, в принципе, при создании нейронной сети, использующей алгоритм обратного распространения ошибок, могут быть применены линейные функции активации. Однако следует при этом помнить, что область значения такой функции простирается от «минус» до «плюс» бесконечности. По этой причине алгоритм обратного распространения может изменять параметры сети так, что они устремятся в бесконечность, доводя скрытые нейроны других слоев до «насыщения». В ситуациях, когда диапазон выходных значений должен или может быть очень большим или требуется «вместить» все данные процесса, информация о котором неизвестна, использование в выходном слое нейронов с линейной функцией активации будет правильным.

Вторая группа эвристик касается того, чем следует руководствоваться при выборе параметра η, названного скоростью обучения (learning rates). Это некая константа, которая при рассматриваемом обучении в процессе минимизации среднеквадратической ошибки не менялась.

Все нейроны многослойной сети в идеале должны обучаться с одинаковой скоростью. Однако последние слои обычно имеют более

высокие значения локальных градиентов, чем начальные слои сети. Исходя из этого, параметру скорости обучения η следует назначать меньшие значения для последних слоев сети и большие для первых. Чтобы время обучения для всех нейронов сети было примерно одинаковым, нейроны с большим числом входов должны иметь меньшее значение параметра обучения, чем нейроны с малым количеством входов. В литературе существует рекомендация, в которой предлагается назначать параметр скорости обучения для каждого нейрона обратно пропорционально квадратному корню из суммы его синаптических связей. От параметра η при обучении с учителем на основе коррекции ошибок зависит характер перемещения (траектория изменения векторов весовых коэффициентов по поверхности ошибок) и, соответственно, «успешность» поиска минимума данной функции («сходимость» процесса обучения). Обратимся для наглядности к линейной нейронной сети и выполним процедуру обучения для случая нейрона с одним входом и одним выходом. Для такой сети поверхность ошибок представляет собой фигуру с одним минимумом, напоминающую чашу. Введем в командное окно:

```
clear, P = [1 -1.2];% Вектор входов
T= [0.5, 1]; % Вектор целей
% Максимальное значение параметра
обучения
% (задаем коэффициент 0,4 либо 1,6)
maxlr = 1.60*maxlinlr(P,'bias');
% Создание линейной сети
net = newlin([-2,2],1,[0],maxlr);
% Расчет функции среднеквадратической
ошибки
w_range=-1:0.2:1; b_range=-1:0.2:1;
ES = errsurf(P,T, w_range, b_range, 'pure-
lin');
% Построение поверхности ошибок
figure(1),clf,
surfc(w_range, b_range, ES)
% Расчет траектории обучении
x = zeros(1,50); y = zeros(1,50);
net.IW{1}=1; net.b{1}= -1;
x(1) = net.IW{1}; y(1) = net.b{1};
net.trainParam.goal = 0.001;
net.trainParam.epochs = 1;
% Цикл вычисления весов и смещения
для одной эпохи
for i = 2:50,
  [net, tr] = train(net,P,T);
  x(i) = net.IW{1};
  y(i) = net.b{1};
end
% Построение линий уровня и траекто-
рии обучении
figure(2),clf,
contour(w_range,b_range,ES,20), hold on
plot(x, y,'-*'),
grid on, hold off,
```

После запуска программы получаем изображение поверхности ошибок (рис. 3.4) и траекторию изменения вектора весовых коэффици-

ентов в двумерном пространстве для различных параметров скорости обучения (рис. 3.5).

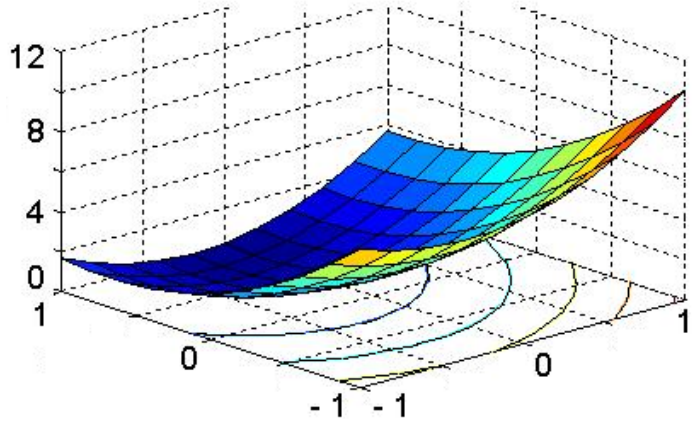
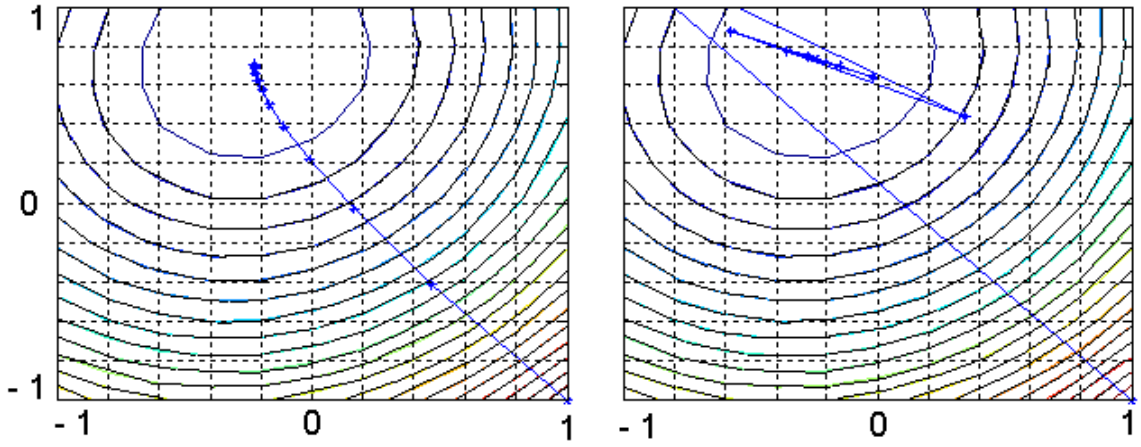


Рис. 3.4. **График поверхности ошибок линейной нейронной сети с одним минимумом, построенный в пространстве, координатами которого являются веса нейрона W. Внизу размещены линии уровня функции (линии постоянства значений поверхности ошибок)**



а) б)

Рис. 3.5. **Траектории изменения вектора весовых коэффициентов при «спуске» в двумерном пространстве для параметра скорости обучения η =0,15 (траектория представляет собой гладкую кривую)(а) и η =0,65 (траектория представляет собой зигзагообразную кривую) (б). На графике знаком "x" отмечены значения весов на различных этапах движения по поверхности ошибок**

Графики свидетельствуют о следующем. Корректировка весов выполняется в направлении минимума поверхности ошибок (к оптимальному значению w^*), то есть в направлении противоположном вектору градиента. От величины положительной константы η зависит то, как

осуществляется переход к точке w^*. При малых значениях параметра скорости обучения движение «медленное» (алгоритм сходится к оптимальному значению достаточно медленно), но зато траектория изменения $w(n)$ соответствует гладкой кривой. Если параметр η велик, то алгоритм ускоряется, но при этом перемещение точки по поверхности ошибок имеет вид «случайного блуждания», а траектория движения принимает зигзагообразный характер (движение снизу-вверх-частично вниз и вправо-резко влево-постепенно вправо и т.п.). Если, увеличивая η, превысить некоторое критическое значение (в этом легко можно убедиться, взяв параметр скорости обучения примерно равным 1,9), то алгоритм становится неустойчивым (то есть расходящимся).

Из представленных простых, но наглядных графиков можно сделать еще ряд важных для практики заключений.

Поверхность ошибок вдоль направления некоторого весового коэффициента может быть достаточно «плоской», поэтому частная производная в этом направлении может быть небольшой по величине. В этом случае, коррекции, применяемые к синаптическим весам, окажутся весьма малыми, и, следовательно, для достижения алгоритмом минимальной ошибки может потребоваться достаточно большое количество итераций. И, наоборот, если поверхность ошибок вдоль направления некоторого весового коэффициента достаточно искривлена, то в этих направлениях частные производные велики по амплитуде и, значит, синаптические веса на каждом шаге будут подвергаться значительной коррекции. Это может привести к прохождению (к «проскоку») мимо точки минимума на поверхности ошибок.

Вообще говоря, направление антиградиента может не соответствовать кратчайшему направлению к точке минимума на поверхности ошибок. В этой ситуации коррекция весовых коэффициентов может привести к движению алгоритма в неправильном направлении, к большой продолжительности вычислений и, как говорят, к малой скорости сходимости метода обратного распространения.

Поскольку параметр скорости обучения, являющийся некой эмпирической константой, создает сложности в обучении сети, то вместо этого простейшего способа задания η в методах стохастической аппроксимации предлагается изменять его со временем. Одной из таких форм изменения этого параметра является следующая: $\eta(n) = c/n$, где c –

константа. При этом надо помнить, что если константа c велика, существует опасность выхода алгоритма из-под контроля на первых шагах его работы. Есть и другие способы изменения $\eta(n)$.

Третья группа эвристик касается того, чем следует руководствоваться при выборе начальных значений синаптических весов и пороговых значений или, по-другому, как, предваряя процедуру обучения, наилучшим образом инициализировать нейронную сеть.

Очевидно, что если синаптические веса принимают большие начальные значения, то нейроны, скорее всего, достигнут «режима насыщения». Если такое случится, то локальные градиенты алгоритма обратного распространения будут принимать малые значения, что в свою очередь, вызовет «торможение» процесса обучения. Если же синаптическим весам присвоить малые начальные значения, алгоритм будет очень «вяло» работать в окрестности координат поверхности ошибок. В частности, это верно для случая антисимметричной функции активации, такой как гиперболический тангенс. Начало координат в этой ситуации является, как говорят, «седловой точкой», то есть точкой, когда «образующие» поверхности ошибок вдоль одной оси имеют положительный градиент, а вдоль другой – отрицательный. По названным причинам нет смысла использовать как слишком большие, так и слишком маленькие начальные значения синаптических весов. Как всегда, «золотая» середина находится между этими двумя крайностями. Поэтому хорошей считается такая «стратегия» инициализации, когда исходные значения весов и порогов выбираются таким образом, чтобы они были «равномерно» распределены в «переходной» области между линейной частью сигмоидальной функции активации и областью насыщения. Для многих нейронных сетей веса и смещения сети задаются на этапе инициализации с помощью специальных функций, когда случайным образом «генерируются» равномерно распределенные значения с нулевым средним и дисперсией, равной величине обратной корню квадратному из количества синаптических связей нейронов.

Четвертая группа эвристик относится к тому, как должны быть предварительно обработаны входные переменные. Рассматривая нормализатор, мы уже отмечали, что для сигналов, поступающих на вход сети надо выполнить масштабирование (шкалирование) данных и осуществить их статистическую обработку. Целесообразно, чтобы входные

данные находились в диапазоне либо между 0 и 1, либо между -1 и 1. Кроме того, отмечалась необходимость удаления содержащейся в векторе сигнала $\alpha_1, \alpha_2, ..., \alpha_N$ постоянной составляющей.

Для оценки целесообразности предварительной обработки входных данных таким образом, чтобы их среднее значение по всему обучающему множеству было близко к нулю, рассмотрим экстремальный случай, когда все входные переменные положительны. В такой ситуации синаптические веса нейронов первого скрытого слоя могут либо одновременно увеличиваться, либо одновременно уменьшаться. Следовательно, вектор весов этих нейронов вынужден будет менять направление, что приведет к зигзагообразному движению по поверхности ошибок. Такая ситуация обычно замедляет процесс обучения и, таким образом, неприемлема. Для минимизации времени обучения следует избегать использования входных значений с ненулевым средним. Для входного вектора **P**, передаваемого нейронам первого скрытого слоя многослойной нейронной сети это условие выполнить просто. Надо вычесть из каждого компонента этого вектора его ненулевое среднее значение до передачи этого вектора в сеть. Однако что делать с сигналами, передаваемыми на следующие скрытые и выходной слои? Ответ на этот вопрос заключается в выборе функций активации, которые используются в сети. Если функция является несимметричной (как в случае униполярной логистической функции), выходной сигнал любого нейрона принадлежит к интервалу [0, 1]. Такой выбор является источником систематического смещения для нейронов, расположенных правее первого скрытого слоя сети. Чтобы обойти эту проблему, следует использовать антисимметричную функцию активации, такую как гиперболический тангенс. В последнем случае выход каждого нейрона может принимать как положительные, так и отрицательные значения из интервала [-1, 1] и, с большой долей вероятности, будут иметь нулевое среднее значение. Обучение методом обратного распространения с антисимметричными функциями активации приведет, вероятнее всего, к более быстрой сходимости, чем в случае применения симметричной функции активации. Чтобы ускорить процесс обучения методом обратного распространения необходимо провести статистическую обработку входных сигналов в следующих двух аспектах. Во-первых, входные переменные, содержащиеся в обучающем множестве, должны быть некоррелированы. Этого

можно добиться с помощью их анализа методом главных компонент. Во-вторых, некоррелированные входные переменные должны быть представлены так, чтобы их ковариация была приближенно равной. К сожалению, вопросы статистической обработки данных выходят за рамки рассматриваемой темы, поэтому мы на них останавливаться не будем.

Пятая группа эвристик касается способности сети к обобщению или, другими словами, проблемы «недообучения-переобучения». При обучении методом обратного распространения в сеть подают обучающую выборку, «загружая» максимально возможное количество примеров, и вычисляют синаптические веса и пороги. При этом мы надеемся, что обученная таким образом сеть будет способна к «обобщению», то есть способной обобщать результат на новые наблюдения. Термин «обобщение» заимствован из психологии. Считается, что сеть обладает хорошей обобщающей способностью (generalize well), если отображение входа на выход, осуществляемое ею, является корректным для данных, никогда не «виденных» сетью в процессе обучения. Нейронная сеть, спроектированная для хорошего «обобщения», должна порождать правильный вывод не только для большинства вводимых образцов из набора тестовых данных, но и для случаев, когда входной сигнал слегка отличается от примеров, использованных для обучения сети. Например, если мы, решая задачу аппроксимации, обучили сеть строить «гладкое нелинейное отображение», то образцы, которые подобны, но не использовались ранее в процессе обучения, тоже должны порождать на выходе функции примерно с той же степенью гладкости, что и ранее. Если при пропускании через сеть обучающего множества ошибка обучения уменьшалась и была небольшой, то и на образцах, не участвующих в обучении, на новых данных, ошибка сети должна оставаться малой и не расти с увеличением эпох. Качество обобщения сети зависит от количества учебных данных и архитектуры сети. Набор учебных данных и число скрытых слоев должно быть характерным для решаемой проблемы, а не намного большим, чем требуется.

К сожалению, нейронная сеть обратного распространения подвержена такому явлению, как переобучение (overtraining). В этом случае обученная сеть хорошо «отслеживает» выходные значения, обучающего множества, но оказывается не способной обобщить закономерность на новые данные. Такое явление также называют избыточным обучением, так как оно характерно для ситуаций, когда сеть обучается на слишком

значительном количестве примеров или используется число скрытых элементов намного больше, чем требуется для обучения. При переобучении сеть, используя «частные подробности и детали», которые не свойственны самой моделируемой функции, но присутствуют в примерах обучения (например, данные, которые возникают благодаря случайным помехам, шуму и т. п.), тщательно «запоминает» эти «детали» в обучаемых данных и, затем, выполняя «отслеживание» частных точек уже других новых данных, поступающих на ее вход, выдает в качестве результата величины, подобные тем, на которых она обучалась. В таких случаях сеть, по мере уточнения решений, обычно вместо гладких кривых или моделей, обладающих определенным запасом гладкости, дает осцилляции или кусочно-ломанные функции. При этом теряется способность к обобщению на аналогичных входных сигналах. Следует предостеречь читателя от поспешного вывода о том, что, для того, чтобы избежать стадии излишнего переобучения, следует «пораньше» остановить сеанс обучения. Сама идея хорошая, но при этом сеть может оказаться «недоученной». Мы уже видели, что при нормальных условиях среднеквадратическая ошибка уменьшается по мере увеличения эпох обучения. При этом график обучения начинается с довольно больших значений, стремительно уменьшается, а затем продолжает убывать все медленнее и медленнее по мере продвижения сети к локальному минимуму на поверхности ошибок. Поэтому, если «рано» остановить обучение, когда среднеквадратическая ошибка еще относительно большая, то получим просто «недообученную» сеть. В то же время, если, продолжая обучения, вовремя не остановить сеанс обучения, то существенно повышается вероятность излишнего переобучения, при котором кривая обучения, после монотонного убывания до некоторого минимума, начинает возрастать после продолжения обучения. После прохождения точки минимума сеть начнет обучаться, чаще всего, «посторонним шумам» содержащимся в обучающейся выборке.

Рассмотрим в качестве примера нейронную сеть типа 1-30-1 с 1 входом, 30 скрытыми нейронами и 1 выходом, которую необходимо обучить аппроксимировать функцию синуса, если обучение сети осуществляется по ее приближенным значениям, которые получены как сумма значений синуса и неких случайных величин, распределенных по нормальному закону. Введем в командное окно:

```
clear                                   % промежуточные результаты
```

```
% Определяем входной  и тестовый сигнал
P = [-0.5:.05:0.5];
test = sin(2*pi*P) + 0.22*randn(size(P));
test1 = sin(2*pi*P);
% Создаем многослойную сеть обратного распростр-я ошибок
netapr = newff([-1.2 1.2],[80,1],{'tansig','tansig'},'traingd');
%  Определяем объектно-ориентированные параметры сети,
% определяющие процесс обучения.
%  Параметр скорости обучения
netapr.trainParam.lr = 0.01;
%  Используемое число эпох
netapr.trainParam.epochs = 3000;
% Количество итераций (эпох),по истечении которых будут представляться
netapr.trainParam.show = 300;
%   Значение целевой ошибки обучения
netapr.trainParam.goal = 1e-3;
%  Проводим обучение сети
netapr1 = train(netapr,P,test);  grid on
%   Строим графики зависимости сигнала от времени
Y = sim(netapr1,P);
figure(2), clf,
h1=plot(P,Y,'LineWidth',2);
xlabel('Time');ylabel('Signal');
hold on,
set(h1,'Color',[1/2,1/2,0])
plot(P,test,'+b','MarkerSize',8,'LineWidth',2),
grid on
legend('outnet', '---sin(t)')
hold on,
set(h1,'Color',[1/2,1/4,0])
plot(P,test1,'LineWidth',2);
```

После запуска программы получаем графики аппроксимируемой функции, приближенных значений, которые использовались для обучения сети и результата аппроксимации (рис. 3.6).

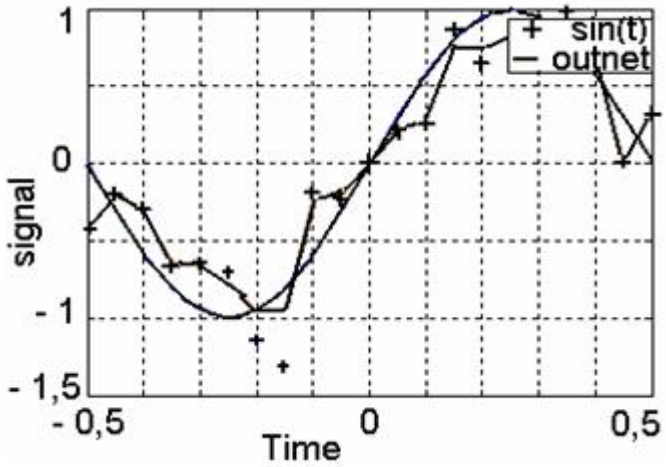


Рис. 3.6. **Графики приближенных значений, которые использовались для обучения сети (+), аппроксимируемой синусоидальной функции (---) и результата, выданного сетью при попытке аппроксимации**

Видно, что в данном случае мы сталкиваемся с явлением переобучения. Исходные данные были «зашумлены», поэтому с точки зрения способности сети «к обобщению» входных данных, нельзя считать что «наилучшей» является кривая, которая в точности проходит через все имеющиеся точки. По этой причине «выданный» сетью график результата аппроксимации, принимающий замысловатую кусочно-ломанную форму и не имеющий никакого отношения к синусоидальной зависимости, является результатом переобучения сети или, другими словами, «излишней подгонки».

Отсутствие обобщения при работе сети особенно неприятно при моделировании сложных функций многих переменных. Поэтому при практической работе с нейронными сетями приходится экспериментировать с большим числом сетей различной архитектуры (изменять количество слоев, число нейронов), обучая каждую по несколько раз и сравнивая полученные результаты. При выборе наилучшей сети из множества «архитектур-кандидатов», позволяющей при обучении на примерах «прошлого» обобщать результаты на «будущее», в настоящее время используют два подхода: перекрестную проверку и «регуляризацию».

В рамках подхода, получившего название перекрестной проверки (cross-validation) имеющиеся в наличии данные сначала случайным образом разбиваются на обучающее множество (training set) и тестовое множество (test set). Обучающее множество, в свою очередь, разбивается на два несвязанных подмножества: а) подмножество для оценивания (estimation subset), используемое для выбора модели; б) проверочное подмножество (validation subset), используемое для тестирования модели. Идея этого подхода состоит в проверке качества модели нейронной сети на данных, отличных от использования для начального оценивания. Для проверки эффективности разных моделей-кандидатов и выбора из них лучшей используется обучающее множество. Сеть обучается на подмножестве для оценивания и «контролируется» на проверочном подмножестве. Чтобы было понятнее, рассмотрим пример. Сформируем обучающее подмножество на интервале входных значений от -1 до 1 с шагом 0.05 в виде суммы функции синуса и погрешности, описываемой случайной величиной, распределенной по нормальному закону с дисперсией 0.01. Затем сформируем проверочное подмножество. Для него определим входы в диапазоне от –0.975 до 0.975 и, чтобы

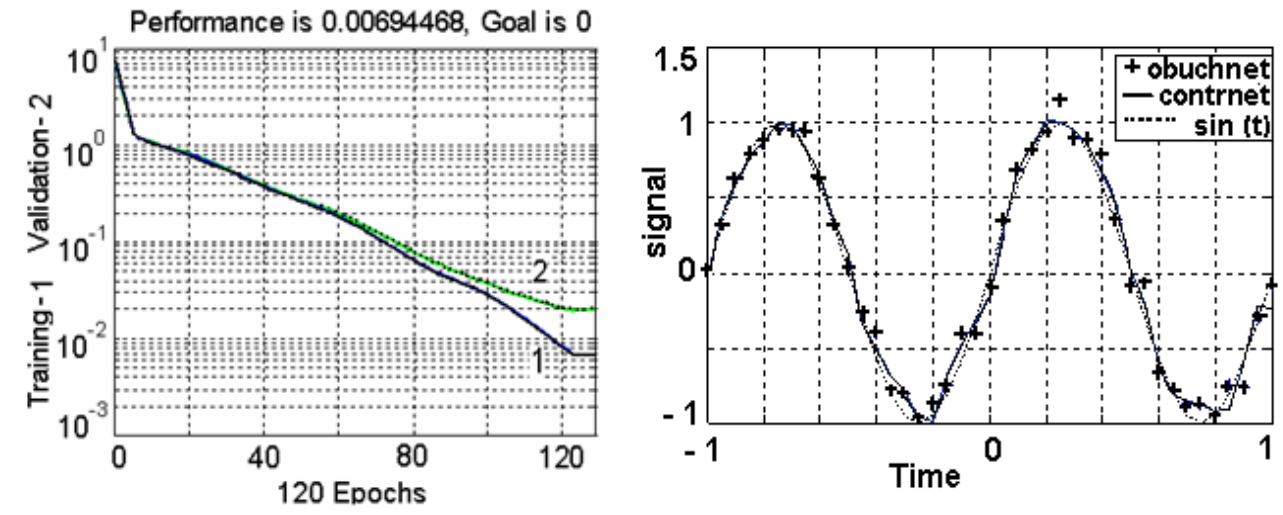
сделать задачу более реалистичной, добавим помеху, распределенную по нормальному закону. Вводим в командное окно:

```
clear
% Определяем сигналы
% обучающего подмножества
p = [-1:0.05:1];
t = sin(2*pi*p)+ 0.1*randn(size(p));
% Определяем сигналы проверочного
%  (контрольного) подмножества
v.P = [-0.975:.05:0.975];
v.T = sin(2*pi*v.P)+0.1*randn(size(v.P));
% Создаем многослойную сеть обратного
распространения ошибок
net = newff([-1 1],[20,1], {'tan-
sig','purelin'},'traingdx');
%  Проводим обучение сети
% Определяем объектно-
ориентированные параметры сети,
% определяющие процесс обучения.
%  Используемое число эпох
net.trainParam.epochs = 300;
% Количество итераций (эпох), по ис-
течении которых будут
% представляться (выводиться) проме-
жуточные результаты
net.trainParam.show = 25;
net = init(net);
[net,tr] = train(net,p,t,[],[],v);
grid on
an = sim(net,p);
t1 = sin(2*pi*p);
% Строим графики зависимостей сиг-
налов от времени
figure(2);
hplot=plot(p,t,'+',p,an,'-',p,t1,':');
grid on
legend('obuchnet', 'contrnet','sin(t)')
xlabel('Time');
ylabel('Signal');
set(hplot,'LineWidth',2);
```

После запуска программы увидим, что в начале работы сети ошибки на оценивающем и проверочном подмножествах будет одинаковыми (рис. 3.7а). Если это не так и они существенно отличаются, то, вероятнее всего, разбиение обучающего множества на два подмножества было выполнено неудачно и подмножества неоднородны.

По мере того, как сеть обучается, обе ошибки, естественно, убывают. Если ошибка для данных проверочного подмножества перестала убывать и даже стала расти, это указывает на то, что сеть выбрана неудачно и возникает явление переобучения. В этом случае обычно советуют уменьшить число скрытых элементов сети, поскольку использованная сеть оказалась более «мощной», чем этого требует задача. В случае, если обе ошибки, - оценивающего и проверочного подмножеств, - не достигают достаточного уровня малости, то это свидетельствует о том, что сеть является «слабой» и имеет место явление недообучения сети. Проведя многократные эксперименты можно выбрать «удачную» сеть, дающую хороший результат на оценивающем и проверочном подмножествах. В таком случае получается относительно гладкая аппрок-

симация (рис.3.7б). И все же, после работы с обучающим множеством существует вероятность того, что отобранная по параметрам эффективности модель окажется излишне переученной. Чтобы предотвратить такую опасность, эффективность обобщения выбранной сети измеряется также на тестовом множестве, которое отличается от проверочного. Перекрестные проверки особенно привлекательны, если требуется создать большую нейросеть, обладающую хорошей способностью к обобщению.



а) б)

Рис. 3.7. **Графики изменения среднеквадратичной ошибки нейронной сети в процессе ее обучения на подмножестве для оценивания и на проверочном подмножестве (а) и графики приближенных значений, которые использовались для обучения сети (+), аппроксимируемой синусоидальной функции (---) и результата, выданного сетью при попытке аппроксимации (б)**

В рамках второго подхода, получившего название «регуляризация», предназначенного для выбора наилучшей сети из множества «архитектур-кандидатов», используется математическая теория приближенного решения некорректных задач, – нового направления в вычислительной математике, – предназначенная, в общем случае, для решения плохо обусловленных задач. Именно в этой теории было введено в научный обиход понятие регуляризирующего алгоритма. В самом первом и весьма грубом приближении о регуляризации нейронных сетей можно сказать следующее. Если мы хотим добиться от сети хорошего обобщения, то мы должны обеспечить некий баланс между достоверностью

данных обучения и архитектурой или, точнее, сложностью нейронной сети. Этого компромисса при обучении с учителем можно достичь, если минимизировать не среднеквадратическую ошибку, как мы это делали ранее, а «общий риск» $R(w) = E_S(W) + \lambda E_C(w)$. Первое слагаемое, – $E_S(W)$, – это стандартная мера эффективности. При обучении методом обратного распространения она определяется как среднеквадратическая ошибка, которая вычисляется по всем выходным нейронам сети на всем обучающем множестве примеров для каждой эпохи. Естественно, что первое слагаемое зависит от самой сети, но, главным образом, от входных данных. Второе слагаемое, $-E_C(w)$, – это «штраф» за сложность (complexity penalty). Оно зависит исключительно от самой сети. Оценка «штрафа» основывается на предварительных знаниях о рассматриваемой модели сети. Формула «общего риска» является одним из утверждений общей теории регуляризации, предложенной академиком А. Н. Тихоновым в 1963 году и широко используемой на практике для реализации на ЭВМ численных алгоритмов решения плохо обусловленных задач. Плохая обусловленность задачи, по существу, означает, что даже большой набор данных может нести в себе удивительно малый объем информации, «нужный» для решения задачи. Плохая обобщающая способность сети – это своего рода «плохая обусловленность», которая может возникать из-за того, что в обучающих примерах может быть недостаточно информации для «реконструкции» отображения «вход-выход» (недостаток информации нельзя восполнить никакой математической хитростью). Вносит неопределенность в данные обучения и наличие шумов. Если задача обучения не удовлетворяет критерию непрерывности, если не известна предварительная информация о функциональной зависимости, то вычисленное отображение с входа на выход может иметь мало общего с достоверным решением задачи.

Главная идея регуляризации состоит в том, чтобы с помощью некоторой неотрицательной функции, которая несет в себе предварительную информацию о решении, за счет нахождения так называемого минимума функционала Тихонова стабилизировать решение, сделать его гладким (гладкость это мера колебания функции) и удовлетворяющим свойству непрерывности. Чтобы найти решение задачи регуляризации и, соответственно, минимизировать «общий риск» надо выбрать оптимальную величину λ, которая называется параметром регуляризации

(regularization parameter). Параметр регуляризации в некотором смысле можно рассматривать как индикатор достаточности (достоверности) входного набора данных для определения минимума и нахождения гладкого решения задачи. Он также характеризует относительную значимость слагаемого штрафа за сложность сети по сравнению со слагаемым, характеризующим меру эффективности. Если параметр λ равен нулю, ограничений на гладкость нет, то процесс обучения методом обратного распространения полностью определяется предоставленными примерами обучения. С другой стороны, если параметр λ является бесконечно большим, то «сложность сети» при определении архитектуры сети является доминирующей, гладкость решения определяющей, а примеры обучения – недостоверными. Однако, использование мощных, сложных сетей приводит к излишнему «заглаживанию» исходной информации и к потере так называемой «тонкой» структуры, очень важной при исследовании процессов. В реальных ситуациях параметр регуляризации принимает некоторое среднее значение. Для регуляризации «сложности сети» в многослойной нейронной сети могут быть использованы следующие подходы. Во-первых: если $y = F(p, w)$ выполняемое отображение входа на выход, то «штраф» за сложность сети может регулироваться тем, что часть «ненужных» весов может «подавляться» (уменьшаться). Действительно, как показывает анализ, в многослойной сети есть веса, которые могут принимать совершенно произвольные значения, но которые, по сути, имеют «малое влияние на сеть». Это, по-другому, «избыточные» веса. Именно эти веса «заставляют» сеть для незначительного уменьшения ошибки выполнять обучение на больших объемах данных и тем самым снижают качество обобщения. Используя различные алгоритмы, анализирующие распределение в пространстве весов, можно выявить «избыточные» веса и либо их «снизить», либо вообще «подавить». Тем самым в нейронной сети остаются и «работают» веса, имеющие высокую ценность для процесса обучения, которые дают высокое качество обобщения. Во-вторых: для упрощения структуры сети («усечения» сети) можно использовать информацию о вторых производных поверхности ошибок. В этом случае удается обеспечить значительно лучший компромисс между сложностью сети и поверхностью ошибок и обеспечить высокое качество обобщения. Поскольку такая более строгая процедура регуляризации более требовательна к вычислительным ресурсам, «отслеживает» взаимодействие двух множеств весов нейро-

нов скрытого и выходного слоев, то обычно в данном случае используют сеть с одним скрытым слоем и одним нейроном в выходном слое. Алгоритмы, используемые сетью, определяют параметры, исключение которых вызовет минимальное увеличение функции стоимости, а затем проводят двухуровневую минимизацию весов, дающую оптимальную степень «повреждения» сети. Нечто подобное выполняется в «оптимальной хирургии мозга». Укажем, что с точки зрения теории аппроксимации нейронные сети с осуществлением регуляризации обладают следующими положительными свойствами: 1) сеть регуляризации является универсальным аппроксиматором в том смысле, что при большом числе скрытых элементов она способна аппроксимировать любую непрерывную функцию с компактным носителем. 2) Сети регуляризации обладают свойством наилучшей аппроксимации линейной функции. Это значит, что для неизвестной линейной функции f всегда существует такой набор коэффициентов, который аппроксимирует функцию f лучше любого другого набора. 3) Решение, обеспечиваемое сетью регуляризации, является оптимальным. Под оптимальностью понимается то, что такая сеть минимизирует функционал, измеряющий «удаленность» решения от своего истинного значения, представленного примерами обучения.

Чтобы ответить на вопрос об эффективности практического применения регуляризирующих алгоритмов, рассмотрим в качестве примера применение нейронной сети для решения задачи аппроксимации функции синуса с аддитивной помехой с использованием процедуры регуляризации сложности сети. Аддитивная помеха представляет собой некоторый случайный процесс, моделируемый с помощью генератора случайных чисел. По ряду соображений изложенных выше, используем сеть, имеющую три слоя, в которой реализованы алгоритмы, дающие оптимальную степень «повреждения» сети. Будем полагать также следующее. Входной слой пусть состоит из числа узлов, количество которых равно размерности r_0 вектора входного сигнала P (то есть количеству независимых переменных в задаче аппроксимации). Второй слой скрытый и состоит из 41 нейрона, которые непосредственно связаны со всеми узлами входного слоя. Выходной слой состоит из единственного линейного нейрона, связанного со всеми узлами скрытого слоя. Остановка процесса обучения производится в случае, когда предельное ко-

личество циклов обучения станет равным 800. Параметр регуляризации равен 0.999. Вводим в командное окно:

```
clear
net = newff([-1 1],[20,1], {'tan-
sig','purelin'},'trainbfg');
% Определяем объектно-ориенти-
рованные параметры сети,
% определяющие процесс обучения.
%  Используемое число эпох
net.trainParam.epochs = 800;
% Количество итераций (эпох), по ис-
течении которых будут
% представляться (выводиться) про-
межуточные результаты
net.trainParam.show = 200;
% Процесс обучения останавливается,
если ошибка обучения стала меньшей
net.trainParam.goal = 1e-4;
net.performParam.ratio = 0.999;
net.performFcn = 'msereg';
% Параметр регуляризации
% Определяем   время
p = [-1:0.05:1];
% Определяем сигнал с помехой
t = sin(2*pi*p)+0.15*randn(size(p));
%  Определяем сигнал без помехи
t1 = sin(2*pi*p);
[net,tr] = train(net,p,t);
an = sim(net,p);
figure(1);
hplot=plot(p,t,'+',p,an,'-',p,t1,':');
legend('signal', 'net output','sin(t)')
set(hplot,'LineWidth',2);
xlabel('Time');
ylabel('Signals');
grid on
```

После запуска программы получаем графики, показанные на рис. 3.8а. Видно, что в ходе осуществления аппроксимации функции синуса с аддитивной помехой с использованием процедуры регуляризации сложности сети удается получить приближенное решение, в котором из-за «сглаживания» удается обеспечить равномерное, практически без появления каких-либо осцилляций, «приближение к искомой функции. Алгоритм с регуляризацией обладает хорошей обобщающей способностью и является робастным, поскольку возмущения с относительно малой энергией незначительно влияют на ошибку оценивания.

Основная трудность при практической реализации нейронной сети для решения задачи аппроксимации сигналов с использованием процедуры регуляризации сложности сети состоит в том, что не всегда удается верно подобрать параметр регуляризации. Не вполне ясно также, когда следует останавливать итерации. Особенно это относится к ситуации, когда мы не располагаем предварительной информацией о самой функции. Для таких случаев были разработаны специальные алгоритмы, позволяющие «автоматически» устанавливать значение параметра регуляризации λ. При этом для получения наилучших обобщающих

свойств нейронной сети осуществляется так называемая Байесовская регуляризация при которой минимизируется комбинация квадратов ошибок и весов с выбором наилучшей. Вводим в командное окно:

```
clear
net = newff([-1 1],[30,1],{ 'tan-
sig','purelin'},'trainbr');
% Определяем объектно-ориентиро-
ванные параметры сети, определяющие
%  процесс обучения.
% Используемое число эпох
net.trainParam.epochs = 200;
% Количество итераций (эпох), по исте-
чении которых будут представляться
% промежуточные результаты
net.trainParam.show = 50;
randn('seed',192736547);
p = [-1:.05:1];
%  Определяем сигнал с помехой
t = sin(2*pi*p)+0.15*randn(size(p));
%  Определяем сигнал без помехи
t1 = sin(2*pi*p);
net = init(net);
net = train(net,p,t);
an = sim(net,p);
figure(1);
hplot=plot(p,t,'+',p,an,'-',p,t1,':');
legend('signal', 'net output','sin(t)')
set(hplot,'LineWidth',2);
xlabel('Time');
ylabel('Signals');
grid on
```

После запуска программы получаем рис. 3.8 б.

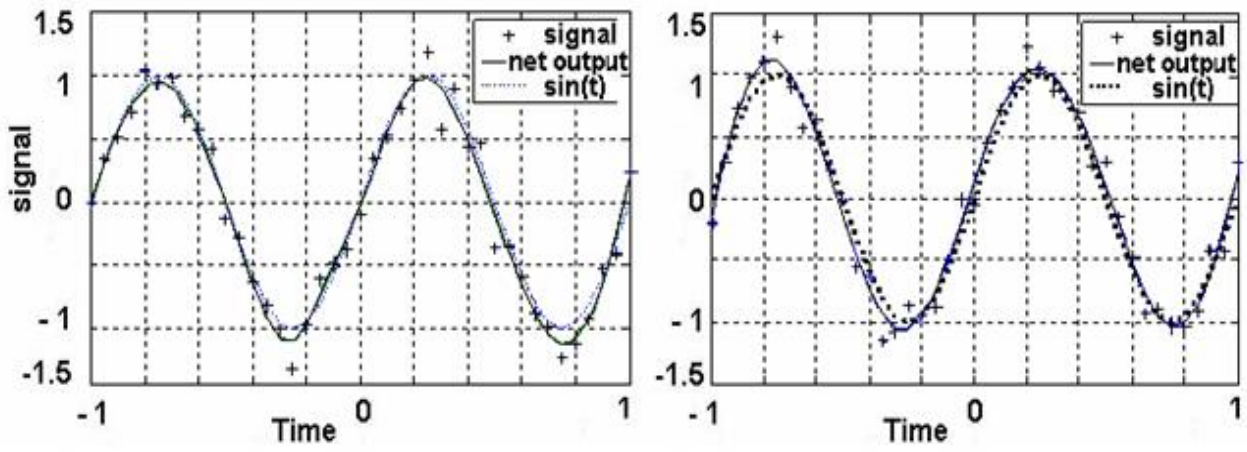


Рис. 3.8. **Графики приближенных значений, которые использовались для обучения сети (+), аппроксимируемой синусоидальной функции (---) и результата, выданного сетью при попытке аппроксимации за конечное число эпох (а) и при аппроксимации для случая Байесовской регуляризации (б)**

Как следует из приведенного графика в процессе регуляризации алгоритму «удается» выделить искомое решение, «полезный сигнал», «отделив» его от погрешностей, вносимых случайными «высокочастот-

ными» помехами. Алгоритм обладает сглаживающим свойством, хорошей обобщающей способностью и также является робастным, поскольку возмущения с относительно малой энергией незначительно влияют на ошибку оценивания.

Графики, показанные на рис. 3.8б свидетельствуют о том, что, настраивая параметр затухания весовых коэффициентов, можно добиться того, что ошибка обучения будет монотонно убывающей функцией от номера итерации. При этом ошибка обобщения убывает только в начале процедуры обучения, а затем, после достижения минимума увеличивается. Это объясняется тем, что в начале сеть обучается на характерные «гладкие» признаки, после чего идет «подстройка» под возмущения, имеющиеся в обучающем множестве.

Наконец, последняя группа эвристик относится к тому, какую следует взять процедуру поиска минимума среднеквадратической ошибки, которая бы позволяла получить минимальную меру несходства между желаемым откликом и откликом, сгенерированным обучаемой нейронной сетью. Мы отмечали, что при обучении с учителем на основе коррекции ошибок «успешность» поиска минимума среднеквадратической ошибки зависит от траектории изменения векторов весовых коэффициентов по поверхности ошибок. Для сети, имеющей два синаптических веса, была показана поверхность ошибок, напоминающую чашу (часть поверхности сферы) с единственным минимумом. Отталкиваясь от случайной начальной конфигурации весов и порогов (то есть от случайно взятой точки на поверхности ошибок) алгоритм обучения постепенно отыскивал глобальный минимум. В многослойной сети с большим числом нейронов каждому из весов и порогов соответствует одно измерение в n- многомерном пространстве. Поэтому, для всевозможных сочетаний весов и порогов ошибку нейронной сети можно представить точкой в n-мерном пространстве. При этом все точки образуют некоторую нелинейную «поверхность», которая с точки зрения математики, называется гиперповерхностью. Графически представить гиперповерхность невозможно, однако можно, используя образное мышление, для этой многомерной поверхности использовать понятия, имеющие некоторую аналогию с трехмерной поверхностью.

В нейронной сети с большим числом нейронов зависящая от синаптических весов в высшей степени нелинейная «гиперповерхность ошибок» может иметь гораздо более сложное строение, напоминающее

«горный рельеф», и обладать рядом неприятных свойств. Характерной особенностью гиперповерхности ошибок, в частности, является наличие не только глобального, но и локальных минимумов (local minima), неких изолированных «впадин» на «горной поверхности» (точек, самых низких в некоторой своей окрестности, но лежащие выше глобального минимума). А поскольку в процессе обучения методом обратного распространения направление спуска по гиперповерхности определяется направлением наибольшего «наклона» поверхности, то всегда существует риск попадания в точку локального минимума, в которой производная равна нулю, а любое изменение синаптических весов будет приводить к увеличению ошибки. В данной точке алгоритм обучения надолго «застрянет» в таком состоянии вплоть до исчерпания лимита итераций. При этом в другой области гиперсферы ошибок могут существовать другие множества синаптических весов и порогов, для которых среднеквадратическая ошибка будет иметь меньшее значение, чем в данной точке локального минимума. Понятно, что остановка процесса обучения в точке локального минимума является весьма нежелательной, особенно если эта точка находится выше точки глобального минимума. Заметим по этому поводу также то, что аналитически невозможно определить положение глобального минимума на поверхности ошибок. По этой причине обучение сети, по сути, заключается в исследовании этой поверхности или, как говорят, в численной оптимизации (numerical optimizing).

Другой характерной неприятностью градиентного метода «спуска» к минимуму гиперповерхности ошибок, является наличие у гиперповерхности «оврагов», т. е. ситуаций, когда поверхность сильно «вытянута» в неком направлении, имеет «склоны», а иногда и «повороты». Это означает, что небольшое изменение некоторых весов и порогов, которые характеризует «склон оврага», приводит к резкому изменению среднеквадратической ошибки, а по остальным переменным, задающим направление «дна оврага», ошибка меняется незначительно.

В методе обратного распространения градиентный «спуск» по оврагам приведет к тому, что результаты приближения к минимуму будут поочередно находиться то на разных «склонах оврага». Если «склоны оврага» достаточно круты, то такие скачки «со склона на склон» точек могут сильно замедлить сходимость алгоритма к минимуму поверхности ошибок.

3.4. Разновидности алгоритмов обучения искусственных нейронных сетей

Интуитивно ясно, что для нейронных сетей ученные уже разработали алгоритмы, позволяющие эффективно находить глобальный минимум на гиперповерхности ошибок, «проходить овраги», добиваясь при этом существенного сокращения объема вычислений. На таких разновидностях алгоритмов обучения искусственных нейронных сетей мы и остановимся. Заметим, что для отыскания минимума гиперповерхности ошибок могут использоваться алгоритмы, которые условно можно поделить на две категории: 1) алгоритмы, использующие для обучения нейронных сетей градиентные методы (методы первого порядка, так как они используют информацию о первых производных функции ошибок). 2) Алгоритмы, использующие для обучения нейронных сетей информацию о вторых производных функции ошибок (методы второго порядка). Существенная разница во всех рассматриваемых алгоритмах состоит в том, как осуществляется выбор направления продвижения к минимуму и как выбирается шаг в данном направлении. В MATLAB устанавливать требуемый алгоритм и параметры обучения нейронной сети позволяют функции обучения, которые с помощью определенных параметров задают параметры, как самого процесса обучения, так и его остановки. Формировать и обучать нейронную сеть можно с помощью следующих градиентных алгоритмов обучения:

1. «Классического» алгоритма обратного распространения, в котором используется вычисление градиента и фиксированная на весь период скорость обучения. Функция, реализующая этот алгоритм – traingd. В данном алгоритме вычисляется вектор антиградиента гиперповерхности ошибок. Этот вектор указывает направление спуска по поверхности из исходной точки, поэтому на каждой итерации ошибка уменьшается и мы «немного» продвигаемся к точке минимума. Последовательность таких шагов, замедляющаяся по мере приближения к «дну», в конце концов, приведет к минимуму того или иного типа. Уже отмечалось, что при большой длине шага сходимость будет более быстрой, но при этом имеется опасность перепрыгнуть через решение или, если гиперповерхность имеет особо вычурную форму, уйти в неправильном направлении. Особенно такая ситуация опасна в случае, когда алгоритм медленно продвигается по «узкому оврагу с крутыми склонами», прыгая с одной

его стороны на другую. Напротив, при маленьком шаге, вероятно, будет движение в правильном направлении, однако при этом потребуется много итераций, пока значение градиента не станет меньшим, чем наперед заданная величина, характеризующая точность процесса обучения.

Понятно, что для «классического» алгоритма обратного распространения, как, впрочем, и для других алгоритмов, важен критерий, по которому следует остановить вычисления. Он вводится для того, чтобы определить момент завершения процедуры обучения. Обычно критерием останова является некоторое условие, выполнение которого показывает, что дальнейшие вычисления не имеют практического смысла. При этом существенной проблемой при выборе критерия является то, что весьма затруднительно его теоретическое определение. При практическом решении данной проблемы используют несколько вариантов ограничений, позволяющих завершить процедуру обучения. Первый вариант – задание максимального числа итераций процедуры обучения. Данный критерий останова не имеет ни математического ни физического обоснования и определяет лишь максимально допустимые временные затраты на процедуру обучения. Тем не менее, данный критерий может быть основным в системах «реального времени». Другой вариант задания критерия останова – введение ограничений на величину градиента. Как известно, в точке минимума среднеквадратической ошибки градиент должен быть равен нулю. При использовании итерационного приближения достижения такого равенства невозможно. По этой причине в качестве критерия обычно выступает условие, при котором значение ошибки становится меньше некоторой наперед заданной величины.

В-третьих. Критерий останова может быть получен на основе оценки изменения весовых коэффициентов между двумя итерациями. В этом случае процедура обучения завершается, если максимальное изменение весовых коэффициентов станет меньше некоторой наперед заданной величины.

Наконец, в качестве критерия останова может быть использовано достижение некоторого, наперед заданного значения функции ошибок. Однако использование этого подхода с точки зрения практического воплощения его в нейронной сети из-за таких факторов как нелинейность и многомерность проблематично, так как предварительное (априорное) определение значения функции невозможно в силу отсутствия детальной информации о возмущениях и помехах. Считается, что «классиче-

ский» алгоритм обратного распространения, в котором используется вычисление градиента и постоянная скорость обучения, имея такое преимущество, как простоту его реализации (простоту специализированного аппаратного обеспечения и скромные требования к оперативной памяти) и вычислительную робастность, обладает низкой скоростью сходимости, что ведет к продолжительности вычислений. Кроме того, он не всегда обеспечивает сходимости к глобальному минимуму и может «застревать» в некоторых точках гиперповерхности ошибок. Это может стать основополагающим в задачах большого объема и при применении его в системах, работающих в режиме «реального времени».

2. Алгоритма обратного распространения, в котором используется вычисление градиента, но скорость обучения при этом меняется (адаптируется). Функция, реализующая этот алгоритм – traingda. Алгоритм основывается на адаптивном подборе коэффициента скорости обучения η с учетом фактической динамики величины функции среднеквадратической ошибки. Скорость обучения в алгоритме меняется в течение процесса обучения. В начале специальным образом подбирается η и алгоритм ведет себя как стандартный алгоритм минимизации среднеквадратической ошибки. Дальнейшая стратегия изменения η состоит в том, что при стремлении к увеличению η контролируется изменение суммарной погрешности по сравнению с предыдущей итерацией. В случае роста последней, ограничивают η. Данный алгоритм является грубой попыткой использования информации о вторых производных, о поверхности ошибок. По сравнению с предыдущим алгоритмом в метод обратного распространения были привнесены некоторые преимущества, поэтому «Traingda» несколько более эффективен, по сравнению с «Traingd».

