

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего профессионального образования
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ

М. В. Бураков

НЕЙРОННЫЕ СЕТИ И НЕЙРОКОНТРОЛЛЕРЫ

Учебное пособие



Санкт-Петербург
2013

УДК 004.032.6

ББК 32.818

Б91

Рецензенты:

Кандидат физико-математических наук,
старший научный сотрудник *В. Г. Курбанов*

(Учреждение Российской Академии наук

«Институт проблем машиноведения (ИПМаш РАН)»;

кандидат технических наук *Д. О. Якимовский*

(Федеральное государственное унитарное предприятие

«НИИ командных приборов»)

Утверждено

редакционно-издательским советом университета
в качестве учебного пособия

Бураков, М. В.

Б91 Нейронные сети и нейроконтроллеры: учеб. пособие / М. В. Бураков. – СПб.: ГУАП, 2013. – 284 с.: ил.
ISBN 978-5-8088-0812-6

В учебном пособии рассматриваются основы теории нейронных сетей и нейроконтроллеров, необходимые для понимания принципов нейросетевых технологий, – мощного средства построения систем автоматизации, которое в последние годы активно используется в инженерной практике. Приводится описание основных структур нейронных сетей регуляторов, рассмотрены примеры решения конкретных задач. Описываются возможности пакета математического моделирования MatLab для анализа и синтеза нейронных регуляторов.

Учебное пособие предназначено для подготовки бакалавров и магистров по направлению 220400 «Управление в технических системах».

УДК 004.032.6

ББК 32.818

ISBN 978-5-8088-0812-6

© Санкт-Петербургский государственный
университет аэрокосмического
приборостроения (ГУАП), 2013

© М. В. Бураков, 2013

ВВЕДЕНИЕ

Понятие «искусственные нейронные сети» оформилось в 1940-е годы благодаря основополагающей работе У. Мак-Каллока и Ч. Питтса [1], в которой была предложена модель мозга как множества нейронов, имеющих одинаковую структуру.

Каждый нейрон реализует некоторую функцию над входными значениями. Если значение функции превышает определенную величину – порог, то нейрон возбуждается и формирует выходной сигнал для передачи его другим нейронам. Мозг получает входную информацию от рецепторов (слуховых, зрительных и других), затем она обрабатывается нейронными структурами, преобразуется в набор управляющих воздействий на организм.

В работе [1] было показано, что сети, состоящие из искусственных нейронов, способны, в принципе, вычислить любую арифметическую или логическую функцию. Авторы предложили использовать искусственные нейронные сети (ИНС), элементами которых являются искусственные нейроны, выполненные на бинарных пороговых преобразователях и функционирующие по принципу «все или ничего». Такие сети оказались способны обучаться распознаванию образов и общению информации, т. е. обладали качествами, присущими живому мозгу.

В 1949 г. Д