3. Алгоритма обратного распространения, в котором используется вычисление градиента, но при этом вводится «инерционная коррекция весов и смещений нейронов сети». Функция, реализующая этот модифицированный алгоритм – traingdm. Одним из способов повышения скорости обучения без потери устойчивости является изменение дельта-правила за счет добавления к нему момента инерции $\Delta W_{ji}(n) = \alpha \Delta W_{ji}(n-1) + \eta \delta_j(n) y_i(n)$, где α – положительная величина называемая постоянной момента (moment constant). В литературе показано, что включение момента в алгоритм обратного распространения при относительно незначительной модификации метода корректировки весов

ведет к ускорению «спуска» по гиперповерхности ошибок в некотором постоянном направлении. С другой стороны, добавление момента приводит к некоторому «стабилизирующему эффекту» для «направлений» меняющих знак (для случая сильно искривленной поверхности ошибок, содержащей много «выпуклостей» и «впадин»). Кроме того, слагаемое момента может предотвратить нежелательную остановку алгоритма в точке какого-либо минимума на поверхности ошибок. Таким образом, включение момента в алгоритм обратного распространение оказывает положительное влияние на работу алгоритма обучения.

4. Модифицированного алгоритма обратного распространения, в котором используется вычисление градиента и реализуется комбинированный метод, позволяющий менять (адаптировать) скорость обучения и производить «инерционную коррекцию весов и смещений нейронов сети». Функция, реализующая этот алгоритм – traingdx.

При описании предыдущего алгоритма предполагалось, что параметр скорости обучения представлен константой η. В общем же случае он может меняться на каждой итерации η_{ji}, как в алгоритме traingda. Сочетание двух алгоритмов, применение различных параметров скорости обучения в разных областях сети и гиперповерхности ошибок позволяет в ряде случаев добиться интересных результатов. Экспериментальные исследования показывают следующее. В случае малых η наблюдается низкая сходимость, но при этом более точно определяется точка минимума на гиперповерхности ошибок. Это интуитивно понятно, так как при малых η поиск минимума выполняется в более широкой области поверхности ошибок. При $\eta \to 0$ выбор больших значений константы момента ($\alpha \to 1$) приводит к увеличению скорости сходимости. С другой стороны, при $\eta \to 1$ малые значения фактора момента ($\alpha \to 0$) повышают устойчивость обучения. Средние значения $\eta \to 0.5$ и $\alpha \to 0.9$ могут приводить к колебаниям среднеквадратической ошибки в процессе обучения, что нежелательно. «Оптимальные» значения скорости обучения составляют порядка 0,1 при $\alpha \approx 0.5$.

5. Алгоритма так называемого «упругого» обратного распространения ошибки (resilient back propagation algorithm, RPROP) Функция, реализующая этот алгоритм – trainrp. Данный алгоритм основывается на подборе коэффициента скорости обучения η и в идейном плане похож на

«Traingda». Однако в нем используется своя, специфическая стратегия изменения шага в процессе обучения многослойной нейронной сети.

Основанные на дифференциальном исчислении и описанные в учебниках по математике классические градиентные алгоритмы обучения и их разновидности, охарактеризованные выше, определяют все частные производные функции ошибки по всем весам сети (градиент) и производят «моментальную оценку» вектора антиградиента. При использовании результатов такой оценки перемещение точки по гиперповерхности ошибок имеет вид «случайного блуждания», а сама сеть обладает статистическими свойствами. В алгоритмах минимизации среднеквадратической ошибки вектор весов «перемещается» по случайной траектории, «выписывает» хаотичную траекторию, имеющую сходство с траекторией броуновского движения. По этой причине их называют стохастическими градиентными алгоритмами. Случайный выбор направления предельно прост, он сокращает до минимума возможность остановки алгоритма в точке какого-нибудь локального минимума, но зато он требует большого числа шагов, имеет низкую скорость сходимости и «плохо чувствует» изменения входных сигналов. К тому же, когда градиент принимает малые значения, алгоритм становится малоэффективным. Медленная сходимость таких методов становится серьезной преградой, если размерность входных данных очень велика.

«Блуждания» алгоритма вызваны тем, что в процессе его работы осуществляется поиск из заданной точки в направлении одной из осей (одного веса), до точки минимума в данном направлении. Затем производится поиск в направлении, параллельном другой оси (другого веса), и т. д. Разумно модифицировать этот метод таким образом, чтобы на каждом этапе поиск точки минимума производился по детерминированной (не случайной) траектории вдоль «наилучшего» направления. Можно ожидать, что предлагаемый метод сходился бы быстрее градиентного метода. Это «наилучшее направление» позволило бы за кратчайшее время проводить наискорейший спуск по гиперповерхности ошибок. Эта естественная и прозрачная идея выбора наилучшего направления приводит нас к целой группе методов, в которых за конечное число шагов «кратчайшим путем» ищется точка минимума гиперповерхности ошибок.

Для выбора нужного направления поиска минимума в ситуации, когда «классическое» определение минимума функции связано со значительными затратами, математики предложили экономичные, или как их

еще называют, оптимальные методы. Такие методы позволяют решить задачу минимизации с требуемой точностью на основе вычислений значения исследуемой функции как можно в меньшем числе точек, гарантируя при этом наилучшую точность при жестко заданном количестве вычислений значений анализируемой функции. Суть таких оптимальных методов состоит в том, что, используя несколько значений функции в определенных точках, анализируемую функцию заменяют в небольшой области значений другой функцией, аппроксимирующей, минимум которой найти значительно проще. В частности, вы уже наверно встречались с тем, что при подобранных параметрах парабола вблизи минимума достаточно точно «заменяла» унимодальную выпуклую функции одной переменной. Аналогично, желая повысить обычно низкую скорость сходимости и одновременно избежать вычислительных сложностей в нейронных сетях с большим количеством нейронов, поступают при создании различных модификаций алгоритмов, названных в литературе алгоритмами метода сопряженных градиентов (conjugate-gradient method). В случае использования квадратичной выпуклой функции (quadratic function) методы сопряженных градиентов дают точное значение минимума за не более чем, κ шагов. При этом эффективность метода сопряженных градиентов возрастает на завершающем этапе поиска минимума, когда последовательные приближения близки к точке минимума, ибо в окрестности минимума функция обычно гладкая и близка к квадратичной выпуклой функции. При использовании таких методов опираются на то, что минимизация квадратичной функции эквивалентна задаче о решении системы линейных уравнений, поэтому для определения направления поиска приходится решать системы линейных уравнений, полученные из каких-то соображений. В методе сопряженных градиентов для минимизации квадратичной функции «генерируют» набор векторов сопряженных направлений $s(n)$ и на основе некоторого «условия» определяют тот, который «попадает» в точку глобального минимума. Этот метод и получил свое название из-за того, что для нахождения минимума последовательно на каждой итерации метода генерируются сопряженные версии векторов градиента квадратичной функции.

При использовании метода сопряженных градиентов для минимизации функции среднеквадратической ошибки следует аппроксимировать гиперповерхность ошибок квадратичной функцией. Затем предпо-

лагается, что начальное направление $s(0)$ задается антиградиентом. Если производная по данному направлению равна нулю, то точка минимума располагается вдоль некоторой прямой. Сопряженное направление выбирается так, чтобы эта производная и дальше оставалась нулевой. Далее, для определения нужного направления поиска следует вычислить так называемую резидуальную ошибку $r(n)$ и множитель масштабирования $\beta(n)$. Важным является то, что для определения $\beta(n)$ в алгоритме сопряженных градиентов используется только информация о градиенте. Это позволяет избежать вычисления матрицы вторых производных (матрицы Гессе), что сопряжено с вычислительными трудностями. Разновидности алгоритмов метода сопряженных градиентов как раз и различаются способом вычисления $\beta(n)$. «Генерируя» комбинации векторов $s(n+1)=r(n+1)+\beta(n+1)s(n)$ и определяя резидуальные ошибки $r(n)$, находим нужное направление $s(n+1)$, при котором $\|r(n)\|<\varepsilon\|r(0)\|$.

Для формирования и обучения нейронной сети используют следующие разновидности алгоритмов метода сопряженного градиента: 1) Алгоритм обратного распространения ошибки, в котором для «спуска» по поверхности ошибок используется формула Флетчера-Ривза (Fletcher-Reeves). Функция, реализующая этот модифицированный алгоритм – traincgf. 2) Алгоритм обратного распространения ошибки, в котором для «спуска» по поверхности ошибок используется формула Полака-Рибьера (Polak-Ribiere). Функция, реализующая этот алгоритм – traincgp. 3) Алгоритм обратного распространения ошибки, в котором для «спуска» по поверхности ошибок используется формула Пауэлла – Биеле (Powell - Beale). Этот алгоритм реализуется функцией – traincgb. 4) Алгоритм обратного распространения ошибки, в котором для «спуска» по поверхности ошибок используется формула Молера. Функция, реализующая этот модифицированный алгоритм – trainscg.

До сих пор мы рассматривали методы первого порядка – так называемые методы минимизации функции среднеквадратической ошибки, использующие лишь первые производные минимизируемой функции. В этих методах для определения направления убывания функции используется лишь линейная часть разложения функции в ряд Тейлора. Если минимизируемая функция дважды непрерывно дифференцируема, а также первые и вторые производные вычисляются достаточно просто,

то возможно применение методов минимизации второго порядка, которые используют квадратичную часть разложения этой функции в ряд Тейлора. Поскольку квадратичная часть разложения аппроксимирует функцию точнее, чем линейная, то естественно ожидать, что методы второго порядка сходятся быстрее, чем методы первого порядка.

Основная идея метода Ньютона (Newton's method) заключается в минимизации квадратичной аппроксимации функции ошибок. Поиск минимума выполняется за ряд шагов в так называемом направлении Ньютона. Если использовать разложение функции ошибки в ряд Тейлора с точностью до членов второго порядка, то обнаружится, что для обеспечения работоспособности метода необходима матрица Гессе (Гессиан), которая представляет собой матрицу из элементов вида $H = \partial^2 E / \partial w_i \partial w_j$. Гессиан должен быть положительно определен для всех n (матрица должна быть несингулярной), что нельзя гарантировать для всех итераций алгоритма. Поскольку вычисление матрицы Гессе требует больших вычислительных затрат, то были разработаны алгоритмы, не требующие вычисления вторых производных на каждой итерации. Алгоритмы, использующие только некоторые оценки матрицы Гессе, получили название квазиньютоновских. Эти алгоритмы используют информацию второго порядка (о кривизне) гиперповерхности ошибок, не требуя точного знания матрицы Гессе. В таких модификациях метода Ньютона положительно определенная оценка матрицы $H^{-1}(n)$ вычисляется напрямую, без обращения самой матрицы. Следует иметь в виду, что при этом Гессиан может быть положительно определен, но плохо обусловлен. Используя такую оценку, квазиньютоновские алгоритмы обеспечивают движение вниз по склону гиперповерхности ошибок, однако вычислительная сложность их при этом измеряется порядком квадрата весов.

Для формирования и обучения нейронной сети можно использовать следующие разновидности алгоритмов квазиньютоновского метода:

1. Алгоритм обратного распространения ошибки, в котором для «спуска» по поверхности ошибок используется алгоритм Бройдена – Флетчера – Гольдфарба – Шано (Broyden – Fletcher – Goldfarb – Shano). Функция, реализующая этот алгоритм – trainbfg. Данный алгоритм в настоящее время считается лучшей формой квазиньютоновских методов. Он дает положительно определенную аппроксимацию Гессиана, сходится на малом числе итераций, хотя требует большого количества

вычислений на каждой итерации и большого объема памяти. Он эффективен в нейронных сетях с небольшим числом слоев.

2. Алгоритм обратного распространения ошибки, в котором для «спуска» по поверхности ошибок используется метод секущих плоскостей, разработан Батитити. Этот алгоритм реализуется функцией – trainoss. Данный алгоритм объединяет в себе идеи метода сопряженного градиента и метода Ньютона. При этом новое направление выбирается без вычисления обратной матрицы $H^{-1}(n)$. Поправка к направлению наискорейшего спуска отыскивается на основе матрицы малого ранга. Сама матрица не хранится, а ее действие на векторы строится с помощью скалярных произведений на нескольких специально подобранных векторах, поэтому алгоритм требует меньших объемов оперативной памяти и вычислительных ресурсов на эпоху по сравнению с trainbfg.

3. Алгоритм обратного распространения ошибки, в котором для «спуска» по поверхности ошибок используется алгоритм Левенберга – Маквардта (Levenberg – Marquardt). Функция, реализующая этот квазиньютоновский алгоритм – trainlm. Алгоритм использует аппроксимацию Гессиана, что существенно проще вычисления матрицы Гессе. Этот алгоритм является самым быстродействующим и пригоден для обучения для настройки нейронных сетей с несколькими сотнями весов и смещений. Дополнительным преимуществом алгоритма является хорошая обусловленность Гессиана и его вычислительная робастность. К сожалению, при его использовании имеется ряд ограничений. Он применим для сетей с одним выходным нейроном и плохо применим для больших сетей. При использовании алгоритма может возникнуть ситуация, когда аппроксимация вблизи точки минимума может стать неадекватной.

Сравнивая метод сопряженных градиентов и квазиньютоновский метод, заметим следующее. В принципе, при нахождении в непосредственной близости от минимума, квазиньютоновские алгоритмы могут иметь более высокую сходимость, чем это возможно при использовании метода сопряженных градиентов. Квазиньютоновские методы, в дополнение к операциям матрично-векторного умножения, требуют хранения элементов матрицы, связанной с вычислением вектора направления $s(n)$. В вычислительном смысле, если размерность вектора весов велика, метод сопряженных градиентов более предпочтителен. В литературе приведены оценки алгоритмов обучения нейронных сетей таблица 3.1).

Таблица 3.1.

Сравнительная оценка алгоритмов обучения нейронных сетей

Алгоритм обучения	Оценка алгоритма по пятибалльной шкале		
	Скорость сходимости	Вычислительная робастность	Требования к памяти
Классический	1	4	2
Сопряженных градиентов	3	3	3
Метод Левенберга-Маркарда	5	5	1

Завершая тему об эвристиках, относящихся к тому, какую следует взять процедуру поиска минимума среднеквадратической ошибки, которая бы позволяла получить минимальную меру несходства между желаемым откликом и откликом, сгенерированным обучаемой нейронной сетью заметим следующее. Если задача нахождения минимума гиперповерхности ошибок является многоэкстремальной (имеется хотя бы одна точка локального минимума, отличная от точки глобального минимума), то с помощью описанных методов удается, вообще говоря, лишь приближение к какой-либо точке локального минимума. Поэтому упомянутые методы часто называют локальными. Такие методы ведут к одному из локальных минимумов, лежащему в окрестности точки начала обучения. Только в ситуации, когда значение глобального минимума известно, удается оценить, находится ли найденный локальный минимум в достаточной близости от искомого решения. Поэтому на практике приходится несколько раз начинать обучение при новых, как правило, случайных, значениях весов. Как говорят, надо проводить «встряхивание» весов. По названным причинам возникает необходимость применения методов глобальной оптимизации. В настоящее время используют два подхода к глобальной оптимизации: метод имитации отжига; генетический алгоритм. Подробности этих методов можно найти в литературе.

4. РАСПОЗНАВАНИЕ ОБРАЗОВ «С УЧИТЕЛЕМ» С ПОМОЩЬЮ НЕЙРОННЫХ СЕТЕЙ

4.1. Общие сведения

Мы рассмотрели несколько характерных классов задач, для решения которых могут успешно применяться нейронные сети. Было установлено, что когда имеется подготовленный набор обучающей информации, состоящий из примеров входных данных и соответствующих им выходов, сеть позволяет установить вид связи между ее входами и выходами. Если «обучение с учителем» проведено достаточно «хорошо», то нейронная сеть приобретает способность моделировать неизвестную функцию, связывающую значения входных и выходных переменных, проводить «обобщение». Можно говорить о том, что устройство, выполненное на основе нейронных сетей, в этом случае поступает «разумно», обретает некоторые «разумные свойства» и имеет, хотя и достаточно слабый, но все же «интеллект». И все же для нас более желанным является программно-аппаратное устройство с «искусственным интеллектом», которое может «думать» как человек, решать те задачи, ответы на которые приходится находить людям в своей жизни. При этом мы отдаем себе отчет, что сами до конца не понимаем, что есть процесс мышления и что значит, что компьютер поступает «разумно».

Деятельность человека, связанная с реальными жизненными потребностями, с решением научных и технических задач, обозначается термином «распознавание образов». Мы обычно не замечаем, что человеческий мозг хорошо приспособлен для такой деятельности. Например, мы можем отличить женское лицо от мужского. Мы легко можем, часто в толпе, узнать знакомое лицо, даже если наш знакомый с момента нашей последней встречи изменился, например, отрастил бороду или надел очки. По телефону мы всегда узнаем знакомый голос даже при наличии помех в линии связи или при наличии постороннего шума. В аудитории, находясь к человеку спиной, мы отличим голос «товарища» от голоса человека, с которым у вас более формальные отношения. Наконец, человек достаточно просто распознает буквенно-цифровые символы, написанные от руки разным почерком, напечатанные принтером или в типографии, несмотря на то, что при этом существуют различные почерки

и используется огромный набор гарнитур шрифта. Из-за важности для деятельности человека первые нейронные сети создавались именно для решения задач распознавания образов. В литературе можно найти множество различных описаний, определяющих понятия «образ» и «распознавание образов». Известно, что распознавание образов всегда предполагает наличие некоторой системы обработки информации, имеющей вход и выход. На вход системы поступают данные от множества различных источников. При этом они, как правило, сложны и часто являются пространственными и (или) временными функциями. Выходная информация сводится к указанию одного из нескольких классов. Распознавание образов формально определяет процесс, при котором получаемый образ (рассматриваемые объекты или входные данные) должен быть отнесен к одному из предопределенных классов (категорий). Для группы лиц образом может быть женское или мужское лицо, лицо ребенка или взрослого. Образ нами устанавливается по чертам человеческого лица, свойственным для данного образа (т.е. по каким-то свойствам, признакам). Хорошо известен пример, когда врач во время приема проводит медицинскую диагностику (классификацию) и, выбирая на основании симптомов наиболее вероятное заболевание, направляет больного для дальнейшего лечения в соответствующее отделение (класс) больницы: терапии, травматологии, стоматологии и т.д. В более широком смысле распознавание образов состоит в автоматическом выявлении некоторых похожих «регулярностей», встречающихся в данных научных исследований, в экономической или технической информации, и указание некоторого класса для рассматриваемых данных. При этом образы формально представляются отдельными точками в многомерном пространстве решений, а все пространство разделяется на отдельные области, каждая из которых ассоциируется с отдельным классом.

Важным в распознавании образов является то, что множество различных входных данных, несущих информацию об одном образе, должны быть отображены на один и тот же класс. Например, все разновидности рукописного или печатного изображения символа буквы {А} должны отображаться в один и тот же класс «А». Все электрокардиограммы, снятые у здоровых людей, должны отображаться в один и тот же класс «норма» независимо от пола, национальности и возраста обследуемых.

Можно говорить о распознавании образов с учителем и без учителя. В первом случае каждому классу ставится в соответствие «метка»,

указывающая класс. При этом если система имеет «учителя», то он знает, к каким классам на самом деле относятся примеры обучающего множества. Система распознавания образов сначала, в процессе обучения, учится отображать входную информацию в «правильные» классы. Затем, она, усвоив некоторое ограниченное число распознаваемых образов, должна умело отображать в соответствующие «правильные» классы и другие новые входные данные, не входящие в обучающее множество. Задача распознавания образов, суть которой состоит в указании принадлежности входного образа (например, речевого сигнала или рукописного символа), представленного вектором свойств (признаков), одному или нескольким предварительно определенным классам, часто называется классификацией образов. Решение подобной задачи распознавания обычно используется в таких ситуациях, когда требуется заменить оператора какой-либо технической системой и обеспечить при этом достижение таких же результатов, которых добивается человек. К известным решениям такой задачи относятся распознавание печатных или рукописных текстов, распознавание речи, классификация объектов по их изображениям и анализ сцен.

Случай распознавания без учителя возникает, если данные, подлежащие анализу, не снабжены метками, указывающими их принадлежность к классу (отсутствует обучающая выборка с метками классов). Случай обучения без учителя возникает при решении задач распознавания на основе данных произвольного характера, когда вообще может быть неизвестно, отличаются ли некоторые классы друг от друга. Обычно для таких ситуаций используют алгоритм кластеризации (clustering algorithm) – метод, благодаря которому данные объединяются (группируются) в небольшие группы (кластеры) по принципу выявления подобия образов в силу выбранной метрики и размещения близких образов в один кластер. По этому же принципу осуществляется отделение несхожих данных. Следует иметь в виду, что алгоритмы кластеризации имеют под собой «биологическую основу». Человеческий мозг, изучая действительность, новые понятия, всегда сравнивает их с уже существующими знаниями. Мы, определяя новое, пытаемся объединить его с чем-то, что нам уже известно в одном кластере, что является основой для понимания нового. Если новое понятие нельзя связать с тем, что мы уже знаем, нам приходится создавать новую структуру, чтобы понять явление, которое выходит за рамки привычных нам представлений. Объединяя но-

вые понятия в кластеры с уже существующими знаниями, а также, создавая новые кластеры, мы на основе новой информации расширяем свои познания и реорганизуем структуру имеющихся у нас знаний. Кластеризация применяется для извлечения новых знаний, сжатия данных, моделирования сложных технологических процессов.

Распознавание образов с обучением (с учителем) предназначено для отнесения неклассифицированных объектов (явлений, ситуаций, процессов) к заранее описанным классам (обучающим группам). По этой причине определение класса, к которому принадлежит новый объект, часто называют задачей классификации (classification), а устройство распознавания – классификатором. О задаче классификации, в общей постановке, можно сказать следующее. Классификация основана на преобразовании данных наблюдений над образами в совокупность чисел и на обработке этих чисел по специальным алгоритмам. Для выполнения этих операций полезными оказываются широко используемые в математике понятия множества и метрического пространства.

Классификацией (или конечным разбиением) множества C, на языке теории множеств, называется конечное семейство $\{C_1, \ ..., \ C_R\}$ непустых попарно непересекающихся подмножеств $C_k \subset C, \ k = \overline{1, R}$ множества C, в объединении дающих все $C \ \underset{k=1}{\overset{R}{\cup}} C_k = C$. Подмножества $C_к$, называются классами эквивалентности (разбиения). Выделение классов (классификация) представляет собой интуитивный акт и производится с учетом характеристических свойств m образов, так что $C_к = \{m: m \ \ имеет \ свойство \ к\}$. Предполагается, что каждый данный образ обладает каким-либо свойством m или совокупностью значений свойств $\{m\}$, определяющих описание данного объекта. Совокупность значений свойств определяет описание каждого образа $I(m)$. Каждому свойству образа можно сопоставить некоторое число, не совпадающее с числами, приписанными другим свойствам. Тогда всей совокупности N свойств образа будет соответствовать совокупность чисел, которую можно рассматривать как координаты некоторой точки в N- мерном пространстве свойств. Другими словами, с точки зрения функционального анализа, образ – это точка, обладающая совокупностью общих свойств, которую можно счи-

тать набором из n компонент и представлять как точку в n-мерном пространстве наблюдений или входных данных (observation space, date).

Как следует из определения, модель образа характеризуется множеством возможных свойств (состояний). Это множество может исследователем назначаться интуитивно. В зависимости от специфики задачи используется много типов «свойств», позволяющих проводить классификацию. Некоторые свойства хорошо поддаются определению и легко интерпретируются для образов. Например, при классификации электродвигателей полезными «свойствами» могут быть номинальная мощность, число оборотов вала, масса двигателя, его габаритные размеры. Очень трудно выбрать систему свойств для интеллектуальной системы обработки образов, которая должна обеспечивать разделимость классов при классификации электрокардиограмм, изображений и рукописных символов. В ряде подобных случаев вообще не удается непосредственно использовать свойства для разделения образов, к примеру, информацию, поступающую от датчиков. В таких случаях, чтобы добиться компактной формы представления и высокой информативности приходится применять различного рода преобразования «свойств», например, вместо «свойств» самого сигнала использовать «свойства», полученные в результате проведения вейвлет-анализа.

Каждое свойство образа может принимать значения из различных множеств допустимых значений. По этой причине в литературе принято выделять типовые разновидности «состояний» образа. Характеристические свойства можно разделить на качественные и количественные.

Качественные свойства, которые часто называют признаками. Они характеризуются тем, что в их отношении можно высказать лишь суждения типа «да – нет», то есть каждый элемент обладает данным признаком или нет. Классы $\{C\}$, составляющие полную систему N несовместимых альтернатив и связанные с качественным состоянием образа, условимся нумеровать индексами ν=1,2,3,…, N. В этом случае образ характеризуется дискретным набором несовместимых состояний с номерами ν=1,2,3,…, N, а упорядоченный набор признаков определяет «список» классов, образующих простейший вид классификации. Заметим, что решения, принимаемые в результате распознавания, совпадают с одним из возможных теоретических ν-состояний образа, но не обязательно совпадают с его действительным неизвестным состоянием. По этой

причине целесообразно использовать различные обозначения для действительных состояний и решений о них. В дальнейшем для обозначения действительных состояний по-прежнему используем индекс ν=1,2,3,…, N. Решения об этих состояниях, полученные в результате распознавания, обозначим $\hat{\nu} = 1,2,...,N$. Далее, обозначим через g=1,2,3,…, G номера признаков. Для каждого g-го признака введем дискретную совокупность возможных состояний с номерами $m_g = 1,2,...,M_g$, где M_g - полное число состояний g-го признака. Здесь индекс ^ указывает на случайный характер появления $\hat{m}_g$, то есть мы полагаем, что во всех случаях состояния $\hat{m}_g$ подвержены ошибкам тех специалистов или технических устройств, которые устанавливают значения признаков. Так, например, если $m_g = 1,2,3$ соответствуют показаниям светофора (красный, желтый, зеленый), то наблюдения $\hat{m}_g = 1,2,3$ водителя дальтоника через запыленное стекло автомашины могут не совпадать с действительными состояниями светофора. При этом номера $\hat{m}_g$ можно связывать с различными нечисловыми характеристиками образа. Так, например, классы яблока, составляющие полную систему несовместимых альтернатив: $\nu = 1$ – спелое или $\nu = 2$ – неспелое – можно распознавать на вкус по состояниям: $\hat{m}_g = 1$ – горькое, $\hat{m}_g = 2$ – кислое, $\hat{m}_g = 3$ – кисло-сладкое, $\hat{m}_g = 4$ – сладкое. Специалиста-дизайнера, работающего в полиграфической промышленности, в задачах распознавания можно рассматривать как $\nu = 1$ – бездарного, $\nu = 2$ – способного, $\nu = 3$ – гениального по его работам с состояниями: $\hat{m}_g = 1$ – плохая, $\hat{m}_g = 2$ – так себе, $\hat{m}_g = 3$ – хорошая. Примером подобной классификации может служить разбиение на классы одиночных импульсов: $\nu = 1$ – прямоугольный, $\nu = 2$ – экспоненциальный, по его состояниям: $\hat{m}_g = 1$ – не дифференцируется в классическом смысле, поскольку имеет разрывы первого рода, $\hat{m}_g = 2$ – непрерывен, всюду дифференцируем. Укажем, что все возможные g-признаки могут быть косвенными, с

состояниями $m_g = 1,2,...,M_g$, не совпадающими по семантическому смыслу с классами ν=1,2,3,…, N образа, и прямыми признаками с состояниями $m_g = 1,2,...,M_g$, совпадающими по смыслу с ν=1,2,3,…, N. Так, например, если яблоко (спелое, неспелое) оценивается на вкус по состояниям m_g (горькое, кислое, сладкое), то такой признак является косвенным. Но этот же g-й наблюдатель может непосредственно судить о состоянии (классе) яблока по прямым признакам m_g (спелое, неспелое), полагая $m = 1$ – спелое или $m = 2$ – неспелое. Его суждения подвержены ошибкам, и поэтому их не следует отождествлять с фактически неизвестными состояниями $\nu = 1$ и $\nu = 2$ яблока. Другой, $(g+1)$-й, наблюдатель вполне может принять противоположные суждения о том же яблоке.

Количественные свойства часто называют параметрами; они характеризуются численными значениями. Их определяют вектором неизвестных параметров ρ, компоненты ρ_S, s=1,2,3,…, S, которого имеют смысл отдельных неизвестных параметров образа; S – полное число неизвестных. Если особенностью качественных состояний была дискретность, то векторы ρ количественных свойств обычно задаются по непрерывным (или кусочно-непрерывным) шкалам. Примером распознавания образов с использованием количественных свойств может служить узнавание человека, произносящего некоторый набор изолированных слов. В качестве свойств, по которым может осуществляться разделение и маркировка фонем, может служить электрический сигнал, формируемый микрофоном и усилителем. Если, к тому же сигнал подвергается цифровой обработке, то есть дискретизируется и квантуется, то получаем набор параметров, позволяющих разделять фонемы произносимых слов. Еще более трудная задача – распознавание связанной речи. В данной ситуации неизвестно, где кончается одно слово и начинается следующее. Кроме того, важно обеспечить «выделение» некоторого сообщения, а не просто каждой фонемы и каждого слова. Здесь уже требуется использовать синтаксические и семантические свойства текста.

Следует иметь в виду, что в литературе по нейронным сетям, свойства, по которым устанавливается образ, не всегда делят на каче-

ственные и количественные. В этих случаях свойства, по которым устанавливается образ, просто называют признаками. При этом их обычно делят на информативные и несущественные. Признаки, по которым устанавливается образ, называются информативными. Помимо информативных, любой объект имеет множество других признаков, не важных для опознавания образа, но которые могут оказывать существенное влияние на решение задачи разделения образов. Так для выделения мужского лица не информативны цвет глаз и волос, но если мы анализируем классы «блондинов» и «брюнетов», то эти признаки информативны. Одна из основных трудностей при распознавании образов состоит в том, что очень часто информативные признаки не могут быть полно и точно описаны на математическом, логическом или каком-либо другом подобном языке, хотя известно, что они существуют, так как исследователь достаточно уверенно и с малой вероятностью ошибки их обнаруживает. Понятно, что распознавание образов ставит два кардинальных вопроса: какие входные данные можно считать уместными; какая предварительная обработка исходных данных приводит к получению «свойств» или «признаков», позволяющих действительно проводить классификацию.

Хотя этим проблемам в научной литературе уделяется много внимания, но, к сожалению методов, позволяющих получить практический ответ на эти вопросы пока нет. До сих пор на этом пути используются априорные знания, интуиция, метод проб и ошибок и опыт, накопленный исследователем. Если имеется значительный объем априорных сведений о регулярностях, составляющих образ, при условии, что эти регулярности просты, имеют детерминированную природу, отыскание «нужных» признаков удается выполнить. Однако, чаще всего приходится иметь дело с ситуациями, когда о регулярностях явных сведений немного, а сами регулярности отличаются существенными флуктуациями и изменчивостью. В этом случае проблема построения пространств информационных признаков не имеет общего решения. Нельзя забывать и о такой проблеме распознавания, как изменчивость образов, когда входные данные, которые должны быть классифицированы как представители одного и того же класса, могут отличаться довольно сильно. Существуют различные причины изменчивости. Источником изменчивости образа может служить шум или искажения, вносимые в образ. К примеру, акустические отражения (эхо) могут изменить сигналы микрофона,

источником которых служит речь. Различия в освещении объекта, а также «осмотр» его под различными углами, могут приводить к получению совершенно различных изображений (фотографий). Люди при распознавании образов мало внимания обращают на такие «мелочи», но при автоматическом распознавании это может оказаться весьма существенным. Трудно поддается описанию изменчивость, свойственная самим образам. Все различные варианты буквы А, выписанные рукой, из-за различного числа концевых точек, пересечений, а также числа и расположения выпуклых и вогнутых линий, могут очень сильно отличаться друг от друга. Все печатные символы, соответствующие одному знаку, имеют в идеале, стандартные размеры и очертания; изменчивость их возникает из-за ограниченности срока службы механизмов, используемых при печати, неравномерным нанесением типографской краски, «загрязнением» буквы, неоднородностью бумаги. «Декоративные» символы могут сильно отличаться от обычных. Написанные от руки печатные символы проявляют ужасающую изменчивость и по форме и по размеру. Во многих задачах образы взаимно проникают друг в друга, делая невозможным их однозначное разделение. Чтобы выйти из такой ситуации, пространство свойств (наблюдений) сначала переводят в некоторое «промежуточное» пространство признаков, из которого уже проще отображать «точку образа» в один из классов. По этой причине, решая задачу о разделении «искаженных помехами и изменчивостью» образов, стараются найти такое отображение признаков в пространстве (как говорят – извлечь признаки), чтобы различным образам соответствовали непересекающиеся области, внутри которых точки расположены достаточно близко друг к другу, а между этими областями существуют «пустые» места.

4.2. Использование нейронной сети для решения задач «автоматической» классификации

4.2.1. Классификация линейно-разделимых множеств

После этих важных вводных замечаний рассмотрим как для решения задач «автоматической» классификации можно использовать нейронные сети. В качестве иллюстрации выберем задачу двухальтернативного распознавания ν=1, 2, когда анализируемый объект, опреде-

ляемый вектором оценок параметров $\hat{\theta}$, с компонентами $\hat{\theta}_S$, s=1,2,3,…, S, на основании решения, принимаемого в процессе распознавания, должен быть отнесен к одному из классов. Чтобы осуществить классификацию образов алгоритм обработки входных данных сети должен реализовать решающее правило. Построение решающего правила, в общем случае, сводится к выбору двух элементов: функции отклика $L(v)$, принимающей различные значения при подстановке конкретной реализации экспериментального материала, и правила принятия решения о классе объекта по значениям функции отклика. В случае двухальтернативного распознавания оба эти элемента удается представить в виде формулы $\hat{C} = \begin{cases} \hat{C}_1, & если \quad \lambda(p) < 0, \\ \hat{C}_2 & если \quad \lambda(p) \geq 0, \end{cases}$ где $\lambda(p)$ – некоторая функция. С геометрической точки зрения уравнение $\lambda(p) = 0$ определяет в пространстве наблюдений линию, разделяющую пространство на две области принятия решений: $\hat{C} = \hat{C}_1$ и $\hat{C} = \hat{C}_2$. Эту линию называют дискриминатной (от лат. discriminatio – различение). Если сказанное перевести на язык нейронных сетей, то это означает, что в задачах классификации мы должны построить решающее правило, согласно которому выходной элемент должен выдавать большой сигнал в случае, если данные наблюдения принадлежат интересующему нас классу, и слабый – в противоположном случае. Чтобы обучить сеть работе согласно такого решающего правила необходимо найти вектор весовых коэффициентов w, для которого будут истинны следующие утверждения:

– $w^T p > 0$ для любого входного вектора P, принадлежащего классу C_1;

– $w^T p \leq 0$ для любого входного вектора P, принадлежащего классу C_2.

Будем считать, что нейронная сеть представляет собой персептрон. Известно, что такая нейронная сеть состоит из линейного сумматора и ограничителя, реализованного в виде пороговой функции вычисления знака. Выход нейрона принимает значения +1, если сигнал на выходе сумматора положителен, и -1, если отрицателен. В этом, простейшем случае, когда в качестве классификатора выступает персептрон и решающее правило состоит в том, что двухкомпонентный входной сигнал P относится к классу C_1, если выход равен единице и к классу C_2 в

противном случае, имеется две области решения, которые разделяет прямая, определяемая уравнением $\sum_{i=1}^{2} w_i p_i + b = 0 \;\; w_1 p_1 + w_2 p_2 + b = 0 \Rightarrow$.

Для заданного входного стимула P, у которого подмножества P_1 и P_2 достаточно далеко удалены друг от друга, можно найти бесконечное множество прямых, соответствующих различным значениям весов w_1 и w_2, которые бы удовлетворят равенству $w_1 p_1 + w_2 p_2 + b = 0$. По этой причине для нейронных сетей были разработаны специальные алгоритмы настройки вектора весов. Обычно для этого используют процедуру адаптации однослойного персептрона на основе коррекции ошибок (error-correction procedure), при которой на различных итерациях весовые коэффициенты адаптируются таким образом, чтобы в конце обучения вектор весов стал ортогонален разделяющей (дискриминантной) линии. В литературе показано, что алгоритм обучения персептрона (адаптации синаптических весов) сходится после определенного числа итераций, осуществляя, таким образом, корректную классификацию образов.

Рассмотрим конкретную задачу. Пусть оператор печатной машины, заступающий на смену, проходит тестирование на специальных автоматизированных устройствах (тренажерах), которые позволяют определять некоторые показатели, которые влияют на его работоспособность и являются причиной ошибок и срывов в работе. Полученные данные поступают на классификатор, выполненный на основе нейронной сети, который должен в автоматическом режиме выдать сигнал о готовности человека к своей профессиональной деятельности. Для определения функционального состояния оператора сеть предварительно проходила «обучение с учителем», который, согласно многочисленных экспериментов и опросов экспертов, установил определенные показатели результативности (эффективности) деятельности операторов.

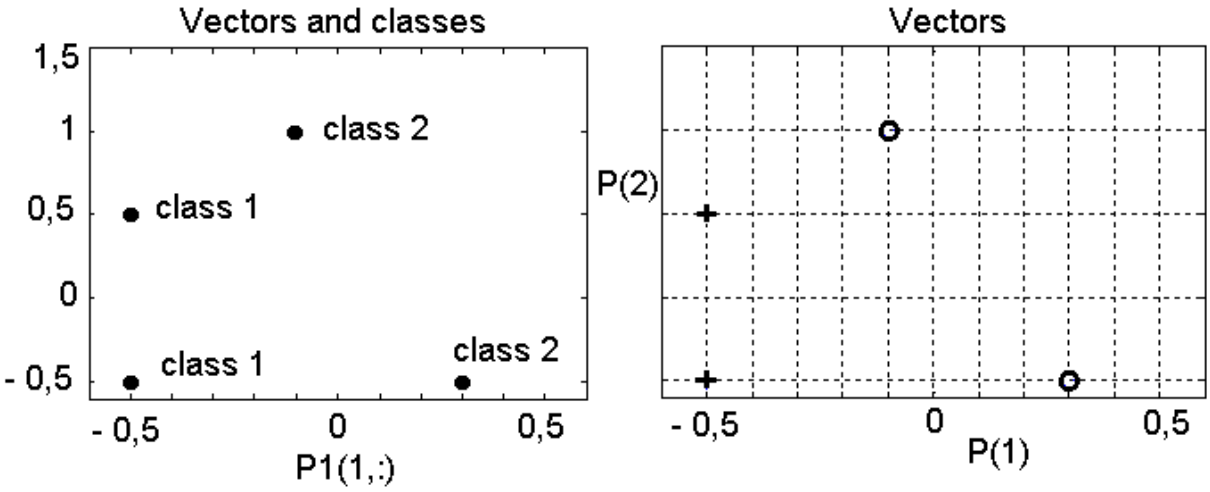
Предположим, что двухвходовому персептрону «учителем» были предъявлены следующие обучающие пары показателей результативности {входной вектор и соответствующее ему значение желаемого отклика}: {P_1, T_1}={(-0.5 -0.5), 1)}; {P_2, T_2}={(-0.5 0.5), 1)}; {P_3, T_3}={(+0.3 -0.5), 0)}; {P_4, T_4}={(-0.1 1), 0)}. При этом, если T = 1, то оператор готов к выполнению работы (попадает в класс C_1, как человек, который готов к данному виду деятельности), в противном случае, при T = 0, деятельность оператора связана с вероятностью возникновения аварийных си-

туаций (он попадает в класс C_2). Чтобы наглядно посмотреть, как функционирует классификатор, как он «строит границу для классов» и «принимает решение», водим в командное окно следующую программу:

```
clear
% Определяем обучающие пары показателей
P1 = [-0.5 -0.5;-0.5 0.5; +0.3 -0.5; -0.1 1]';
% Задаем вектор индексов классов
Tc = [1 1 2 2];
disp( ' Матрица связности, которая показывает')
disp( ' к какому классу (первая цифра)')
disp( ' принадлежит вектор (вторая цира)')
Tind=ind2vec(Tc)
disp( ' Полная матрица связности,  в которой показаны к какому классу ')
disp( ' (строка матрцы) принадлежит ')
disp( ' вектор (столбец матрицы)')
Tind1=full(Tind)
plot(P1(1,:),P1(2,:),'.','markersize',30)
for i=1:4,
text(P1(1,i)+0.1,P1(2,i)+0.1,sprintf('class %g',Tc(i))),
end
axis([-0.6 0.6 -0.6 1.5])
title(' Vectors and  classes')
xlabel('P1(1,:)')
%  Определяем сигнал
P = [ -0.5 -0.5 +0.3 -0.1;  ...
     -0.5 +0.5 -0.5 +1.0];
% Определяем с помощью "учителя" к какому
%  классу относится каждое наблюдение
T = [1 1 0 0];
% Определяем расположение точек на плоскости
figure(2);
plotpv(P,T);
axis([-0.6 0.6 -0.6 1.5])
title(' vectors ')
grid on
% Создаем сеть
net = newp([-1 1;-1 1],1);
% Задаем веса
% Проводим обучение сети
net.adaptParam.passes = 3;
net = adapt(net,P,T);
figure(3);
plotpv(P,T);
plotpc(net.IW{1},net.b{1});
title(' vectors and discrimin. line ')
% Вводим параметр, который хотим классифицировать
p = [0.7; 1.2];
a = sim(net,p);
figure(4);
plotpv(p,a);
axis([-0.6 0.6 -0.6 1.5])
point = findobj(gca,'type','line');
set(point,'Color','red');
% Строим график с новым вектором
hold on;
plotpv(P,T);
plotpc(net.IW{1},net.b{1});
title(' vectors and discrimin. line ')
hold off;
```

После запуска программы получаем матрицу связности, которая показывает, к какому классу (первая цифра) на этапе обучения принадлежат входные данные (вторая цифра). Tind = (1,1) (1,2) (2,3) (2,4) свидетельствует о том, что к первому классу относятся первая и вторая

реализация вектора вектор, а ко второму – третья и четвертая. Полная матрица связности Tind1 = 1 1 0 0
0 0 1 1 также отражает тот факт, что в соответствии с первой строкой матрицы к первому классу относятся первый и второй вектор (первый и второй столбцы равны 1), а ко второму (в соответствии с второй строкой матрицы) – третий и четвертый. На рис. 4.1а показано, что для обучения использованы входные переменные персептрона, которые принадлежат двум различным классам. При этом, как видно из рисунка, где показано размещение входных векторов в виде точек на плоскости, образы значительно отделены друг от друга, что позволяет провести их корректную классификацию.



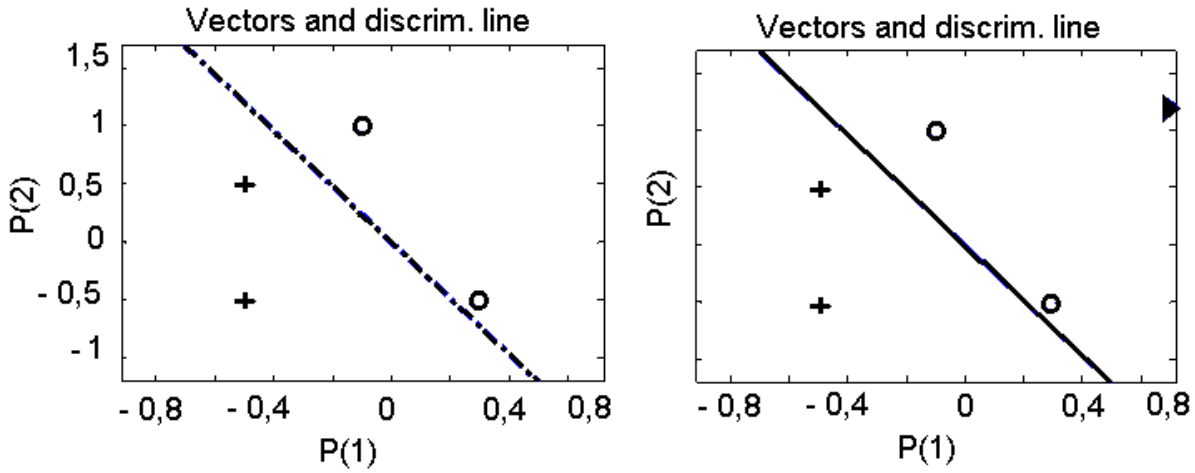
а) б)

Рис. 4.1. **Присвоение классов (а) обучающим векторам и расположение входных данных в двухмерном пространстве, которые нейронная сеть должна научиться разделять их на классы (б)**

При обучении использованы четыре входных вектора, имеющие по два признака. Поскольку количество признаков определяет размерность пространства, из которого выбираются все вводимые данные, то рис. 4.1б иллюстрирует, что для обучения двухвходового персептрона используется двумерное пространство, где исходные данные для классификации представлены в виде точек, помеченных маркерами + и ●.

В процессе обучения сеть проводит корректировку весовых коэффициентов и «создает» линию (границу принятия решения), позволяющую разделять образы на два класса (рис. 4.2а). После того, как персептрон настроил весовые коэффициенты, он может правильно относить

новые внешние стимулы к одному из двух классов. На рис. 4.2б. показано, как после обучения нейронная сеть классифицировала новый входной вектор P = [0.7; 1.2], соответствующий, согласно условию задачи, показателям, полученным на тренажере оператором. Автоматический классификатор, как мы видим, выдал сигнал о неготовности человека к своей профессиональной деятельности, поскольку попадание теста в класс C_2 свидетельствует о большой вероятности возникновения при его работе в данном состоянии аварийных ситуаций.



а) б)

Рис. 4.2. **Дискриминантная линия (граница принятия решения), разделяющая входные данные на классы, которая была получена в процессе обучения (а) и отнесение нового показателя (в виде треугольника) к одному из классов (б)**

4.2.2. Классификация линейно-неразделимых множеств

При разделении образов на соответствующие классы мы рассмотрели случай, когда пространство оказывается двумерным. Если бы мы работали с сетью, классифицирующей три признака, то пространство признаков было бы трехмерным. Для разделения классов в этом случае необходима бы была плоскость. В случае n признаков пространство признаков оказывается n -мерным. При решении задач классификации при размерности больше двух для разделения образов на соответствующие классы требуется уже не прямая линия или плоскость, а n-мерная гиперповерхность. При этом если для деления на классы требуется одна прямая, одна плоскость или одна гиперповерхность, то гово-

рят, что проблема разделения на классы является линейной. Если же для классификации требуется несколько прямых или гиперповерхностей, то проблема разделения входных данных на классы считается нелинейной. Следует иметь в виду, что для того, чтобы персептрон функционировал корректно, два класса C_1 и C_2 должны быть линейно разделимы (linearly separable). Это означает, что для правильной классификации образы должны быть значительно отделены друг от друга, чтобы поверхность решений могла представлять собой одну гиперплоскость. Если два класса сдвинуты на очень близкое расстояние, признаки расположены так, что классифицируемые точки «пересекаются», персептрон не дает корректной классификации. Следующая программа наглядно демонстрирует ситуацию, когда классифицируемые образцы «пересекаются» или, точнее говоря, линейно-неразделимы.

```
% Пример некорректной классификации
P = [ -0.5 -0.5 +0.3 -0.1 -0.8; ...
     -0.5 +0.5 -0.5 +1.0 +0.0 ];
T = [1 1 0 0 0];
plotpv(P,T);
net = newp([-40 1;-1 50],1);
hold on
plotpv(P,T);
linehandle=plotpc(net.IW{1},net.b{1});
net.adaptParam.passes = 3;
linehandle=plotpc(net.IW{1},net.b{1});
for a = 1:25
  [net,Y,E] = adapt(net,P,T);
  linehandle =
plotpc(net.IW{1},net.b{1},linehandle);
 drawnow;
end;
```

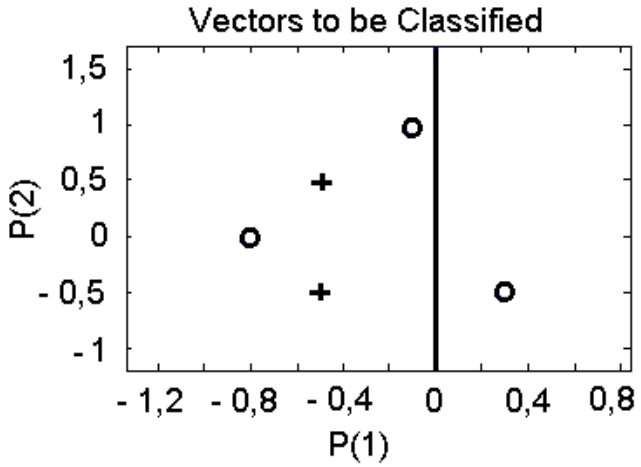


Рис. 4.3. **Пример, когда линия, разделяющая образы на классы, проведена не верно и нейронная сеть некорректно осуществляет классификацию входных данных**

Как видно из рис. 4.3 после получения дискриминантной линии некоторое число «нуликов» и «крестиков» попадает в чужеродные области принятия решения. Следовательно, в этом случае персептрон, несмотря на все «усилия», осуществляет некорректную классификацию.

Элементарный однослойный персептрон не может классифицировать входные образы, которые линейно-неразделимы. Однако такие случаи на практике встречаются не так уж редко. Подсчитано, что для n нейронов, входами которых являются двоичные переменные, вероятность того, что образы окажутся линейно разделимыми, весьма мала даже для небольшого числа переменных.

В литературе давно обсуждается задача, известная как классификация ирисов. В ней рассматривается три вида цветков; для каждого измерены длина и ширина чашелистика, длина и ширина лепестка. Цель решения задачи, когда имеются цветы, экземпляры которых легко отнести к другому классу, состоит в том, чтобы научиться принимать решение о типе нового (полученного в результате селекции и пока неизвестного) цветка по данным измерений чашелистика и лепестка. Подобной «тестовой» задачей является известная задача об «исключающем ИЛИ» (XOR-exclusive OR problem), которую можно рассматривать как частный случай более общей задачи классификации точек единичного гиперкуба (unit hypercube). Проблема XOR является линейно-неразделимой, поскольку на выходе данного логического элемента появляется логическая единица тогда, когда только одно из вводимых значений [входных пар (0, 0), (0, 1), (1, 0), (1,1)] равно 1. Входные данные (0, 0), (1, 1), а также (0, 1), (1, 0) располагаются, если рассматривать их расположение на плоскости, в противоположных углах единичного квадрата и принадлежат разным классам, поэтому «точки» на плоскости классификации не удается разделить одной линией.

Понятно, что персептрон в принципе не может решить нелинейную проблему, поскольку при использовании всего одного нейрона с двумя входами граница решений в пространстве весов всегда будет линейной. Действительно, несмотря на то, что положение и ориентация этой линии в пространстве входов определяется синаптическими весами, всегда для точек, расположенных по одну сторону этой линии, выход нейрона будет равен единице, а для остальных точек – нулю. Для случая XOR, имея две пары точек, расположенных в противоположных углах квадрата и относящихся к разным классам, мы, естественно, не сможем по-

строить прямую линию так, чтобы точки одного класса лежали по одну сторону от разделяющей поверхности. Выход из создавшейся ситуации состоит в использовании многослойной сети обратного распространения. Если представить картину расположения точек в противоположных углах квадрата, напрашивается решение об использовании для разделения данных (в их двумерном виде) двух прямых. Для реализации такого решения нейронная сеть должна работать в два этапа, что потребует два скрытых слоя нейронов. Первый из этих слоев должен «строить» две прямые и должен состоять из двух нейронов, которые бы воспринимали по два значения входных данных и имели бы два выхода. Этот слой должен проводить границы решений так, чтобы точки, попадающие в один класс, находились либо «внутри» этих прямых, либо снаружи. Чтобы, в конце концов, правильно определить класс, второй слой, для которого входные данные уже линейно разделимы, должен моделировать прямую. По этой причине второй слой, должен иметь один нейрон, который, объединяя информацию о прямых первого слоя, давал на выходе сети однозначный сигнал 1, если входные данные соответствуют точкам пространства входных сигналов, расположенным между указанными линиями, или 0 – в противоположном случае.

Скрытые нейроны первого слоя, при таком подходе к проблеме, играют важную роль в работе многослойной сети. Они в ходе обучения, преобразуя и разделяя данные на два новых класса, «выявляют характерные, общие черты» данных обучения. В этой связи скрытые нейроны первого слоя можно рассматривать как нелинейные обнаружители признаков (nonlinear feature detector), которые призваны, за счет преобразования входной информации, отобразить классы исходного входного линейно-неразделимого пространства в пространство активности второго скрытого слоя, где их линейная разделимость более вероятна.

Обобщая сказанное и рассуждая подобным образом, можно, предложить общие рекомендации по созданию сетей, осуществляющих классификации линейно – неразделимых входных данных. Чтобы было легче отделить входные точки друг от друга в новом пространстве, чем в исходном, можно, применять сети, преобразование в которых осуществляется за счет использования выпуклых областей пространства, а не двух линий. Известно, что область считается выпуклой, если для каждой из двух ее точек соединяющий их отрезок целиком лежит в этой области. Если в первом скрытом слое взять три нейрона, то он, проводя границы

решений, будет таким образом строить «треугольник», чтобы точки, попадающие в один класс, находились либо «внутри» его, либо снаружи треугольника. Включение шести нейронов позволит для нелинейного обнаружения признаков использовать шестиугольники. Включением достаточного числа нейронов в первый скрытый слой, может быть образован выпуклый многоугольник (многогранник) желаемой формы, способный отобразить классы исходного входного линейно-неразделимого пространства в пространство активности второго скрытого слоя, где имеет место их линейная разделимость. Можно также сказать о следующем. Точки, не составляющие выпуклой области, не могут быть отделены от других точек пространства двухслойной сетью из-за того, что возможности нейрона второго слоя ограничены. Более общей в этом случае является трехслойная сеть, у которой ограничения на выпуклость отсутствуют. Нейрон третьего слоя принимает в качестве входа набор выпуклых многоугольников, даже если их логические комбинации не выпуклы. Объединяя различные области, выпуклые и невыпуклые, можно осуществлять разделение на классы входных данных (образов) любой формы, всякий раз выдавая на выходе нейронной сети 1, когда входной вектор принадлежит одной из них.

Проиллюстрируем сказанное на следующих примерах. Введем в командное окно программу, демонстрирующую, как осуществляется двухслойной нейронной сетью (с двумя нейронами в первом скрытом слое и с одним во втором) решение задачи об «исключающем ИЛИ».

```
clear
% Определяем входной и тестовый сигнал
disp(' Входной сигнал для обучения сети')
P = [0 1 0 1;0 0 1 1]
disp(' Желательный отклик')
test = [0 1 1 0]
% Определяем обучающие пары показателей
P1 = [0 0;1 0; 0 1; 1 1]';
% Задаем вектор индексов классов
Tc = [1 2 2 1];
disp(' Матрица связности, которая показывает к какому ')
title(' vectors ')
grid on
%Создаем многослойную сеть
netbm = newff([-0.01 1.01; -0.01 1.01],[2,1],{'tansig','tansig'},'traingd');
% Определяем объектно-ориентированные параметры
% сети, определяющие процесс обучения.
%  Параметр скорости обучения
netbm.trainParam.lr = 0.03;
%  Используемое число эпох
netbm.trainParam.epochs = 5000;
% Количество итераций (эпох), по истечении которых будут представляться
% промежуточные результаты
```

```
disp('классу (первая цифра) принадле-')
disp( ' жит вектор (вторая цифра)')
Tind=ind2vec(Tc)
disp(' Полная матрица связности,  в ко-
торой показаны к какому классу ')
disp(' (строка матрицы) принадлежит ')
disp( ' вектор (столбец матрицы)')
Tind1=full(Tind)
plot(P1(1,:),P1(2,:),'.','markersize',30)
for i=1:4,
text(P1(1,i)+0.1,P1(2,i)+0.1,sprintf('class
%g',Tc(i))),
end
axis([-0.1 1.6 -0.1 1.4])
title(' Vectors and  classes')
xlabel('P1(1,:)')
T = test;
% Определяем расположение точек на
плоскости
figure(2);
plotpv(P,T);
axis([-0.1 1.6 -0.1 1.4])
netbm.trainParam.show = 500;
% Значение целевой ошибки обучения
netbm.trainParam.goal = 1e-5;
%  Проводим обучение сети
netbm1 = train(netbm,P,test);
%  Рассчитываем отклик сети
% на входной  сигнал
disp('Полученный отклик сети на входной
сигнал')
y1 = sim(netbm1,P)
EN=test-y1;
disp( ' Среднеквадратическая ошибка в
итоге ')
disp( ' работы нейронной сети')
err1 =mse(EN)
% Вводим параметр, который хотим
классифицировать'новый сигнал, который
disp('  хотим классифицировать')
pnov = [1 0;0 1]
disp( 'Класс, куда отнесен новый входной
сигнал')
a = sim(netbm1,pnov)
```

Обратим внимание на следующее обстоятельство. Поскольку веса и смещения инициализируются случайным образом и пошаговая корректировка синаптических весов продолжается до тех пор, пока сеть не достигнет устойчивого состояния, результаты расчетов по программам, графики ошибок имеют статистическую природу и могут отличаться от одного запуска программы к другому. Учитывая стохастический характер работы алгоритмов нейронной сети, следует, в случае первоначальной неудачи, несколько раз запустить программу.

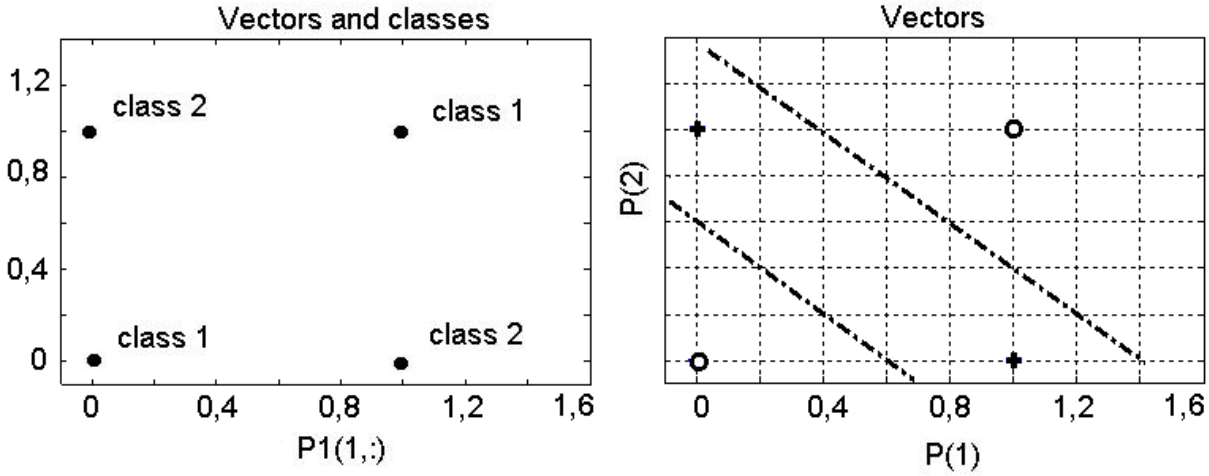
После запуска программы получаем два варианта геометрической трактовки обучающих входных данных (рис. 4.4а,б). На рис. 4.4б совокупность обучающих данных промаркирована «крестиками» и «ноликами» и наглядно показано, что интуитивно в качестве решающего правила можно выбрать две прямые линии, которые проводят границы решений так, чтобы точки, попадающие в один класс, находились либо «внутри» этих двух прямых, либо снаружи.

После запуска программы в командное окно также выводится информация, позволяющая судить о верности введенных данных, о том к какому классу относятся обучающие примеры, о ходе обучения, о точности совпадении желаемого с действительным и прочее.

Входной сигнал для обучения сети: P = 0 1 0 1
0 0 1 1.

Желательный отклик: test = 0 1 1 0.

Матрица связности, которая показывает, к какому классу (первая цифра) принадлежит вектор (вторая цифра): Tind = (1,1) (2,2) (2,3) (1,4).



а) б)

Рис. 4.4. **Классификация обучающих входных данных в двухмерном пространстве при решении проблемы «Исключающего ИЛИ» (а) и расположение входных данных в вершинах квадрата и возможный вариант разделения их на классы за счет использования двух прямых (б)**

Полная матрица связности, в которой показаны к какому классу (строка матрицы) принадлежит вектор (столбец матрицы): Tind1 = 1 0 0 1
0 1 1 0.

Полученный отклик сети на входной сигнал:
y1 = 0.0020 0.9794 0.9645 0.0014.

Среднеквадратическая ошибка в итоге работы нейронной сети:
err1 = 4.2354e-004.

После того, как двухслойная нейронная сеть настроила свои весовые коэффициенты, она может правильно относить новые внешние стимулы к одному из двух классов. В командное окно выводятся результат

того, как после обучения нейронная сеть классифицировала новый входной вектор P соответствующий, согласно условию задачи.

Новый сигнал, который хотим классифицировать: pnov = 1 0
0 1.

Класс, куда отнесен новый входной сигнал: a = 0.9794 0.9645.

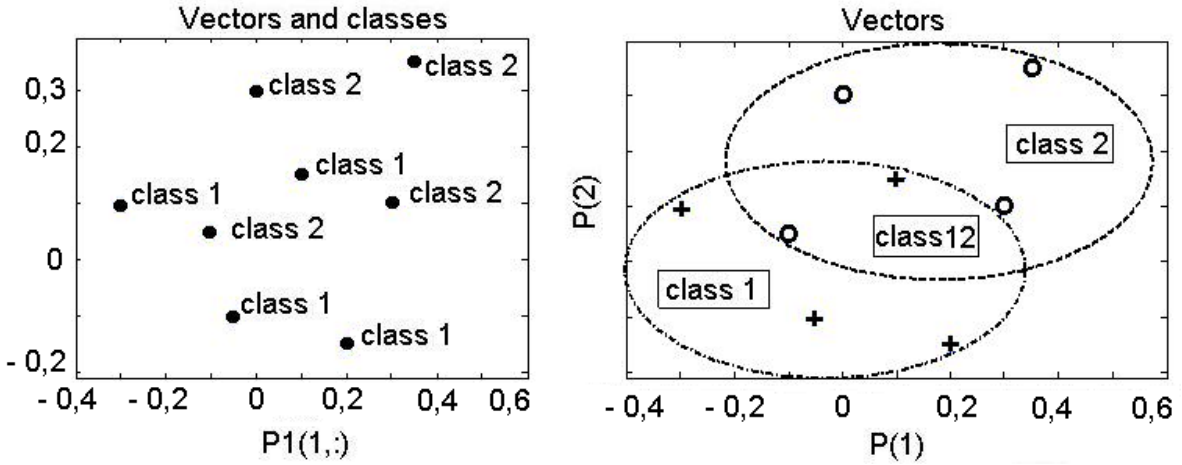
Если приять во внимание, что в идеале мы должны были получить на выходе сети два сигнала по 1, то можно утверждать, что сеть уверенно классифицирует ранее линейно-неразделимые входные данные. Предположим теперь, что мы хотим осуществить классификацию линейно-неразделимых входных данных, но таких, что некоторое число «ноликов и крестиков» попадает с чужеродные области так, что их нельзя разделить прямыми. В этом случае, как уже отмечалось, целесообразно для разделения на классы, для нелинейного обнаружения признаков использовать выпуклые многоугольники. Предположим, что включение 20 нейронов в первый скрытый слой окажется достаточным, чтобы был образован выпуклый многоугольник желаемой формы, способный отобразить классы исходного входного линейно - неразделимого пространства в пространство активности второго скрытого слоя, где имеет место их линейная разделимость. Вводим в командное окно программу.

```
clear
%  Определяем входной  и тестовый
сигнал
disp(' Входной сигнал для обучения
сети')
P = [-0.05 -0.3 -0.1 0.0 0.1 0.2 0.35 0.3; ...
-0.1 0.1 0.05 0.3 0.15 -0.15 0.35 0.1]
disp(' Желательный отклик')
test = [1 1 0 0 1 1 0 0]
% Определяем обучающие пары показа-
телей
P1 = [-0.05 -0.1; -0.3 0.1; -0.1 0.05; 0 0.3; .
0.1 0.15; 0.2 -0.15; 0.35 0.35; 0.3 0.1]';
% Задаем вектор индексов классов
Tc = [1 1 2 2 1 1 2 2];
disp( ' Матрица связности, которая пока-
зывает к какому классу ')
disp(' (первая цифра) принадлежит ')
disp( ' вектор (вторая цифра)')
axis([-0.4 0.6 -0.21 0.4])
title(' vectors ')
%Создаем многослойную сеть
netbm = newff([-1 1;-1
1],[20,1],{'tansig','tansig'},'traingd');
% Определяем объектно-
ориентированные параметры сети,
% определяющие процесс обучения.
%  Параметр скорости обучения
netbm.trainParam.lr = 0.03;
% Используемое число эпох
netbm.trainParam.epochs = 5000;
% Количество итераций (эпох),по исте-
чении которых будут
% представляться (выводиться) проме-
жуточные результаты
netbm.trainParam.show = 500;
% Значение целевой ошибки обучения
netbm.trainParam.goal = 1e-5;
```

```
Tind=ind2vec(Tc)
disp(' Полная матрица связности,  в ко-
торой показаны к какому классу ')
disp(' (строка матрицы) принадлежит')
disp( ' вектор (столбец матрицы)')
Tind1=full(Tind)
plot(P1(1,:),P1(2,:),'.','markersize',30)
for i=1:8,
text(P1(1,i)+0.01,P1(2,i)+0.01,sprintf('class
%g',Tc(i))),
end
axis([-0.4 0.6 -0.21 0.4])
title(' Vectors and  classes')
xlabel('P1(1,:)')
T = test;
% Определяем расположение точек на
плоскости
figure(2);
plotpv(P,T);
```

```
%  Проводим обучение сети
netbm1 = train(netbm,P,test);
% Рассчитываем отклик сети
%  на входной  сигнал
disp( 'Полученный отклик сети на вход-
ной  сигнал')
y1 = sim(netbm1,P)
EN=test-y1;
disp(  ' Среднеквадратическая ошибка в
итоге работы нейронной сети ')
err1 =mse(EN)
% Вводим параметр, который хотим
классифицировать
disp(' Новые сигналы, которые ')
disp(' хотим классифицировать')
pnov = [-0.0 0.3;-0.15 0.35]
disp('Классы, куда отнесены новые вход-
ные  сигналы')
a = sim(netbm1,pnov)
```

После запуска программы получаем два варианта геометрической трактовки обучающих входных данных (рис. 4.5а,б).



а) б)

Рис. 4.5. **Классификация обучающих входных данных в двухмерном пространстве (а) и расположение входных данных в двух областях, каждая из которой ассоциируется с определенным классом, для ситуации, когда в каждой области имеются точки, «попадающие» одновременно в два класса (class12) (б)**

На рис. 4.5б совокупность обучающих данных, помеченных «крестиками» и «ноликами», наглядно демонстрирует, что некоторое число точек попадает в чужеродные области принятия решений и имеется большая вероятность ошибочного решения. Это подтверждает необходимость использования для классификации средств, связанных с выпуклыми многоугольниками и с преобразованием входного пространства за счет применения в первом скрытом слое сети большого числа нейронов. Любая совокупность точек, находящихся в рассматриваемой плоскости, образует «точечное множество» M. Говорят, что совокупность всех точек этой плоскости образует двумерное евклидово пространство $E^{(2)}$. То, что множество M содержится в пространстве $E^{(2)}$, символически обозначается, как $M \subset E^{(2)}$. Из рисунка видно, что в пространстве $E^{(2)}$ имеется два подпространства M_H и M_K (ноликов и крестиков), состоящих из конечного или счетного множества точек, образующих фигуры (эллипсы) на плоскости. Если некоторая точка P входит в множество M, то говорят, что она принадлежит множеству M и обозначают $P \in M$. Разделение на классы можно легко выполнить, если бы подпространства M_H и M_K не содержали точек, одновременно входящих в них.

Если сети в процессе обучения удается образовать выпуклый многоугольник такой формы, чтобы нолики были внутри его, а крестики – вне, чтобы отсутствовало, как для эллипсов «пересечение» (класс12), то нейронная сеть способна отобразить классы исходного входного линейно-неразде-лимого пространства в пространство активности второго скрытого слоя, где имеет место их линейная разделимость. После запуска программы также выводится информация, позволяющая судить о верности введенных данных, о том к какому классу относятся обучающие примеры, о ходе обучения, о точности процесса классификации и прочее:

Входной сигнал для обучения сети: P =

-0.0500 -0.3000 -0.1000 0 0.1000 0.2000

-0.1000 0.1000 0.0500 0.3000 0.1500 -0.1500

0.3500 0.3000

0.3500 0.1000

Желательный отклик: test = 1 1 0 0 1 1 0 0.

Матрица связности, которая показывает, к какому классу (первая цифра) принадлежит вектор (вторая цифра):

Tind = (1,1) (1,2) (2,3) (2,4) (1,5) (1,6) (2,7) (2,8).

Полная матрица связности, в которой показаны к какому классу (строка матрицы) принадлежит вектор (столбец матрицы): Tind1 =

1 1 0 0 1 1 0 0

0 0 1 1 0 0 1 1.

Полученный отклик сети на входной сигнал: y1 = 0.8865 0.9073 0.0653 -0.0152 0.9148 0.8993 -0.0084 0.0315.

Среднеквадратическая ошибка в итоге работы сети: err1 = 0.0056.

После того, как двухслойная нейронная сеть настроила свои весовые коэффициенты и может правильно относить новые внешние стимулы к одному из двух классов, в командное окно выводится результат того, как сеть классифицировала новый входной вектор P соответствующий, согласно условию задачи:

Новые сигналы, которые хотим классифицировать: pnov = 0 0.3000

-0.1500 0.3500.

Классы, куда отнесены новые входные сигналы: a = 0.9635 0.3366.

Полученные результаты свидетельствуют о том, что удалось успешно обучить сеть и она, если учесть, что в идеале мы должны были получить на выходе сети два сигнала 1 и 0, достаточно «уверенно» классифицирует предъявляемые ей новые образцы входных данных.

Рассмотрим теперь случай, когда мы хотим осуществить классификацию таких линейно-неразделимых входных данных, при которых множества «ноликов и крестиков» образуют области довольно сложной формы, которые вряд ли удастся разделить выпуклыми многоугольниками. Как отмечалось, подобные невыпуклые области решений могут быть реализованы на трехслойной сети, когда поступающие на входы нейрона третьего слоя выпуклые многоугольники путем определенных логических комбинаций, осуществляемых этим нейроном, образуют невыпуклый многоугольник. Вводим в командное окно программу:

```
clear
% Определяем вход-й и тестовый сигнал
disp( 'Входной  сигнал для обучения сети')
P = [0 -0.05 -0.15 -0.25 -0.35 -0.45 -0.4 -
0.375 -0.3 -0.2 ...
-0.1 -0.15 -0.25 -0.35 -0.4  -0.45 -0.35 -0.2 -
0.1 -0.3 ...
-0.275 -0.25 -0.3 -0.15 -0.1 -0.05 0.05 0.125
ориентированные параметры сети,
% определяющие процесс обучения.
%  Параметр скорости обучения
netbm.trainParam.lr = 0.03;
%  Используемое число эпох
netbm.trainParam.epochs = 5000;
% Количество итераций (эпох),по истечении которых будут
```

```
0.05 -0.05 ...
-0.025 -0.15 -0.275 -0.3 -0.2 -0.25 -0.125
0.025 0 0.05; ...
0.35 0.3 0.4 0.3 0.35 0.3 0.2 0.1 0.05 0 ...
-0.05 -0.075 -0.1 -0.125 -0.2 -0.3 -0.35 -0.4 -
0.375 -0.45 ...
0.25 0.225 0.1 0.15 0.1 0.2 0.1 0.05 -0.05 0 ..
-0.1 -0.125 -0.15 -0.225 -0.2 -0.3 -0.325 -
0.25 -0.35 0.05]
disp(' Желательный отклик')
test = [0 0 0 0 0 0 0 0 0 0 ...
      0 0 0 0 0 0 0 0 0 0 ...
1 1 1 1 1 1 1 1 1 1 ...
1 1 1 1 1 1 1 1 1 1]
%  Определяем расположение точек на
плоскости
figure(2);
plotpv(P,test);
axis([-0.6 0.2 -0.5 0.5])
title(' vectors ')
%Создаем многослойную сеть
netbm = newff([-0.6 0.6; -0.6
0.6],[20,10,1],...
{'tansig','tansig','tansig'},'traingd');
%  Определяем объектно-
%  представляться (выводиться) про-
межуточные результаты
netbm.trainParam.show = 500;
%   Значение целевой ошибки обучения
netbm.trainParam.goal = 1e-5;
% Проводим обучение сети
netbm1 = train(netbm,P,test);
%  Рассчитываем отклик сети
% на входной  сигнал
disp( 'Полученный отклик сети на
входной сигнал')
y1 = sim(netbm1,P)
EN=test-y1;
disp(  ' Среднеквадратическая ошибка в
итоге ')
disp(' работы нейронной сети')
err1 =mse(EN)
% Вводим параметр, который хотим
классифицировать
disp(' Новый сигнал, который ')
disp(' хотим классифицировать')
pnov = [-0.2 0.1 -0.3;0.35 0 -0.1]
disp(  'Класс, куда отнесен новый вход-
ной  сигнал')
a = sim(netbm1,pnov)
```

После выполнения программы получаем геометрическую трактовку совокупности обучающих данных, промаркированных «крестиками» и «ноликами», показывающую, что нейронная сеть используется в этом случае для решения сложной задачи классификации. Как видно из рис. 4.6, первый класс входных данных (нолики) располагается в области, по форме напоминающей символ ∑, а «крестики» находятся в области имеющей очертания в виде знака 2. Результат работы программы:

Отклик сети на входной сигнал:

```
y1 =   0.0221   -0.0056   -0.0347    0.1992   -0.1019    0.0522
       0.0013    0.0046    0.1074   -0.1023    0.1658    0.0236
       0.1812   -0.0934    0.0898   -0.1113    0.1902   -0.0935
       0.2770   -0.0659    0.7531    0.9605    0.8213    0.9692
       0.9666    0.9268    0.9970    0.9998    0.9982    0.8661
       0.9729    0.7593    0.7184    0.9093    0.9566    0.9288
```

0.8303 0.9948 0.7305 0.9986.

Среднеквадратическая ошибка в итоге работы сети: err1 = 0.0167. После того, как трехслойная нейронная сеть настроила свои весовые коэффициенты и может правильно относить новые внешние стимулы к одному из двух классов, в командное окно выводятся результаты того, как сеть классифицировала новый входной вектор P, соответствующий согласно условию задачи: новый сигнал, который хотим классифицировать: pnov = -0.2000 0.1000 -0.3000

0.3500 0 -0.1000.

Класс, куда отнесен новый входной сигнал: a = -0.2201 0.9998 0.0054.

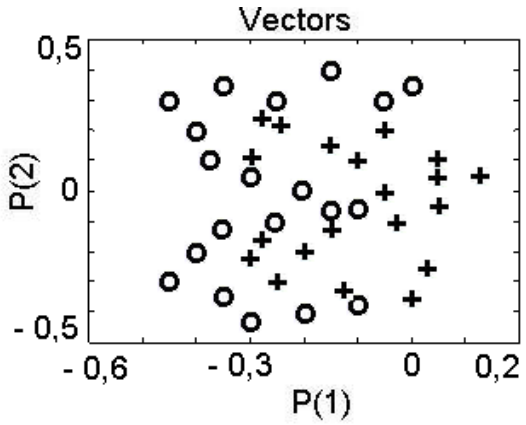


Рис. 4.6. **Расположение множеств обучения, соответствующих ситуации, когда сомнительно проведение классификации. Область, где расположились «нолики» имеет форму, напоминающую символ ∑, а «крестики» находятся в области подобной очертаниям знака 2**

Данный пример свидетельствует о том, что использование трех слоев позволяет получать области решений в виде многоугольников любой наперед заданной формы.

Сделаем небольшое отступление от темы, необходимое для лучшего понимания излагаемого ниже материала. Вы, наверно, видели песочные часы. Из воронки, имеющей небольшой диаметр отверстия, вниз сыплется мелкий песок, образуя внизу горку, по своей форме напоминающей колокол. Если мы, неважно пока каким образом, сделаем «срез» этой горки песка плоскостью, проходящей через вершину, то получим проекцию, график которой будет представлять некоторую монотонно изменяющуюся кривую колоколообразной формы, спадающей от центра к краям. Она отражает количество песка, попавшего на поверх-

ность (функцию накопленного песка) в зависимости от уклонения от точки, соответствующей центру воронки. Если процесс высыпания песка считать случайным процессом, то кривая, характеризующая очертания горки песка, будет называться функцией (плотностью) распределения вероятности (дифференциальной). В общем случае плотности распределения будут различаться положением места, где «сконцентрирована» масса вероятностей, степенью выпуклости или вогнутости горки и тем, как она «размазана» по области определения. Для характеристики этих названных свойств были введены количественные меры. Для описания положения центра распределения в теории вероятности используют такие понятия как математическое ожидание и мода. Они являются мерой положения центра. Наряду со «средними тенденциями» обращают внимание и на степень рассеивания данных. Стандартной мерой рассеивания является дисперсия. Но, поскольку эта мера является квадратичной величиной, а для сравнения часто нужна линейная мера, то используют среднее квадратичное отклонение или стандартное отклонение, за которое принято значение корня квадратного из дисперсии. От опыта к опыту форма эмпирических распределений обычно варьируется, но к определенным условиям существуют устойчивые по форме распределения, некие абстракции, которые теоретически описывают существующие на практике формы распределений. Аналитически их выражают определенными уравнениями и часто им придают статус теоретических законов. В начале XIX века знаменитый Фридрих Гаусс открыл закон нормального распределения случайных погрешностей, получивший его имя. Длительное время нормальное распределение Гаусса утверждалось в качестве «эталонного» распределения. Хотя, как выяснилось, нормальное распределение является скорее частным случаем, чем «общей» нормой распределения вероятностей случайных величин, большинство статистических методов оказалось «приспособленным» именно на применение к нормально распределенным случайным величинам. Например, известна одномерная механическая модель, реализующая функцию плотности нормального распределения. В ящик, со стороны задней стенки было набито много гвоздей, образующих по форме в совокупности треугольник. Сверху, в такой ящик сыпалась дробь, которая, ударяясь о гвозди и скатываясь вниз, образовывала на дне ящика колоколообразное тело, подобное кривой нормального распределения. Подробности о сказанном вы можете найти в учебниках по теории вероятности.

Вернемся к теме распознавания образов. Рассмотрим случай, когда в задаче классификации имеющиеся в пространстве входных сигналов «нолики и крестики» имеют распределение Гаусса. Чтобы лучше представить себе, о чем идет речь, предположим, что мы рядом разместили две воронки. Пусть из одной сыплется «красный» сахар (создаются нолики со своими координатами), а из другой – «синяя» соль (генерируются крестики). Ясно, что будут образованы две «горки» синяя и красная, из соли и сахара. При этом будет существовать красно-синяя область, где одновременно будет и «красный» сахар и «синяя» соль, то есть горки будут перекрываться. Для многослойной нейронной сети, выступающей в качестве классификатора, это означает, что ей в процессе обучения необходимо научиться разделять два перекрывающихся двумерных класса с гауссовым распределением, обозначенных цифрами 1 и 2. Понятно, что если трехмерные графики гауссовых распределений существенно перекрываются, то высока вероятность неправильной классификации. Теоретическое решение данной задачи в статистике известно как байесовский классификатор (Bayes classifier) или байесовская процедура проверки гипотез.

Если случайный вектор P принадлежит классам 1 и 2 и используется критерий отношения правдоподобия, то в теории говорят, что для байесовского классификатора вероятность правильной классификации (probability of correct classification) составляет 0,815. Эксперименты, проведенные с нейронными сетями, показали, что эффективность классификации для подобной ситуации для многослойной сети с двумя скрытыми нейронами достаточно близка способности классификации байесовского классификатора и составляет величину 0,801. Для «оптимальной» сети, когда каждое из обучающих множеств состояло из 1000 примеров, выбранных из классов C_1 и C_2 с одинаковой вероятностью, вероятность корректной классификации составила 0,797.

Таким образом, многослойная сеть обладает почти «идеальной» способностью решать задачу классификации множеств. При этом получаемое решение близко к оптимальному, полученному для вероятности корректной классификации двух перекрывающихся двумерных гауссовских распределений, представляемых классами C_1 и C_2. Вероятность корректной классификации, близкую к 0,797, нейронная сеть способна

добиваться в этом случае без предварительных знаний о предметной области, о законах распределения входных данных.

4.3. Использование нейронных сетей для автоматического распознавания буквенно-цифровых печатных символов

Способность скрытых нейронов многослойной сети выступать в роли детектора признаков и «выявлять» характерные черты данных обучения (вследствие нелинейного преобразования входных данных в новое пространство признаков, в котором классы легче отделить друг от друга, чем в исходном пространстве) широко используется для автоматического распознавания буквенно-цифровых печатных символов, считанных сканером. Известно, что технология считывания буквенно-цифровых изображений в упрощенном виде состоит в следующем. Относительно неподвижного оригинала, расположенного обычно на стеклянной подложке, перемещается каретка со светочувствительными элементами и фокусирующей оптической системой. Световой поток, отражаясь от поверхности непрозрачного листа с буквами, принимается матрицей фоторецепторов (ПЗС - матрицей), и затем переводится в форму электрических сигналов. Вы знаете, что использование фоточувствительных ПЗС - матриц (фоточувствительных матриц на основе приборов с переносов заряда – ФПЗС) буквально преобразило технологию считывания изображений. Совместимость технологии ФПЗС с базовыми структурами полупроводниковой микроэлектроники сделали ее экономичной и обеспечивающей массовый выпуск недорогих сканеров. Современные компьютерные средства, применяющие сканеры с сенсорами на ПЗС, «стараются» распознавать широкую номенклатуру объектов автоматического считывания. В настоящее время имеется возможность делать это не только для типографских символов, нанесенных на бумагу высокого качества, но и для символов, напечатанных на «плохой» бумаге печатными машинками, различного рода принтерами. Символы построчно печатающих устройств, как известно, подвержены значительно большей изменчивости, чем полиграфические буквы или цифры.

Изображения буквенно-цифровых символов образуют пространственное распределение энергетической освещенности на плоскости. Математически это распределение можно представить непрерывной функцией двух пространственных переменных $E(x, y)$. Обычно такое изо-

бражение представляют в виде двумерного массива точек. Точка на плоскости (или, как говорят на 2–D сетке) называется пикселем. Пиксель представляет энергетическую освещенность в соответствующем месте сетки. Положение пикселя задается с помощью общепринятого обозначения для матриц: первый индекс обозначает положение в строке; второй – в столбце. В простейшем случае изображение может состоять из набора отдельных пикселей, которые имеют значения либо единица, либо ноль. Вообще говоря, каждый пиксель представляет не просто точку на изображении, а скорее прямоугольную область, элементарную ячейку сетки со средней энергетической освещенностью. Поэтому следует иметь в виду, что при больших размерах пикселей имеет место плохое пространственное разрешение, а также возникают искажения, отвлекающее наше внимание от содержания изображения. При достаточно малых размерах возникает ощущение пространственно непрерывного изображения. В этой связи будем полагать, что для решения конкретных задач в сканерах используют матрицы датчиков имеющих такое количество элементов, чтобы обеспечить достаточно высокое пространственное разрешение при сканировании символов.

Предположим, что поставлена задача создать нейронную сеть, которая бы могла классифицировать 26 символов латинского алфавита, заданных набором их черно-белых пиксельных изображений. В качестве источника информации для системы распознавания предполагается использовать прибор с зарядовой связью, который выполняет считывание изображения каждого символа, находящегося в поле зрения. Будем при этом считать, что в результате считывания изображения с помощью ФПЗС для каждого символа латинского алфавита на прямоугольной сетке размером 5×7 пикселей определяется буквоподобный «электрический» образ. Для дальнейшего распознавания образ представлялся матрицей размера 5×7, у которой отдельные пиксели, в зависимости от вида символа имеют значения либо единица, либо ноль (предполагается, что белому цвету оригинала соответствует значение 0, а черному – 1). В целом, для нейронной сети входные значения определяются 26 векторами входа, каждый из которых содержит 35 элементов. Этот массив данных часто называют алфавитом. Проектируемая нейронная сеть должна, после прохождения процесса обучения, с максимальной точностью уметь распознавать даже изображения плохого качества («зашумленные» или, по-другому, на которые одновременно попали и «прилип-

ли» крупинки разного размера белой соли и черного перца). Данная задача классификации символов является частным случаем схемы кодирования «один из М» (one – from – m coding scheme). Это означает, что желаемый отклик сети (составленная из нулей и единиц вектор-строка, состоящая из М элементов) будет иметь *к* - й элемент равным единице только тогда, когда входной сигнал принадлежит *к* - му классу (в противном случае элемент вектор-строки будет равен нулю). В нашем случае имеется 26 классов (по числу символов в алфавите), поэтому в вектор-строке, указывающей принадлежность символа одному из 26 классов, только один элемент из 26 будет равен 1, а остальные – 0. К примеру, для латинской буквы В единице должен быть равен второй элемент, а остальные – нулю. Введем в командное окно программу, демонстрирующую классифицируемые символы и способы их представления.

```
clear,
[alphabet, targets] = prprob;
% Вводим номер символа, который хотим отобразить (от 1 до 26)
i = 5;
ti = alphabet(:, i);
letter{i} = reshape(ti, 5, 7)';
figure(1),clf
plotchar(ti );
disp('Цифровое изобр. символа в виде ')
disp('двумерного массива точек (пикселов) на прямоугольной сетке')
letter{i}
td = targets(:, i);
targetletter{i} = reshape(td, 26, 1)';
disp('Представление вектора классов, у которого позиция 1 в строке означает ')
disp('класс вводимого символа (число,')
disp(' соответствующее номеру симвла)')
targetletter{i}
```

После запуска программы для символа А получаем:

1) Цифровое представление символа в виде двумерного массива точек (пикселов) на прямоугольной сетке:

```
0  0  1  0  0
0  1  0  1  0
0  1  0  1  0
1  0  0  0  1
1  1  1  1  1
1  0  0  0  1
1  0  0  0  1
```

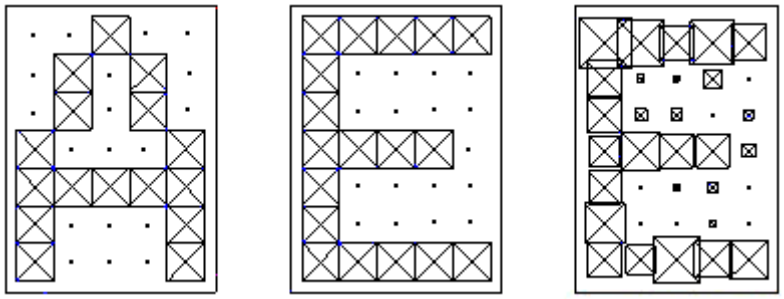
2) Прямоугольное изображение (рис. 4.7а), которое посредством функция plotchar интерпретирует матрицу. Каждый элемент матрицы представляется в виде квадратика, размер которого соответствует величине элемента матрицы. 3) Представление вектора классов, у которого пози-

ция 1 в строке означает класс вводимого символа (число, соответствующее номеру символа): 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0.

Цифровое представление символа E в виде двумерного массива точек (пикселов) на прямоугольной сетке будет иметь вид:

1	1	1	1	1
1	0	0	0	0
1	0	0	0	0
1	1	1	1	0
1	0	0	0	0
1	0	0	0	0
1	1	1	1	1

При представлении матрицы в виде квадратиков получаем (рис. 4.7б).



а) б) в)

Рис. 4.7. **Изображение символов A (а) и E (б) и «зашумленного» символа E (в) в виде квадратиков. Область, в которой элементы матрицы цифрового изображения равны нулю, отмечены точкой**

В этом случае представление вектора классов, будет иметь вид:
0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0.

Если в командное окно ввести нижеследующую программу:

```
clear,
[alphabet, targets] = prprob;
% Вводим номер символа, который хотим отобразить (от 1 до 26)
i = 5;
ti = alphabet(:, i) + randn(35,1)*0.2;
letter{i} = reshape(ti, 5, 7)';
figure(1),clf
plotchar(ti );
disp('Цифровое изображение символа в виде двумерного массива точек ')
disp(' на прямоугольной сетке ')
letter{i}
td = targets(:, i);
targetletter{i} = reshape(td, 26, 1)';
disp(' Представление вектора классов, у которого позиция 1 в строке означает ')
disp(' класс вводимого символа (число, ')
disp('соответствующее номеру символа)')
targetletter{i}
```

то в результате ее выполнения получим преднамеренно искаженный, за счет шумов для случайно выбираемых пикселей («зашумленный»), символ E в виде матрицы и изображения квадратиками (рис. 4.7в).

1.4269	1.2845	0.9454	1.2064	0.9782
0.9937	-0.2187	-0.1457	0.5139	0.0643
0.9355	0.3233	0.3241	-0.0701	-0.2658
0.8533	1.0517	0.9312	0.9092	0.3564
0.9007	0.0554	-0.1548	-0.2700	0.0358
1.1304	-0.0389	-0.0666	0.1892	-0.0741
0.9471	0.8015	1.2923	1.0016	1.0610

Чтобы научить нейронную сеть решению задачи классификации латинских букв последовательно проделаем следующее. Сначала определим весь набор цифровых изображений идеализированных символов алфавита «alphabet» (размера 35×26) и массив целевых векторов «targets». Затем создадим классическую двухслойную нейронную сеть с обратным распространением ошибок (с помощью команды newff). Сеть первоначально обучим работать в отсутствии шума, задавая при этом в качестве критерия остановки обучения максимальное число циклов обучения – 500, либо вариант достижения допустимой средней квадратичной погрешности, равной 0.1.Введем в командное окно:

```
clear,
% Задаем весь набор символов алфавита
% цифровых изображений и массив целевых векторов
[alphabet,targets] = prprob;
[R,Q] = size(alphabet);
[S2,Q] = size(targets);
S1 = 10;
% Создаем двухслойную нейроннуюсеть
net=newff(minmax(alphabet),[S1 S2],...
{'logsig' 'logsig'}, 'traingdx');
net.LW{2,1} = net.LW{2,1}*0.01;
net.b{2} = net.b{2}*0.01;
% Определяем символы и соответствующие им классы, характеризуемые
% вектор строкой с одной единичкой
P = alphabet;
T = targets;
ные параметры сети, определяющие
%  процесс обучения.
% Используемый алгоритм оценки качества обучения сети
% (функционал качества обучения сети)
net.performFcn = 'sse';
% Используемое число эпох
net.trainParam.epochs = 500;
% Количество итераций (эпох), по истечении которых будут
% представляться (выводиться) промежуточные результаты
net.trainParam.show = 20;
% Значение целевой ошибки обучения
net.trainParam.goal = 0.1;
% Коэффициент инерции
net.trainParam.mc = 0.95;
% Проводим обучение сети
```

```
% Определяем объектно-ориентирован-      [net,tr] = train(net,P,T);
```

Чтобы нейронная сеть могла распознавать символы при воздействии на изображения «белой соли и черного перца» и не была при этом чувствительной к воздействию шума, протестируем ее на образцах входных данных, использующих две идеальные и две «зашумленные» копии изображений символов. Целевые выходные векторы будут также состоять из четырех копий векторов классов. Преднамеренно искаженные случайными «добавками» векторы пусть имеют шум со средним значением 0.1 и 0.2. Такой подход к обучению позволит сети приобрести свойство обобщения входных данных, то есть правильно распознавать зашумленные символы и хорошо распознавать «идеальные» изображения символов. При обучении с шумом максимальное число циклов обучения сократим до 300, а допустимую погрешность увеличим до 0.6.

```
% Создаем для тестирования изображе-
ний с шумом и без него новую сеть
netn = net;
% Определяем ее объектно-
ориентированные параметры
netn.trainParam.goal = 0.6;
netn.trainParam.epochs = 300;
% Создаем четыре набора  классов,
% характеризуемых вектор строкой с
одной единичкой
T = [targets targets targets targets];
% Определяем четыре набора обучаю-
щих символов (двух идеальных и двух
% зашумленных копий векторов алфа-
вита) и осуществляем обучение за 10
% проходов
for pass = 1:10
alphabetnois1 =alphabet + randn(R,Q)*0.1;
alphabetnois2 =alphabet + randn(R,Q)*0.2;
P = [alphabet, alphabet, ...
(alphabetnois1), ...
(alphabetnois2)];
%  Проводим обучение сети на образцах
символов
[netn,tr] = train(netn,P,T);
disp(' Матрицы цифровых изображений
символов с шумом, посредством ')
disp(' которых осуществлялось  ')
disp(' дополнительное обучение сети')
% Вводим номер символа, который хо-
тим отобразить (от 1 до 26)
i = 5;
ti1 = alphabetnois1(:, i);
letter1{i} = reshape(ti1, 5, 7)';
letter1{i}
ti2 = alphabetnois2(:, i);
letter2{i} = reshape(ti2, 5, 7)';
letter2{i}
end
```

Для того, чтобы гарантировать, что идеальные цифровые изображения символов классифицируются по-прежнему правильно, имеет смысл снова повторить обучение нейронной сети без шума. После повторного обучения уже можно оценить эффективность нейронной сети

при решении задачи распознавания символов, представленных в виде цифровых изображений. С этой целью поступим следующим образом. Изучим работу двух нейронных сетей: первой сети – обученной на идеальных последовательностях, и второй сети – прошедшей обучение на зашумленных последовательностях. Проверку функционирования выполним на 100 образцах цифровых изображений символов при различных уровнях шума. При этом анализ эффективности проведем для ряда случаев, когда к элементам исходных матриц изображений с шагом 0.05 добавляется «шум», который первоначально имел среднее значение равное 0 и стандартное отклонением 0…0.5. Для каждого уровня шума и каждого символа сформируем 100 зашумленных цифровых изображений символов и вычислим выход сети. Выходной сигнал обработаем специальной М-функцией compet (функцией конкуренции), которая формирует матрицу с единичными элементами, индексы которых соответствуют индексам наибольших элементов каждого столбца. Делается это с той целью, чтобы только один из 26 элементов вектора выхода был классифицирован как верный. После этого выполним оценку количества ошибочных классификаций, и вычислим процент ошибки. Для наглядности процесса оценивания эффективности нейронной сети при решении задачи распознавания символов, представленных в виде цифровых изображений, в конце концов, построим график погрешности сети от уровня входного шума. Вводим в командное окно:

```
tic,
noise_range = 0:.05:.5; max_test
= 20;
network1 = []; network2 = []; T =
targets;
 for noiselevel = noise_range
   errors1=0; errors2=0;
  for i=1:max_test
   P = alphabet +
randn(35,26)*noiselevel;
   A = sim(net,P);   AA = com-
pet(A);
   errors1 = errors1 +
sum(sum(abs(AA - T)))/2;
   An = sim(netn,P);  AAn =
compet(An);
errors2 = errors2 +
sum(sum(abs(AAn - T)))/2; echo
off
  end
  network1 = [network1 er-
rors1/26/100];
  network2 = [network2 er-
rors2/26/100];
end
toc
figure(4),clf
plot(noise_range,network1*100,'-
-',noise_range,network2*100);
grid on
```

После запуска программы получаем график (рис. 4.8).

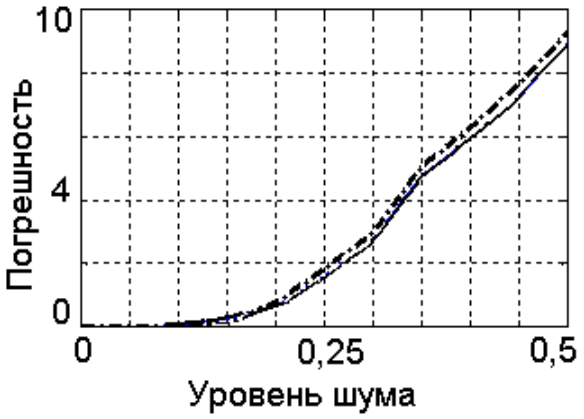


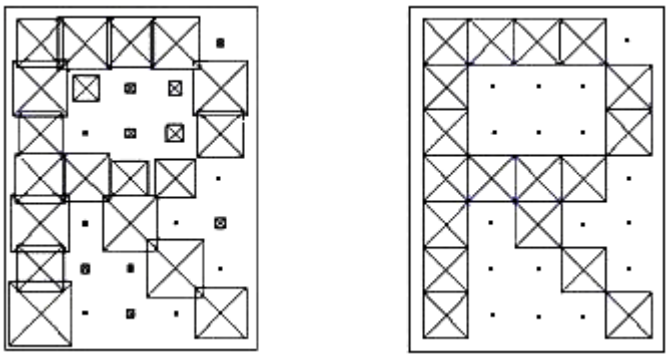
Рис. 4.8. **График зависимости погрешности сети от уровня входного шума**

В заключение, проверим, как работает обученная нейронная сети при распознавании конкретных преднамеренно искаженных символов. Сформируем зашумленный вектор входа для символа R. Вводим в командное окно следующую программу:

```
% Номер зашумленного символа
% который надо распознать
disp(' Надо распознать цифровое изо-
бражение символа с номером ')
nn=18
noisysim = alphabet(:,nn) +
randn(35,1)*0.2;
figure(6),clf
plotchar(noisysim);
A2 = sim(net,noisysim); A2 = compet(A2);
disp(' Номер символа цифровое изобра-
жения, который был  распознан сетью ')
answer = find(compet(A2) == 1)
figure(7),clf
plotchar(alphabet(:,answer))
```

После ее запуска получаем следующее.
Надо распознать цифровое изображение символа с номером nn = 18.
Номер символа цифрового изображения, который был распознан сетью answer = 18.

Это свидетельствует о том, что нейронная сеть без ошибок классифицировала зашумленный вектор (рис. 4.9а), верно указав, что это есть цифровое изображение символа R (рис. 4.9б). Аналогичные оценки эффективности распознавания символов можно выполнить и для других букв латинского алфавита.



а) б

Рис. 4.9. **Изображение «зашумленного» (а) и «не зашумленного» (б) символа R в виде квадратиков**

Таким образом, уже при наличии двух скрытых слоев нейронная сеть успешно обучается и приобретает способность правильно обобщать данные. При относительно небольшом уровне шума (малом слое соли и перца) нейронная сеть успешно классифицирует буквы (число неудачных попыток классификации изображений символов невелико).

4.4. Классификация образов с помощью RBF-сети

Многослойные нейронные сети, представленные в предыдущих разделах, выполняют преобразования множества входных данных $P \in E^{(R)}$ в выходные $Y \in E^{(M)}$ объединенными усилиями многих нейронов. По этой причине алгоритм обратного распространения, реализующий стохастическую аппроксимацию (stochastic approximation) или классификацию множеств (разделимость образов семейством поверхностей), можно назвать «глобальным».

Построение нейронной сети может выполняться и в рамках «локального» подхода. В этом случае отображение входного множества в выходное осуществляется за счет неких «локальных преобразований», выполняемых путем «подгонки» нейронов к ожидаемым значениям. В соответствии с таким подходом вместо глобальной аппроксимации кривой нейронная сеть решает задачу приближения функции «по точкам». При задаче классификации «локальное» обучение эквивалентно нахождению такой разделяющей гиперповерхности в многомерном пространстве, которая бы наиболее точно соответствовала отдельным данным обучения. При этом множество функций «локального» типа реализуется

скрытыми нейронами сети. Обычно нейроны реализуют колоколообразные функции с компактным носителем, изменяющиеся вокруг некоторого «центра» *c* и принимающие ненулевые значения только в окрестности этого центра. Подобные функции, определяемые в виде $F(x) = F(\|x - c\|)$, называют радиальными базисными функциями (radial-basis function). Роль нейрона в «локальных» сетях с радиальными базисными функциями заключается в отображении радиального (от лат radius - расположенного по радиусу) пространства вокруг одиночной заданной точки (центра) либо вокруг группы таких точек. Суперпозиция (взвешенное суммирование) всех сигналов, поступающих от скрытых нейронов, которая выполняется выходным нейроном, позволяет получить отображение всего многомерного пространства. Нейронные сети, построенные на основе радиальных базисных функций (radial-basis function network), называют сокращенно RBF. Теоретическую основу функционирования RBF-сети, при решении задач классификации, составляет теорема Ковера о разделимости образов (cover's theorem on the separability of patterns), которая утверждает следующее. Нелинейное преобразование сложной задачи классификации образов в пространство более высокой размерности повышает вероятность разделимости образов. Или то же самое, но другими словами: нелинейные проекции образов в некоторое многомерное пространство могут быть линейно разделены с большей вероятностью, чем при их проекции в пространство с меньшей размерностью. То есть, доказано, что каждое множество образов, случайным образом размещенных в многомерном пространстве, является φ- разделяемым с вероятностью 1 при условии большой размерности этого пространства. Здесь $\varphi(x)$ обозначает вектор радиальных функций в N-мерном входном пространстве. Для практики это означает, что применение достаточно большого количества скрытых нейронов, реализующих радиальные функции $\varphi(x)$, гарантирует решение задачи классификации при построении всего лишь двухслойной сети. При этом скрытый слой, состоящий из $m1$ нейронов, должен реализовывать вектор радиальных функций $\varphi(x) = [\varphi_1(x), \quad \varphi_2(x), \quad ..., \quad \varphi_{m1}(x)]^T$. А выходной слой должен состоять из единственного линейного нейрона, который выполняет суммирование выходных сигналов от скрытых нейронов с весовыми коэффициентами, заданными вектором w.

Рассмотрим подробнее понятие разделимости образов в RBF-сети. Пусть P – множество из N образов (векторов) $p_1, p_2, ..., p_N$, каждый из которых принадлежит одному из классов – P_1 или P_2. Эта дихотомия (разбиение на два класса) точек называется разделимой по отношению к семейству гиперповерхностей, если в семействе существует поверхность, которая отделяет точки класса P_1 от точек класса P_2. Для каждого образа $p \in P$ определим вектор, состоящий из множества действительнозначных функций $\{\varphi_i(p)\}$, вида $\varphi(p) = [\varphi_1(p), \quad \varphi_2(p), \quad ..., \quad \varphi_{m1}(p)]^T$.

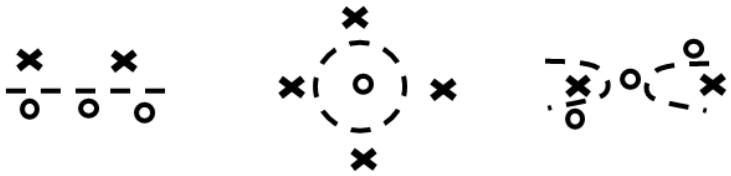
Предположим, что образ p является вектором в m_0- мерном входном пространстве. Тогда векторная функция $\varphi(p)$ отображает точки m_0-мерного пространства в новое пространство размерности m_1. Поскольку функции $\varphi(p)$ играют роль скрытых нейронов в сетях прямого распространения, то их часто называют скрытыми, а все пространство, образованное скрытыми функциями – скрытым пространством (hidden space) или пространством признаков (feature space).

Дихотомия $\{P_1, \quad P_2\}$ множества P называется $\varphi(p)$- разделимой, если существует m_1 -мерный вектор весов w, для которого можно записать: $\begin{matrix} w^T\varphi(p) > 0, & p \in P_1, \\ w^T\varphi(p) < 0, & p \in P_2. \end{matrix}$ Гиперплоскость, задаваемая уравнением $w^T\varphi(p) = 0$, описывает поверхность в φ- пространстве (то есть в скрытом пространстве). Обратный образ этой поверхности определяет разделяющую поверхность во входном пространстве.

Теоретические исследования показывают, что в качестве разделяющих поверхностей, в зависимости от вида функции $\varphi(p)$, могут выступать гиперплоскости (hyperplane), квадрики (quadrice) и гиперсферы (hypersphere). Для конфигурации из пяти точек в двумерном пространстве показаны примеры разделяющих поверхностей (рис. 4.10).

Сети радиального типа, когда нейроны линейно разделяют пространство на две категории, как известно, не всегда решают стоящие на практике задачи. По этой причине целесообразно использовать функции $\varphi(p)$, когда осуществляется сферическое разделение пространства или реализуется квадратично - разделимая дихотомия. В этом случае необходимо, чтобы функция $\varphi(p)$ имела квадратичный вид. При этом целе-

сообразно, чтобы аргумент функции активации каждого скрытого нейрона представлял собой Евклидову норму (расстояние) между входным вектором и центром радиальной базисной функции.



а) б) в)

Рис. 4.10. **Три примера φ- разделимых дихотомий для множеств, состоящих из пяти точек в двумерном пространстве: а) линейно-разделимая дихотомия; б) сферически разделимая дихотомия; в) квадратично-разделимая дихотомия**

Наиболее часто в качестве желательной радиальной базисной функции выбирают гауссову функцию с центром, размещенным в точке c $\varphi(p) = \exp\left\{-\frac{(p-c)^2}{2\sigma^2}\right\}$. В этом выражении σ – параметр, от значения которого зависит ширина функции. Параметр σ, а его также называют сглаживающим фактором, отклонением, определяет насколько «острой» будет гауссова функция. Если эти функции выбраны слишком острыми, сеть не будет «накрывать» входные данные (интерполировать данные между известными точками) и потеряет способность к обобщению. Если же гауссовы функции взяты чересчур широкими, сеть не будет «замечать» мелкие, но важные детали. Как правило, σ-параметр выбирается таким образом, чтобы «колпак» каждой гауссовой функции захватывал «несколько» соседних центров.

Чтобы обучить гауссовы RBF-сети решать задачу классификации образов надо располагать информацией о трех типах параметров: 1) о значениях весов скрытого и выходного слоя w_{ij}; 2) о «центрах» c_i радиальных базисных функций $\varphi_i(p)$; 3) об отклонениях (радиусах базисных функций) σ_{ij}. Следовательно, возможными вариантами обучения RBF-сети являются следующие: а) задаются по каким-то правилам центры и отклонения для радиальных функций, а, затем, вычисляются веса (оп-

тимизируются параметры линейного выходного слоя); определяются по каким-то правилам центры и отклонения, а для коррекции весов используется обучение с учителем. При этом будет достаточно удобным, если центры базисных функций будут на основе «индикатора близости» «привязаны» к существующим объединениям «точек» входного пространства, а ширина базисной функции будет примерно одинаковой. Обычно расположение центров соответствует объединенным группам данных (кластерам), реально присутствующим в исходных данных. В этом случае потребуется лишь корректировать веса, поскольку ни центры, ни ширина радиальной базисной функции меняться в процессе обучения сети не будут. Такой подход обеспечит хорошее «покрытие» всех точек входного пространства, делая сеть нечувствительной к выбору центров. В случае «случайного» выбора центров радиальных базисных функций (при использовании изотропных функций Гаусса), при малом их числе «представление» исходных данных может оказаться неудовлетворительным. При обучении с учителем выбор центров радиальных базисных функций может проводиться на основе процедуры градиентного спуска по поверхности ошибок.

Представление RBF-сети будет не полным без упоминания того, что сеть «идеально» работает лишь в случае неограниченного объема входных данных. На практике не существует неограниченной выборки данных об образах. В этой связи нахождение весов (решение уравнения) при больших размерностях вектора **P** становится затруднительным. К этому следует добавить, что в некоторых классах задач, если количество точек данных обучающего множества значительно превышает количество степеней свободы самого физического процесса (говорят, что задача оказывается избыточно определенной), может возникнуть проблема, которая в прикладной математике известна как «плохая обусловленность» (ill –posed).

Чтобы избежать подобных неприятностей используют регуляризацию (подбирают архитектуру сети с меньшим количеством нейронов в скрытом слое) и уменьшают количество используемых базисных функций. При этом отыскивается решение в пространстве меньшей размерности, которое с достаточной точностью приближается к точному решению. Процесс обучения в этом случае происходит с помощью так называемой итерационной процедуры, когда на каждом ее шаге добавляется

по одному нейрону в скрытый слой до тех пор, пока сумма квадратов ошибок не станет меньше наперед заданного значения.

Отметим, что по сравнению с многослойными сетями сеть RBF использует всего один скрытый слой, что избавляет нас от ряда вопросов, касающихся количества скрытых слоев. Кроме того, параметры линейной комбинации в выходном слое можно рассчитать, используя методы линейной оптимизации. При этом сеть RBF обучается быстро и не испытывает трудностей с локальными минимумами, так мешающими процессу обучения в алгоритме обратного распространения.

Рассмотрим несколько примеров, демонстрирующих, как нейронная сеть RBF осуществляет классификацию образов. Предположим снова, что мы хотим осуществить классификацию линейно-неразделимых входных данных, которые представлены на рис. 2.37. В этом случае некоторое число «нуликов и крестиков» попадает с чужеродные области и их нельзя разделить прямыми. Введем в командное окно программу:

```
clear
% Опр-ем входной и тестовый сигнал
disp('Входной сигнал для обучениясети')
P = [-0.05 -0.3 -0.1 0.0 0.1 0.2 0.35 0.3; ...
-0.1 0.1 0.05 0.3 0.15 -0.15 0.35 0.1]
disp(' Желательный отклик (класс)')
test = [1 1 0 0 1 1 0 0]
% Определяем обучающие пары показателей
P1 = [-0.05 -0.1; -0.3 0.1; -0.1 0.05; 0 0.3;
.0.1 0.15; 0.2 -0.15; 0.35 0.35; 0.3 0.1]';
% Задаем вектор индексов классов
Tc = [1 1 2 2 1 1 2 2];
disp(' Матрица связности, которая показывает к какому классу (первая цифра')
disp('принадлежит вектор (вторая цифра)')
Tind=ind2vec(Tc)
disp(' Полная матрица связности, в которой показаны' к какому классу)
disp(' (строка матрицы)' принадлежит)
disp(' вектор (столбец матрицы)')
Tind1=full(Tind)
axis([-0.4 0.6 -0.21 0.4])
title(' Vectors and  classes')
xlabel('P1(1,:)')
T = test;
% Определяем расположение точек на плоскости
figure(2);
plotpv(P,T);
axis([-0.4 0.6 -0.21 0.4])
title(' vectors ')
% Параметр обучения σ (отклонения)
SPREAD = 1;
netbm1 = newrb(P,T,GOAL,SPREAD);
% Число нейронов в скрытом слое
disp(' Число нейронов в скрытом слое ')
netbm1.layers{1}.size
% Рассчитываем отклик сети
% на входной сигнал
disp('Полученный отклик сети на входной сигнал')
y1 = sim(netbm1,P)
EN=test-y1;
disp(  ' Среднеквадратическая ошибка в
```

```
plot(P1(1,:),P1(2,:),'.','markersize',30)
for i=1:8,
text(P1(1,i)+0.01,P1(2,i)+0.01,sprintf('class %g',Tc(i))),
end
%Создаем радиальную базисную сеть
% Устанавливаем значение среднеквадратической ошибки обучения
GOAL = 0.01;
итоге работы нейронной сети')
err1 =mse(EN)
% Вводим параметр, который хотим классифицировать
disp(' Новый сигнал, который ')
disp(' хотим классифицировать')
pnov = [-0.05 0.35;-0.1 0.35]
disp('Классы, куда отнесены новые входные сигналы')
a = sim(netbm1,pnov)
```

После запуска программы получаем.

Входной сигнал для обучения сети: P =
-0.0500 -0.3000 -0.1000 0 0.1000 0.2000 0.3500 0.3000
-0.1000 0.1000 0.0500 0.3 0.1500 -0.1500 0.3500 0.1000.

Желательный отклик (класс): test = 1 1 0 0 1 1 0 0.

Количество нейронов и ошибка в начале процесса обучения:
NEWRB, neurons = 0, SSE = 1.41072.

Число нейронов в скрытом слое (после обучения): ans = 7.

Полученный отклик сети на входной сигнал:
y1 = 1.0000 1.0000 -0.0000 -0.0000 1.0000 1.0000 0 -0.0000.

Среднеквадратическая ошибка в итоге работы нейронной сети:
err1 = 4.1359e-025.

Новый сигнал, который хотим классифицировать:
pnov = -0.0500 0.3500
-0.1000 0.3500.

Классы, куда отнесены новые входные сигналы: a = 1 0.

Результаты примера свидетельствуют о том, в процессе обучения нейронная сеть RBF очень точно настроила свои весовые коэффициенты что позволило ей получить почти «идеальный» отклик на выходе сети. Сеть успешно классифицировала новый входной вектор P соответствующий условию задачи.

Рассмотрим случай, подобный тому, когда мы осуществляли классификацию линейно-неразделимых входных данных, при которых множества «нуликов» и «крестиков» образуют области сложной формы (рис. 4.6). Ранее для классификации образов мы применяли трехслойную сеть; теперь используем сеть RBF. Вводим в командное окно:

```
clear
% Опред-ем входной  и тестовый сигнал
disp('Входной сигнал для обучения
сети') P = [0 -0.05 -0.15 -0.25 -0.35 -0.45 -
0.4 -0.375 -0.3 -0.2 ...
-0.1 -0.15 -0.25 -0.35 -0.4  -0.45 -0.35 -0.2
-0.1 -0.3 ...
-0.275 -0.25 -0.3 -0.15 -0.1 -0.05 0.05
0.125 0.05 -0.05 ...
-0.025 -0.15 -0.275 -0.3 -0.2 -0.25 -0.125
0.025 0 0.05; ...
0.35 0.3 0.4 0.3 0.35 0.3 0.2 0.1 0.05 0 ...
-0.05 -0.075 -0.1 -0.125 -0.2 -0.3 -0.35 -0.4
-0.375 -0.45 ...
0.25 0.225 0.1 0.15 0.1 0.2 0.1 0.05 -0.05 0
-0.1 -0.125 -0.15 -0.225 -0.2 -0.3 -0.325 -
0.25 -0.35 0.05]
disp(' Желательный отклик')
test = [0 0 0 0 0 0 0 0 0 0 ...
      0 0 0 0 0 0 0 0 0 0 ...
1 1 1 1 1 1 1 1 1 1 ...
1 1 1 1 1 1 1 1 1 1]
% Определяем расположение точек на
плоскости
figure(2);
plotpv(P,test);
axis([-0.6 0.2 -0.5 0.5])
title(' vectors ')
T=test;
%Создаем радиальную базисную сеть
% Устанавливаем значение среднеквад-
ратической ошибки обучения
GOAL = 0.01;
% Параметр обучения (отклонения)
SPREAD = 0.6;
netbm1 = newrb(P,T,GOAL,SPREAD);
% Число нейронов в скрытом слое
disp(' Число нейронов в скрытом слое ')
netbm1.layers{1}.size
% Рассчитываем отклик сети
%  на входной  сигнал
disp('Полученный отклик сети на вход-
ной сигнал')
y1 = sim(netbm1,P)
EN=test-y1;
disp(' Среднеквадратическая ошибка в
итоге работы нейронной сети ')
err1 =mse(EN)
% Вводим параметр, который хотим
классифицировать
disp(' Новый сигнал, который ')
disp(' хотим классифицировать')
pnov = [0.005 -0.275;0.35 0.25]
disp('Классы, куда отнесены новые вход-
ные сигналы')
a = sim(netbm1,pnov)
```

После запуска программы получаем:

Входной сигнал для обучения сети: P =

0	-0.0500	-0.1500	-0.2500	-0.3500	-0.4500
0.3500	0.3000	0.4000	0.3000	0.3500	0.3000
-0.4000	-0.3750	-0.3000	-0.2000	-0.1000	-0.1500
0.2000	0.1000	0.0500	0	-0.0500	-0.0750
-0.2500	-0.3500	-0.4000	-0.4500	-0.3500	-0.2000
-0.1000	-0.1250	-0.2000	-0.3000	-0.3500	-0.4000
-0.1000	-0.3000	-0.2750	-0.2500	-0.3000	-0.1500
-0.3750	-0.4500	0.2500	0.2250	0.1000	0.1500
-0.1000	-0.0500	0.0500	0.1250	0.0500	-0.0500

```
    0.1000    0.2000    0.1000    0.0500   -0.0500         0
   -0.0250   -0.1500   -0.2750   -0.3000   -0.2000   -0.2500
   -0.1000   -0.1250   -0.1500   -0.2250   -0.2000   -0.3000
   -0.1250    0.0250         0    0.0500
   -0.3250   -0.2500   -0.3500    0.0500.
  Желательный отклик: test =
    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0
    1    1    1    1    1    1    1    1    1    1
    1    1    1    1    1    1    1    1    1    1.
NEWRB, neurons = 0, SSE = 7.24247
NEWRB, neurons = 25, SSE = 0.952194
  Число нейронов в скрытом слое ans =    39
  Полученный отклик сети на входной  сигнал:y1 =
    0.0000    0.0000    0.0000    0.0000    0.0000   -0.0000
    0.0000    0.0000    0.0000   -0.0000    0.0000   -0.0000
    0.0000    0.0000   -0.0000    0.0000    0.0000   -0.0000
    0.0000    0.0000    1.0000    1.0000    1.0000    1.0000
    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000
    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000
    1.0000    1.0000    1.0000    1.0000.

  Среднеквадратическая ошибка в итоге  работы нейронной сети:
err1 =  3.6410e-010
  Новый сигнал, который хотим классифицировать:
pnov =    0.0050   -0.2750
          0.3500    0.2500.
  Классы, куда отнесены новые входные  сигналы: a =   -2.0140    1.0000.
```

Результаты примера свидетельствуют о том, что и на этот раз с помощью нейронной сети RBF удалось получить почти «идеальный» отклик на выходе сети, а затем, ей удалось успешно классифицировать новый входной вектор P соответствующий условию задачи. Введем в командное окно следующую программу:

```
clear
% Определяем входной  и тестовый
сигнал
GOAL = 0.01;
% Параметр отклонения
SPREAD = 1;
```

```
P = -1:.1:1;
T = [-.9602 -.5770 -.0729  .3771
.6405  .6600  .4609  .1336 ...
 -.2013 -.4344 -.5000 -.3930 -.1647
.0988  .3072  .3960 ...
  .3449  .1816  -.0312  -.2189  -.3201];
%Создаем радиальную базисную
сеть
% Устанавливаем значение средне-
квадратической ошибки обучения
net = newrb(P,T,GOAL,SPREAD);
disp('Число нейронов в скрытом слое ')
net.layers{1}.size
figure(1), clf,
plot(P,T,'sr','MarkerSize',8,'MarkerFaceColor','y')
hold on;
Pnov = -1:.01:1;
Y = sim(net,Pnov);
plot(Pnov,Y,'LineWidth',2), grid on
```

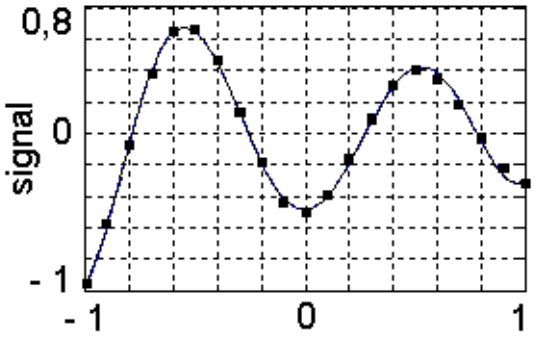


Рис. 4.11. **График аппроксимации кривой по точкам, полученный с помощью нейронной сети RBF**

Можно убедиться в том, что, используя 6 нейронов в скрытом слое, нейронная сеть RBF осуществляет хорошую аппроксимацию довольно сложной функции (рис. 4.11).

В литературе описано решение нейронной сетью RBF задачи дихотомической классификации данных, полученных на основе двух равновероятных перекрывающихся двумерных распределений Гаусса, соответствующих классам C_1 и C_2. Класс C_1 характеризовался вектором среднего значения [0, 0] и дисперсией равной единице; класс C_2 задавался вектором [0, 2] и дисперсией, равной 4. Пример подобен тому, который рассматривался в многослойной сети. В результате компьютерного эксперимента было получено, что, используя регуляризацию, вероятность корректной классификации находилась в диапазоне значений [0.7…0.78].

По сравнению с классификатором Байеса и оценками для многослойной нейронной сети, это несколько худший результат.

4.5. Использование машины опорных векторов для решения задач классификации образов

Подобно многослойным сетям, обучаемым по алгоритму обратного распространения ошибки, и нейронным сетям RBF, на основе радиальных базисных функций, для решения задач классификации образов можно использовать еще одну категорию сетей прямого распространения – так называемые машины опорных векторов (support vector machine – SVM). Машина опорных векторов, как, впрочем, и сеть RBF, базируется на следующих двух идеях: 1) для распознавания образов требуется осуществить нелинейное отображение входного вектора в пространство признаков (feature space) более высокой размерности; 2) для разделения признаков, полученных в п. 1, необходимо построить оптимальную гиперплоскость.

Идея 1 реализуется в соответствии с теоремой Ковера о разделимости образов. Если входное пространство состоит из нелинейно-разделимых образов (nonlinearly separated patterns), то, в соответствии с этой теоремой, такое многомерное пространство можно преобразовать в новое пространство признаков, в котором образы станут линейно разделимыми с высокой вероятностью. При этом необходимо, чтобы преобразование было нелинейным и размерность пространства признаков была большой. Размерность скрытого пространства признаков преднамеренно увеличивается таким образом, чтобы в этом пространстве можно было построить поверхность решений в виде гиперплоскости. На конструируемую гиперплоскость накладываются специальные ограничения, связанные с выбором подмножества примеров обучения. Как следует из названия, основная идея создания этой машины состоит в выборе некоторого подмножества обучающих примеров в качестве опорных. Опорные векторы представляют собой подмножество обучающих данных, отбираемых алгоритмом. Это подмножество представляет устойчивые свойства всей обучающей выборки. При этом алгоритм обучения минимизирует количество положительных и отрицательных обучающих примеров, которые попадают на границу разделения, что, в первом приближении, минимизирует вероятность ошибок классификации (а не минимизирует среднеквадратическую ошибку, как в многослойных нейронных сетях). По этой причине вторая идея (о построении оптимальной разделяющей гиперплоскости) в машине опорных векторов несколько отличается от

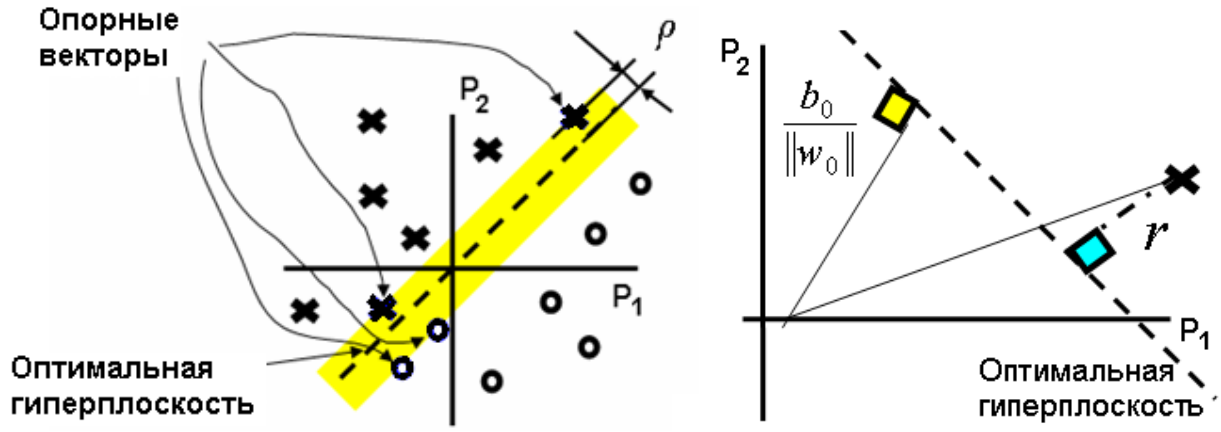
того, что было сказано по этому поводу ранее. Фундаментальным отличием является то, что разделяющая гиперплоскость определяется теперь как линейная функция от векторов, взятых из пространства признаков, а не из исходного входного пространства, как было ранее. В этой связи машина опорных векторов может обеспечить хорошее качество обобщения в задаче классификации, не обладая предварительными (априорными) знаниями о предметной области конкретной задачи. Именно это свойство является уникальным для машины опорных векторов. Теория SVM не требует эвристик, используемых при разработке нейронных сетей на основе многослойных персептронов и на основе радиальных базисных функций. Для машин опорных векторов размерность пространства признаков определяется числом опорных векторов, отобранных из обучающих данных. В связи с этим в машине опорных векторов за счет использования специальных приемов (использования «штрафной» гиперплоскости в скрытом пространстве вместо оптимальной гиперплоскости в пространстве данных) и некоторого увеличения объема вычислений (работа машины более медленная по сравнению с другими типами сетей) удается избежать «проклятия размерности», характерного при использовании пространств высокой размерности.

Поскольку SVM обучается на предложенном множестве данных с целью извлечения опорных векторов, то имеет смысл перейти к рассмотрению того, как можно получить опорные векторы и того, как построить оптимальную гиперплоскость для образов.

Рассмотрим множество обучающих примеров $\{(p_i, d_i\}_{i=1}^{N}$, где p_i – входной вектор для примера i; d_i – соответствующий ему желаемый отклик. Чтобы доступно объяснить идею, положенную в основу машин опорных векторов, предположим, что класс C_1, представленный подмножеством $d_i = 1$, и класс C_2, описанный подмножеством $d_i = 0$, линейно-разделимы. В этом случае уравнение поверхности решений в форме гиперплоскости, выполняющей это разделение, записывается следующим образом: $w^T p + b = 0$, где p – входной вектор; w – настраиваемый вектор весов; b – порог. Расстояние между ближайшей точкой из набора данных и гиперплоскостью, проведенной для вектора

весов и определяемой $w^T p + b = 0$, называется границей разделения (margin of separation) и обозначается символом ρ (рис. 2.44).

Основной целью машины опорных векторов является поиск конкретной гиперплоскости, для которой граница разделения будет максимальной. При этом условии поверхность решения называется оптимальной гиперплоскостью. На рис. 4.12а оптимальная гиперплоскость в двумерном пространстве входных сигналов показана штриховой линией.



а) б)

Рис. 4.12. Г**еометрическая интерпретация линейно-разделимых векторов, расположенных в двумерном пространстве в виде крестиков и ноликов, и опорных векторов (а); интерпретация алгебраических расстояний от точек входных данных до оптимальной гиперплоскости (б)**

Оптимальную гиперплоскость, представляющую поверхность решений в пространстве входных сигналов, можно записать уравнением $w_0{}^T p + b_0 = 0$, где w_0 и b_0 – оптимальные значения весов и порогов. Алгебраическую меру расстояния от точки p до оптимальной гиперплоскости определяет функция $g(p) = w_0{}^T p + b_0$. Если обозначить через r алгебраическое расстояние от точки p до оптимальной гиперплоскости, то $g(p) = w_0{}^T p + b_0 = r\|w_0\|$ или $r = g(p)/\|w_0\|$ (рис. 4.12б).

Если задано множество примеров обучения $\{(p_i, d_i\}_{i=1}^N$, то для него обязательно найдутся точки данных, которые ближе всего лежат к поверхности решений и, в этой связи, являются самыми сложными для

классификации. Они лучше всего «указывают» на размещение поверхности решений. Если мы возьмем две точки, принадлежащие к различным классам, то параметры w_0 и b_0 можно выбрать таким образом, чтобы удовлетворялись ограничения $\begin{matrix} {w_0}^T p_i + b_0 \geq 1 \quad для \quad d_i = 1, \\ {w_0}^T p + b_0 \leq -1 \quad для \quad d_i = 0. \end{matrix}$

Конкретные точки $\left\{(p^{(s)}{}_i, d^{(s)}{}_i\right\}$, для которых выполняются первое и второе ограничение со знаком равенства, называются опорными векторами. Алгебраическое расстояние от опорного вектора $p^{(s)}$ до оптимальной гиперплоскости равно $r = \dfrac{g(p^{(s)})}{\|w_0\|} = \begin{cases} \dfrac{1}{\|w_0\|}, & если \quad d^{(s)} = 1, \\ \dfrac{-1}{\|w_0\|}, & если \quad d^{(s)} = 0, \end{cases}$ где знак «плюс» означает расположение точки $p^{(s)}$ с «положительной» стороны от оптимальной гиперплоскости, а «минус» – с «отрицательной». При этом, если ρ – оптимальное значение границы разделения между классами, составляющими множество примеров, то $\rho = 2r = \dfrac{2}{\|w_0\|}$. Это соотношение означает, что максимизация границы разделения между классами эквивалентна минимизации Евклидовой нормы вектора w. При этом оптимальная гиперплоскость является единственной и оптимальный вектор весов w_0 обеспечивает максимально возможное разделение между положительными и отрицательными примерами.

Для поиска оптимальной гиперплоскости на обучающем множестве $\{(p_i, d_i\}_{i=1}^N$ требуется найти оптимальные значения вектора весовых коэффициентов w_0 и порога b_0. Если вместо w_0 использовать w, то разработка эффективной вычислительной процедуры для определения оптимальных значений вектора весовых коэффициентов с минимальной Евклидовой нормой сводится к нахождению вектора весов и порогов, удовлетворяющих следующим двум условиям: а) «формирования» разделяющей гиперплоскости с наибольшей границей разделения ρ $d_i(w^T p_i + b) \geq 1 \quad для \quad i = 1,2,...,N$; б) максимизации функции стоимо-

сти $Z(w) = w^T w$. В математике задачи такого рода решают с помощью метода множителей Лагранжа. Опустив достаточно сложные выкладки, можно отметить, что, если определить оптимальные множители Лагранжа $\alpha_{0,i}$, то для заданного множества примеров обучения $\{(p_i, d_i\}_{i=1}^N$ оптимальный вектор весовых коэффициентов и порога определяется соотношениями $w_0 = \sum_{i=1}^{N} \alpha_{0,i} d_i p_i$, $b_0 = 1 - w_0^T p^{(S)}$ *для* $d^{(S)} = 1$.

До сих пор, стараясь обеспечить понимание идеи машины опорных векторов, мы рассматривали линейно-разделимые образы. Теперь сконцентрируем наше внимание на более общем случае – неразделимых множеств. В этом случае, как вы уже наверно догадались, невозможно построить разделяющую гиперповерхность, полностью исключающую ошибки классификации. По этой причине «конструируют» оптимальную гиперплоскость, которая сводит к минимуму вероятность ошибок классификации на множестве обучающих примеров. Для этой цели вводится величина ξ_i, называемая фиктивной переменной (slack variable), которая определяет отклонение точек от идеального состояния линейной разделимости. Если $0 \le \xi_i \le 1$, то точка (крестик или нолик) попадает в область границы разделения с «нужной» стороны поверхности решений (на рис. 4.12б в полосу со стороны «своих» крестиков или ноликов). Если $\xi_i > 1$, то точка попадает в область границы разделения с «чужой» стороны разделяющей гиперплоскости. Граница разделения классов становится «мягкой» и допускает некоторые «нарушения» со «своей» и «чужой» стороны поверхности решений. В этом случае решается задача поиска «мягкой» разделяющей гиперплоскости, минимизирующей ошибку классификации на множестве обучающих примеров. Это можно обеспечить, если минимизировать сумму $Z(\xi) = \sum_{i=1}^{N} I(\xi_i - 1)$ неких функций индикатора, определяемых следующим образом $I(\xi) = \begin{cases} 0, & \xi \le 0, \\ 1, & \xi > 0. \end{cases}$

Для поиска оптимальной «мягкой» гиперплоскости на обучающем множестве $\{(p_i, d_i\}_{i=1}^N$ требуется найти оптимальные значения вектора весовых коэффициентов w и порога b_0, удовлетворяющие следующим

двум условиям: а) «формирования» «мягкой» разделяющей гиперплоскости, имеющей минимум ошибок классификации на множестве обучающих примеров $d_i(w^T p_i + b) \geq 1 - \xi_i$ для $i = 1, 2, ..., N$; б) максимизации функции стоимости $Z(w, \xi) = w^T w + C \sum_{i=1}^{N} \xi_i$. Последнее выражение оптимизируется по w и ξ_i, при условии $\xi_i \geq 0$. Если для линейно-разделимых образов максимизация границы разделения между классами была эквивалентна построению оптимальной гиперплоскости, обеспечивающей за счет опорных векторов максимально возможное разделение между положительными и отрицательными примерами (наибольшее ρ), то при решении задачи классификации неразделимых подмножеств требуется также свести к минимуму верхнюю границу ошибок тестирования. При этом задаваемый пользователем положительный параметр C обеспечивает компромисс между «сложностью» машины SVM (количеством опорных векторов) и количеством неразделимых точек ξ_i. Например, установив значения точек $\xi_i = 0$, получим случай линейно-разделимых множеств. И в этом случае, используя метод множителей Лагранжа для заданного множества примеров обучения $\{(p_i, d_i)\}_{i=1}^{N}$, можно найти оптимальный вектор весовых коэффициентов и порога.

Обычно, чтобы обеспечить машине способность решать задачу классификации множеств, близкую к оптимальной, как отмечалось, входное пространство преобразовывают в пространство признаков. Для этой цели используют множество нелинейных преобразований $\{\varphi_j(p)\}_{j=1}^{m1}$. Причем размерность пространства признаков $m1$ является более высокой по сравнению с размерностью входного пространства $m0$: $m1 >> m0$. В результате преобразований $\{\varphi_j(p)\}_{j=1}^{m1}$ при предъявлении вектора входного сигнала p в пространстве признаков «формируется» «образ». Этим образам, применительно к задаче поиска поверхности решений для линейно разделимых множеств, соответствует вектор весов $w = \sum_{i=1}^{N} \alpha_{0,i} d_i \varphi(p_i)$, который формирует поверхность решений в

пространстве признаков $\sum_{i=1}^{N}\alpha_{0,i}d_i\varphi^T(p_i)\varphi(p)=0\Rightarrow\sum_{i=1}^{N}\alpha_{0,i}d_iK(p_i,p)$, где $K(p_i,p)$- ядро скалярного произведения. Разложение ядра скалярного произведения $K(p_i,p)$ (представление его в виде ряда вида $K(p\ ,p')=\sum_{i=1}^{N}\lambda_i\varphi_i(p)\varphi_i(p')$ с положительными коэффициентами, называемыми собственными значениями – eigenvalue) позволяет построить поверхность решений, которая во входном пространстве является нелинейной, однако ее образ в пространстве признаков является линейным. Поэтому, если для данного множества обучающих примеров $\{(p_i,d_i\}_{i=1}^{N}$найти множитель Лагранжа $\{\alpha_i\}_{i=1}^{N}$, которые максимизируют функцию $Q(\alpha)=\sum_{i=1}^{N}\alpha_i-\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_jd_id_jK(p_i,p_j)$, при ограничениях 1) $\sum_{i=1}^{N}\alpha_id_i=0$, 2) $0\le\alpha_i\le C$, то тогда можно найти соответствующее оптимальное значение весов w_0, связывающее пространство признаков с выходным пространством $w_0=\sum_{i=1}^{N}\alpha_{0,i}d_i\varphi(p_i)$. В данном выражении первый компонент вектораw_0 представляет собой оптимальный порог. В зависимости от того, что используется в качестве функций $\{\varphi_j(p)\}_{j=1}^{m1}$ ядра скалярного произведения$K(p_i,p)$, различают машины опорных векторов полиномиальные $K(p,p_i)=(p^Tp_i+1)^\gamma$ и на основе радиальных базисных функций $K(p,p_i)=\exp(-1/2\sigma^2\|p-p_i\|^2$.

В обоих случаях степень γ и ширина σ^2 предварительно задаются пользователем при решении задачи. При этом в машинах опорных векторов на основе радиальных базисных сетей количество требуемых функций и расположение их центров определяется автоматически по числу опорных векторов.

Рассмотрим теперь некоторые результаты компьютерных экспериментов, получаемых с помощью двоичного классификатора с помощью машины опорных векторов на основе радиальных базисных сетей.

Пусть решается задача классификации 120 данных, выбранных из двух неразделимых подмножеств. Данные имеют распределение Гаусса. При этом у одного подмножества (ноликов) вектор среднего (центр группирования данных) равен (0,0) и он имеет дисперсию равную единице. Другое распределение (крестики) группируется вблизи точки (0,2) и характеризуется дисперсией, равной 4. Расположение обучающего множества показано на рис. 4.13а.

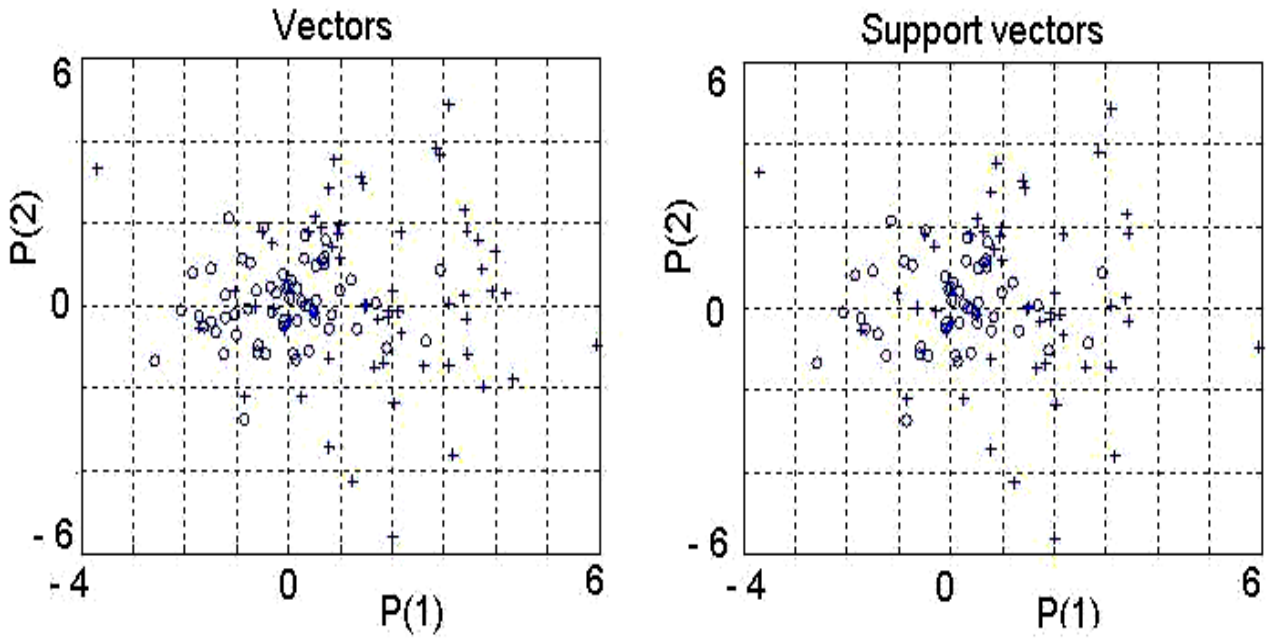


Рис. 4.13. **Расположение на плоскости данных обучающего множества (а). Область, в которой расположились «нолики» имеет распределение Гаусса с центром группирования данных (0,0) и дисперсией 1. Крестики имеют среднее значение вблизи точки (0,2) и характеризуется дисперсией, равной 4. Расположение на плоскости точек опорных векторов (б). Их количество, по сравнению с обучающим множеством, уменьшилось на 20 единиц. Были «отброшены» крестики и нулики, не представляющие интерес для формирования границы разделения классов.**

После прохождения обучения нейронная сеть, использующая SVM, определила примерно 100 опорных векторов, показанных на рис. 4.13б.

Можно выполнить расчеты для классификации двух перекрывающихся гауссовых распределений, представленных классами C_1 и C_2 в

500 точках при обучении 500 при проверке работы сети. Вероятность корректной классификации при этом составляет примерно 0,814. ошибка классификации составила примерно 20 процентов. Если сравнить эту цифру с оптимальным байесовским классификатором (0,815) и с тем, что было получено ранее для многослойной сети (0,797), то нетрудно заметить, что машина опорных векторов обладает способностью решать задачу классификации множеств по результатам практически сравнимым с оптимальными. Практически идеальная классификация подтверждается и построением границы решений (рис. 4.14). Граница решений байесовского классификатора представляет собой круг с центром в точке (2/3, 0) и радиусом 2,34. машина опорных векторов «строит» примерно такую же границу решений между классами.

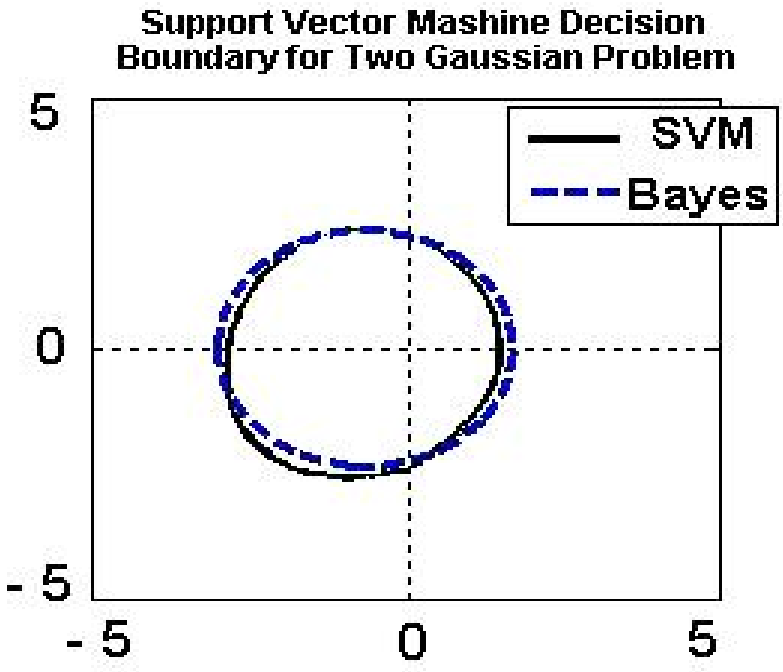


Рис. 4.14. **Графическое изображение границ решений при классификации двух перекрывающихся Гауссовских распределений.**

Нейронные сети могут работать в ситуациях, когда имеется определенная известная информация, а требуется получить некоторую пока неизвестную информацию.

5. НЕЙРОННЫЕ СЕТИ С НЕКОНТРОЛИРУЕМЫМ ОБУЧЕНИЕМ

5.1. Общие сведения и понятия задач таксономии

Мы рассмотрели нейронные сети управляемого разделения классов, в которых «учитель», на основе имеющихся у него знаний, всегда мог сформировать и передать обучаемой нейронной сети желаемый отклик, соответствующий данному входному вектору. То есть в таких сетях обучающие данные всегда содержат вместе с входными переменными соответствующие им выходные значения. Из-за этого сети управляемого разделения классов относительно просто было обучить: достаточно было «сообщить» ей обучающие данные и указать, чего мы от нее добиваемся. Этот процесс очень напоминал обучение ребенка. Показав ребенку изображение буквы «А», мы спрашиваем его: «Какая это буква?» и сообщаем ребенку тот ответ, который бы мы хотели от него получить: «Это буква А». Ребенок запоминает этот «входной» пример вместе с верным ответом (желаемым откликом). Если мы будем повторять процесс предъявления букв снова и снова, то все буквы, после непродолжительного времени обучения, будут твердо запомнены и уверенно называться при предъявлении различных образцов ребенку. При обучении сети мы действуем аналогично. У нас имеется некоторая база данных, содержащая некоторые примеры. Предъявляя их, мы получаем от сети некоторый ответ. Вычисляя разность между желаемым и реальным ответом, настраивая веса (или другими словами – «тренируя» сеть), мы «готовим» ее к «работе» на новых данных. При этом вся информация, которую использует сеть для решения задачи, содержится в наборе примеров (входных данных и задаваемых учителем желаемых откликов). Поэтому при управляемом обучении качество работы сети напрямую зависит от количества примеров в обучающей выборке, а также насколько полно эти примеры описывают решаемую задачу. Конечно, для успешного функционирования сети с учителем также необходим эксперт, который на предварительном этапе для каждого входного образа определит эталонный выходной сигнал.

Теперь мы переходим к изучению искусственных нейронных сетей, в которых осуществляется неконтролируемое обучение (обучение без

учителя, unsupervised). Главной чертой таких сетей является то, что они не получают информацию о желаемом выходном сигнале и поэтому «самостоятельны», поскольку обладают способностью классифицировать данные без вмешательства внешнего учителя, или корректора, контролирующего процесс обучения. Поскольку нет информации о желаемом выходном сигнале, отсутствует «супервизор», управляющий коррекцией весов, в сетях без учителя невозможно сформировать и критерий настройки, основанный на отличии (рассогласовании) реальных и требуемых выходных сигналов. Для обучении без учителя сеть имеет лишь известные входные векторы. Поэтому сети такого типа, после обучения только на входных данных, «автоматически» создают классы, основываясь лишь на некой мере качества, на некоторых представлениях о том, что данные чем-то «похожи» или «не похожи» друг на друга.

На первый взгляд такая постановка задачи может показаться странной. Чему можно учить, если тебе не известен желаемый результат обучения и у тебя нет информации о том, к какому классу следует научиться относить входной образец? Ответ заключается в том, что при обучении без учителя сеть должна сама научиться понимать, как некое «разумное» существо, «структуру», суть данных, выявлять в них то, что можно считать, по совокупности индивидуальных свойств и особенностей, однородным, подобным, типичным или несхожим. То есть, анализируя предъявленные объекты, сеть должна, как бы сделав шаг на новую и более важную «интеллектуальную ступень» обработки информации, сама научиться обнаруживать закономерности в объектах (сама опознавать образы), определять в ходе обучения множества подобных объектов (имеющих что-то общее), а также выявлять отношения между ними. Отнесение объекта к классу в этом случае есть результат сравнения с типичными представителями класса и выбора из них ближайшего.

При произвольном распределении объектов в многомерном пространстве задача таксономии (разделения объектов на несколько компактных и в чем-то схожих групп) является достаточно сложной. Кроме того, ситуация усложняется еще и тем, что число этих групп (классов), в ряде важных для практики случаев, является неизвестным. Для решения такой задачи с помощью нейронных сетей было предложено несколько подходов. Классическим считается разбиение пространства объектов на кластеры и построение так называемой самоорганизующейся карты.

Мы уже отмечали, что каждый объект задается точкой κ-мерного пространства, в котором каждая координата представляет один признак. Образ в этом пространстве отображается совокупностью точек, обладающих общими информативными признаками. Известно также, что образы, обладающие малой изменчивостью, то есть малыми вариациями значений признаков, располагаются в κ -мерном пространстве «компактно» и характеризуются небольшим «объемом». Поэтому напрашивается простое решение задачи распознавания без учителя: классифицировать предъявляемые объекты в соответствии с тем, какое положение они занимает в κ-мерном пространстве (какую область в гиперпространстве) и «проводить» разделяющие гиперповерхности так, чтобы они разделяли образы с неизвестной классовой принадлежностью в соответствии с некими правилами предоставления каждому классу своего «объема». В этом случае, если ввести понятие расстояния, образы можно считать сходными, если они находятся на небольшом расстоянии друг от друга и от центра их «объема», и, наоборот» – существенно отличными, если они разделены значительным расстоянием в κ-мерном пространстве.

Для решения задач распознавания без учителя, при анализе данных, которые «учитель» не наделил «метками» с именем класса, для выделения в них структур, классов, образов, множества подобных объектов полезным оказывается понятие кластера. Под кластером (cluster) в изучаемой совокупности объектов будем понимать группу «схожих», связанных с ближайшими «соседями» и упорядоченных объектов, которые в выборке данных по совокупности индивидуальных особенностей можно считать однородными. Говоря о кластере, надо иметь в виду следующее. В простейшем случае кластер может быть элементарным, состоящим из одного объекта (одного исходного наблюдения, одной точки). Чаще всего кластер является аггломеративным (объединительным) и включает в себя несколько объектов, число которых задается исследователем. Характерной особенностью такого кластера является его связанность. Объединение объектов в кластеры должно удовлетворять следующим требованиям: а) объекты внутри одного кластера должны быть в некотором смысле подобны; б) кластеры, подобные в некотором смысле, должны размещаться близко один от другого. Однако, надо при этом помнить, что разные процедуры кластерного анализа для одних и тех же данных могут давать различное разбиение объектов на кластеры

(как по их количеству, так и по составу). Чтобы облегчить уяснение сказанного перейдем к простейшему графическому представлению данных, которое наглядно и способствует изучению введенных формальных понятий. Для этого изобразим в двумерном пространстве в виде «крестиков» некие входные данные, предъявленные нейронной сети для обучения без учителя (рис. 5.1).

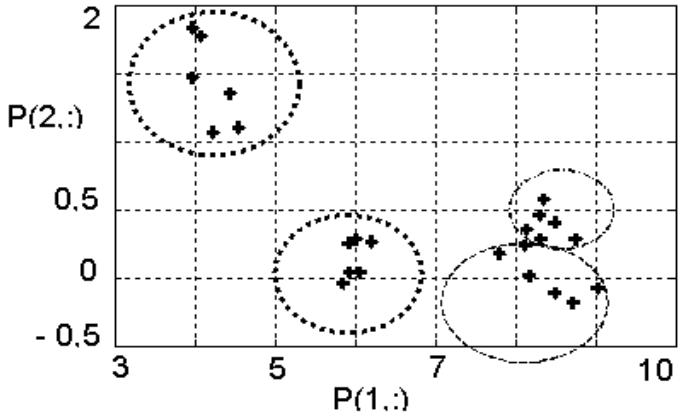


Рис. 5.1. **Пример графического изображения входных данных, предъявленных нейронной сети для обучения без учителя**

Если абстрагироваться от конкретного смысла переменных, то расположение данных на плоскости свидетельствует о следующем. Общая картина расположения «крестиков» показывает отсутствие однородности в выборке и наличие в ней нескольких групп данных. Однородность пространства означает, что все точки пространства равноправны и в нем нет мест, обладающих особыми свойствами. Множества «крестиков», представляющих входные данные в каждой группе, обладают «связанностью». Данные каждой группы образуют отдельное «облако» крестиков (для наглядности на рисунке эти «облака» выделены кривыми шаровидной или эллипсовидной формы), которое принимает на плоскости определенные формы. Пространственная форма, которую принимает совокупность экспериментальных точек, называется следом.

Вообще говоря, каждое «облако» крестиков (как некоторое множество точек с индексом M) образует, в пространстве E^n «тело» – компактное подмножество P_M, внутренность которого не пуста. При характеристике «облаков» можно говорить о том, что непустое компактное подмножество P_M точечного евклидового пространства E^n может быть

выпуклым (если оно вместе с каждой парой своих точек a и b содержит весь соединяющий их отрезок [a,b]). Если в пространстве E^n введены прямоугольные декартовы координаты p_1, ..., p_n ,а a и b – точки из E^n с координатами a_1, ..., a_n и b_1, ..., b_n, то расстояние от точки a до точки b будет определяться формулой $|a-b| = \sqrt{\sum_{i=1}^{n}(a_i - b_i)^2}$, а множество $B^n(a,r) = \{p \in E^n, |p-a| \le r\}$ будет представлять собой n-мерный замкнутый шар радиуса r с центром в точке a. Для каждого множества $B^n(a,r)$ с диаметром (максимальной мерой близости между объектами) $diamP = \sup_{a,b \in P} d(a,b)$ всегда существует наименьший $B^n_{\min}(a,r)$ шар (шар минимального объема), содержащий $B^n(a,r)$.

Таким образом, в точечном евклидовом пространстве E^n можно выделить несколько «тел» – компактных подмножеств $\{P_M\}$ в виде замкнутых шаров минимального радиуса $\{r_M\}$ с центром в точке $\{a_M\}$, которые разделят n-мерные данные на определенные группы $\{p\} = P_1 \cup P_2 \cup ... \cup P_M$. Это семейство подмножеств (семейство минимальных «шаров», рассматриваемых в совокупности и характеризуемых определенными своими свойствами) и представляет собой кластеры. Между этими телами, при условии, что они компактны, можно провести в пространстве E^n гиперповерхности (семейство границ), которые и разделят неоднородные данные на классы. Для $n = 2$ и $n = 3$ термин «гиперповерхность» преобразуется в термины кривая и поверхность, соответственно. Во входных данных, представленных в двумерном пространстве в виде «крестиков» на рис. 5.1, можно выделить 4 кластера. При этом, как видно на рис. 5.1 , при разбиении данных на кластеры может возникнуть ситуация, когда имеет место пересечение нескольких разных следов. В этом случае возникает некоторая общая область пересечения из-за чего трудно провести границу разделения классов и не всегда можно принять решение о том, к какому из кластеров принадлежит изображенный на плоскости в виде точки объект.

Каждый кластер («облако» данных) может быть представлен центроидом – точкой, соответствующей усредненной характеристике раз-

мещения всех образцов в кластере. Из физики известно, что в любой системе частиц имеется одна замечательная, обладающая рядом важных свойств точка С – центр масс, или центр инерции. Введение этой точки позволяет рассматривать движение системы, состоящей из многих частиц, как движение одной частицы. Положение точки С относительно точки О, принятой за начало отсчета в некоторой инерциальной системе, характеризуется радиус вектором, определяемым формулой: $\vec{r}_C = \left(\sum_{i=1}^{N} m_i \vec{r}_i \Big/ M \right)$, где m_i и $\vec{r}_i$ – масса и радиус-вектор каждой частицы системы; $M = \sum_{i=1}^{N} m_i$ - масса всей системы; N – число частиц.

Пользуясь аналогией и полагая единичной «массу» каждой точки (частички) кластера можно говорить, что одним из ключевых параметров кластера, характеризующим размещение всех образцов в кластере, является «опорный представитель» кластера (центр кластера входных образцов, центр группирования, типичный представитель), как некий аналог центра масс. Положение такого опорного представителя w_{oi} для каждого кластера P_i определяется формулой $w_{oi} = \sum_{p \in P_i} p \Big/ \|P_i\|$. Наличие опорного представителя позволяет численно оценивать насколько близка к w_{oi} каждая точка кластера с координатами $p_1, \ ..., \ p_n$. Мерой близости, позволяющей судить о том насколько далеко отстоит от опорного представителя точка кластера, может служить Евклидова мера $d(p,w) = \sum_i (p_i - w_{oi})^2$.

5.2. Нейронные сети Кохонена

5.2.1. Понятие карт самоорганизации

Из всего сказанного вырисовывается идея решения проблемы автоматической классификации без учителя, касающаяся первого и наиболее важного шага – извлечения признаков. Для того, чтобы искусственная нейронная сеть могла автоматически относить предъявленные ей входные данные к одному из классов, она должна сначала «выявить»

все имеющиеся кластеры, потом настроить свои векторы весов, чтобы они «показывали» на центры кластеров, а затем, оценив «близость» каждой точки к центру группирования, «отнести» к ближайшему опорному представителю предъявленный вектор входа и на основании этого определить класс, к которому принадлежит объект с соответствующим значением вектора признаков. При этом, если число предполагаемых классов M, то задачу классификации без учителя можно сформулировать в виде: найти M опорных представителей $\{w_{oi}\}$, позволяющих разбить входные n-мерные данные $\{p\} = P_1 \cup P_2 \cup ... \cup P_M$ на определенные классы $\{C^M\}$ таким образом, чтобы минимизировать общую сумму мер близости $\min\left\{\sum_m d(p^m, w^{m(p)})\right\}$.

Известно, что деятельность человеческого мозга практически целиком сосредоточена в его коре. Каждый фрагмент информации, поступающий от слуховых, зрительных, моторных сенсоров, отражается только в «свои» области головного мозга. При этом отличительным свойством является то, что такие сенсорные входы, как нервные окончания тактильной системы, зрения, слуха, отображаются на области церебральной коры головного мозга меньшей размерности, в виде неких топологически упорядоченных карт. Многомерные, сходные по контексту образы при этом также группируются, из-за чего участки головного мозга, работающие с близко расположенными областями информации, также расположены достаточно близко друг к другу и «взаимодействуют» посредством самых коротких «связей».

Исследования в области нейробиологии, которые свидетельствовали о том, что за упорядочивание ячеек коры мозга, ответственных за извлечение признаков, отвечает некоторый самоорганизующийся процесс, привели к использованию на практике искусственных «топографических» карт, создаваемых посредством самоорганизации. Основной целью карт самоорганизации (self-organizing map – SOM) является преобразование векторов входных сигналов, имеющих произвольную и, как правило, большую, размерность в одно- или двухмерную дискретную карту. При этом такое преобразование осуществляется в топологически упорядоченной форме, когда нейроны помещаются в узлах решетки.

Топологическую карту входных образов обычно реализуют в виде двухслойной нейронной сети Кохонена, которая имеет слой входных нейронов и слой Кохонена (слой активных нейронов). Нейроны входного слоя служат для ввода признаков распознаваемых образов. Они предназначены и для того, чтобы распределять данные входного вектора (N-мерный входной сигнал) между выходными активными нейронами сети. Активные нейроны слоя Кохонена предназначены для формирования кластеров, поэтому их часто называют кластерными нейронами. В случае одномерной самоорганизующейся карты Кохонена активные нейроны расположены в цепочку.

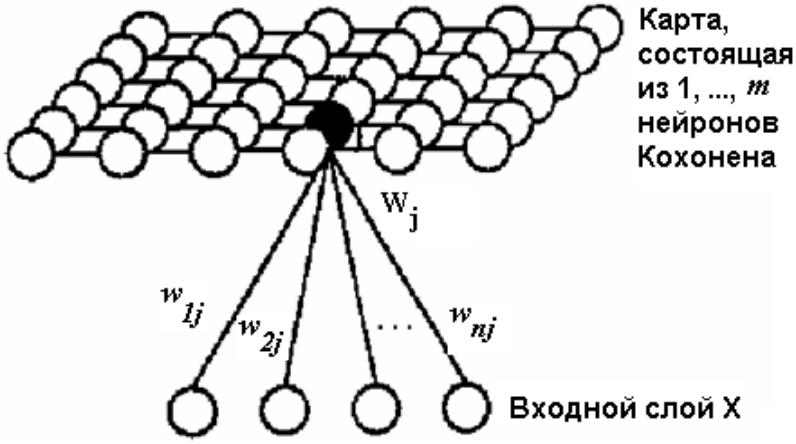


Рис. 5.2. **Топологическая карта нейронной сети Кохонена, кластерный слой которой образует двухмерную сетку**

В двухмерном случае (рис. 5.2) они образуют двухмерную сетку (в форме квадрата или прямоугольника). Количество активных нейронов Кохонена определяется тем, сколько кластеров (и, соответственно, классов) мы ожидаем выявить среди неизвестных входных образов по завершении процесса самообучения. Обычно число кластерных нейронов выбирают меньшим, чем число учебных образцов, поскольку целью исследований является получение упрощенной характеристики данных.

Каждый активный нейрон характеризуется своим положением в выходном слое и весовым коэффициентом. Положение активных нейронов, в свою очередь, характеризуется некоторой метрикой (расстоянием между ними). Следует учитывать и такую особенность активных нейронов. Если в многослойных нейронных сетях в работе одновременно участвовали все нейроны слоя, то нейроны слоя Кохонена «возбуждаются» (участвуют в работе сети) по одному. В этой связи удобно интерпретировать весовые значения каждого задействованного в работе сети ней-

рона $W_{j,i}$ как некие значения координат, описывающих позицию кластера в пространстве входных данных, а процесс коррекции (подбора значений) весов как перемещение (упорядочивание местоположения) «возбужденного» нейрона. Процесс обучения разбиению входных векторов на классы при самоорганизации, как и в случае «с учителем», заключается в том, что для выявления во входных данных образов и нахождения в них «опорных представителей» используется итерационное подстраивание весов сети.

5.2.2. Основные механизмы создания карты признаков

Чтобы сеть могла самоорганизоваться и сформировать топологически упорядоченную карту признаков используют три основных процесса: конкуренции, кооперации и синаптической адаптации.

Пространственное местоположение (то есть координаты) «возбужденных» нейронов решетки является неким «индикатором» встроенных признаков, содержащихся во входных примерах. Если карта будет иметь М нейронов, по числу классов, то чем большее значение «принимает» состояние нейрона m_0, тем больше «уверенность» в том, что входной вектор принадлежит классу $m_0 \in C$. В начале обучения, когда вообще нет сведений о том, какие образы имеются во входных данных, все кластерные нейроны можно рассматривать как претенденты на «награды». В качестве награды им дают входные векторы. Когда выставляется «награда» в виде учебного вектора, то путем анализа тех сигналов, которые генерируют все нейроны слоя Кохонена, проводится «конкурс» (конкурентная борьба, конкуренция) всех кластерных нейронов. В результате конкуренции объявляется «нейрон-победитель». Он, по мнению «конкурсной комиссии», находится в некотором смысле «ближе всего» к предъявленному вектору.

Пусть m размерность входного пространства, а входной вектор, обозначаемый как $p = [p_1, p_2, ..., p_m]^T$, случайным образом выбирается из входного пространства. Алгоритм, ответственный за формирование самоорганизующихся карт, начинает свою работу с инициализации весов сети. Обычно это происходит с помощью придания синаптическим весам значений, к примеру, сформированных генератором случайных чисел. При этом вектор синаптических весов каждого из нейронов сети

будет иметь ту же размерность, что и входное пространство. Будем считать, что синаптический вес J кластерного нейрона определяется следующим образом: $w_j = [w_{J1}, w_{J2}, ..., w_{Jm}]^T$, J = 1, 2, …, L, где L, – общее количество активных нейронов. Для того, чтобы подобрать «победителя», то есть наилучший (ближе всего расположенный) вектор w_J, соответствующий входному сигналу p (награде), нужно сравнить скалярные произведения $w_J^T p$ для J = 1, 2, …, L, и выбрать наибольшее значение (вспомните, что минимизация Евклидова расстояния между векторами математически эквивалентна максимизации скалярного произведения $w_J^T p$). Выбирая нейрон с наибольшим скалярным произведением $w_J^T p$ («возбужденный» нейрон) мы тем самым определяем местоположение нейрона (центр «возбужденного» нейрона) и топологическую окрестность вокруг него.

Для идентификации i-го нейрона «победителя» (с вектором весов w наиболее близким к входному вектору p) из всех нейронов кластерного слоя, который лучше всего соответствует предъявленному входному сигналу, удобно использовать (что и предложил Кохонен) его индекс $i(p)$, который можно определить из соотношения $i(p) = \arg\min \| p - w_J \|$, J = 1, 2, …, L. В таком случае, вместо того чтобы искать местоположение «возбужденного» нейрона путем решения общих уравнений, оказалось достаточным сосредоточить «внимание» нейронной сети на определении величины $i(p)$, являющейся «индикатором» встроенных образов, содержащихся во входных примерах.

Говоря о том, что конкретный нейрон i становится победителем (winning) или наиболее подходящим (best-matching) для данного входного вектора p, следует обратить внимание на то, что в результате процедуры конкуренции между отдельными нейронами сети непрерывное m-мерное входное пространство отображается в L-мерное дискретное выходное пространство и сеть Кохонена формирует единственный «сильный» отклик в виде индекса победившего нейрона (то есть его позицию в решетке). Для «возбужденного» нейрона также рассчитывается вектор синаптических весов, который является самым близким (в Евклидовом

смысле) к данному входному вектору p и, тем самым, нейрон-победитель как бы «запоминает вес» образа.

После того, как определился наиболее близкий по отношению начальным весам входной вектор p_1, победивший нейрон получает право уточнить свои веса в «направлении» других векторов p. Уточнение весов, в простейшем виде, может проводиться по формуле, минимизирующей меру близости для множества остальных входных объектов $\{p\}: w_J(n+1) = w_J(n) + \eta[p - w_J(n)]$, где η – параметр скорости обучения (learning – rate parametr). Согласно этого выражения, вектор весов преобразуется так, что на следующем шаге итерации расстояние между ним и входным вектором, подобным выигравшему, станет еще меньше. Понятно, что в ходе обучения, вероятнее всего, победивший нейрон выиграет соревнование и в том случае, когда будет представлен новый входной вектор, близкий к предыдущему. И наоборот, когда будет подан на вход вектор, существенно отличающийся (по величине расстояния между ним и вектором весов) от предыдущего, то его победа маловероятна, и другой нейрон начнет создание опорного представителя. В конечном счете, каждая группа близких между собой векторов окажется «привязанной» к одному из нейронов кластерного слоя, то есть будет происходить кластеризация данных. Чтобы происходило обучение и других нейронов («неудачников»), победивший нейрон, помимо права уточнить свои веса в «направлении» вектора p, также получает право на кооперацию с незадействованными до этого нейронами слоя Кохонена, на привлечение к «сотрудничеству» (к обновлению весов) нейронов, расположенных в его топологической окрестности. Это дает возможность всем «похожим» векторам p участвовать в формировании более качественного опорного представителя класса. Кроме того, подключение «нейронов-неудачников» к процессу обучения позволяет поделить входное пространство на большее число меньших по размерам областей участвующих в формировании кластеров. «Возбужденный» нейрон-победитель проводит обучение, а, следовательно, и уточнение весов, для пространственно близких к нему кластерных нейронов. Весовые значения «сотрудничающих» нейронов (нейронов из ближайшего окружения нейрона победителя) подлежат обновлению, если нейрон, участвующий в кооперации, находится на карте внутри круга заданного радиуса с центром в нейроне-победителе.

Обозначим для конкретизации символом $h_{j,i}$ топологическую окрестность (функцию кооперации, соседства) с центром в победившем нейроне i, состоящую из множества кооперирующихся нейронов, типичный представитель которой имеет индекс j. Пусть $d_{j,i}$ – латеральное расстояние (lateral distance) между победившим нейроном (i) и вторично возбужденным нейроном (j). Термин «латеральный» отражает то обстоятельство, что в нейронной сети с самоорганизацией имеет место взаимодействие между нейронами слоя (латеральные связи). Он также подчеркивает сходство с процессами латерального взаимодействия нейронов человека. Для обеспечения кооперации между соседними нейронами необходимо, чтобы $h_{j,i}$ была зависима от латерального расстояния между победившим нейроном (i) и вторично возбужденным нейроном (j). Используя нейробиологическое обоснование, можно в качестве функции соседства между j и i нейронами (топологической окрестности) $h_{j,i}$ взять функцию Гаусса: $h_{j,i(p)} = \exp(\frac{d_{j,i}^2}{2\sigma^2})$. Параметр σ, определяющий уровень, до которого нейроны из топологической окрестности победившего участвуют в процессе обучения, называют эффективной шириной (effective width) топологической окрестности. Топологическая окрестность в виде наиболее подходящей с биологической точки зрения функции Гаусса является симметричной относительно точки максимума, определяемой при $d_{j,i} = 0$. Максимум этой функции достигается в победившем нейроне. Амплитуда топологической окрестности $h_{j,i}$ монотонно уменьшается с увеличением латерального расстояния $d_{j,i}$, достигая нуля при $d_{j,i} \to \infty$. В случае одномерной решетки латеральное расстояние $d_{j,i}$ является целым числом, равным модулю $|j - i|$. В случае же двумерной решетки это расстояние определяется соотношением $d_{j,i} = \|r_j - r_i\|^2$, где дискретный вектор r_j определяет позицию возбуждаемого нейрона j, а r_i - победившего нейрона i. Оба эти измерения проводятся в дискретном выходном пространстве.

Еще одним уникальным свойством процесса кооперации в алгоритме SOM является то, что размер топологической окрестности на входе итераций со «временем» уменьшается. Это требование удовлетворяется за счет постепенного уменьшения ширины σ функции топологической окрестности $h_{j,i}$. В зависимости от номера итерации («дискретного времени») величина σ экспоненциально убывает $\sigma(n) = \sigma_0 \exp(-n/\tau_1)$, n = 0, 1,…, где σ_0 – начальное значение; τ_1 - некоторая временная константа. Исходя из этого, топологическая окрестность имеет форму, зависящую от «времени», то есть $h_{j,i(p)}(n) = \exp(-\frac{d_{j,i}^2}{2\sigma^2(n)})$, n = 0, 1,… .

Таким образом, при увеличении количества итераций n эффективная ширина окрестности $\sigma(n)$ экспоненциально убывает, а вместе с ней соответственно сжимается и топологическая окрестность. Поэтому $h_{j,i(p)}(n)$ часто называют функцией окрестности (neighborhood function).

Приступим теперь к изучению последнего механизма, используемого при формировании карты признаков, – синаптической адаптации.

Чтобы сеть могла самоорганизоваться, вектор синаптических весов w_J нейрона j должен изменяться в соответствии с входным вектором p. Если использовать для уточнения весов только классическое «правило Хэбба», основывающееся на феномене привычки, то вес связи (синаптическая сила) между двумя нейронами, одновременно «возбужденными», должен возрастать. Это не отвечает целям классификации и это правило в самоорганизующейся сети использовать нельзя: если изменения в связях происходят только в одном направлении, то это приведет к тому, что нейронная сеть «войдет в насыщение». По этой причине правило Хэбба несколько видоизменяют, вводя слагаемое $w_J g(y_j)$ «забывания» (forgetting term). Вследствие этого изменение вектора весов нейрона в решетке происходят следующим образом: $\Delta w_J = \eta y_J p - w_J g(y_j) \rightarrow \Delta w_J = \eta y_J p - \eta y_j$, где $g(y_j)$ – некоторая положительная скалярная функция отклика y_j; η – параметр скорости обучения. Если при этом принять $g(y_j) = h_{j,i(p)}$, то обновленный вектор синаптических весов $w_J(n+1)$ в момент «времени» $(n+1)$ (для данного

вектора синаптических весов $w_J(n)$ в момент «времени» n) можно определить в следующем виде: $w_J(n+1) = w_J(n) + \eta(n)h_{j,i(p)}[p - w_J(n)]$. Это выражение, применяемое ко всем нейронам решетки, лежащим в топологической окрестности победившего нейрона i, и отражающее эффект перемещения вектора синаптических весов w_i в сторону входного вектора p, свидетельствует о следующем.

При итерационном представлении данных обучения (входных векторов) каждый нейрон карты, участвуя в конкуренции, старается «захватить» один из обучающих векторов (в качестве опорного) и сформировать вокруг него группу ему подобных (расположенных ближе к центроиду соответствующего кластера, чем ко всем остальным кластерам). Поскольку, в общем случае, преобразование m-мерного входного пространства на L-мерную сетку проводится в виде итеративной процедуры, то в результате обучения (точнее самообучения) сети входные образы постепенно формируют на топологической карте дискретный набор «опорных представителей», позволяющий выявлять кластеры в данных. Благодаря коррекции весов в окрестности победившего нейрона, векторы синаптических весов нейронов кластерного слоя будут стремиться «приближаться» и к другим входным векторам. Из-за этого, с одной стороны, близкие друг к другу во входном пространстве входные векторы будут группироваться на карте вблизи одного нейрона. С другой стороны, нейроны, находящиеся «далеко» от обучающих векторов, никогда не будут «выигрывать» и их веса не будут вообще или будут слабо корректироваться при обучении. В результате этого при итерационном представлении данных обучения будет происходить выявление и топологическое упорядочивание (topological ordering) признаков (образов) во входном пространстве. Группы в некотором смысле похожих наблюдений будут формироваться в кластеры. При этом близким в исходном многомерном пространстве векторам будут «соответствовать» близкие узлы на карте. Т.е., получаемая карта признаков в ходе обучения будет становиться все более топологически упорядоченной в том смысле, что пространственное положение нейрона в решетке будет все более точнее соответствовать конкретной области пространства P (или отдельному образу). При этом будет сохраняться не только структура разбиения входных векторов на кластеры, но и отношения топологической бли-

зости между ними. Это значит, что индекс нейрона $i(p)$ (вектор синаптических весов w_i) можно рассматривать как указатель на наиболее подходящий класс для данного входного вектора p. При этом корректируемые нейроны для одинаковых образов будут стремиться иметь одинаковые веса, свойственные для опорного представителя образа. Нужно отметить, что все входные вектора в пределах одного кластера неразличимы и им соответствует один и тот же кластерный нейрон. Следует также подчеркнуть, что отображение m-мерного входного пространства в L-мерное дискретное выходное пространство реализуется в результате рекуррентной (итеративной) процедуры самообучения (unsupervised learning). Отличительная особенность этого отображения – формирование кластеров (Cluster) или классов: по завершении процесса самообучения на стадии реального использования сети Кохонена неизвестные входные образы относятся к одному из выявленных кластеров (классов).

Резюмируя сказанное, можно утверждать, что входное пространство P аппроксимируется (приближается) указателями на «победивший» нейрон или координатами вектора w_i, а сама топологическая карта обеспечивает «правильное» представление образов, характеризующих пространство входных векторов. Как показали исследования, даже для линейно-неразделимых данных из входного пространства самоорганизующаяся карта способна извлечь «наилучший» набор образов.

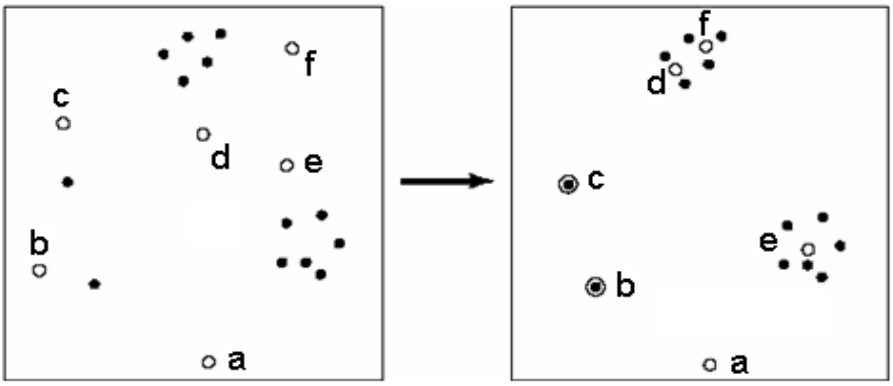


Рис. 5.3. **Изображение того, что нейронная сеть Кохонена не совсем точно разбивает на кластеры входные данные разной плотности**

Дополним сказанное следующим. В результате компьютерных экспериментов было обнаружено, что на топологической карте признаков

области с низкой плотностью данных представляются более «тщательно» нежели области с высокой входной плотностью.

На рис. 5.3 точками обозначены вектора p обучающего множества, а кружками – вектора весовых коэффициентов. Из рисунка видно, что каждому из одиночных векторов выделяется свой кластер (c, b), в то время как такой же кластер формируется для более «плотной» группы векторов (e). Не соответствуют плотности обучающих векторов и кластеры (d,f). Эта «неспособность» алгоритма SOM дать правильное представление о распределении входных данных послужила основанием к модификации самоорганизующихся алгоритмов. Отметим, что к процессу конкуренции может быть добавлен «механизм совести», улучшающий соответствие плотностей распределения. Если нейрон побеждает слишком часто, то он «чувствует себя виноватым» (feels guilty) и выводит себя из конкуренции. В модифицированном алгоритме конкуренции, помимо прочего, каждый из нейронов отслеживает, сколько раз он победил в соревновании (то есть сколько раз его вектор синаптических весов был ближе других нейронов к вектору входного сигнала в смысле Евклидова расстояния). Поэтому каждый нейрон, независимо от его положения в решетке, получает шанс «выиграть соревнование».

5.3. Компьютерное моделирование работы искусственных нейронных сетей SOM

Проиллюстрируем работу искусственных нейронных сетей SOM с помощью компьютерного моделирования. Рассмотрим следующую задачу. В полиграфическом оборудовании 12 раз проводились измерения двух параметров P_1 и P_2. Результаты измерений представлены в виде двух 12 компонентных векторов: P_1 = [0.1 0.3 1.2 1.1 1.8 1.7 0.1 0.3 1.2 1.1 1.8 1.7]; P_2 = [0.2 0.1 0.3 0.1 0.3 0.2 1.8 1.8 1.9 1.9 1.7 1.8]. Определить в каких режимах работало оборудование на момент проведения каждого измерения параметров. Как известно, сеть Кохонена в ходе неуправляемого обучения способна распознать кластеры во входных данных, отнести все измерения к тем или иным кластерам и, тем самым, произвести определение классов (режимов функционирования оборудования). Введем в командное окно следующую программу:

```
clear,
%Создаем самообучающуюся сеть Ко-
                                    3],'hextop',...
                                    'linkdist',Olr,Osteps,Tlr,Tnd);
```

```
хонена
% Компоненты входного вектора
% будут располагаться в диапазонах[0
1.9; 0 2]
%Создаем сеть  с 2x3=6 нейронами
%Топология размещения нейронов в уз-
лах сетки:
%'hextop'- в углах гексагональной (шес-
тиугольной) сетки
%'gridtop'- в углах прямоугольной ре-
шетки
%'randtop'- координаты нейронов распо-
лагаются случайно
%В сети - 'linkdist'- линейное расстояние
между нейронами
% Параметры обучения сети Кохонена:
% Olr- коэффициент обучения на этапе
упорядочивания позиций ней-
% ронов (на фазе упорядочивания)
Olr=0.9;
% Osteps- число шагов для фазы упоря-
дочивания
Osteps=1000;
% Tlr- коэффициент обучения на этапе
уточнения
%позиций нейронов (на фазе подстрой-
ки карты)
Tlr=0.01;
% Tnd- расстояние между нейронами
для фазы подстройки
Tnd=0.5;
netcoch = newsom([0 1.9; 0 2], [2
disp(  '   Параметры карты Кохонена')
netcoch.layers{1}
%Определяем входной   сигнал
disp('Входной  сигнал для обучения сети')
P = [0.1 0.3 1.2 1.1 1.8 1.7 0.1 0.3 1.2 1.1
1.8 1.7; ...
0.2 0.1 0.3 0.1 0.3 0.2 1.8 1.8 1.9 1.9 1.7
1.8]
%Представляем графически начальное
расположение на карте
% векторов входа и нейронов
figure(1), clf,
plot-
som(netcoch.iw{1,1},netcoch.layers{1}.dist
ances), hold on
plot(P(1,:),P(2,:),'*k','markersize',10), grid
on
%Обучаем сеть Кохонена
%Число циклов обучения для фазы под-
стройки
tic, netcoch.trainParam.epochs = 5000;
%Периодичность вывода информации
tic, netcoch.trainParam.show = 500;
netcoch1 = train(netcoch,P); toc
figure(2),
plot(P(1,:),P(2,:),'*','markersize',10), hold on
plot-
som(netcoch1.iw{1,1},netcoch1.layers{1}.d
istances)
netcoch1.IW{1}
a = sim(netcoch1,P)
```

После запуска программы получаем:

Параметры карты Кохонена ans = dimensions: [2 3] distanceFcn: 'linkdist' distances: [6x6 double] initFcn: 'initwb' netInputFcn: 'netsum' positions: [2x6 double] size: 6 topologyFcn: 'hextop' transferFcn: 'compet' userdata: [1x1 struct].

Из этого описания следует, что сеть формирует кластерный слой нейронов (карту Кохонена) размером 2x3, состоящий из 6 нейронов. В

качестве меры близости анализируемых объектов применена функция линейного расстояния между нейронами в слое 'linkdist'. По умолчанию задается функция инициализации слоев нейронов initwb, которая устанавливает начальные значения весов нейронов в соответствии со средними значениями диапазонов векторов входа, и топология размещения нейронов в узлах сетки hextop. Для расчетов используется функция обработки входов netsum, а также функция конкуренции compet, которая формирует разреженную матрицу с единичными элементами, индексы которых соответствуют индексам наибольших элементов каждого столбца матрицы.

Входной сигнал для обучения сети P =

0.1000 0.3000 1.2000 1.1000 1.8000 1.7000 0.1000 0.3000
0.2000 0.1000 0.3000 0.1000 0.3000 0.2000 1.8000 1.8000
1.2000 1.1000 1.8000 1.7000
1.9000 1.9000 1.7000 1.8000

Появляющийся вслед за входными данными график наглядно демонстрирует, что начальные значения весов нейронов задаются как среднее значение концов промежутка изменения координат входных векторов.

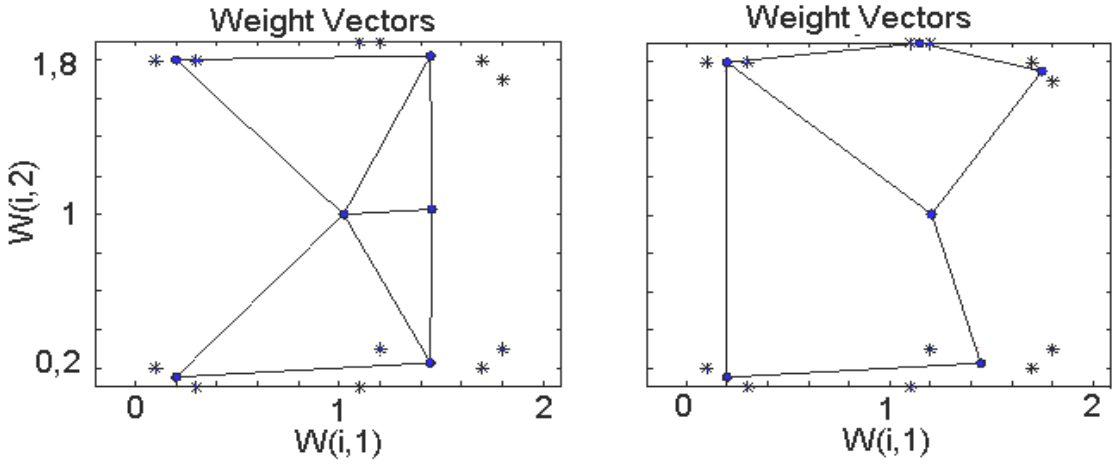
Обучается сеть Кохонена методом последовательных приближений. На первом этапе, нейроны кластерного слоя упорядочиваются так, чтобы отражать входное пространство. Начиная от начального расположения в центре, нейроны-победители постепенно «перемещаются» по карте в такие позиции, чтобы «уловить кластеризацию» обучающих данных. Этот процесс для двумерных и трехмерных данных обычно представляется визуально: чтобы представить на карте относительное расположение нейронов их соединяют линиями, в результате чего получается графическое изображение называемое картой. Группы данных, образующие кластер, представляются единственным нейроном-победителем (опорным элементом кластера). Кластерные нейроны-победители организуются таким образом, чтобы находиться в таком положении карты Кохонена, чтобы «точки» входных данных, расположенные «близко» друг от друга в пространстве входов, располагались также «близко» друг от друга на топологической карте. Сначала позиции опорных элементов кластеров не очень точны, из-за чего карта выглядит сильно «измятой», но постепенно в ходе обучения она «разворачивается и расправляется». На втором этапе обучения производится подстрой-

ка карты (уточнение позиций выигравших нейронов). На этом этапе используется более детальное уточнение позиций нейронов, окружающих выигравший, и постепенное убывание коэффициента обучения, чтобы коррекция позиции кластерного нейрона на карте становилась все более тонкой. При этом похожие входные данные активируют группы близко расположенных нейронов на топологической карте, они уточняют «свои позиции» и, тем самым, внутри участков карты выявляются все более тонкие различия, что, в конце концов, приводит к тонкой настройке каждого нейрона карты.

После того, как сеть Кохонена была обучена (за время обучения 79 секунд – Elapsed time is 79.334000 seconds) на результатах измерения двух параметров P_1 и P_2 была получена топологическая карта, покрывающая все входное пространство (рис. 5.4а).

Значения весов нейронов, которые задают местоположение опорных элементов кластеров в пространстве входных данных, равны:

ans = 0.2007 0.1500; 1.4500 0.2250; 1.0213 0.9990
1.4529 1.0212 ; 0.2007 1.8000; 1.4500 1.8250



а) б)

Рис. 5.4. **Топологические карты Кохонена, построенные на входных данных (*) с размещением нейронов (●) в точках, соответствующих узлам прямоугольной (а) и гексагональной (б) сетки**

Данные на рис. 5.4 наглядно свидетельствуют о том, что нейронной сети удалось в процессе обучения разделить карту на отдельные кластеры и тем самым распознать структуру данных. Если вернуться теперь к содержательному смыслу решаемой задачи, то нейронам тополо-

гической карты можно присвоить содержательные по смыслу метки и для выпуклых областей входного пространства сформировать классы. Анализ результатов, полученных при решении задачи a = (1,1) (1,2) (2,3) (2,4) (2,5) (2,6) (5,7) (5,8) (6,9) (6,10) (6,11) (6,12), свидетельствует о том, что по результатам 12 измерений двух параметров P_1 и P_2 (вторая цифра в скобках) оборудование работало в следующих четырех режимах (первая цифра в скобках) - 1, 2, 5, 6.

Если размещать нейроны в углах прямоугольной решетки, то карта будет иметь вид, показанный на рис. 5.4б. В этом случае сеть Кохонена несколько по-другому образовала кластеры и, вследствие этого, можно говорить о том, что в работе полиграфического оборудования использовано 5 режимов работы: (1,1) (1,2) (2,3) (2,4) (2,5) (2,6) (3,7) (3,8) (5,9) (5,10) (6,11) (6,12).

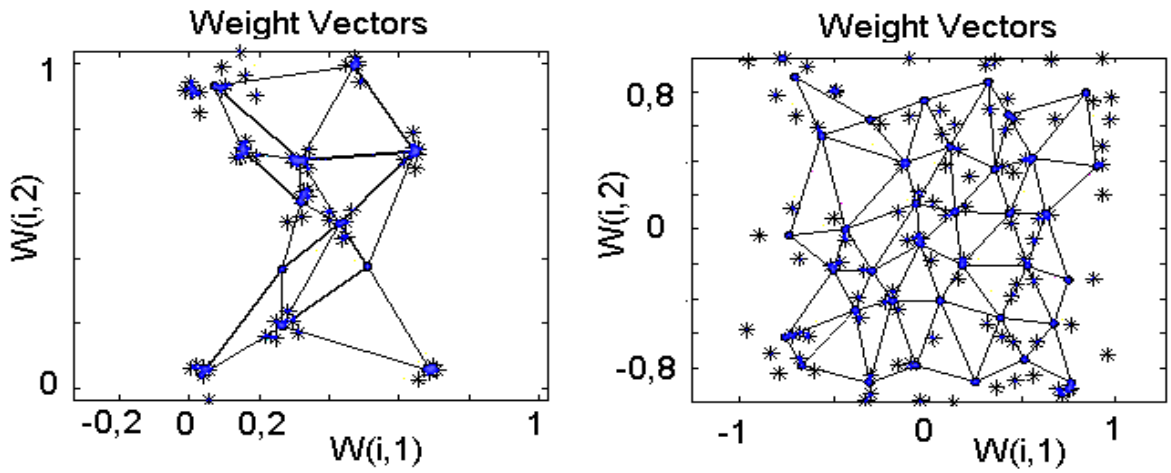
Процесс перемещения нейронов по карте к центрам группировки данных можно интерпретировать и таким образом. В процессе обучения нейронной сетью была сформирована «рыбацкая сеть» с «большими» крючками в узлах. Если забросить такую сеть в реку, где движутся «рыбки-векторы» (*), то на каждом крючке одного узла (●) окажутся «рыбки», проплывающие вблизи «своего» крючка и составляющие своеобразный кластер (рис. 5.4). Из сравнения рис. 5.4а и 5.4б понятно, что в зависимости от того как «плели» сеть, как формировали ячейки сети, так и «ловилась рыбка» на крючки, так и формировались кластеры.

Еще одним подтверждением того, что нейронной сети в процессе обучения удается распознать структуру многомерных входных данных, разделить карту на отдельные кластеры (входные образы) служит следующий компьютерный эксперимент. Создадим сами программным путем 11 кластеров, имеющих в своем составе («в облаке») по 6 точек и убедимся в том, что сеть Кохонена проецирует входные образы на области меньшей размерности с сохранением топологии. После ввода в командное окно программы

```
clear,
%Создаем самообучающуюся сеть Кохонена
% Компоненты вх-го вектора будут
% располагаться в диапазонах[0 1.9; 0 2]
%Создаем сеть с 4x3=12 нейронами
%Топология размещения нейронов в узnetcoch.layers{1}
%Определяем входной  сигнал
disp('Входной  сигнал для обучения сети')
% Фрагмент программы по созданию кластеров
X = [0 1; 0 1];   % Диапазон значений
```

```
лах сетки:
%'hextop'- в углах гексагональной (шестиугольной) сетки
%'gridtop'- в углах пря-моугольной решетки
%'randtop'- координаты нейронов располагаются случайно
%В сети -  'linkdist'- линейное расстояние между нейронами
%    Параметры обучения сети Кохонена:
% Olr- коэффициент обучения на этапе упорядочивания
% позиций нейронов (на фазе упорядочивания)
Olr=0.9;
% Osteps- число шагов для фазы упорядочивания
Osteps=1000;
% Tlr- коэффициент обучения на этапе уточнения
%позиций нейронов (на фазе подстройки карты)
Tlr=0.01;
% Tnd- расстояние между нейронами для фазы подстройки
Tnd=0.5;
netcoch = newsom([0 1.9; 0 2], [4 3],'hextop',...
'linkdist',Olr,Osteps,Tlr,Tnd);
disp('Параметры карты Кохонена')
clusters = 11;     % Количество создаваемых кластеров
points = 6;      % Число точек в кластере
% Стандартное отклонение точек от центра кластера
std_dev = 0.03;
P = nngenc(X,clusters,points,std_dev);
%Представляем графически начальное расположение
% на карте векторов входа и нейронов
figure(1), clf,
plot-
som(netcoch.iw{1,1},netcoch.layers{1}.distances), hold on
plot(P(1,:),P(2,:),'*k','markersize',10), grid on
%Обучаем сеть Кохонена
%Число циклов обучения для фазы подстройки
tic, netcoch.trainParam.epochs = 5000;
%Периодичность вывода информации
tic, netcoch.trainParam.show = 500;
netcoch1 = train(netcoch,P); toc
figure(2),
plot(P(1,:),P(2,:),'*','markersize',10), hold on
plot-
som(netcoch1.iw{1,1},netcoch1.layers{1}.distances)
netcoch1.IW{1}
a = sim(netcoch1,P)
```

примерно через 8 минут получаем следующую двухмерную карту Кохонена (рис. 5.5а). Из рис. 5.5а наглядно видно, что входные данные, образующие кластер, представляются на карте единственным «нейроном-победителем». При этом веса, ассоциированные с этими нейронами, принимают значения вблизи «центра облака» входных сигналов, для которых данный нейрон является «победителем».



а) б)

Рис. 5.5. **Топологические карты Кохонена, построенные на входных данных (*), состоящих из созданных программным путем 11 кластерах (а) и на входных данных, имеющих почти однородное распределение по пространству (б), с размещением нейронов (●) в узлах, соответствующих углам гексагональной сетки**

Пусть входные векторы выбираются случайным образом так, чтобы обеспечить почти однородное распределение данных в некотором квадрате. Возьмем для примера 200 компонентный входной вектор и построим карту размером 6x6=36. Вводим в командное окно программу:

```
clear,
%Создаем самообучающуюся сеть Кохонена
% Компоненты входного вектора
% будут располагаться в диапазонах[0 1.9; 0 2]
%Создаем сеть  с 6x6=36 нейронами
%Топология размещения нейронов в узлах сетки:
%'hextop'- в углах гексагональной (шестиугольной) сетки
%'gridtop'- в углах прямоугольной решетки
%'randtop'- координаты нейронов располагаются случайно
%В сети -  'linkdist'- линейное расстояние между нейронами
% Параметры обучения сети Кохонена:
netcoch = newsom([-1 1; -1 1], [6 6],'hextop',...
'linkdist',Olr,Osteps,Tlr,Tnd);
disp(' Параметры карты Кохонена')
netcoch.layers{1}
%Определяем входной сигнал
disp('Входной сигнал для обучения сети')
P = rands(2,100);
%Представляем графически начальное расположение
% на карте векторов входа и нейронов
figure(1), clf,
plotsom(netcoch.iw{1,1},netcoch.layers{1}.distances), hold on
plot(P(1,:),P(2,:),'*k','markersize',10), grid on
%Обучаем сеть Кохонена
```

```
% Olr- коэффициент обучения на этапе упорядочивания позиций нейронов
% (на фазе упорядочивания)
Olr=0.9;
% Osteps- число шагов для фазы упорядочивания
Osteps=1000;
% Tlr- коэффициент обучения на этапе уточнения позиций нейронов
% (на фазе подстройки карты)
Tlr=0.01;
% Tnd- расстояние между нейронами для фазы подстройки
Tnd=0.5;
%Число циклов обучения для фазы подстройки
tic, netcoch.trainParam.epochs = 5000;
%Периодичность вывода информации
tic, netcoch.trainParam.show = 500;
netcoch1 = train(netcoch,P); toc
figure(2),
plot(P(1,:),P(2,:),'*','markersize',10), hold on
plotsom(netcoch1.iw{1,1},netcoch1.layers{1}.distances)
netcoch1.IW{1}
a = sim(netcoch1,P)
```

После запуска программы получаем двухмерную карту Кохонена (рис. 5.5б). Из рисунка видно, что карта покрывает все входное пространство. Однако из-за того, что входные данные распределены неравномерно, между ними имеются области «пустот» различных размеров карта получилась не регулярной, а несколько «измятой».

Проиллюстрируем теперь работу алгоритма SOM на примере нейронной сети, состоящей из 100 кластерных нейронов, упорядоченных в форме двумерной решетки размером 10x10. Эта сеть обучается на 200 двумерных входных векторах P компоненты p_1 и p_2 которых равномерно распределены в областях $\{(-1 < p_1 < 1);(-1 < p_2 < 1)\}$. Для инициации сети синаптические веса выбирались случайным образом. В этом случае, как следует из рис. 5.6, входные данные более однородные и в значительной мере распределены равномерно.

В начале обучения синаптические веса выбирались случайным образом и имели почти однородное распределение в центральной части квадрата. На этапе обучения, который называется фазой упорядочивания, карта представляет собой достаточно регулярную сетку (нейроны отображаются в правильном порядке, поэтому линии, соединяющие соседние нейроны образуют правильные квадраты), покрывающую все входное пространство. Это свидетельствует о том, что входные данные распределены в значительной степени равномерно. Однако, при внимательном рассмотрении распределения входных данных можно обнаружить локальные неравномерности во входном распределении. На этапе

сходимости распределение нейронов на карте «уточняется» и приближается к «истинному» распределению входных векторов (может быть за исключением граничных эффектов). Таким образом, алгоритм SOM достаточно точно «извлек» топологию распределения входного сигнала.

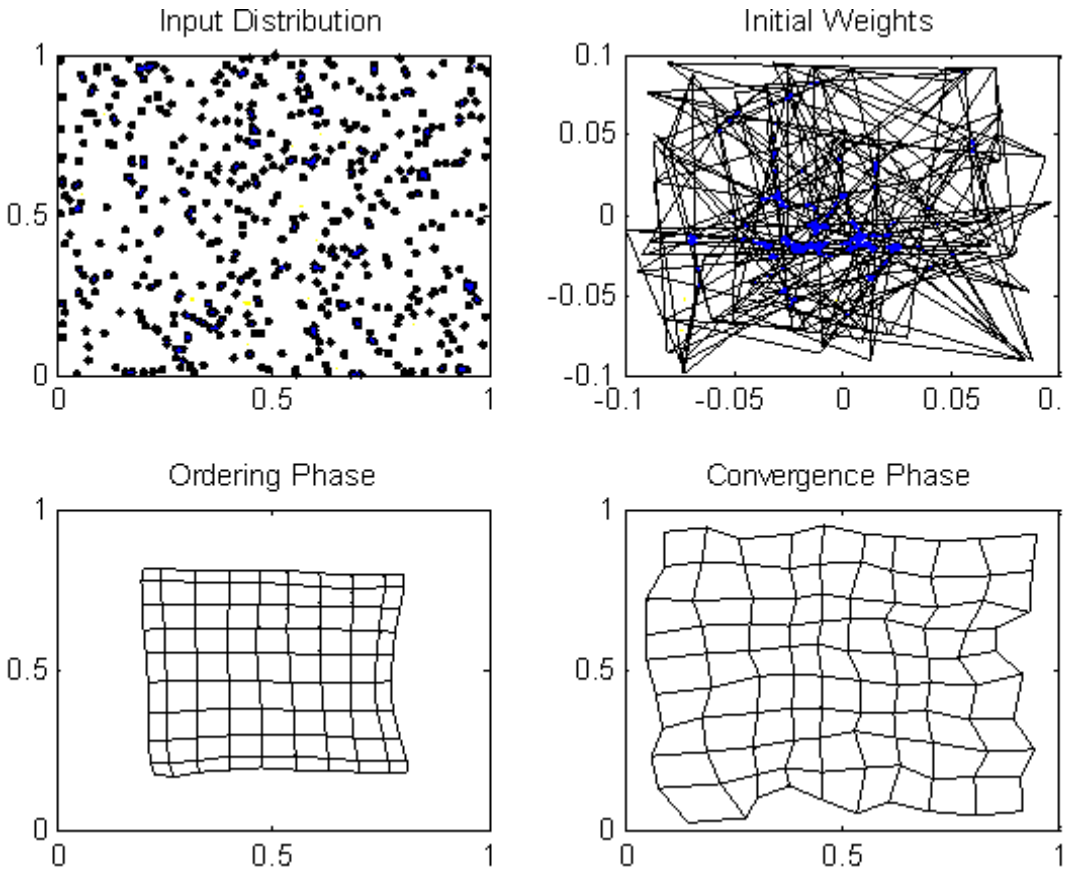


Рис. 5.6. **Распределение входных данных, начальное состояние весов двумерной решетки, состояние карты в конце этапа упорядочивания и этапа сходимости**

Учитывая то, что в исходном состоянии сеть находилась в состоянии полного беспорядка а, затем, в результате следования определенным предписаниям постепенно достигла организованного представления входных данных, нетрудно предположить, что для наилучшей реализации алгоритма SOM потребуется дать ответы, хотя бы в форме эвристик, на следующие вопросы: 1) *Сколько нейронов должно быть в слое?* Считается, что число нейронов в слое Кохонена должно соответствовать ожидаемому числу классов входных сигналов. Поскольку заранее бывает неизвестно, какие кластеры встретятся в данных, то рекомендуется провести несколько экспериментов, позволяющих ориентиро-

вочно оценить размер карты и выбрать их число, позволяющее придать кластерам какой-то смысл. 2) *Как на этапе упорядочивания разумно выбрать число шагов (Osteps) и коэффициент обучения η (Olr)?* Чтобы происходило топологическое упорядочивание векторов весов, в литературе рекомендуется брать Osteps=1000 (не менее). Выбор коэффициента обучения (параметра скорости обучения) влияет как на устойчивость получаемого решения, так и на скорость обучения. Процесс обучения ускоряется при выборе η близким к единице. Однако при этом существует опасность «перепрыгивания» из своего кластера в чужой и, соответственно, получения неустойчивого решения. Напротив, при $\eta \to 0$ скорость обучения будет медленной, однако вектор весовых коэффициентов при подаче на вход сети различных сигналов, относящихся к одной группе, будет «упорно» стремиться к центру кластера (к опорному вектору) и можно «не заметить» входных данных, относящихся к данному кластеру. В литературе рекомендуется параметр скорости обучения брать из диапазона [0.9-0.1]. Считается лучшим $\eta = 0.1$. 3) *Как на этапе сходимости задать параметр скорости обучения η (Tlr)?* Для точного упорядочивания входного пространства на этапе сходимости требуется точная подстройка карты, поэтому параметр скорости обучения в этом случае должен иметь малое значение. Чтобы избежать «застревания» сети в метаустойчивых состояниях рекомендуется выбирать Tlr=0.01. При этом количество итераций, достаточное для этапа сходимости, должно примерно в 500 раз превышать количество кластерных нейронов сети. 4) *Как правильно выбрать параметр σ (1/Tnd), называемый эффективной шириной топологической окрестности?* Поскольку этот параметр определяет уровень, до которого нейроны из топологической окрестности победившего участвуют в процессе обучения, то при малых σ можно «не захватить» далеко расположенные нейроны. Рекомендуется параметр Tnd брать из диапазона [0.5-0.1]. Осуществляя компьютерное моделирование, мы брали входное и выходное пространство двумерным. Нейронная сеть Кохонена способна работать и в случае, когда размерность входного пространства P больше размерности выходного. Покажем это на примере, когда входные данные двумерны и взяты из равномерного распределения внутри квадрата, а карта отображается на одномерную решетку, состоящую из 100 нейронов. После обучения на входных данных и представления пространства P его проекцией на

пространство более низкой размерности (снижения размерности) получена следующая одномерная карта (рис. 5.7).

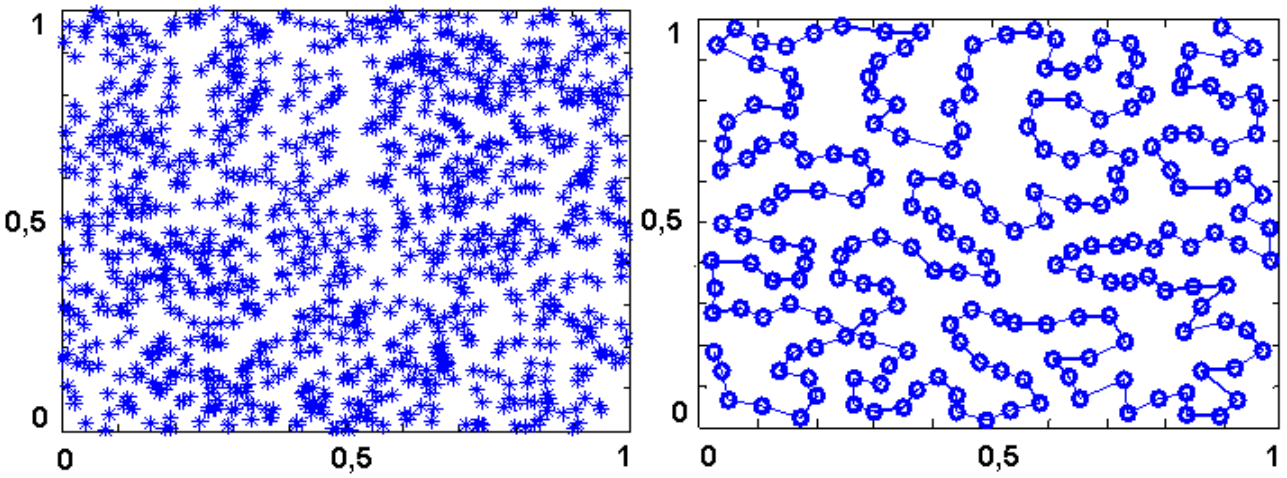


Рис. 5.7. **Распределение двумерных входных данных и состояние одномерной решетки в конце этапа сходимости**

Из рисунка видно, что одномерная карта разворачивается таким образом, чтобы заполнить пространство входных данных. Кривая, соединяющая соседние нейроны карты на рис. 5.7, является кривой Пеано. Как несложно заметить одномерная карта, хотя и отображает входное пространство, но дать достаточно хорошую аппроксимацию двумерного входного пространства не может. Представление данных посредством кривой Пеано не дает корректного достаточно равномерного покрытия квадрата входных данных. Для визуального анализа степени сходства входных данных и выявления групп, близких между собой векторов, можно строить трехмерные карты Кохонена в которых нейроны расположены в трехмерном пространстве.

На рис. 5.8 показаны результаты работы алгоритма SOM на нейронной сети, состоящей из 36 кластерных нейронов, упорядоченных в форме трехмерной решетки. Сеть обучалась на 100 трехмерных входных векторах P компоненты p_1, p_2 и p_3 которых были равномерно распределены в кубе $\{(-1<p_1<1);(-1<p_2<1);(-1<p_3<1)\}$. Кластерные нейроны трехмерной карты, как следует из рис. 5.8, относительно равномерно распределены по пространству, что свидетельствует о покрытии всего трехмерного пространства входных данных. Упомянем, что можно строить карты и большей размерности, но их графическое представление будет соответствовать только трем последним координатам.

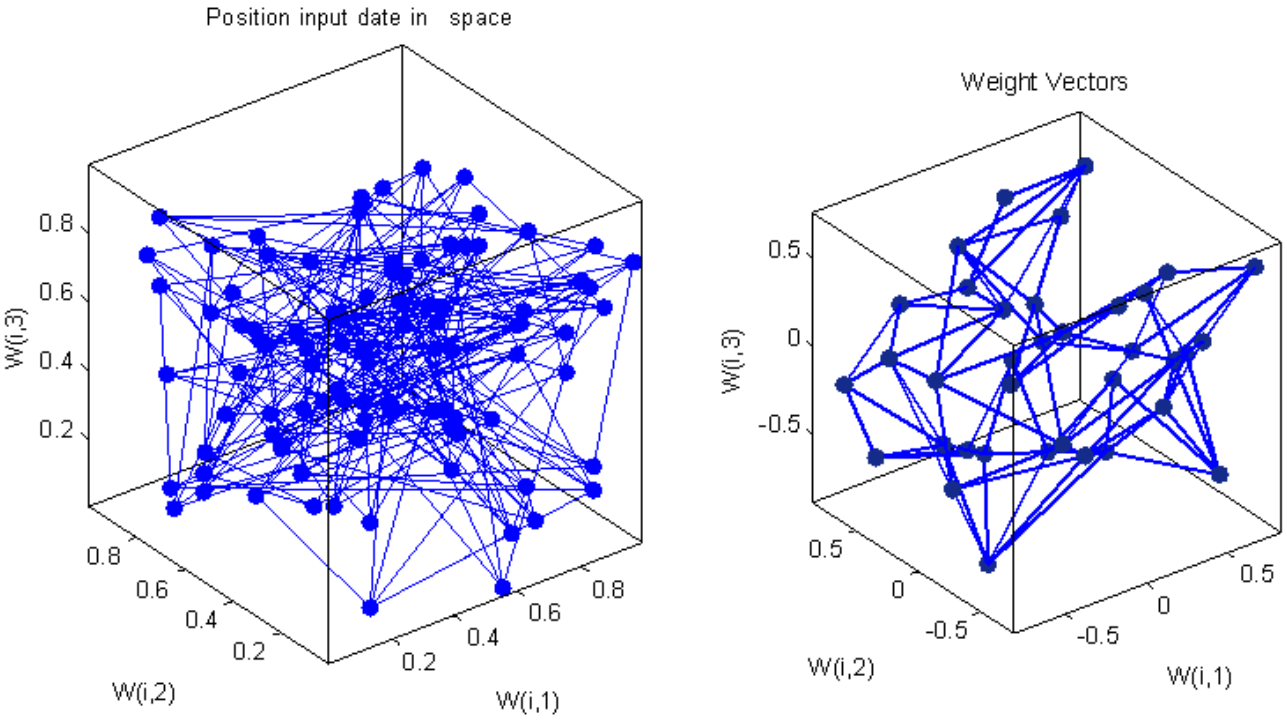


Рис. 5.8. **Расположение входных данных и кластерных нейронов трехмерной решетки Кохонена в конце этапа сходимости**

В задачах, связанных с исследованием топологической структуры данных, объединением их в кластеры (в группы объектов сходных в некотором смысле между собой) используют также слой Кохонена (слой «соревнующихся» нейронов). Его применяют для двумерного случая входных данных в задачах автоматического выявления центров кластеров. Вводим в командное окно программу:

```
clear,
% Фрагмент программы по созданию кластеров
X = [0 1; 0 1];   % Диапазон значений
clusters = 8;
% Количество создаваемых кластеров
points = 10;      % Число точек в кластере
% Стандартное отклонение точек от центра кластера
std_dev = 0.05;
P = nngenc(X,clusters,points,std_dev);
%Представляем графически расположение на карте векторов входа
figure(1), clf,
% Коэффициент "справедливости"
Clr=0.0001;
net = newc(minmax(P),8,Klr,Clr);
% Формирование графика исходных данных и вектора весов
w = net.IW{1};
figure(2), clf,
plot(P(1,:),P(2,:),'*b');
title('Position  neurons in 0 time '),
hold on;
circles = plot(w(:,1),w(:,2),'ob');
%Задание количества циклов обучения
net.trainParam.epochs = 1000;
%Периодичность вывода информации
```

```
plot(P(1,:),P(2,:),'*b');
title('Input Vectors');
xlabel('p(1)');
ylabel('p(2)');
%%Создаем слой Кохонена
% Компоненты входного вектора будут
% располагаться в диапазонах[0 1; 0 1]
% В слое будет 10 нейронов
% Коэффициент обучения Кохонена
Klr=0.05;
tic, net.trainParam.show = 100;
%Обучение сети
net = train(net,P);toc
w = net.IW{1};
figure(3), clf,
plot(w(:,1),w(:,2),'.','markersize',30);
title('Position data and neuron in space '),
hold on;
plot(P(1,:),P(2,:),'*b');
a = sim(net,P)
```

После запуска программы получаем результаты проведения кластеризации исходных двумерных данных (рис. 5.9). Из рис. 5.9 видно, что каждый нейрон слоя Кохонена можно истолковывать как центр группировки определенных входных данных, а его весовой вектор как обобщенный вектор, представляющий целый класс входных сигналов.

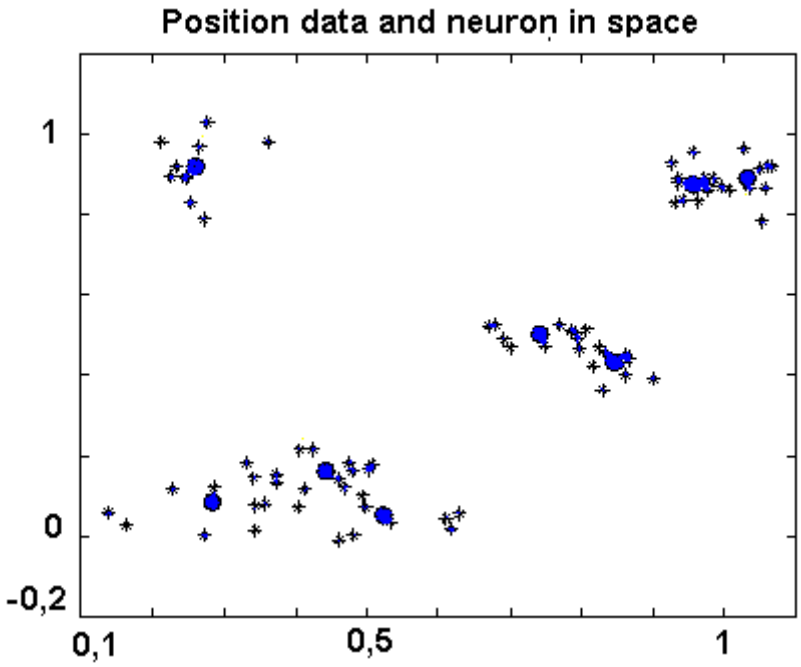


Рис. 5.9. **Исходные входных данные (*) и выявленные центры кластеров (●).**

5.4. Особенности LVQ нейронных сетей

Одной из модификаций самоорганизующихся сетей Кохонена являются нейронные сети для кластеризации и классификации векторов, осуществляющие квантование вектора обучения и поэтому названные LVQ (learning vector quantization). Векторное квантование – это прием

обучения с учителем, когда за счет использования «знаний» о структуре входных данных каждому входному образу *p* ставится в соответствие желаемый выходной сигнал. При этом первый и наиболее важный шаг классификации множеств – извлечение признаков – выполняется без учителя за счет использования самоорганизующихся карт Кохонена. Комбинация самоорганизующейся карты со схемой обучения с учителем формирует гибридную по своей сущности сеть, которая в принципе может осуществлять адаптивную классификацию множеств.

LVQ –сеть получают путем добавления к входному слоев нейронов Кохонена и Гроссенберга (линейного слоя). Каждый компонент входного сигнала подается на все нейроны Кохонена. Каждый нейрон слоя Кохонена соединен со всеми нейронами слоя Гроссенберга. Как оказалось, объединение разнотипных нейронных структур в единой архитектуре приводит к свойствам, которых нет по отдельности у каждой сети. По этой причине LVQ –сеть, называемая также сетью встречного распространения, по своим характеристикам существенно превосходят возможности сетей с одним скрытым слоем нейронов. Так, время их обучения при решении задач распознавания и кластеризации более чем в 100 раз меньше времени обучения аналогичным задачам сетей с обратным распространением. Это является важным фактором в технических приложениях, где долгая обучающая процедура невозможна. Другим определяющим свойством сети встречного распространения являются ее хорошие способности к обобщению, что позволяет получать правильный выход даже при неполном или зашумленном входном векторе. Это дает возможность эффективно использовать данную сеть для распознавания и восстановления образов.Такой гибридный подход к решению задач классификации, вообще говоря, может иметь различные формы, в зависимости от того, как реализована схема обучения с учителем. Однако чаще всего используют квантователь вектора обучения.

Известно, что при соответствующей структуре входных векторов входное пространство $P \in R^N$ можно поделить на несколько (m) четких областей, для каждой из которых имеется опорный вектор w_{cm}. При этом множество векторов P разбивается на конечное число классов C_m, каждому из которых присваивается свой код. Каждому классу соответствует свой опорный вектор. Множество возможных опорных векторов

$\{w_{cm}\}, m = \overline{1, M}$ часто называют кодовой книгой (code book) устройства квантования (классификатора). Отдельные опорные векторы w_{cm} называют кодовыми словами (code word).

Минимальные искажения при кодировании (при разбиении пространства на «ячейки»), как установлено в литературе, можно получить на основе правила «ближайшего соседа» (nearest – neighbor quantizer), которое основывается на Евклидовой метрике. Устройство векторного квантования, формирующее во входном пространстве «ячейки» на основе правила ближайшего соседа называется квантователем Вороного (Voronoi). В каждой такой ячейке имеется опорный вектор, называемый вектором Вороного. Каждая из ячеек содержит только те точки входного пространства, которые ближе всего расположены к вектору Вороного.

Алгоритм SOM обеспечивает приближенный метод вычисления векторов Вороного без учителя. При этом на карте признаков «появляются» векторы синаптических весов нейронов. Поэтому для «подстройки» карты используется дополнительный «механизм», основанный на использовании информации о классе, для того, чтобы «откорректировать» вектор Вороного и тем самым улучшить качество проведения границ классификатором. Пусть $\{w_J\}$ – множество векторов Вороного, а p_i - множество входных векторов. Предположим при этом, что количество входных векторов намного превосходит количество векторов Вороного, что, как правило, имеет место на практике. Входные векторы p случайно выбираются из входного пространства. Предположим также, что вектор Вороного w_C является самым близким к поданному на вход сети вектору. Обозначим символом C_{W_C} класс, ассоциируемый с вектором Вороного w_C, а символом C_{P_i} – метку класса входного вектора p_i. В этом случае, при адаптивной классификации множеств, использующей самоорганизующуюся карту и квантование вектора обучения, корректирование вектора Вороного w_C осуществляется следующим образом:

- Если $C_{W_C} = C_{P_i}$, то $w_c(n+1) = w_c(n) + \alpha_n[p_i - w_c(n)]$.
- Если $C_{W_C} \neq C_{P_i}$, то $w_c(n+1) = w_c(n) - \alpha_n[p_i - w_c(n)]$.

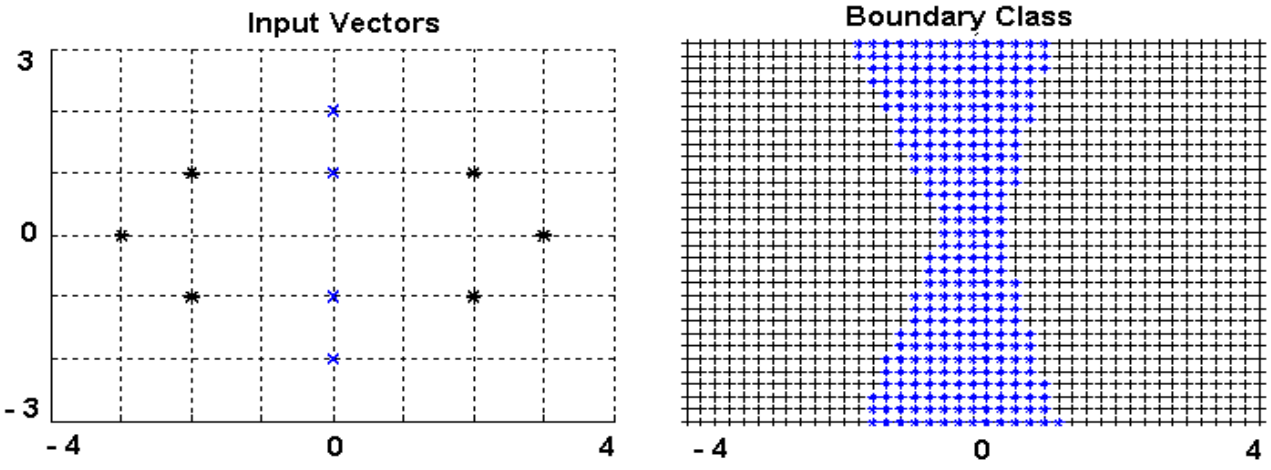
Таким образом, если метка класса входного вектора p и вектора Вороного w_C согласуются, то последний смещается в направлении пер-

вого. Если же метки классов этих векторов не согласуются, то вектор Вороного смещается в сторону, противоположную вектору p. Соответственно, вектор весов нейрона-победителя w_C, который ближе всего расположен к предъявляемому входному вектору, смещается в направлении последнего, если вектор относится к одному с ним классу, и удаляется от него в противном случае. Для демонстрации работы алгоритма LVQ решим задачу. Задано 10 векторов входа. Необходимо, используя LVQ-сеть, во-первых, сгруппировать эти вектора в 4 кластера, а, во-вторых, соотнести эти кластеры к одному из двух выходных классов. Для определенности будем считать, что первый конкурирующий слой, которой имеет 4 нейрона по числу кластеров, а второй линейный слой – 2 нейрона по числу классов. Вводим в командное окно программу:

```
clear,
%Определяем входной   сигнал
disp('Входной  сигнал для обучения сети')
P = [-3 -2 -2  0 0 0 0  2 2 3;...
0 1 -1 2 1 -1 -2 1 -1 0]
%Определяем входной   сигнал
disp(' Классы для обучения сети')
Tc = [1 1 1 2 2 2 2 1 1 1]
%Строим график расположения векторов
входа
I1 = find(Tc==1); I2 = find(Tc==2);
figure(1), clf, axis([-4,4,-3,3]), hold on
plot(P(1,I1),P(2,I1),'*k','markersize',10), grid
on
plot(P(1,I2),P(2,I2),'xb','markersize',10)
title('Input Vectors');
disp(' Массив целевых векторов, в кото-
ром кол-во единиц в строке к общему')
disp('кол-ву нулей и единиц дает долю ')
disp(' входных векторов в каждом классе')
disp(' Например: 6 единиц/10 знаков даст
долю =0,6')
T = full(ind2vec(Tc))
%%Создаем сеть встречного распростра-
нения LVQ
% Компоненты входного вектора будут
net.trainParam.show = 100;
%Параметр скорости обучения
net.trainParam.lr = 0.05;
%Обучаем сеть
tic, net = train(net,P,T); toc
disp('Веса нейронов конкурирующего
слоя,которые определяют положение ')
disp('центров кластеров')
V = net.IW{1,1}
%Строим график распределения векто-
ров входа по кластерам
I1 = find(Tc==1); I2 = find(Tc==2);
figure(2), axis([-4,4,-3,3]), hold on
P1 = P(:,I1); P2 = P(:,I2);
plot(P1(1,:),P1(2,:),'*k','markersize',10)
plot(P2(1,:),P2(2,:),'xb','markersize',10)
plot(V(:,1),V(:,2),'or','markersize',10)
title('Input Vectors in Clusters');
disp('Веса нейронов линейного слоя,ко-
торые указывают как центры кластеров
распределены по классам')
net.LW{2}
% Проверка функционирования сети
Y = sim(net,P),
disp(  'Массив полученных в результате
проверки классов')
```

```
% располагаться в minmax диапазонах
% В скрытом слое будет 4 нейрона
Sout=4;
% Доля принадледности элементов  к первому классу
Dp1=0.6;
% Доля принадледности элементов  к второму классу
Dp2=0.4;
% Коэффициент обучения
Klr=0.05;
% Функция обучения 'learnlv2'
net = newlvq(minmax(P),Sout,[Dp1 Dp2],Klr,'learnlv2');
disp(  '  Параметры сети')
net.inputWeights{1}
%Задаем параметры обучения сети:
%Число циклов обучения
net.trainParam.epochs = 2000;
%Периодичность вывода информации
Yc = vec2ind(Y)
%Построение границ, разделяющих классы
x = -4:0.2:4;
y = -3:0.2:3;
% Вычисление массива координат прямоугольной сетки
[X,Y]= meshgrid(x,y);
P = cat(3,X,Y);
[n1,n2,n3] = size(P);
P = permute(P,[3 2 1]);
P = reshape(P, [n3 n1*n2]);
Y = sim(net,P);
Yc = vec2ind(Y);
I1 = find(Yc==1); I2 = find(Yc==2);
figure(3), clf,
plot(P(1,I1),P(2,I1),'+k'), hold on
plot(P(1,I2),P(2,I2),'*b')
title('Boundery Class');
```

После запуска программы получаем (рис. 5.10).



а) б)

Рис. 5.10. **Исходные входных данные которые необходимо классифицировать(а) и граница решений, построенная на карте (б).**

Для распознавания сети LVQ предъявлены данные, в которых можно легко распознать 4 кластера, два из которых относятся к одному классу, а два – к другому (рис. 5.10а). В процессе обучения определялись «нейроны-победители» слоя Кохонена. Множество сходных между

собой входных векторов «активировали» один и тот же нейрон, в результате чего были сформированы 4 опорных вектора, характеризующих 4 кластера, имеющихся во входных данных. Кроме того, в сети проводилось определение весов слоя Гроссберга (обучение с учителем, использующее заданные желаемые выходы). При этом веса корректировались лишь в том случае, если нейрон линейного слоя был соединен с нейроном Кохонена, имеющим ненулевой вход. Величина коррекции веса была пропорциональна разности между весом и требуемым выходом нейрона Гроссберга. В результате обучения были получены веса нейронов конкурирующего слоя, которые определяют положение центров кластеров V = 0.3566 -0.0525 -0.4345 -0.0950 0.1027 -0.1151 0 0.2353 и веса нейронов линейного слоя, помечающие, как центры кластеров распределены по классам 1 1 0 0 0 0 1 1. Этот результат свидетельствует о том, что сеть LVQ после обучения обеспечила верное разбиение входных сигналов на кластеры и, соответственно на классы. Последовательная коррекция весовых коэффициентов обеспечивает также построение границ между классами, как медиатрис между опорными векторами (рис. 5.10б).

Группировка данных в некоторые классы, которым присваивается свой код или индекс, формирование кластеров со своим опорным вектором (центром кластера), внутри которых данные различаются незначительно, можно использовать для обработки больших массивов данных с целью «сжатия данных». Если массивы данных внутри кластера отличаются незначительно и, из-за вариации их внутри кластера, небольшие потери информации несущественны, то каждую группу входных векторов одного кластера можно представить «эталонным образом». Как известно, последний определяется вектором весов «нейрона-победителя» и «номером», а точнее кодом (индексом). Если количество нейронов сети значительно меньше, чем количество вариантов входных образов, то для кодирования «эталонов» потребуется значительно меньший объем информации, чем для кодирования всех входных образов.

Известно, что каждый день посредством вычислительных машин и различных средств коммуникации запоминается, преобразуется и передается в цифровом виде огромное количество информации. Правительственные программы информатизации формируют массивы данных о принятых законодательными органами законах, о содержащихся в библиотеках изданиях и публикациях. Интернет заполнен информацией о

новостях, периодических изданиях, информацией о товарах, каталогах, которыми фирмы снабжают своих партнеров и пр. Кабельное телевидение по заказу пользователей по каналам передает свою развлекательную видеоинформацию. При этом значительная часть передаваемых данных является графической и звуковой информацией. Работа с речевыми сигналами, изображениями, представленными в цифровом виде, всегда связана с необходимостью хранения, обработки и передачи больших массивов цифровых данных. Это требует значительных аппаратных вычислительных ресурсов и существенных временных затрат. Поэтому значительный практический и коммерческий интерес приобретают средства сжатия данных (или как еще говорят «компрессии данных») для их передачи и хранения.

Сам термин «сжатие данных» в обиходе означает уменьшение объема данных, используемых для представления определенного количества информации. При этом слова «данные» и «информация» не являются синонимами. Вам это хорошо известно из жизни, когда два разных человека рассказывают одну и ту же историю. Один ведет рассказ многословно, другой – точно и лаконично. Несмотря на то, что рассказчики используют разное количество слов (данных) информация у них практически одна и та же (конкретный факт). Это означает, что какой-то вариант рассказа истории содержит данные (слова) которые, либо несут несущественную информацию, либо повторяют уже известное. Другими словами, один рассказчик использует избыточные данные.

Важным вопросом, с которым сталкиваются разработчики «дружественных» SILK- интерфейсов (Speech –речь, Image – образ, Language – язык, Knowledge – знание) и речевых технологий – это выбор оптимального метода параметрического представления и хранения речевого сигнала. Поскольку «речь» содержит значительную малоинформационную случайную составляющую, а для ее передачи нужны высокоскоростные аппаратные средства, и для хранения «речи» требуется довольно большой объем памяти, то целесообразно «сжимать» звуковую информацию.

Для удаления избыточности из речевого сигнала, а, следовательно, для его «сжатия» может быть применены нейронные сети векторного квантования. Основная идея работы таких устройств состоит в том, чтобы отобразить неограниченное множество векторов признаков речевого сигнала на конечное множество наиболее типичных акустических со-

стояний. При этом многомерный вектор параметров заменяется индексом соответствующего опорного элемента кодовой книги. Кроме того, закрепляя за опорным элементом кодовой книги определенный звук, можно установить связь с фонетическим уровнем. Естественно, что важным вопросом векторного квантования является разработка соответствующей кодовой книги, необходимой для квантования. Если количество элементов (количество нейронов сети) значительно меньше вариантов образов речевого сигнала, то потребуется значительно меньший объем информации, чем для кодирования всех векторов признаков речевого сигнала. При уменьшении количества нейронов в сети «сжатие» информации увеличивается. В результате процедуры параметрического представления речи мы получаем речевой сигнал, описанный последовательностью некоторых векторов признаков, в котором удалена избыточная информация. Подобные методы сжатия звуковых сигналов находят также применение в активно развивающихся интеллектуальных системах, предназначенных для целей телекоммуникаций и различных информационных сервисов. Примером использования сети LVQ для уменьшения количества информации, представляющей конкретный образ, может также считаться компрессия (с потерями) изображений. При «сжатии» сигналов изображений с помощью нейронных сетей векторного квантования поступают следующим образом. Входное изображение разбивается на кадры размером m x n пикселов. Каждый кадр представляется в виде вектора, содержащего $3mn$ компонент. С помощью полученных таким образом данных обучают нейронную сеть, состоящую из N нейронов. В процессе предъявления очередного кадра выбирается номер нейрона победителя и формируется кодовая книга. После обучения веса и номера нейронов-победителей для каждого кадра сохраняются в файле. Поскольку количество нейронов намного меньше количества кадров, то можно получить существенное сокращение объема данных, описывающих изображение. Системы такого типа позволяют получить степень компрессии изображений порядка 16 при значении коэффициента сигнал/шум около 27 дБ.

6. ДИНАМИЧЕСКИЕ НЕЙРОННЫЕ СЕТИ

6.1. Общие сведения

Рассмотрим теперь отдельную группу искусственных нейронных сетей, которые являются рекуррентными, имеют обратные связи между различными слоями нейронов и обладают «ассоциативной памятью». Для таких сетей, прежде всего, характерно то, что они представляют собой динамические системы. То есть такие сети являются системами, состояния которых изменяются во времени.

Для лучшего понимания природы систем подобного рода и понятий, употребляемых в таких сетях, вспомните, что с динамическими явлениями вы уже встречались на ранних этапах своего обучения и в жизни. Вы учили, например, что во время пуска электрического двигателя его ротор из установившегося состояния покоя переходит в состояние равномерного движения. При этом отмечали, что в процессе пуска имеет место переходный процесс. Он характеризуется промежутком времени, в течение которого происходит переход от одного установившегося состояния (соответствующего покою) к другому установившемуся состоянию (соответствующему равномерному вращению ротора). Во время пуска двигатель некоторое время пребывал в переходном состоянии. С физической точки зрения переходное состояние характеризовалось тем, что происходило изменение энергетических условий работы двигателя. Заметим, что переходные процессы, вызванные, к примеру, изменением нагрузки на валу, и изменения энергетики при переходе от одного установившегося состояния к другому имеют также место во всех системах автоматического регулирования поведения электродвигателей. Известно также, что практическая пригодность схем на операционных усилителях в большой степени зависела от того, как в них протекают переходные процессы. Итак, мы вспомнили, что для динамических систем очень важен переходный процесс. Наш экскурс в «глубины памяти» был бы не полным, если бы мы не отметили несколько принципиально важных фактов, вытекающих из оценок поведения динамических систем в неустановившемся (переходном) режиме.

Во-первых. Для достижения требуемого качества переходного процесса динамическая система должна быть структурно реализована в виде замкнутых контуров, то есть в форме системы с обратной связью. Хорошо известно, что обратная связь оказывают существенное влияние на длительность переходного процесса. Кроме того, было установлено, что динамические системы с обратной связью обладают тем преимуществом, что в них влияние многих негативных факторов может быть существенно снижено.

Во-вторых. Для динамической системы очень важно свойство устойчивости, то есть способности системы возвращаться в состояние устойчивого равновесия после исчезновения внешних сил, которые вывели ее из этого состояния. Неустойчивые динамические системы абсолютно непригодны к применению на практике, поскольку они не способны возвращаться к своему прежнему состоянию после устранения причин, вызвавших изменение этого состояния.

И в-третьих. Поскольку в динамической системе состояния изменяются во времени, то для представления эволюции системы всегда нужно правильно описывать переход из одного состояния в другое. В качестве математической модели, описывающей динамику системы, в настоящее время широко используют понятие (модель) пространства состояний (state –space model). В данной модели для удобства выкладок все состояния системы собраны в вектор $x(t)$, размерности n x 1, который называется вектором состояний. При таком подходе наблюдаемое состояние системы (вектор состояния $x(t)$) в конкретный момент времени t представляется точкой в N - мерном пространстве состояний. Изменение состояния системы во времени (описывающее движение точки N - мерном пространстве состояний) представляется кривой в пространстве состояний, каждая точка которой (явно или неявно) характеризуется меткой времени наблюдения. Такая кривая называется траекторией (trajectory) или орбитой (orbit). Семейство траекторий, соответствующих различным начальным состояниям, называют портретом состояний (state portrait). Если представить, что пространство состояний как бы «течет» в каждой своей точке, подобно жидкости, по определенной траектории, то полезным окажется понятие поток динамической системы.

В курсе «Физика» вы уже получали «портреты» некоторых пространств состояния, в частности, двумерный портрет электрических ко-

лебаний, происходящих в последовательном соединении катушки индуктивности, конденсатора и резистора. На таком «портрете» вы видели, что при малых величинах сопротивления этой электрической цепи «изображающая» динамический процесс точка перемещалась по одной и той же замкнутой кривой на плоскости. Поэтому «портрет» на экране осциллографа выглядел в виде эллипса. При увеличении сопротивления резистора в электрическом контуре траектории, образующие портрет, «навивались», «стягиваясь» к некоторой точке. Форма траектории представляла собой логарифмическую спираль, наматывающуюся на начало координат. Точка, лежащая в начале координат, в которую сходились все траектории, называлась точкой устойчивого фокуса (рис. 6.1).

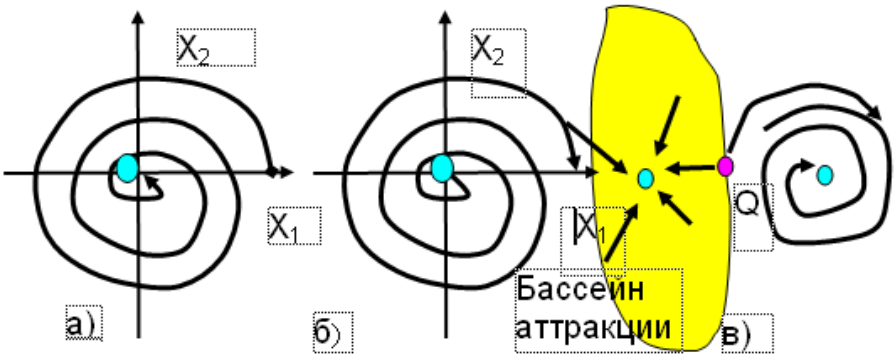


Рис. 6.1. **Портреты состояний некоторых динамических систем в пространстве: а) устойчивый фокус (притягивающая точка); б) неустойчивый фокус (отталкивающая точка); в) изображение бассейна аттракции и сепаратисы**

Исследования в области нелинейных динамических систем показали, что, существует несколько портретов систем (для систем второго порядка их, к примеру, 6), которые дают визуальное описание внутренних тенденций динамической системы, представляют перемещения в любой точке пространства состояний. Было установлено, что в таких портретах существуют притягивающие точки, когда орбиты всех точек из ее некоторой окрестности сходятся к ней, а также отталкивающие точки, когда орбиты всех достаточно близких к ней точек удаляются от нее. Из рис. 6.1 несложно понять следующее. Если точка пространства состояний находится в начале координат, то отклонение ее от этого положения под действием каких то сил (возмущений) может привести либо к тому, что система вернется в эту начальную точку, либо уйдет от нее. Как по-

ведет себя система в конкретной ситуации, зависит от того, является ли система устойчивой. Заметим, что понятие устойчивости нелинейных динамических систем является достаточно сложным и имеет несколько толкований. Для нелинейных динамических систем, и изучаемых нами рекуррентных, имеющих обратные связи между различными слоями нейронов нейронных сетей, под этим обычно подразумевают устойчивость по Ляпунову (stability in the sense of Lyapunov). Идею фундаментальной теории устойчивости русского математика и инженера Ляпунова можно изложить, рассматривая портрет на плоскости, таким упрощенным образом. Если невозмущенному движению системы соответствует состояние покоя (то есть нахождение точки в начале координат), то это движение устойчиво, если все траектории, начинающиеся внутри окружности достаточно малого радиуса δ, не выходят из окружности или, другими словами, траектория, описываемая вектором состояния, всегда достигает исходной точки при устремлении времени к бесконечности. Невозмущенное движение асимптотически устойчиво, если все возможные траектории, начинающиеся внутри окружности радиуса δ, стремятся к началу координат. Невозмущенное движение неустойчиво, если существуют траектории, начинающиеся как угодно близко к началу координат, выходящие за пределы некоторой фиксированной окружности с радиусом δ. Устойчивые и асимптотически устойчивые движения характеризуют равновесные состояния динамической системы. Последние в пространстве состояний, согласно теории Ляпунова, определяют локальные энергетические минимумы. Они в пространстве состояний представлены точками или совокупностью точек «стабильности», называемых аттракторами. Аттрактор может быть точечным и состоять из одной точки в пространстве состояний. Он может представлять собой множество точек, например, принимать форму периодической орбиты в виде эллипса; в этом случае часто говорят об устойчивом предельном цикле. «Ближайшее окружение» «тяготеет» к аттрактору. По этой причине каждый из аттракторов окружен собственной четко очерченной областью «притяжения» (рис. 6.1), на которую распространяется его «действие». Такая область называется бассейном аттракции (basin of attraction). Граница, отделяющая один бассейн притяжения от другого, называется сепаратрисой (separatrix). Начальное состояние динамической системы всегда находится в бассейне какого-либо аттрактора. В

случае, показанном на рис. 6.1 имеют место два аттрактора, а граница бассейна представлена кривой, на которой расположена точка Q.

6.2. Нейронная сеть Хопфилда

Область знаний, в которой нейронные сети рассматриваются как нелинейные динамические системы и основной упор делается на проблему устойчивости (stability), называется *нейродинамикой*. Поэтому, после рассмотрения общих вопросов, приступим теперь к изучению нейродинамической нейронной сети, которую в литературе обычно называют нейронной сетью Хопфилда. В такой сети в ходе динамического (рекуррентного) обучения выполняется размещение аттракторов в определенных «местах» пространства состояний, в результате чего появляется возможность «запоминать» некоторые векторы и использовать сеть в виде некого устройства «памяти».

Сеть Хопфилда состоит из множества нейронов, представленных в виде одного слоя. Нейроны имеет биполярную функцию активации со значениями ± 1. Каждый нейрон имеет один вход, на который подается входной сигнал (вектор). Сигнал с выхода каждого нейрона, помимо того, что он как обычно поступает на выход сети, подается на все другие нейроны, одновременно с входным. Тем самым в сети осуществляется обратная связь выхода с входом. Выход каждого нейрона соединяется с соответствующим входом через элемент единичной задержки, в результате образуется множество «задержанных» выходных сигналов, формирующих систему с множеством обратных связей (multiple – loop feed back system). При этом нейрон в сети Хопфилда не имеет обратной связи (автосвязи) с самим собой, что соответствует $w_{ii} = 0$. Веса сети являются симметричными, то есть вес связи между i-м и j- м нейронами равен весу связи между i-м и j- м нейронами $w_{ij} = w_{ji}$. Отсутствие автосвязи и симметричность матрицы весов являются достаточными (но не необходимыми) условиями сходимости переходных (итерационных) процессов в сети Хопфилда.

На вход сети подаются образы, описываемые N-мерными (N – число нейронов) векторами $p = (p_1, p_2, ..., p_N)^T$ с компонентами, принимающими двоичные значения: -1 и +1. После того, как на вход подан N -

мерный вектор, нейронная сеть формирует выход, который, по обратной связи подается на вход. Этот процесс повторяется много раз, поэтому благодаря обратной связи при подаче сигнала на вход в сети возникнет переходный процесс, который будет продолжаться до тех пор, пока система выходов не установится в одно из состояний равновесия. Сеть Хопфилда является рекуррентной в том смысле, что для каждого входного сигнала нейрона его выход повторно, многократно используется в качестве ввода до тех пор, пока не будет достигнуто устойчивое состояние. Рекуррентные динамические системы в процессе своей работы (во время переходного процесса) могут находиться либо в «хаотическом» движении, либо в состоянии неких осцилляций, либо стремиться к некоторому устойчивому состоянию. Для получения ответа на вопрос, что, в конце концов, произойдет с сетью, была использована теория Ляпунова.

Переходный процесс в нейронной сети, состоящей из N взаимосвязанных нейронов, меняющей в процессе вычислений свои состояния, в теоретическом плане, можно представить системой нелинейных дифференциальных уравнений первого порядка. В связи с этим работа сети может интерпретироваться как решение задачи минимизации функции энергии (Ляпунова). При этом эволюция во времени сети Хопфилда будет представлять собой некоторую траекторию в пространстве состояний, приходящую в некоторую фиксированную точку функции энергии. Доказано, что если матрица весовых коэффициентов является симметричной, то функция энергии (Ляпунова) сети Хопфилда является монотонно убывающей функцией времени. При этом сеть Хопфилда является глобально асимптотически устойчивой, что обеспечивает обязательное движение системы к локальным точкам минимума функции энергии. Эти фиксированные локальные точки минимума функции энергии являются аттракторами. Тем самым изменение состояния какого-либо нейрона инициализирует изменение энергетического состояния сети в направлении минимума ее энергии вплоть до достижения локальных энергетических минимумов в пространстве состояний. В пространстве состояний локальные энергетические минимумы представлены точками стабильности, называемыми аттракторами из-за тяготения к ним ближайшего окружения.

Сказанное по поводу обучения сети Хопфилда можно пояснить с помощью «карты» энергии (energy contour map) или, по-другому, в терминах «ландшафта энергии» (energy landscape). Функция энергии сети

Хопфилда представляет условно «гористую местность», на вершине которой находится шар, представляющий таким образом входной сигнал. Шар, после активизации сети, начинает катиться вниз по склону, пока не остановится в какой-либо локальной впадине. Данная впадина «лучше всего подходит» определенному предъявленному сети вектору. Другой входной вектор, обладая иной «потенциальной» энергией, может спускаться с другой скоростью и достигнуть иной, нежели у первого, низины. Но в обоих случаях это будут устойчивые точки локального минимума энергии на «карте» энергии. Кроме того, другие шары, попавшие на склон по которому двигался «пробный» шар, обязательно повторят его «путь» и, вероятней всего, попадут в ту же точку локального минимума. Все шары, попавшие на «свой» склон, будут «притянуты» к одной и той же точке минимума (к одному и тому же аттрактору).

В случае, когда на вход нейронной сети подаются двоичные биполярные значения: -1 и +1, динамика работы нейронной сети «направит» траекторию в пространстве состояний в некоторую фиксированную точку (аттрактор), расположенную в одном из углов единичного гиперкуба, в котором функция энергии минимальна.

Следует иметь в виду, что активация сети Хопфилда может осуществляться асинхронно, когда на каждом такте только один случайно выбранный нейрон изменяет свое состояние, и синхронно, когда все нейроны изменяют свое состояние одновременно

Способность сети создавать аттракторы послужила основанием для использования ее в качестве устройства ассоциативной памяти.

Вы, наверное, много раз уже слышали слово ассоциация: с конкретным человеком мы связываем его e-mal, адрес; взгляд на фотографию друга у нас вызывает воспоминания его имени. Ассоциация – это явление, когда одно представление вызывает по сходству, смежности или противоположности другое. При ассоциации образуются некие связи между явлениями, что вызов одного из них приводит к вызову другого. Немаловажным является и то обстоятельство, что с помощью ассоциаций мы можем решать такую важную задачу, как восстановление искаженной или зашумленной информации. По части слова, оставшейся на клочке бумаги, или по сокращению, в виде заглавных букв, мы сможем понять все слово и определить о чем в нем идет речь. Мы в состоянии вспомнить человека, даже если увидели часть его лица.

Нейронная сеть Хопфилда оказалась способной решать некоторые задачи из области ассоциаций. В частности она способна «запомнить», а затем и «воспроизвести», пользуясь «ассоциациями», сохраненный в ней «образец» при подаче на нее разумного подмножества информации об этом образце. Более того, ассоциативная сеть позволяет корректировать ошибки, а также аннулировать «лишнюю» информацию, не относящуюся к образцу, если таковая содержится в нем. Если, к примеру, известен некоторый набор двоичных сигналов, которые считаются образцовыми (цифровых, звуковых или прочих данных, описывающих некоторые объекты или характеристики процесса), то нейронная сеть должна уметь из поданного на ее вход неидеального (зашумленного) сигнала, выделить («вспомнить по частичной информации») соответствующий образец (если такой есть) или «дать заключение» о том, что входной сигнал не соответствует ни одному из образцов.

Идея работы ассоциативной памяти состоит в следующем. Требуемые для сохранения образцы «записываются» в ячейки «фундаментальной памяти» (fundamental memories). В качестве таких ячеек выступают фиксированные точечные аттракторы с четко определенными бассейнами аттракции. Бассейны делят в процессе записи пространство состояний на соответствующее множество четко очерченных областей. Пусть теперь нейронной сети представлен вектор, "похожий" (в некотором выбранном смысле) на один из запомненных ранее образцов, и, соответственно, содержащий, хотя и частичную, но существенную информацию об одной из ячеек фундаментальной памяти. Этот вектор можно представить как начальную точку в пространстве состояний. Полагая, что эта стартовая точка находится достаточно близко к точке, представляющей «запрашиваемый» образец (то есть стартовая точка лежит в бассейне аттракции, принадлежащем «запрашиваемой» точке), можно предположить, что в ходе переходного процесса сеть проведет вычисления таким образом, что, в конце концов, траектория движения точки придет к состоянию ячейки фундаментальной памяти. Когда сеть «опознает» предъявленный ей входной сигнал, она сформирует вектор выходных сигналов, компоненты которого совпадут с соответствующими компонентами фундаментальной ячейки памяти. Тем самым на выходе сети появится «запрашиваемый» образец, причем в полном виде и без искажений. Такой способ организации памяти и выборки данных назы-

вается адресацией по содержимому, в отличие от адресации по номеру ячейки памяти, принятому в большинстве электронных устройств.

Сети Хопфилда, выступающие в качестве устройств ассоциативной памяти, имеют две фазы работы: фазу сохранения и фазу извлечения.

Предположим, что необходимо сохранить множество N-мерных векторов (двоичных слов), которые обозначены как $\{\xi_\mu\}$, $\mu = 1, 2, \ldots, M$. компоненты этих векторов принимают двоичные биполярные значения -1 и +1. Назовем эти векторы ячейками фундаментальной памяти, указывая этим на то, что в них сохраняются образы, представленные сети.

Работа сети Хопфилда во время фазы сохранения (storage phase) состоит в следующем. Предъявляем сети входной вектор, принадлежащий ячейке фундаментальной памяти. Мысленно разорвав цепочку обратной связи, рассчитываем значения выходов. Замкнув обратную связь, предоставляем сети возможность, меняя свое состояние, самостоятельно осуществить переходный процесс. Остановить переходный процесс (процесс релаксации) после того, как выходной вектор перестанет меняться, то есть по достижении минимума функции энергии.

Чтобы формировать в пространстве состояний траекторию движения к точке минимума функции энергии, совпадающей с решением задачи, расчет весовых коэффициентов сети проводят по формуле, являющейся обобщением постулата обучения Хебба. Синаптический вес, направленный от нейрона i к нейрону j, определяется так: $W = \frac{1}{N}\sum_{\mu=1}^{M} \xi_\mu \xi_\mu - MI$, где $\xi_{\mu,j}\xi_{\mu,i}$ - векторное произведение вектора ξ_μ самого на себя; I – единичная матрица.

Уместно еще раз упомянуть о том, что матрица W является симметричной, то есть $W^T = W$.

Во время фазы извлечения на вход сети Хопфилда, в качестве начального ее состояния, подается N-мерный вектор, называемый пробным или зондом (probe). Этот вектор представляет собой неполную или зашумленную версию какой-нибудь ячейки фундаментальной памяти сети. В ходе итераций в соответствии с некоторым динамическим правилом для всех нейронов j сети случайно, но на некотором фиксированном уровне, проверяют индуцированное локальное поле v_j. Если в некото-

рый момент времени v_j окажется больше нуля, то нейрон j будет переведен в состояние +1 (если, конечно, он уже не находился в этом состоянии). Аналогично, если в некоторый момент времени v_j окажется меньше нуля, то нейрон переключится в состояние -1. Если же v_j будет равен нулю, нейрон j останется в своем прежнем состоянии, независимо от того включенное оно или отключенное. Тем самым в сети осуществляется коррекция, производимая случайным образом. Процедура коррекции будет выполняться до тех пор, пока изменения v_j не перестанут происходить. Это значит, что, начиная с некоторого произвольного пробного вектора p, нейронная сеть создаст инвариантный во времени вектор состояний y, отдельные элементы которого будут удовлетворять условию устойчивости: $y_i = \mathrm{sgn}\left(\sum_{i=1}^{N} w_{ji} y_i + b_j \right)$. Данное условие устойчивости также называют условием выравнивания (alignment condition). Удовлетворяющий ему вектор состояния y называют устойчивым состоянием или фиксированной точкой пространства состояний.

Таким образом, если операция извлечения информации выполняется асинхронно, сеть Хопфилда всегда сходится к некоторому устойчивому состоянию. К сожалению, ячейки фундаментальной памяти не всегда являются устойчивыми. Более того, в сети Хопфилда могут появляться ложные состояния, являющиеся устойчивыми состояниями, но отличными от ячеек фундаментальной памяти. Считается, что пока количество M ячеек фундаментальной памяти будет оставаться малым, по сравнению с количеством N нейронов, нейронная сеть будет устойчивой в вероятностном смысле. По этой причине количество ячеек M фундаментальной памяти является мерой емкости памяти сети, а величина $\alpha = M/N$, называемая параметром загрузки (load parametr), характеризует отношение сигнал /шум. В статистической физике утверждается, что качество извлечения из памяти сети Хопфилда ухудшается при увеличении параметра загрузки. Критическим считается значение $\alpha = 0.14$ (отношение сигнал/шум примерно равно 7 или 8,45 дБ). При превышении данной критической отметки, восстановление данных из памяти становится невозможным. Критическое значение $M_{kp} = 0.14N$

определяет емкость памяти с ошибками. Емкость памяти с вспоминанием практически без ошибок (наибольшее количество ячеек фундаментальной памяти в сети при требовании корректного вспоминания) определяется формулой $M_{\max} = n/2\ln N$.

Ассоциативная память может быть реализована и в виде нейронной сети с положительной обратной связью и амплитудным ограничением (positive feedback system with amplitude limitation). Нейронную сеть такого типа часто называют сетью на основе модели BSB (brain-state-in-box), поскольку ее состояние ограничено гиперкубом [+1,-1] с центром в начале координат. Структура и принцип работы сети BSB аналогичны структуре и принципу работы сети Хопфилда. Отличие состоит в том, что в сети BSB допускается наличие собственных обратных связей (замыкание у нейрона выхода и входа). Кроме того, наличие положительной обратной связи приводит к тому, что подаваемые на входы сети сигналы усиливаются до тех пор, пока все нейроны не войдут в насыщение.

Равновесные состояния модели BSB определены не только в некоторых углах единичного гиперкуба, но и в начале координат. Из-за этого любые изменения вектора состояния, независимо от того насколько малыми они будут, усиливаются положительными обратными связями и «уводят состояние» от начала координат в направлении некоторой устойчивой конфигурации. Сети BSB могут использоваться для кластеризации двоичных данных. Это следует из того факта, что устойчивые углы единичного гиперкуба выступают в роли точечных аттракторов с четко выраженными бассейнами аттракции. Эти бассейны делят пространство состояний на соответствующее множество четко очерченных областей. Поэтому в BSB реализуется алгоритм кластеризации без учителя, в котором все устойчивые углы единичного гиперкуба представляют собой кластеры рассматриваемых данных.

6.3. Компьютерное моделирование ассоциативных нейронных сетей

Рассмотрим теперь некоторые результаты компьютерного моделирования ассоциативных нейронных сетей. Пусть нейронная сеть состоит из четырех нейронов. Вводим в командное окно программу:

```
clear,
disp('Вх-й сигнал для обучения
сети')
%Определяем входной двумерный
сигнал, состоящий из 4
% векторов по 2 компоненты
T = [1 -1; -1 1;  1 1; -1 -1]'
%Строим график расположения
векторов входа
figure(1), clf, plot(T(1,:),
T(2,:),'*b','MarkerSize',15)
axis([-1.1 1.1 -1.1 1.1])
xlabel('a(1)'), ylabel('a(2)')
title('Input Vectors');
%Создаем сеть BSB
net = newhop(T);
disp('Синаптические веса и пороги
сетиво время фазы сохранения')
W= net.LW{1,1}
b = net.b{1,1}
Ai = T;
%Проверяем работу сети во время
фазы извлечения
% В слое будет 4 нейрона
NN=4;
disp('Выходной сигнал и задержки')
disp('во время фазы извлечения')
[Y,Pf,Af] = sim(net,NN,[],Ai)
%Строим график, иллюстрирующий, как во
%время фазы извлечения пробные случайные
данные стремятся к ячейкам
% фундаментальной памяти
 figure(2), clf, plot(T(1,:), T(2,:),'*r', 0,0,'rh'),
hold on, axis([-1.1 1.1 -1.1 1.1])
xlabel('a(1)'), ylabel('a(2)')
new = newhop(T);
[Y,Pf,Af] = sim(net,4,[],T);
for i =1:20
   a = {rands(2,1)};
   [Y,Pf,Af] = sim(net,{1,20},{},a);
   record = [cell2mat(a) cell2mat(Y)];
   start = cell2mat(a);

plot(start(1,1),start(2,1),'kx',record(1,:),record(2,:))
end
```

После запуска программы получим графическое изображение результатов реализации ассоциативной памяти (рис. 6.2).

Из рис. 6.2 видно, что во время фазы сохранения были получены 4 ячейки фундаментальной памяти [1 -1], [-1 1], [1 1], [-1 -1], со своими бассейнами аттракции. В правом нижнем углу пунктиром показана примерная конфигурация одного из бассейнов аттракции. Векторы, поступившие на вход ассоциативной нейронной сети, в зависимости от того в какой бассейн аттракции они попали, движутся в пространстве состояний к точкам минимума энергии, где они достигают состояния устойчивости. Пути перехода сети из начального в конечное состояние (пути достижения устойчивого состояния) обладают некоторой неопределенностью, связанной с наличием «случайностей» в алгоритме вычислений. Потоки динамической системы, подобно жидкости, протекают по определенным траекториям к «своей» точке пространства состояний. На рис. 6.2 направление отдельных потоков показано пунктирной линией со

стрелкой на конце. Карта потоков демонстрирует определенную симметрию по отношению к четырем устойчивым состояниям. При этом несложно заметить, что все входные векторы в фазе извлечения информации «сходятся» к устойчивым точкам пространства состояний. Результаты расчетов, подтверждают тот факт, что для сходимости синаптические веса сети должны быть симметричными $w_{ij} = w_{ji}$: (1.1618 0.0000; 0.0000 1.1618). Устойчивые углы квадрата фактически представляют собой «кластеры» рассматриваемых данных. Для рассматриваемого случая ассоциативная нейронная сеть формирует четыре бассейна аттракции (рис. 6.3).

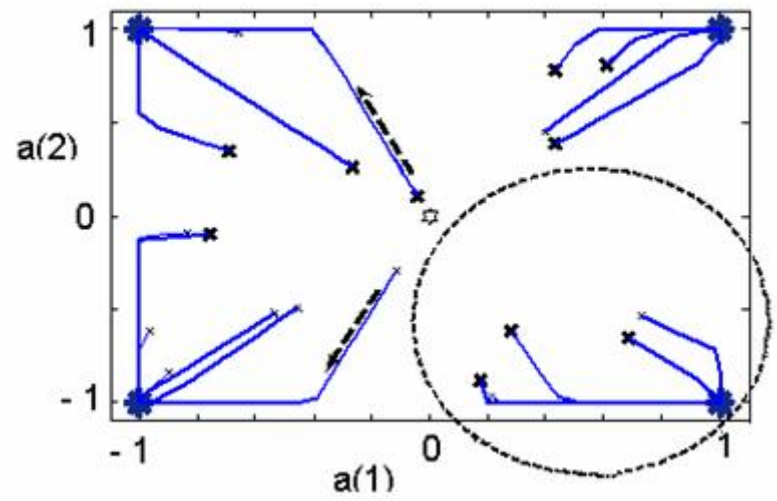


Рис. 6.2. **Визуальное представление процессов кластеризации двоичных данных на плоскости: (+) – входные данные; ячейки фундаментальной памяти (аттракторы) расположены в углах квадрата (*); траектории стремятся к точечным аттракторам**

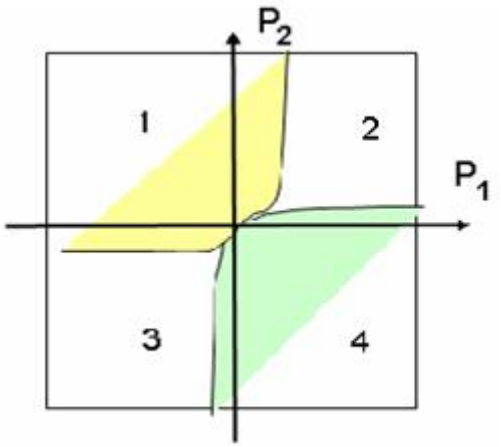


Рис. 6.3. **Бассейны (области) аттракции при четырех фундаментальных ячейках памяти, в углах квадрата**

Следует обратить внимание, что сепаратрисы (границы, отделяющие один бассейн притяжения от другого) не совпадают с серединами квадрата (с осями координат).

Рис. 6.4 наглядно свидетельствуют о том, что от начального состояния пробного вектора в конкретном бассейне аттракции определяет траекторию движения к ячейке фундаментальной памяти. Особый интерес представляют траектории, когда начальное состояние расположено в одном квадранте, а траектории заканчиваются в другом (рис. 6.4а и г).

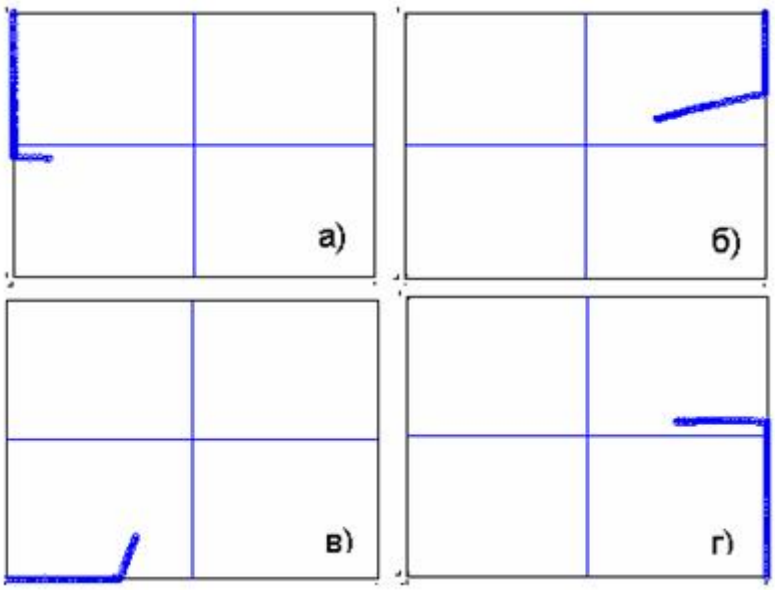


Рис. 6.4. **Траектории при начальном положении входного вектора в бассейне аттракции 1– 4 (а – г соответственно)**

Изучим теперь работу ассоциативной сети при подаче на ее вход трехмерных векторов. Для этого введем в командное окно программу.

```
clear,
disp('Вх-й сигнал для обучения сети')
%Определяем входной двумерный сигнал,
состоящий из 3 векторов
% по 2 компоненты
T = [+1 +1;-1 +1;-1 -1]
 %Строим график расположения векторов
входа
figure(1), clf,
axis([-1 1 -1 1 -1 1])
set(gca,'box','on'); axis manual;  hold on;
%Строим график, иллюстрирующий
то, как во время фазы извлечения пробные
% случайные данные стремятся
% к ячейкам фундаментальной памяти
figure(2), clf,
axis([-1 1 -1 1 -1 1])
set(gca,'box','on'); axis manual;  hold on;
plot3(T(1,:),T(2,:),T(3,:),'*b','MarkerSize',15)
title('Hopfield Network State Space')
xlabel('a(1)');
ylabel('a(2)');
```

```
plot3(T(1,:),T(2,:),T(3,:),'*b','MarkerSize',15)
title('Input Vectors');
xlabel('a(1)');
ylabel('a(2)');
zlabel('a(3)');
view([37.5 30]);
 %%%%Создаем сеть BSB
net = newhop(T);
disp('Синаптические веса и пороги сети во
время фазы сохранения ')
W= net.LW{1,1}
b = net.b{1,1}
%Проверяем работу сети во время фазы
извлечения
% на входном сигнале
a = {rands(3,1)};
[y,Pf,Af] = sim(net,{1 10},{},a);
clear,
disp('Вх-й сигнал для обучения сети')
%Определяем входной двумерный сигнал,
состоящий из 3 векторов
% по 2 компоненты
T = [+1 +1;-1 +1;-1 -1]
 %Строим график расположения векторов
входа
figure(1), clf,
axis([-1 1 -1 1 -1 1])
set(gca,'box','on'); axis manual;  hold on;
plot3(T(1,:),T(2,:),T(3,:),'*b','MarkerSize',15)
title('Input Vectors');
xlabel('a(1)');
ylabel('a(2)');
zlabel('a(3)');
view([37.5 30]);
 %%%%Создаем сеть BSB
net = newhop(T);
disp('Синаптические веса и пороги сети во
время фазы сохранения ')
W= net.LW{1,1}
zlabel('a(3)');
view([37.5 30]);
record = [cell2mat(a) cell2mat(y)];
start = cell2mat(a);
hold on
plot3(start(1,1),start(2,1),start(3,1),'bx', ...
  record(1,:),record(2,:),record(3,:))
 color = 'rgbmy';
for i=1:25
  a = {rands(3,1)};
  [y,Pf,Af] = sim(net,{1 10},{},a);
  record=[cell2mat(a) cell2mat(y)];
  start=cell2mat(a);
  plot3(start(1,1),start(2,1),start(3,1),'kx', ...
    re-
cord(1,:),record(2,:),record(3,:),color(rem(i,5)+1))
end
%Строим график, иллюстрирующий
то, как во время фазы извлечения пробные
% случайные данные стремятся
% к ячейкам фундаментальной памяти
figure(2), clf,
axis([-1 1 -1 1 -1 1])
set(gca,'box','on'); axis manual;  hold on;
plot3(T(1,:),T(2,:),T(3,:),'*b','MarkerSize',15)
title('Hopfield Network State Space')
xlabel('a(1)');
ylabel('a(2)');
zlabel('a(3)');
view([37.5 30]);
record = [cell2mat(a) cell2mat(y)];
start = cell2mat(a);
hold on
plot3(start(1,1),start(2,1),start(3,1),'bx', ...
  record(1,:),record(2,:),record(3,:))
 color = 'rgbmy';
for i=1:25
  a = {rands(3,1)};
  [y,Pf,Af] = sim(net,{1 10},{},a);
```

```
b = net.b{1,1}
%Проверяем работу сети во время фазы извлечения
% на входном сигнале
a = {rands(3,1)};
[y,Pf,Af] = sim(net,{1 10},{},a);
  record=[cell2mat(a) cell2mat(y)];
  start=cell2mat(a);
  plot3(start(1,1),start(2,1),start(3,1),'kx', ...
    re-
cord(1,:),record(2,:),record(3,:),color(rem(i,5)+1)
)
```

После запуска программы получим трехмерное графическое изображение результатов реализации ассоциативной памяти (рис. 6.5). Из рис. 6.5 следует, что в этом случае в сети было создано две ячейки фундаментальной памяти с выраженными бассейнами аттракции.

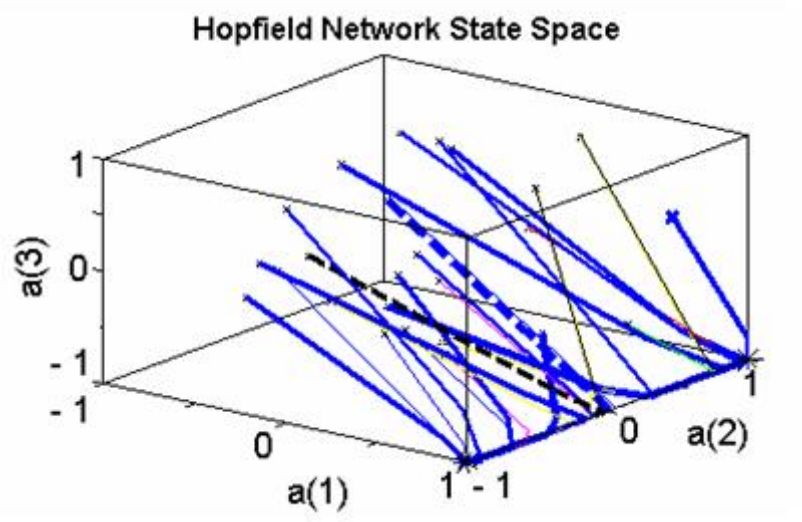


Рис. 6.5. **Визуальное представление процессов кластеризации двоичных данных в пространстве: ячейки фундаментальной памяти (аттракторы) расположены в углах куба (*); траектории стремятся к точечным аттракторам**

Динамическое изменение состояния сети (во время фазы извлечения информации) обеспечивают изменение ее состояния в направлении точечных аттракторов системы. Появление на входе сети какого-либо сигнала приводит к тому, что сеть по истечении некоторого времени окажется в одном из устойчивых состояний находящихся в вершине куба. Нейронная сеть и в этом случае может быть использована в целях кластеризации без учителя (для структурирования данных и формирования понятий). Кроме того, сеть может быть реализована в качестве устройства ассоциативной памяти, которая включает в себя восстановление сохраненных образов при предъявлении ей неполных или зашум-

ленных версий входных данных. Следует обратить внимание на то, что в кубе использованы только две устойчивые вершины. Остальные углы единичного куба являются потенциальными местоположениями ложных (spurious) состояний, также называемых ложными аттракторами. Ложные состояния представляют отличающиеся от ячеек фундаментальной памяти устойчивые состояния сети, из-за которых ухудшается качество извлечения состояния сети. Для иллюстрации работы нейронной сети, выступающей в качестве ассоциативной памяти, проведем компьютерный эксперимент по запоминанию и извлечению цифроподобных черно-белых образов. На вход сети подавались ячейки фундаментальной памяти, каждая из которых представляла цифроподобный черно-белый образы, составленный из 120 пикселей (рис. 6.6).

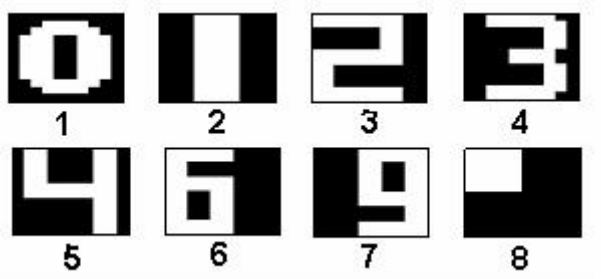


Рис. 6.6. **Множество цифроподобных черно-белых образов, используемых для компьютерного моделирования ассоциативной нейронной сети**

Для входов, применяемых к сети, предполагалось, что белому цвету соответствует значение +1, черному – -1. На основе этих образов во время фазы запоминания создавалась (в асинхронном режиме) матрица синаптических весов сети. Для проверки способности сети корректно восстанавливать из информации, сохраненной в матрице синаптических весов, во время первой части фазы воспоминания на вход сети подавались ячейки фундаментальной памяти. Желаемый образ восстанавливался сетью. Затем, чтобы продемонстрировать способность сети работать в качестве устройства ассоциативной памяти, на вход сети подавался образ, преднамеренно искаженный при помощи воздействующих на пиксели «помех» в виде случайно выбираемых и инверсно изменяемых точек. На рис. 6.7а,б показаны результаты эксперимента для цифры 3 при различном уровне помех.

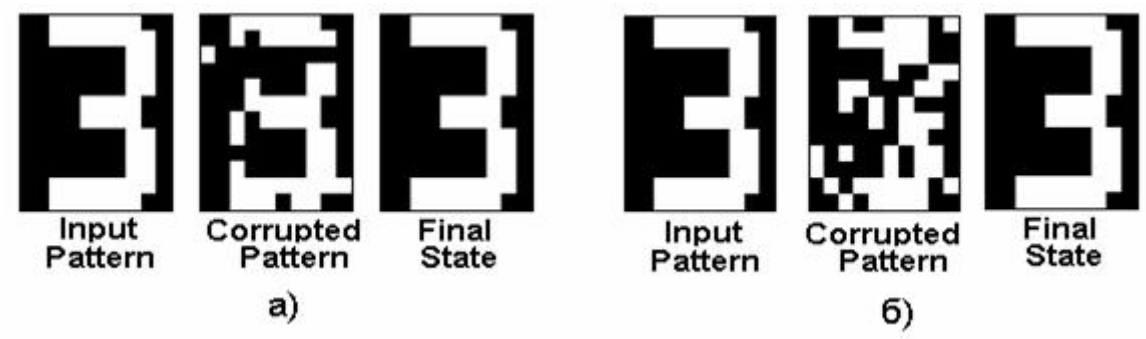


Рис. 6.7. **Изменение (а) и восстановление при большом уровне помех (б) образа цифры 3 нейронной ассоциативной сетью во время фазы вспоминания**

Образ, показанный на рис. 6.7а в центре, представляет собой искаженную помехами версию цифры 3. Этот образ подавался на вход сети во время фазы воспоминания. По мере увеличения количества итераций сеть «восстанавливала» искаженный образ, увеличивая сходство между выходом сети и цифрой 3. После 35 итераций выход сети, показанный на рис. 6.7а справа, сошелся к абсолютно корректной форме цифры 3. На рис. 6.7б показано успешная сходимость к фундаментальной ячейке памяти при достаточно сильном уровне помех.

Примерно за такое же количество итераций при относительно небольшом уровне помех к корректной форме сходились искаженные образы цифр 0, 1, 2, 4, 6, 9, и точка. Эксперименты подтвердили и тот факт, что для сети ассоциативной памяти представляют проблему ложные состояния или ложные аттракторы. Рис. 6.8а показывает, что при достаточно высоком уровне помех искаженная версия фундаментальной ячейки (образ цифры 2) сходится к ложному «зеркальному состоянию» цифры 2. Рис. 6.8б свидетельствует о том, что при достаточно высоком уровне помех сеть может сходиться к некорректной ячейке. В этом случае на вход сети подавался искаженный образ цифры 2, а после 50 итераций он сошелся к ячейке фундаментальной памяти цифры 6.

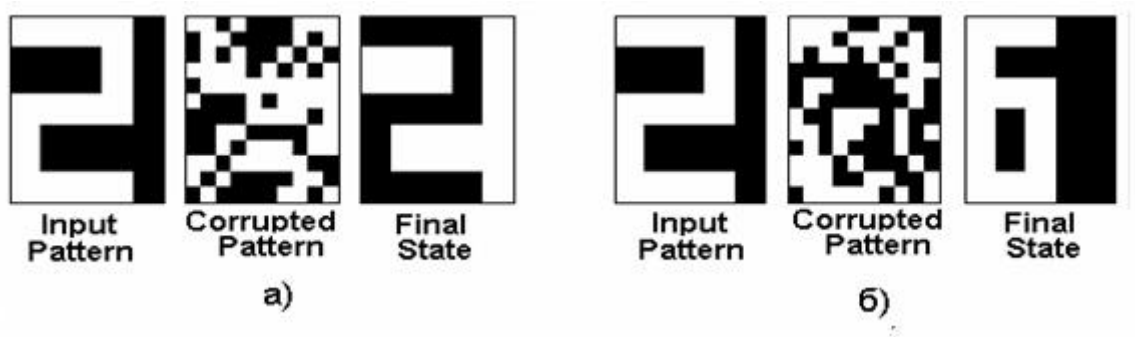


Рис. 6.8. **Схождение образа цифры 2 в нейронной ассоциативной сети во время фазы вспоминания к ложному «зеркальному состоянию» (а) и к фундаментальной ячейке образа 6 (б)**

Таким образом, динамические нейронные сети, обладающие большим количеством точечных аттракторов в пространстве большой размерности при относительно небольшом уровне помех способны восстанавливать сохраненные образцы при предъявлении им неполных или зашумленных данных. Чтобы сеть успешно сходилась к одному из эталонов, необходимо, чтобы количество образов, запомненных в сети, не превышало емкость сети, а векторы, сохраненные сетью, должны быть слабо коррелированы.

7. ПРИМЕНЕНИЕ НЕЙРОННЫХ СЕТЕЙ И НЕЙРОТЕХНОЛОГИЙ

7.1. Общие сведения

Искусственные нейронные сети, как было показано ранее, обладают некоторыми уникальными свойствами, которые практически невозможно получить при применении традиционных компьютеризированных программно-аппаратных средств. Несмотря на достаточно поверхностное сходство, искусственные нейронные сети демонстрируют некоторые свойства, присущие мозгу: массовый параллелизм; распределенность вычислительных процессов; возможность концептуальной обработки информации; низкое энергопотребление; толерантность к ошибкам. Нейронно-сетевые устройства могут функционировать в условиях, когда их входные данные не доопределены или определены нечетко. При этом отклик сети может быть до некоторой степени не чувствителен к небольшим изменениям входных сигналов или, другими словами, функционирование нейронных сетей робастно. По сравнению с иными информационными системами нейронные сети оказываются более предпочтительными когда нет возможности получить достаточное количество экспериментальных данных ввиду их высокой стоимости (при построении моделей тяжелых производственных аварий, пожаров и т. п.), неполноты и противоречивости, высокой зашумленности. Поставленная задача может быть решена нейросетью даже в тех случаях, когда входная информация не рассматривалась ранее при обучении (при условии, что обрабатываемые данные не выходят за предъявляемые к ним ограничения). При этом нейронные сети обладают способностью извлекать сущность из входных сигналов (абстрагироваться). Как отмечалось, нейронная сеть оказывается избирательно чувствительной к областям скопления данных. Преимущество нейронных сетей состоит и в их способности «выделять общие принципы» (проводить обобщение) при предъявлении некоторого набора обучающих векторов с неполным набором данных, получать некоторые тенденции при сравнении входных сигналов и откликов нейронной сети. При этом выявленные формальным путем тенденции или зависимости могут быть реально прослежены и имеют определенное «физическое» объяснение. Эта особенность выделять

образ сквозь шум и искажения позволяет преодолеть требования строгой точности, предъявляемые к обычным компьютеризированным системам. Важно отметить, что нейронная сеть делает обобщения автоматически благодаря своей структуре, а не с помощью «человеческого интеллекта», представленного в форме специально написанных компьютерных программ. Благодаря таким способностям искусственные нейронные сети обладают уникальными потенциальными возможностями по решению практических задач. Особенно их применение важно там, где человеческий интеллект малоэффективен, а вычисления с помощью традиционных компьютеризированных программно-аппаратных средств трудоемки и физически неадекватны (плохо отражают реальные физические процессы, происходящие в технических объектах). Актуальность применения нейронных сетей возрастает также тогда, когда появляется необходимость решения плохо формализуемых задач.

Нейронные сети, несмотря на то, что они не свободны от недостатков, широко применяют для решения ряда «интеллектуальных» задач. Получить общие представления о характере востребованности нейросетевых технологий позволяет приведенный ниже перечень таких задач (хотя он далеко не полон). Типовые задачи, решаемые с помощью нейронных сетей следующие: а) распознавание образов и автоматизация процессов классификации (кластеризации); б) обработка информации в областях, связанных с высокими технологиями и наукой; в) автоматизация процессов управления техническими системами; г) нахождение оптимальных путей передачи информационных потоков между «узлами» сетей связи и коммуникационных систем и пр.

7.2. Основные направления применения искусственных нейронных сетей

Описать подробно все примеры применения нейронных сетей даже в названных областях не представляется возможным, поэтому мы рассмотрим лишь некоторые примеры эффективного применения искусственных нейронных сетей.

Первый пример эффективного применения искуственных нейронных сетей касается разработки средств эффективного взаимодействия человека с компьютеризированными системами, созданию «дружест-

венных» SILK (Speech – речь, Image – образ, Language – язык, Knowledge – знание) интерфейсов. Это одно из приоритетных направлений развития искусственного интеллекта в настоящее время. Большие усилия разработчиков направлены на автоматизацию средств ввода-вывода информации в компьютер, распознавания звуковой и визуальной информации, иных потоков данных, которые поступают от человека и от окружающей его среды. При этом для решения проблемы человеко-машинного взаимодействия стараются использовать дополнительные «каналы» получения информации, к примеру, жесты, артикуляцию губ, мимику лица, направление взгляда и т. д. В результате стали разрабатываться так называемые многомодальные интерфейсы, в которых появилась возможность выбора того, какой коммуникативный канал (звуковой, визуальный, тактильный и пр.) или какой тип информации удобно использовать в данный момент. К примеру, вам наверно уже известно, что новейшие системы безопасности контролируют доступ в здание, используя множество каналов.

Нейронные сети являют собой те функциональные узлы интеллектуальной информационной системы, которые уже в настоящее время позволяют: а) при выполнении многомодальной обработки сигналов выявить характеристики поступающего потока данных, лучше всего отражающие входящую информацию; б) реализовать (по каждой модальности) наиболее совершенные методы получения признакового описания сигналов; в) осуществить, как само распознавание, так и (или) принятие решений в условиях недостаточной (зашумленной) информации. Уже было упомянуто, что нейронные сети применяются для распознавании печатных и рукописных букв и символов, предъявленного текста, для автоматизированного распознавания акустических речевых команд, слитной речи, для анализа и классификации изображений (снимков) и т. д. Согласованное использование информации от аудио- и видеораспознавателей позволяет получить результат совместного распознавания, который превышает точность распознавания по каждой из модальностей. При этом окончательное решение о распознанном сообщении может приниматься с учетом весовых коэффициентов каждой модальности, которые меняются в зависимости от окружающих условий (уровня шума, освещения и т. д.)

В качестве примера человеко-машинного интерфейса интеллектуальной информационной системы можно привести синтезатор речи, ко-

торый, можно сказать, исторически первым продемонстрировал эффективность применения нейронных сетей. Интерфейс состоит из двух модулей. На вход первого модуля, который представлял собой многослойную нейронную сеть, подавались сигналы, соответствующие 29 буквам английского алфавита (каждая буква представлялась 7 двоичными разрядами). Второй модуль представлял устройство, которое преобразовывало поступающие на его вход электрические сигналы в акустические (синтезировало звуки). Каждый нейрон выходного слоя нейронной сети выдавал сигнал на акустический синтезатор. В связи с тем, что в английском языке связь между буквами и фонемами не является точной, а зависит от контекста, влияющего на произношение той или иной буквы, в первом модуле сеть, помимо прочего, «вычисляла» влияние на рассматриваемую букву трех предыдущих и трех последующих букв. Обучение сети состояло в предъявлении текста, пошаговом его анализе и указании правильных фонем. После обучения система становилась способной образовывать последовательность фонем произвольного текста, то есть озвучивать любой текст.

Разновидностью описанного человеко-машинного интерфейса может служить система, предназначенная для моделирования изменений при разговоре мышц лица (мимики) или артикуляции губ. Такая система предназначена для компьютерной анимации, а также способствует пониманию механизмов синхронизации разных видов информации у человека. В качестве визуальных признаков могут служить геометрические характеристики лица и параметры губ (ширины рта, толщины верхней и нижней губ, видимость языка и зубов). Интерфейс, объединяющий речь, мимику и направление взгляда, может оказаться особенно полезным в системах, создаваемых для помощи людям со специфическими нуждами (например, инвалидам).

Второй пример эффективного применения искусственных нейронных сетей связан с интеллектуальной обработкой изображений, получаемых с помощью датчиков, с неким машинным «восприятием» изображаемых объектов, с реализуемым технически «зрением» оптических изображений. Помимо проведения «сжатия» изображений, о чем уже шла речь, с помощью нейронных сетей возможны классификация объектов по их изображениям, изучение различных «сцен», анализ визуальной информации, полученной в различных диапазонах длин электромагнитных волн (в рентгеновском, видимом, инфракрасном и т.д.). При этом

интеллектуальная система технического «зрения», выделяя различные классы объектов в изображении, классифицируя их, распознавая возможные виды расположения отдельных объектов (например, SMD резисторов на печатной плате) и их соединений, может выполнять проверку, в условиях производства, качества изготовления платы, комплектности деталей (установлен ли SMD резистор в требуемое место печатной платы или нет, правильно ли он отцентрирован, нет ли нарушенных или пропущенных соединений), классифицировать дефекты. Техническое «зрение» может быть также активным, когда компьютерная система становится способной выполнять те же задачи, что и биологическая зрительная система. В этом случае она приобретает еще в большей степени «интеллектуальные свойства», учится формировать и фокусировать «свои взгляды», чтобы определять по изображениям, «что присутствует в мире и где это находится». Примером подобной системы может служить обладающий машинным зрением подвижный промышленный робот, который используется для выполнения некоторых технологических операций (например, доставки рулонов бумаги к печатной машине). Робот перемещается на колесах, приводимых в движение электродвигателем, и имеет возможность поворачивать в любую сторону или разворачиваться на месте. Для обеспечения перемещений робота по промышленному помещению он оборудован системой навигации, построенной на основе распознавания образов различных участков помещения. На «теле» робота установлены датчики, определяющие расстояние до объектов, встретившихся на пути робота и окружающих его. Любое промышленное помещение, в котором перемещается робот, может быть разделено на ряд типовых объектов: коридор; перекресток; тупик; левый поворот; правый поворот; левый угол; правый угол. Сигналы навигации робота, полученные от датчиков и характеризующие расстояния до ближайших объектов, подаются на вход трехслойной нейронной сети. После обработки поступающих в сеть сигналов на выходе нейронной сети появляется комбинация сигналов, которая расшифровывается декодером и принимается решение об опознании того местоположения, где в данный момент находится робот. Центральная система управления роботом, имеющая задание на проведение каких-либо действий в той или иной части промышленного помещения, на основании информации о местонахождении робота принимает решение о направлении движения и подает управляющие сигналы на электродвигатели колес робота. В

результате обеспечивается возможность перемещения робота по производственному помещению без участия оператора, который только дает роботу задание на выполнение тех или иных операций.

Нейронные сети могут быть использованы для создания памяти, адресуемой по содержанию. В модели вычислений фон Неймана обращение к памяти доступно только посредством адреса, который не зависит от содержания памяти. Более того, если допущена ошибка в вычислении адреса, то может быть найдена совершенно иная информация. Ассоциативная память, или память, адресуемая по содержанию, доступна по указанию заданного содержания. Содержимое памяти может быть вызвано даже по частичному входу или искаженному содержанию. Ассоциативная память может быть эффективно применена при создании мультимедийных информационных баз данных.

Известно, что при разработке сложных технических систем, состоящих из огромного количества деталей и узлов, используют информационно-справочные системы, хранящие сведения обо всех этих деталях и узлах. В этих системах для кодирования наименований изделий используют специальную индексацию. Такой подход часто оказывается неэффективным, поскольку для нахождения нужной информации требуется достаточно много времени. В компании «Боинг» была создана нейросетевая информационно-справочная система, использующая геометрические сведения о детали, принятые в CAD- системах автоматического проектирования. Нейронная сеть, используя входные сигналы, отражающие различные комбинации очертаний, изгибов и отверстий, разбиение образов на группы, осуществляла поиск требуемых деталей.

Наконец, нейронные сети могут быть применены для управления реальными, по своей сути, существенно нелинейными объектами. Они хорошо подходят для расчета такого входного воздействия, при котором объект следовал бы по желаемой траектории управления, задаваемой эталонной моделью. При этом решение задачи управления представляет собой динамический процесс настройки сети.

Так, в частности, нейросетевые устройства могут быть использованы в системах векторного управления асинхронным электродвигателем (электроприводом). Для подобной системы управления, построенной по классическим методам, крайне необходимо точное определение параметров электродвигателя. Вследствие нагрева машины происходит изменение активных сопротивлений статора и ротора, что приводит к воз-

никновению ошибок оценивания магнитного потока, а это, в свою очередь, сказывается на снижении энергетической эффективности процесса преобразования энергии и даже потере устойчивости всей системы в целом. Для такой ситуации целесообразность применения нейросетевой системы управления обусловлена тем, что только такая система способна обеспечить устойчивость управления при изменении параметров объекта, поскольку только она устойчива к изменению параметров окружающей среды, а также может извлекать и использовать скрытую информацию. Кроме того, избыточность нейронной сети и параллельность обработки информации позволит получить высокие надежность и быстродействие.

Реализовать систему управления электроприводом целиком на нейронной сети весьма трудно, поскольку требуется сеть очень большой размерности, что влечет за собой значительный объем вычислений при обучении и моделировании. Поэтому целесообразно реализовать «гибридный» подход к созданию системы управления, когда нейросетевые устройства, как отдельные компоненты системы управления, осуществляют наиболее подходящие им функции.

В системе векторного управления асинхронным двигателем, из-за того, что часть параметров состояния электродвигателя не может быть измерена, используются специальные устройства, получившие названия наблюдатели состояния. Они воспроизводят не поддающиеся измерению переменные состояния. В частности, в системах управления электродвигателем часто применяются наблюдатели магнитного потока, которые в качестве входных сигналов используют информацию о напряжении (токе) в обмотках статора и скорости вращения вала асинхронного двигателя. Вы помните, что в случае векторного управления напряжение, подаваемое на двигатель, имеет вид широтно-модулированного сигнала (ШИМ-сигнала), а ток в обмотках асинхронного двигателя представляет собой так называемый «токовый коридор», в котором сигнал пульсирует с некоторой небольшой амплитудой вокруг заданного значения. Поэтому является целесообразным, чтобы в системе векторного управления асинхронным двигателем нейросетевой наблюдатель потокосцепления ротора осуществлял «сглаживание» сигнала с «токовым коридором» (выделять «гладкую» составляющую токового сигнала) и, на этой основе, рассчитывал такие потокосцепления, которые бы обесто-

чивали требуемое изменение частоты вращения вала и электромагнитного момента на валу при всех режимах работы электродвигателя.

Для решения первой задачи нейронная сеть, реализованная в виде отдельного модуля, обучалась на токовых сигналах, полученных для упрощенной модели асинхронного двигателя, без помех и с шумами. Искусственная нейронная сеть обучалась решать задачу аппроксимации, когда по некоторому количеству отсчетов сигнала «токового коридора» на выходе сети формировалось значение «гладкой составляющей». Как показали исследования, подобную задачу может решать двухслойная сеть. Кроме того, в обучающую выборку была «внесена информация» о нагреве обмоток статора и ротора, что приводит к вариации их сопротивлений и даже к их двукратному изменению. Обученная таким способом на упрощенной модели асинхронного двигателя нейронная сеть продемонстрировала хорошие обобщающие свойства при сглаживании сигналов реального двигателя.

Для решения второй задачи использовалась трехслойная нейронная сеть, выполненная в виде отдельного модуля. Частота токов ротора, напрямую определяется приложенным моментом нагрузки, поэтому именно эта переменная используется в качестве первой входной величины нейронной сети. Кроме того, на вход сети подавались сигналы токов фаз. Выходными сигналами нейронной сети являлись проекции потокосцепления ротора в неподвижной системе координат. При обучении нейронной сети использовались данные, соответствующие всему диапазону работы электродвигателя: изменению скорости вращения вала от нуля до скорости холостого хода; изменению электромагнитного момента двигателя от отрицательного до положительного максимума.

В результате было установлено, что искусственная нейронная сеть, обученная для решения задачи наблюдения за потокосцеплением ротора асинхронного электродвигателя, адекватно работает во всем диапазоне скоростей и моментов, а также проявляет свойство толерантности к вариациям активного сопротивления ротора и статора.

7.3. «Умные» датчики

В литературе, чтобы подчеркнуть их особую роль в оказании помощи человеку, к «интеллектуальным» иногда относят устройства, помогающие анализировать данные и выдавать их оператору, с комфорт-

ной для него скоростью, оценки, полученные в результате обработки приходящей информации. Такие функциональные узлы могут подстраиваться под требования и привычки пользователя, поэтому их даже называют «думающими», «разумными», «умными» (smart). Чаще всего такие устройства используются в компьтеризированных системах сбора информации. Поскольку такие устройства относят к «интеллектуальным», то обратим и на них внимание и рассмотрим их особенности.

Основой нашего познания окружающего мира являются экспериментальные данные, получаемые путем измерений. Лорду У. Кельвину приписывают следующее изречение: «Когда вы можете измерить то, о чем говорите, и сможете записать это в цифрах, вы знаете, о чем говорите». Большое значение придавали измерениям и многие другие великие ученные. Д. И. Менделеев говорил: «Наука начинается с тех пор, как начинают измерять». Г. Галилею принадлежит высказывание: «Считай то, что считаемо, измеряй то, что измеряемо; а то, что не измеряемо, делай измеряемым». Наше время, когда все возрастающий поток различных сведений и данных обрушивается на человека и на работе и в быту, можно с большим основанием назвать «веком информации». Существенной частью этого потока является измерительная информация, сопровождающая многие виды человеческой деятельности. Каждое мгновение на земле проводится громадное количество измерений, объем и точность которых непрерывно возрастают. Можно говорить об индустрии измерений, на выполнение и обработку которых затрачиваются огромные средства.

Одним из средств, предназначенных для сбора, обработки и ввода первичной информации в измерительную систему являются датчики. Поэтому мировой рынок всех датчиков оценивается величиной порядка 15-30 млрд. евро в год. Текущий рост продаж обычных датчиков составляет примерно 4% в год, а новейших конструкций – 10%. Промышленно развитые страны, как на государственном, так и на корпоративном уровне, уделяют вопросам создания надежных, точных, коммерчески привлекательных датчиков огромное внимание. Все понимают, что следствием усовершенствования датчиков будет улучшение качества и увеличение производительности в промышленности и других областях. Хорошие датчики это также одна из предпосылок для ускоренного внедрения новейших безлюдных технологий, развития робототехнических устройств, составляющих базу для построения гибких производственных

систем. При этом существует даже мнение, что прогресс в области автоматизации технологических процессов в настоящее время сдерживается качеством измерительных систем.

В настоящее время промышленностью выпускается огромная номенклатура датчиков, измеряющих параметры безопасности, текущие координаты, температуру, давление, скорость потока, влажность, ускорение, уровень жидкости, концентрацию ионов или растворенного кислорода и пр. Особое место при этом занимают датчики, предназначенные для восприятия и преобразования в электрический сигнал информации о пространственном перемещении механических и светоизлучающих объектов, в том числе и рабочих органов оборудования и робототехнических комплексов. Канули в лету датчики, представляющие собой сложные механические системы. Их заменили малогабаритные (миниатюрные) полупроводниковые и оптоэлектронные датчики, которые имеют низкую энергоемкость, обеспечивают большой ресурс работы и отказоустойчивость, имеют высокую чувствительность к входному воздействию, достаточную точность и достоверность считывания, а также совместимы по виду, уровню сигналов, по питанию с микроэлектронными схемами. Такие датчики относительно просто изготовить стандартными методами технологии полупроводникового приборостроения.

Проникновение микропроцессорных средств в измерительную технику позволило существенно улучшить многие характеристики средств измерений, придало им новые свойства, открыло пути решения задач, которые ранее вообще не ставились. С помощью микропроцессоров, встроенных в измерительные устройства, достигается многофункциональность, упрощение управлением измерительной процедурой, автоматизация регулировок, самокалибровка, выполнение статистической обработки результатов наблюдений.

Современную микроэлектронику в последние годы трудно представить без такой важной составляющей, как микроконтроллеры. Микроконтроллеры незаметно завоевали весь мир. Поскольку микроконтроллерные технологии очень эффективны, то в настоящее время на помощь человеку пришла целая армия электронных помощников. С применением микроконтроллеров появляются весьма привлекательные дополнительные возможности у устройств, которые выполнялись на традиционных элементах. Устройства, собранные с применением микроконтроллеров, фактически не требуют регулировки; достаточно несколь-

ко изменить программу. В свою очередь, использование микроконтроллера привело к развитию аппаратно-программных средств обработки данных, к появлению плат сбора данных, которые при подключении к шине могли осуществлять с недоступной для человека скоростью сбор, обработку, анализ и отображение информации. Такие измерительные инструменты, помогающие оператору анализировать данные и выдавать, с комфортной для него скоростью, оценки, полученные в результате обработки приходящей информации, подстраиваясь под требования и привычки пользователя, в литературе стали именовать «думающими», «разумными», «умными» (smart). Естественно, что когда появились датчики со встроенными микроконтроллерами, которые позволяли проводить комплексную обработку информации с низким уровнем шума и помех непосредственно в самом датчике, то их, по аналогии, называли «умными» датчиками. При этом все понимали, что такое представление данных, полученных от датчиков, весьма условно можно назвать «интеллектуальным». Весь интеллект «умного» датчика определяется программным обеспечением, заложенным в память микроконтроллера.

Обобщенная структурная схема «умного» датчика (smart sensor) показана на рис. 7.1.

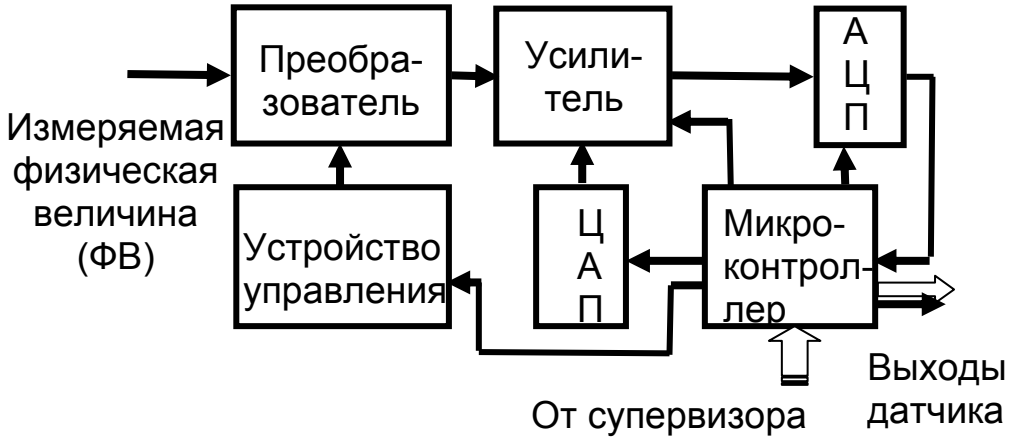


Рис. 7.1. **Обобщенная структурная схема «умного» датчика**

Принцип действия умного датчика состоит в следующем. Измеряемая физическая величина (к примеру, температура, давление, уровень радиации, ускорение и т. п.) поступает на чувствительный элемент датчика, где преобразуется в аналоговый сигнал, связанный определенной зависимостью с этой величиной. Электрический сигнал с преобразо-

вателя, поступает на усилитель, где его уровень повышается до требуемой величины, и, затем, посредством аналого-цифрового преобразователя (АЦП) формируется цифровой сигнал. Этот сигнал поступает на микроконтроллер и, через него на выход датчика. Во время выполнения процедуры измерений преобразователь находится под управлением микропроцессора. Благодаря этому, микроконтроллер, посредством устройства управления, может, меняя восприимчивость чувствительного элемента преобразователя, осуществлять изменение масштаба воспринимаемой физической величины (переключать диапазон). При этом может проводиться, под управлением микроконтроллера, мультиплексирование входных аналоговых сигналов. Кроме того, может выполняться коррекция выходного сигнала преобразователя с учетом температуры окружающей среды. Для повышения точности измерений под управлением микроконтроллера могут выполняться градуировочные операции (самокалибровка), компенсация (исключение) систематической погрешности. Микроконтроллерная обработка поступившего сигнала позволяет «умному» датчику получить несколько полезных свойств. Во-первых, это возможность получения от нелинейного преобразователя линейной зависимости между физической величиной и выходным сигналом. Для этого при помощи контроллера вводят поправки в соответствии с таблицей соответствия, хранящейся в памяти. Это позволяет создавать «умные» датчики с очень малой нелинейностью. Микроконтроллер, многократно обрабатывая единичные измерения, может уменьшить влияние на входной сигнал случайных погрешностей, или даже сформировать оценки таких вероятностных характеристик анализируемой переменной, как среднеквадратическое значение, дисперсия, среднеквадратическое отклонение. Если пользователя будут интересовать не непосредственно получаемые при измерении значения физических величин, а его новые, вычисленные на основании формул значения, то микроконтроллер может выдать на выход «умного» датчика найденное значение, умноженное на константу или его величину в логарифмических единицах.

Выходной сигнал «умного» датчика может быть как аналоговым, так и цифровым. Использование аналогового сигнала связано с тем, что обычно в технологических установках используют двухпроводные токовые линии типа line zeros, в которых для того, чтобы отличить обрыв кабеля от нулевого значения сигнала, последнее передается током 4 мА.

Может быть использован также цифровой выходной сигнал, соответствующий определенному протоколу обмена данных.

Известно, что одной из основных тенденций развития электроники является снижение мощности, потребляемой устройством. В этой связи «умный» датчик может работать в нескольких режимах. Основным является активный режим, при котором задействованы все ресурсы. Когда входная физическая величина в течение достаточно длительного времени не претерпевает изменения, датчик может переводиться в режим ожидания, при котором задействованы лишь преобразователь и АЦП. В режиме ожидания мощность, потребляемая умным датчиком, снижается в 5-10 раз. Переход из этого режима в активный может выполняться по командам микроконтроллера или супервизора. Датчик может переводиться, по каким-то соображениям, в режим останова. В этом случае «отключаются» все узлы датчика. В таком режиме потребляемая мощность снижается на три порядка и составляет единицы микроватт. Возврат в активный режим осуществляется по командам супервизора.

Вход для поступления управляющих сигналов от устройства управления более высокого уровня (супервизора) позволяет включать «умный» датчик в составе большой (общей) системы управления. В дополнение к данным о сигналах тревоги, свидетельствующим о превышении заданного значения сигнала, встроенная память микроконтроллера может быть использована для протоколирования измеряемых данных с последующей передачей протоколов на главное управляющее устройство. Обычно «умные» датчики должны подключаться к любому современному порту и соответствовать требованиям “Plug and Play”.

Питание «умного» датчика не может быть идеальным, поэтому обычно в нем учитывается тот факт, что напряжение источника питания датчика может быть подвержено как прогнозируемым, так и случайным изменениям. Считается, что датчик должен обладать свойством восстанавливать свою работоспособность как при кратковременных отключениях напряжения питания, так и при его «просадках» ниже допустимого значения. На опасных производствах, где существует риск взрыва, в электрическую цепь «умного» датчика включают предохранительное устройство, которое предназначено для ограничения подачи электроэнергии до такого уровня, при котором не возможно искрообразование. Умные датчики могут быть предварительно откалиброваны и его на-

стройки могут быть запомнены. Это важно при работе датчиков в сложных условиях. Пример исполнения «умного» датчика показан на рис. 7.2.

Рис. 7.2. **Пример исполнения «умного» датчика**

Заметим, что среди большой номенклатуры датчиков, используемых в современном полиграфическом оборудовании, важная роль отводится тонкопленочным магниторезистивным датчикам, в которых используется магниторезистивный эффект, т.е. изменение электрического сопротивления материала под воздействием внешнего магнитного поля. Они применяются для измерения напряженности постоянного и переменного магнитного поля (магнитометры), определения величины тока, в качестве устройств гальванической развязки, датчиков углового и линейного положений, тахометров, комбинированных головок воспроизведения для магнитных дисков. Основными элементами структуры датчика являются два ферромагнитных слоя, изготовленные из сплавов Co, Ni, Fe и разделенные прослойкой немагнитного металла – Cu, Ag, Au и др. В качестве фиксирующего слоя, создающего обменное взаимодействие с ближайшим ферромагнитным слоем для его фиксации, обычно используются пленки FeMn, FeIr, NiO.

В концепцию «умных» датчиков «вписываются» также матричные формирователи изображений систем считывания и обработки информации. К числу таких устройств можно отнести узел обработки изображений буквенно-цифровых символов. Структурная схема «умного» датчика такого типа показана на рис.7.3.

Комплементарная металл-окисел-полупроводниковая (КМОП) фотоприемная матрица содержит большое число светочувствительных

ячеек (расположенных по строкам и столбцам), накапливающих заряд, прямо пропорциональный интенсивности света. Черно-белое изображение фиксируется в матрице в виде двоичного кода, зависящего от X и Y координаты, и тем самым, из поступающего изображения символов формируется матрица двоичных электрических сигналов. Эти электрические сигналы, под воздействием устройства управления (УУ) КМОМ-матрицей, построчно «переносятся» по строкам на вход усилителя. Выводимый из матрицы электрический сигнал, содержащий информацию о полученном изображении, непрерывно изменяется во времени и его форма зависит от яркости изображения вдоль считываемой строки. Чтобы такой сигнал можно было обработать цифровым способом, после усиления полученный сигнал преобразовывается в цифровой код, т.е. аналоговый сигнал изображения преобразуется в ряд коротких импульсов, амплитуды которых равны величине аналогового сигнала в заданных временных интервалах. В ходе дискретизации сигнала получаются отсчеты, которые подаются на микроконтроллер. Таким образом, после завершения цикла работы в микроконтроллер поступает информация об уровне освещенности всех ячеек матрицы в виде числовой последовательности, где каждое число показывает состояние одной ячейки.

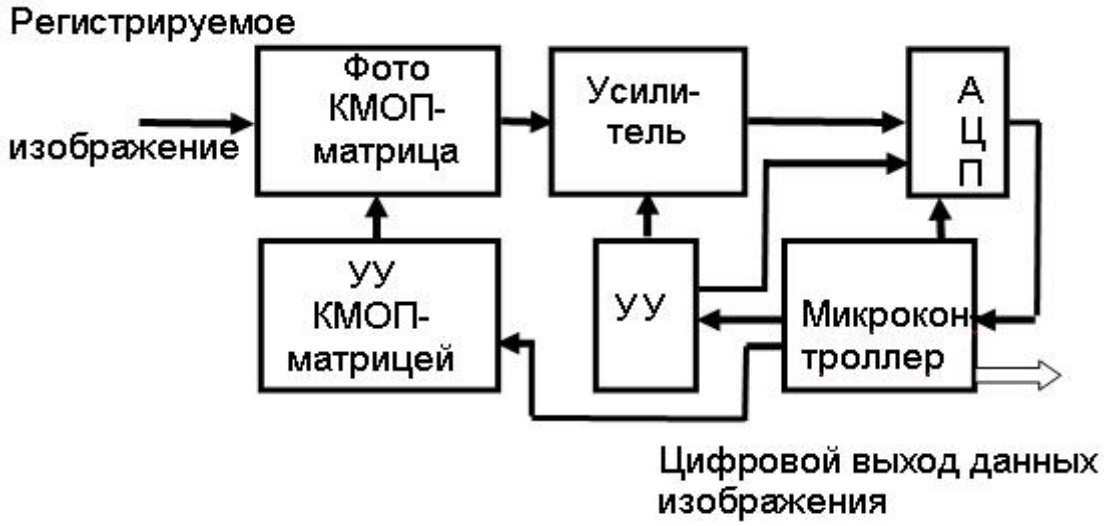


Рис. 7.3. **Структурная схема устройства обработки изображений буквенно-цифровых символов**

Микроконтроллер осуществляет обработку бинарной картины, поступающей от фоточувствительной матрицы. При этом, его первой задачей является исправление дефектов в изображении (пропущенных чер-

ных точек в сигнатуре символов, паразитных черных точек из-за пыли, пропущенных переходов от «белого к черному» из-за низкого качества печати и пр.) с помощью алгоритмов «заполнения» и «удаления». В качестве других задач микроконтроллера при обработке изображений могут быть автоматическое поддержание баланса белого, гамма-коррекция или специальное кодирование изображения для защиты от помех с помощью различных преобразований.

«Умные» матричные формирователи изображений систем считывания и обработки информации могут применяться для автофокусировки объектива. В таком случае с помощью специальной оптической системы из одного изображения делают два. Оба изображения подаются на две фотоматрицы. В результате сравнения сигналов двух изображений в микроконтроллере формируется корреляционный сигнал, максимум которого соответствует наибольшей четкости отображаемой картины. По корреляционному сигналу микроконтроллера исполнительное устройство автоматически устанавливает объектив в требуемое положение.

«Умные» матричные формирователи изображений могут быть применены для определения значения (сочетания выдержки и диафрагмы) экспозиции (exposure value) – количества света, попадающего на матрицу во время формирования одного кадра. При этом измерение яркости изображения, необходимое для того, чтобы правильно определить экспозицию, может выполняться несколькими способами. При матричном режиме используются данные об интенсивности света во всем множестве точек матрицы. При центрально-взвешенном режиме участвуют в измерении ячейки матрицы, расположенные в центральной части матрицы. При точечном режиме используется область ячеек соответствующая примерно 3% общей площади матрицы. Во всех названных случаях полученные данные с помощью микроконтроллера анализируются по специальным алгоритмам и, затем, рассчитывается экспозиция.

7.4. Исполнение нейронных сетей в виде микросхем

7.4.1. Общие сведения

Нейронные сети демонстрируют свою эффективность при решении таких «интеллектуальных» задач, как распознавание образов, кластеризация данных, ассоциативный поиск информации в базах данных и в ря-

де других применений. Хотя нейронные сети могут быть реализованы в форме программ на универсальных компьютерах, но для практических применений более желательным является их исполнение в форме электронных схем, выполненных на специализированных нейронных процессорах (нейрочипах). Нейрочип – специализированный микропроцессор, оптимизированный для массового выполнения нейронных операций: скалярного умножения и нелинейного преобразования сигналов, изготовленный по технологии микроэлектроники. Нейрочипы способны выполнять операции умножения входного сигнала на число (на вес входа), складывать сигналы и вычислять выходной сигнал, на основе заложенной в нейрон функции активации. Кристалл, который обеспечивает выполнение нейросетевых алгоритмов в реальном масштабе времени, часто называют также нейропроцессором. Промышленность уже разработала и выпускает ряд цифровых, аналоговых и гибридных нейрочипов, которые работают соответственно с цифровыми, аналоговыми или одновременно аналоговыми и цифровыми сигналами. Различные варианты нейрочипов выпускаются разными фирмами: Intel, IBM, Siemens, Fujitsu, Philips, AT&T, НТЦ «Модуль». Обычно «электронные нейронные сети» используют в компьютеризированном оборудовании в качестве ускорителей при решении соответствующих классов задач (обработки речевых сигналов и изображений, для распознавания образов и т. п.).

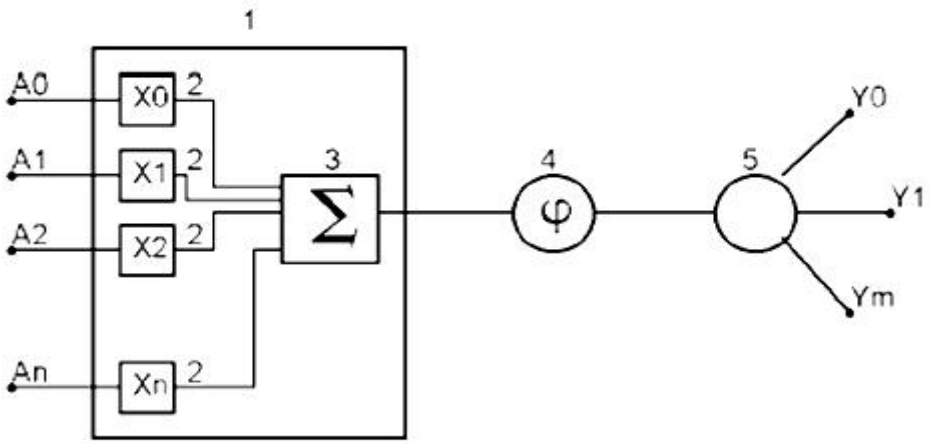


Рис. 7.3. **Структура формального нейрона, который должен быть реализован в виде интегральной микросхемы**

Элемент нейронной сети (формальный нейрон) (рис. 7.3), который должен быть реализован в виде интегральной микросхемы, в общем

случае должен содержать входной сумматор 1, последовательно связанный с нелинейным преобразователем сигналов 4 и точкой ветвления 5, предназначенной для передачи выходного сигнала по нескольким адресам. В состав сумматора входит схема умножения входного сигнала на вес 2 и блок суммирования сигналов 3. Его функционирование происходит следующим образом. Входной сумматор 1 получает на входы Ai вектор входного сигнала {0, .1, ….n }. Каждая компонента входного вектора умножается в блоках 2 на вес входа (вес синапса) xi, а затем результаты суммируются в блоке 3 и передаются на вход нелинейного преобразователя 4. Нелинейный преобразователь 4 на основе выходного сигнала сумматора вырабатывает выходной сигнал нейрона в соответствии с заложенной в него функцией активации y = .(x). Выходной сигнал нейрона через точку ветвления 5 передается на входы других формальных нейронов. Тем самым в искусственном нейроне должно быть технически реализовано умножение и сложение сигналов и передача суммы сигналов нейрону, расположенному справа. Соответственно, множество нейронов должно параллельно вычислять скалярное произведение и осуществлять передачу сигналов для всей совокупности нейронов (рис. 7.4).

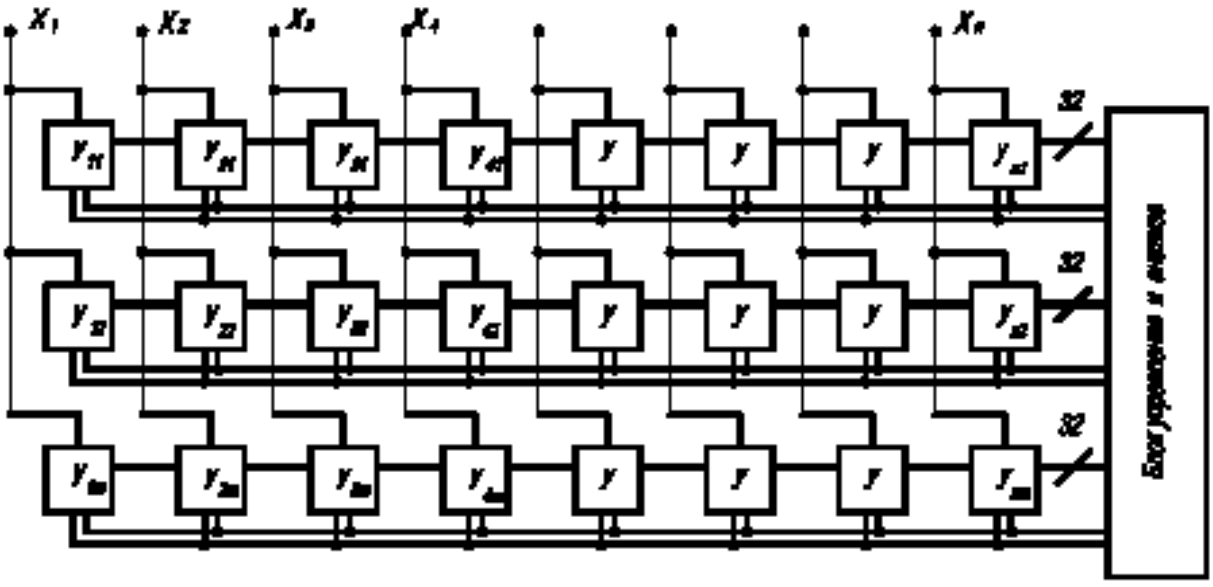


Рис. 7.4. **Модель совокупности нейронов в виде мультипроцессорной системы, реализующая нейронную сеть в виде интегральной схемы**

Мультипроцессорная система характеризуется большим количеством связей, поэтому серьезной проблемой при создании искусственных нейронных сетей в виде электронных цепей является проблема межсоединений. При этом следует учитывать, что полупроводниковые интегральные компоненты являются планарными конструкциями, поэтому такую проблему можно решить только путем использования многослойных структур, в частности, трассировочных матриц, состоящих из параллельных и взаимно пересекающихся проводников, выполненных в изолированных друг от друга технологических слоях. Связь проводников, в том числе и с другими элементами микросхемы, осуществляется через колодцы, выполненные в слоях изоляции микросхемы. Эти особенности технологии делают обоснованным выбор плоской архитектуры нейронной сети, согласованной с современной планарной полупроводниковой технологией ее изготовления. Плоскую нейронную сеть можно сеть каскадировать, то есть входы одного модуля сети можно соединить с выходами другого. Используя «плоскую» структуру нейронной сети можно построить и многослойную сеть (рис. 7.5).

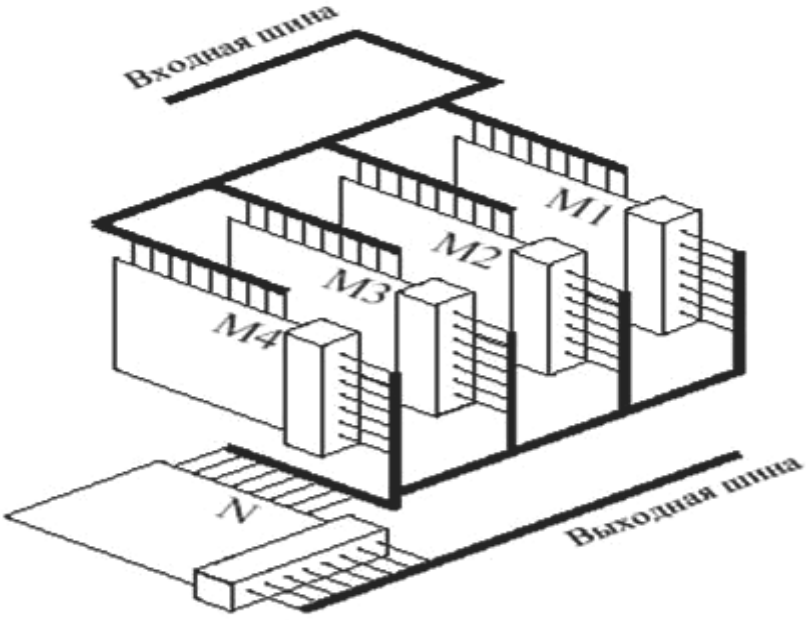


Рис. 7.5. **Пример многослойной структуры**

Для реализации вычислителей, осуществляющих быстродействующее перемножение векторов (входного вектора и столбца матрицы весов), а также устройств хранения информации целесообразно использовать схемы, в которых в качестве базового элемента выступает тран-

зистор на основе нанокристаллов Si. Это позволяет разместить на одном кристалле плоскую нейронную сеть гигабайтного объема. Параллельный алгоритм вычислений, реализованный в плоской нейронной сети, позволяет проводить распознавание изображений в реальном масштабе времени. Перспективны для применения в интеллектуальных системах оптические нейронные сети. Однако при их реализации возникает большое число проблем, которые требуется еще решить, прежде чем оптические нейронные сети получат практическое применение. Однако, учитывая, что большое число исследователей работает над этими проблемами можно надеяться на быстрый прогресс в этой области.

7.4.2. Классификация и основные параметры нейропроцессоров

В качестве нейропроцессоров используют: 1) специализированные нейрочипы; 2) заказные кристаллы (ASIC – Application-Specific Integrated Circuits); 3) встраиваемые микроконтроллеры (mC); 4) процессоры общего назначения (GPP); 5) перепрограммируемые логические интегральные схемы (FPGA, ПЛИС); 6) процессоры цифровой обработки сигналов (ПЦОС); 7) транспьютеры – микропроцессоры с собственной внутренней памятью и каналами связи с аналогичными устройствами, ориентированными на работу в многопроцессорной среде.

Для оценки производительности нейропроцессоров применяется ряд специальных показателей (параметров): 1) MMAC – миллионов умножений с накоплением в секунду; 2) CUPS (Connections Update per Second) – число измененных значений весов в секунду (оценивает скорость обучения); 3) CPS (Connections per Second) – число соединений (умножений с накоплением) в секунду (оценивает производительность); 4) CPSPW = CPS/Nw, где Nw – число синапсов в нейроне; 5) CPPS – число соединений примитивов в секунду: CPPS = CPS•Bw•Bs, где Bw, Bs – разрядность чисел, отведенных под веса и синапсы. Ориентация процессоров на выполнение нейросетевых операций обуславливает повышение скоростей обмена между памятью и параллельными арифметическими устройствами, а также уменьшение времени весового суммирования (умножения и накопления) за счет применения фиксированного набора команд типа регистр-регистр.

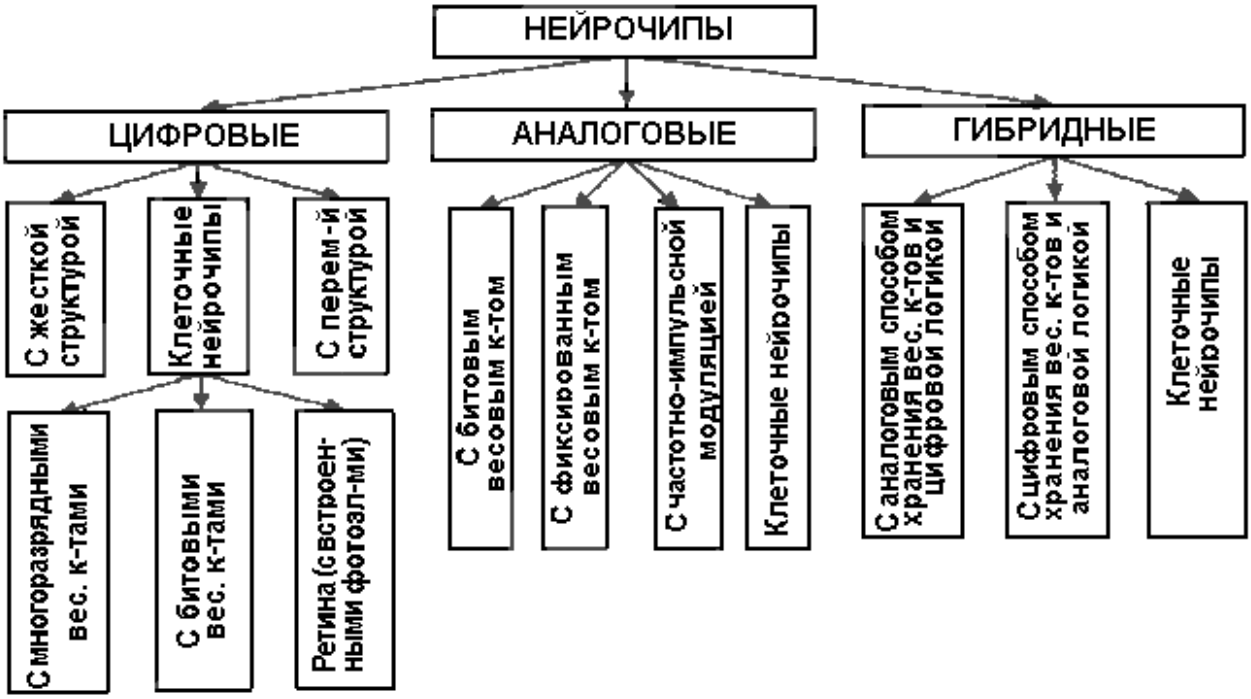


Рис. 7.6. **Классификация нейрочипов**

Таблица 7.1.

Характеристика некоторых специализированных нейрочипов.

Наименование	Изготовитель	Разрядность, бит	Макс-е число синапсов	Число слоев	Примечание
CNAPS-1064	Adaptive Solutions	16	128 Кбайт	64	-
ETANN 80170NW	Intel	64 входа	Два банка весов64x80	3 слоя по	Аналоговая
100 NAP Chip	HNC	32	512 Кбайт	4	4 процессора с плав. ариф-й
NC 3001	NeuriGam	16	4096 шт.	32	-
ZISC 036	IBM	64-разряд-ные вх-е век-торы	-	36 нейро-нов	Векторно-прототипный нейрочип
MD-1220	Micro Dev.	16	64 шт.	8	8 нейронов
MT 19003 - NISP	Micro Circuit Engineering	16-разр. умн.;35разр. сумматор	-	1	RISC-процессор
NLX420(NLX 110, 230)	Adaptive Logic	16	1 Мбайт	16	16 процессорных элементов
OBL Chip	Oxford Computer	16	16 Мбайт	-	-

Основное отличие нейрочипов от других процессоров состоит в обеспечении высокого параллелизма вычислений (и, как следствие, резкого увеличения производительности нейрочипов) за счет применения специализированного нейросетевого логического базиса или конкретных архитектурных решений.

Таблица 7.2.

Производительность некоторых специализированных нейрочипов

Наименование нейрочипа	Конфигурация	CPS	CPSPW	CPPS	CUPS
NLX420	32-16, 8 bit mode	10M	20K	640M	-
100 NAP	4 chips, 2 M wts, 16 bit mantissa	250M	125	256G	64M
WSI (Hitachi)	576 neuron Hopfield	138M	3.7	10G	-
N64000 (Inova)	64-64-1, 8 bit mode	871M	128K	56G	220M
MA16	1 chip, 25 MHz	400M	15M	103G	-
ZISC036	64 8 bit element inp. Vector	-	-	-	-
MT19003	4-4-1-, 32 MHz	32M	32M	6.8G	-
MD1220	8-8	9M	1M	142M	-
NI 1000	256 5 bit element inp. Vector	40 000 vec in sec.	-	-	-
L-neuro-1	1-chip, 8 bit mode	26M	26K	1.6G	32M
NM6403	8 bit mode, 50 MHz	1200M	150M	77G	-

По ряду основных признаков нейропроцессоры классифицируют как (рис. 7.6):

1) *по типу логики* нейропроцессоры разделяют на цифровые, аналоговые и гибридные;

2) *по типу реализации нейросетевых алгоритмов* – с полностью аппаратной реализаций и с программно-аппаратной реализацией (когда нейроалгоритмы хранятся в ПЗУ);

3) *по характеру реализации нелинейных преобразований* – нейропроцессоры с жесткой структурой нейронов (аппаратно реализованных) и нейрокристаллы с настраиваемой структурой нейронов (перепрограммируемые);

4) *по гибкости структуры нейронных сетей* – нейропроцессоры с жесткой и переменной нейросетевой структурой.

В отдельные классы следует выделить так называемые систолитические и нейросигнальные процессоры. Систолические процессоры (процессорные матрицы) – это чипы, как правило, близкие к обычным RISC-процессорам и объединяющие в своём составе некоторое число процессорных элементов. Вся же остальная логика, как правило, должна быть реализована на базе периферийных схем. У нейросигнальных процессоров ядро представляет собой типовой сигнальный процессор, а реализованная на кристале дополнительная логика обеспечивает выполнение нейросетевых операций (например, дополнительный векторный процессор и т.п.).

Промышленное производство специализированных нейрочипов ведется во многих странах мира. Большинство из них ориентируются на закрытое использование, поскольку они создаются для конкретных прикладных систем, однако среди нейрочипов достаточно и универсальных кристаллов (табл. 7.1). Показатели производительности некоторых нейрочипов приведены в табл. 7.2.

7.4.3. Нейропроцессоры на основе принципов цифровой обработки сигналов

Почти повсеместный переход современных систем управления на цифровые стандарты привел к необходимости обрабатывать с высокой скоростью большие объемы информации. Сложная обработка и фильтрация сигналов, например распаковка сжатых аудио- и видеоданных, маршрутизация информационных потоков, требует применения достаточно производительных вычислительных систем. Подобные системы могут быть реализованы на различной элементной базе, но наибольшее распространение получили устройства с применением цифровых сигнальных процессоров и ПЛИС. Программируемая логика способна работать на более высоких частотах, но поскольку управление реализовано аппаратно, то изменение алгоритмов работы требует перепрограммиро-

вания интегральной схемы. Помимо обработки и фильтрации данных ПЦОС могут осуществлять маршрутизацию цифровых потоков, выработку управляющих сигналов и даже формирование сигналов системных шин ISA, PCI и др. ПЛИС и ПЦОС получили значительное распространение и в качестве нейропроцессоров. Особенностью использования ПЦОС и ПЛИС в качестве элементной базы нейрокомпьютеров является то, что ориентация в выполнении нейросетевых операций обуславливает, с одной стороны, повышение скоростей обмена между памятью и параллельными арифметическими устройствами, а с другой – уменьшение времени весового суммирования (умножения и накопления) за счет применения фиксированного набора команд типа регистр – регистр.

Нейрокомпьютеры на базе ПЦОС обладают высокой степенью специализации. В ПЦОС широко используются методы сокращения длительности командного цикла, характерные для универсальных RISC-процессоров, такие как конвейеризация на уровне отдельных микроинструкций и инструкций, размещение операндов большинства команд в регистрах, использование теневых регистров для сохранения состояния вычислений при переключении контекста, разделение шин команд и данных (гарвардская архитектура).

Важно отметить, что реализация нейровычислителей высокой пространственной размерности требует все более производительной элементной базы. Для преодоления возникающих трудностей разработчики используют два подхода: улучшение характеристик уже имеющихся процессоров и увеличение производительности путем разработки новых архитектур.

Сравнительно широко распространенные в последнее время гибридные нейропроцессоры часто реализуют на ПЛИС, когда блок обработки данных выполняется на ПЦОС, а логика управления на ПЛИС. Среди множества фирм в мире, занимающихся разработкой и выпуском различных ПЛИС, лидерство делят фирмы Xilinx и ALTERA. Выделить продукцию какой-либо одной из этих фирм невозможно, так как по техническим характеристикам они различаются очень мало.

Отметим, что реализация нейропроцессоров на основе ПЛИС требует участия эксперта на топологической стадии проектирования. Построение нейрокомпьютеров на их основе хотя и дает высокую гибкость создаваемых структур, но пока еще проигрывает другим решениям по производительности.

Кроме нейроускорителей на базе ПЛИС и ПЦОС в последнее время все большее распространение находят нейроускорители на базе специализированных нейросигнальных и нейросетевых процессоров.

7.5. Нейрокомпьютеры

Нейрокомпьютер – это вычислительная система, построенная на основе нейропроцессора (ов), использующая архитектуру, предусматривающую параллельные потоки однотипных команд и множественные потоки данных, и предназначенная для реализации нейросетевых алгоритмов в реальном масштабе времени.

Нейроэмулятор – это система, построенная на базе каскадного соединения универсальных SISD-, SIMD- или MISD-процессоров (Intel, AMD, Sparc, Alpha, Power PC и др.) и реализующая типовые нейрооперации (взвешенное суммирование и нелинейное преобразование) на программном уровне.

Нейроускоритель – это нейрокомпьютер, реализованный в виде карты или модуля с распараллеливанием операций на аппаратном уровне. Нейроускорители могут быть построены на основе ПЦОС (Texas Instruments Inc., Analog Devices, Motorola), ПЛИС и (или) на основе специализированного (ых) нейрочипа (ов).

С точки зрения конструктивной реализации нейрокомпьютеры можно подразделить на изделия, реализованные в виде карт и модулей, и конструктивно-автономные системы.

Нейрокомпьютеры, изготовленные в виде карт (виртуальные нейрокомпьютеры), как правило, предназначены для установки в слот расширения стандартного ПК. С другой стороны – нейрокомпьютеры в виде модулей соединяются с управляющей Host-ЭВМ по стандартному интерфейсу или шине. Нейрокомпьютеры в виде карт и модулей с использованием нейроускорителей представляют мощное средство, которое обеспечивает современному инженеру-исследователю и аналитику реализацию нейросетевых алгоритмов в реальном масштабе времени.

При построении нейрокомпьютеров на базе ПЦОС и ПЛИС используют ПЦОС или ПЛИС, объединенные между собой в соответствии с архитектурой, которая обеспечивает параллельность выполнения вычислительных операций. Обычно, такие нейрокомпьютеры строятся на основе гибкой модульной архитектуры, которая обеспечивает простоту

конфигурации системы и наращиваемость вычислительной мощности путем увеличения числа процессорных модулей или применения более производительных ПЦОС. Системы реализуются, в основном, на базе модулей стандартов ISA, PCI, VME.

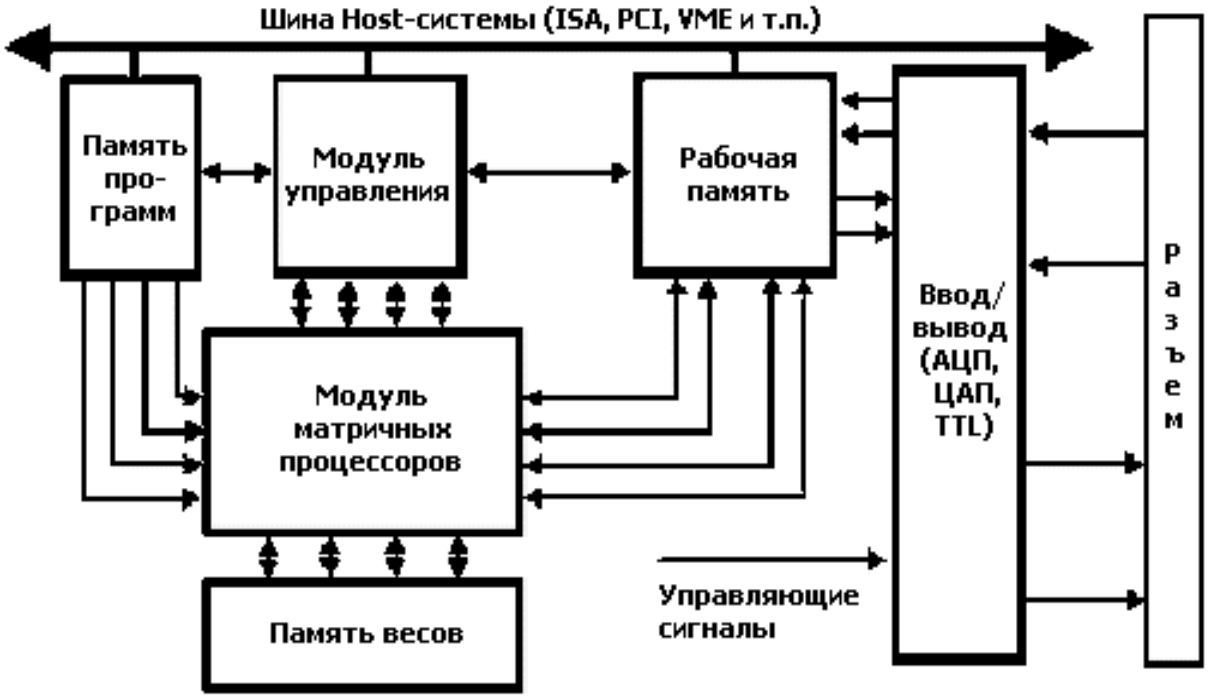


Рис. 7.7. **Обобщенная функциональная схема нейрокомпьютера, реализованного на основе ПЦОС или (и) ПЛИС**

На рис. 7.7. приведена обобщенная функциональная схема нейрокомпьютера, реализованного на основе ПЦОС или (и) ПЛИС. Основные функциональные элементы этих нейрокомпьютеров: модуль матричных ПЦОС; рабочая память; память программ; модуль обеспечения ввода/вывода сигналов (включающий АЦП, ЦАП и TTL линии); модуль управления, который может быть реализован на основе специализированного управляющего ПЦОС (УП), на основе ПЛИС или иметь распределенную структуру, при которой функции общего управления распределены между матричными ПЦОС. Считается, что реализация нейрокомпьютеров и специализированных вычислителей с массовым параллелизмом на базе ПЦОС и ПЛИС является эффективным решением задач цифровой обработки сигналов, обработки видео- и аудиоданных и построения технических систем управления. В последние годы наряду с нейроускорителями на базе ПЛИС и ПЦОС широкое распространение

получили нейроускорители на базе специализированных нейрочипов. На рис 7.8. приведена функциональная схема нейрочипа ZISC компании IBM, а на рис. 7.9а – изображение внешнего вида нейрочипа NeuroMatrix NM6403 компании Модуль.

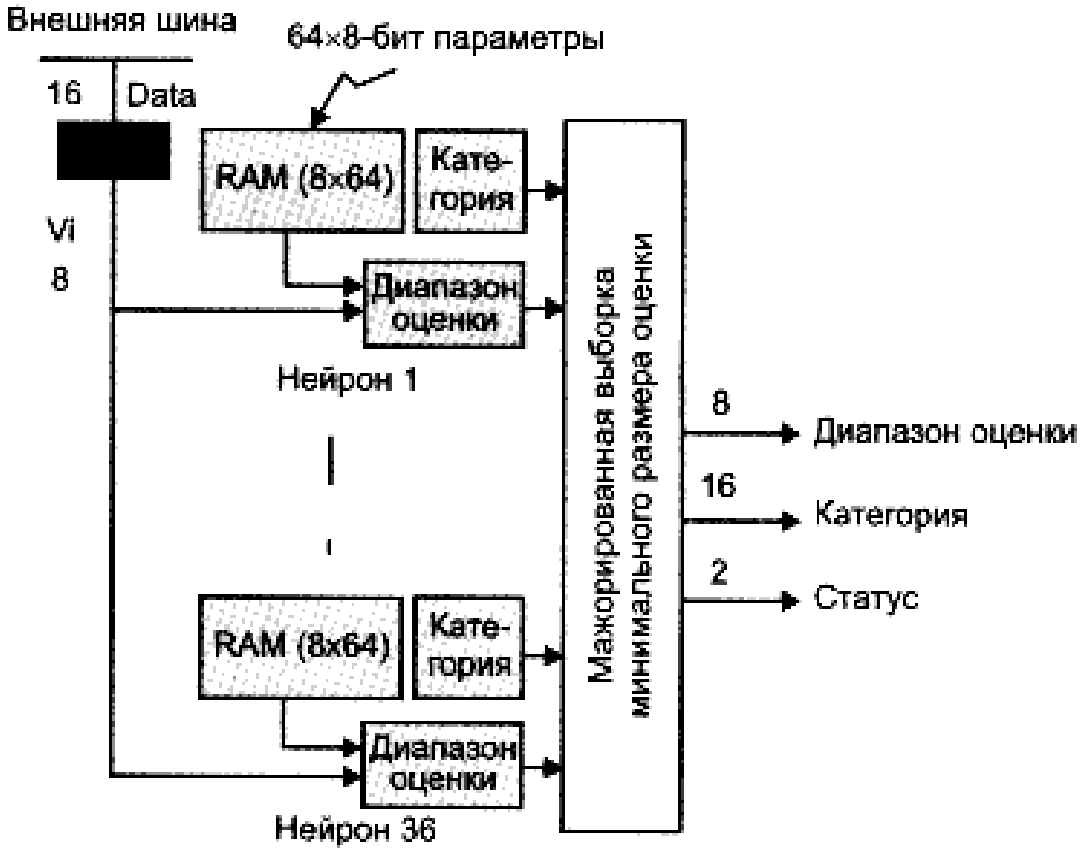


Рис. 7.8. **Функциональная схема нейрочипа ZISC компании IBM**

Рассматривая подходы к аппаратной реализации нейрокомпьютеров необходимо отметить, что, несмотря на широкое распространение различных высокопараллельных ускорителей для различных задач, число моделей полнофункциональных нейрокомпьютеров невелико, а коммерчески доступны из них единицы. Обусловлено это тем, что большинство из них реализованы для спецприменений. Компания Siemens стала первой европейской фирмой, выпустившей нейрокомпьютеры. Сфера применения нейрокомпьютеров Siemens – распознавание речи, изображений, образов, ускорение работы программных эмуляторов. Что касается нейрокомпьютера Synapse, то за один час самообучения он достигает таких же результатов, что и нейронные сети в обычном ком-

пьютере за целый год. Эти системы обладают скалярной многопроцессорной архитектурой и наращиваемой памятью.

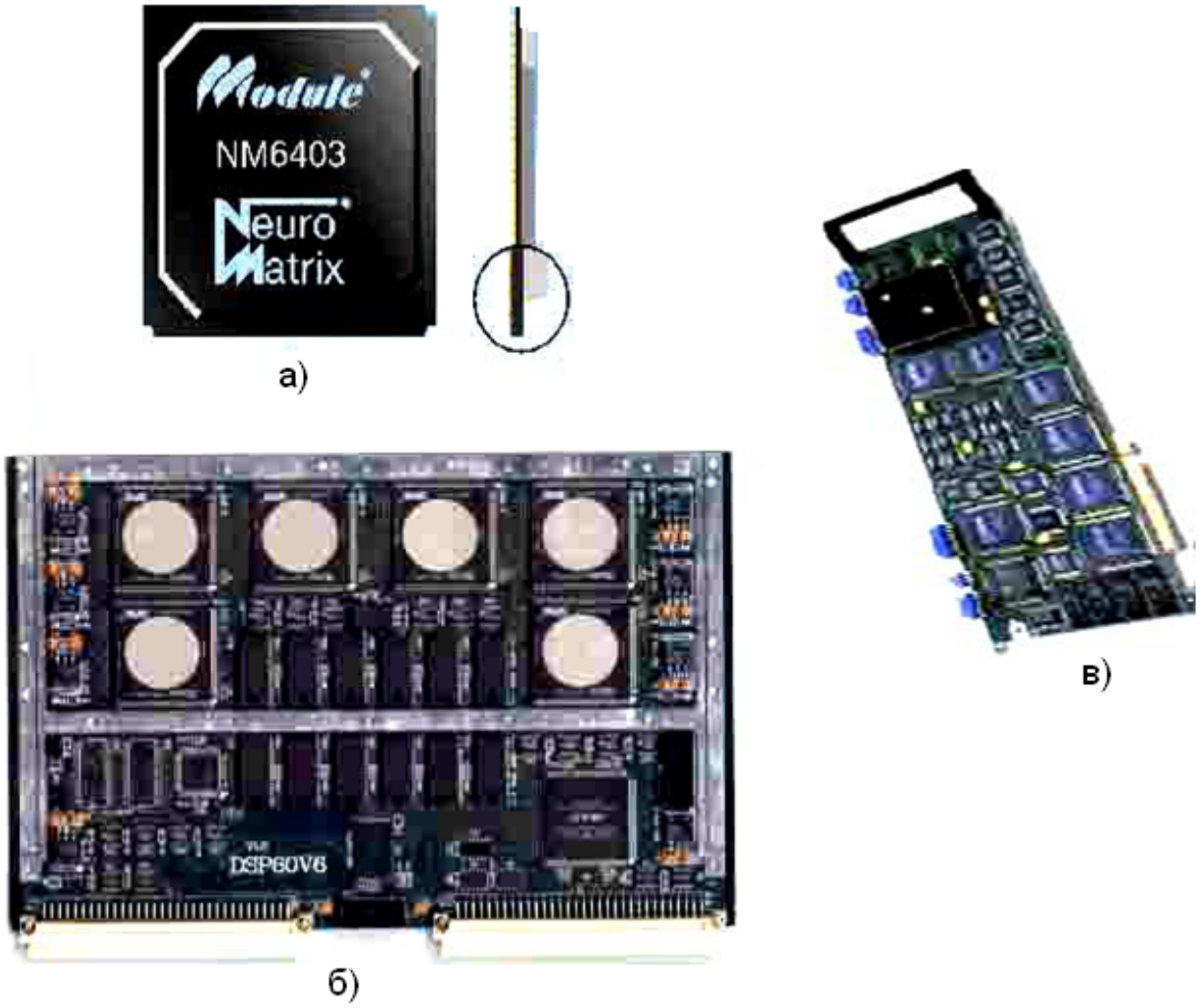


Рис. 7.9. **Нейрочип NeuroMatrix NM6403 компании Модуль (а); нейрокомпьютер DSP60V6 компании Инструментальные системы (б); нейрокомпьютер Synapse 2 компании Siemens (в)**

На рис. 7.9б. приведено изображение нейрокомпьютера Synapse 2 компании Siemens, а на рис. 7.9в. – нейрокомпьютер DSP60V6 компании Инструментальные системы. Исследования показали, что производительность выполнения нейросетевых операций на нейрокомпьютере Synapse 2, по крайней мере, на три порядка выше производительности традиционных вычислительных систем. Нейрокомпьютер позволяет моделировать нейронные сети с количество синапсов до 64000000, а гибкость архитектуры практически не ограничивает разнообразность нейросетевых парадигм.

Рекомендуемая литература

1. Барский А.Б. Нейронные сети: распознавание, управление, принятие решений. – М.: Финансы и статистика, 2004. – 264 с.
2. Галушкин А.В. Теория искусственных нейронных сетей. – М.: ИПРЖР, 2000. – 284 с.
3. Дьяконов В.А., Круглов В.В. Математические пакеты расширения MATLAB. Специальный справочник. – СПб.: Питер, 2001. – 480 с.
4. Дьяконов В. П. MATLAB 6.5 SP1/7 + Simulink 5/6. Основы применения. М.: СОЛОН Пресс, 2005. – 800 с.
5. Егупов Н.Д. Методы робастного, нейро-нечеткого и адаптивного управления. – М.: Изд-во МГТУ им. Н. Э. Баумана, 2002 г. – 744 с.
6. Круглов В.А., Борисов В.В. Искусственные нейронные сети. Теория и практика. – М.: Горячая линия – Телеком, 2002. – 382 с.
7. Медведев В.С., Потемкин В.Г. Нейронные сети. MATLAB 6. М.: ДИАЛОГ-МИФИ, 2002. – 496 с.
8. Назаров А.В., Лоскутов А.И. Нейросетевые алгоритмы прогнозирования и оптимизации систем. – СПб.: Наука и техника, 2003. – 384 с.
9. Осовский С. Нейронные сети для обработки информации. – М.: Финансы и статистика, 2002. – 236 с.
10. Ротштейн А.П. Интеллектуальные технологии идентификации: нечеткая логика, генетические алгоритмы, нейронные сети. – Винница: УНИВЕРСУМ-Винница, 1999. – 320 с.
11. Суровцев И.С., Клюкин В.И., Пивоварова Р.П. Нейронные сети. – Воронеж: ВГУ, 1994. – 224 с.
12. Тархов, Д. А. Нейронные сети. Модели и алгоритмы. Кн. 18. – М.: Радиотехника, 2005. – 256 с.
13. Тихонов А.Н., Арсенин В.Я. Методы решения некорректных задач. – М.: Наука, 1979. – 187 с.
14. Хайкин С. Нейронные сети: полный курс. – М.: И.Д. Вильямс, 2006. – 1104 с.
15. Хьюбер Дж. П. Робастность в статистике. – М: Мир, 1984. – 304с.
16. Шахнов В.А., Власов А.И., Кузнецов А.С., Поляков Ю.А. Нейрокомпьютеры: архитектура и схемотехника. – М.: Изд-во Машиностроение, 2000. – 64 с.

Содержание

НАВЧАЛЬНЕ ВИДАННЯ

Вдовьонков Володимир Юрійович
Гоков Олександр Михайлович
Жидко Євген Анатолійович

ОСНОВИ ЕЛЕКТРОТЕХНІКИ ТА ЕЛЕКТРОНІКИ. ІНТЕЛЕКТУАЛЬНІ КОМПОНЕНТИ НА ОСНОВІ ШТУЧНИХ НЕЙРОННИХ МЕРЕЖ

Навчальний посібник

Частина 5

(рос. мовою)

Відповідальний за випуск **Лапта С. І.**
Відповідальний редактор **Сєдова Л. М.**

Редактор **Голінська О. Г.**
Коректор **Бриль В. О.**

План 2008 р. Поз. №51-П.
Підп. до друку 25.12.2008 Формат 60 × 90 1/16. Папір MultiCopy. Друк Riso.
Ум.-друк. арк. 16,75. Обл.-вид. арк. 20,94. Тираж 400 прим. Зам. № 908

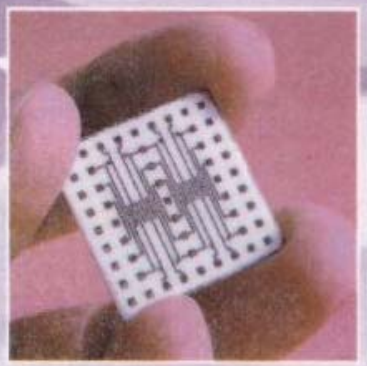
Видавець і виготівник — видавництво ХНЕУ, 61001, м. Харків, пр. Леніна, 9а

Свідоцтво про внесення до Державного реєстру суб'єктів видавничої справи
Дк №481 від 13.06.2001 р.

Вдовёнков В. Ю.
Гоков А. М.
Жидко Е. А.

ОСНОВЫ ЭЛЕКТРОТЕХНИКИ И ЭЛЕКТРОНИКИ. ИНТЕЛЛЕКТУАЛЬНЫЕ КОМПОНЕНТЫ НА ОСНОВЕ ИСКУССТВЕННЫХ НЕЙРОННЫХ СЕТЕЙ

УЧЕБНОЕ ПОСОБИЕ



Излагаются вопросы, связанные с искусственными нейронными сетями. Изучается функционирование нейронных сетей при решении задач аппроксимации, распознавания образов, кластеризации и «автоматической» классификации. Рассмотрены основы применения нейронных сетей и нейротехнологий. Приведен широкий круг характерных практических примеров, в том числе идей по нейроуправлению объектами, созданию многомодальных интерфейсов, сжатию сигналов.

ИЗДАТЕЛЬСТВО ХНЭУ
ХАРЬКОВ 2008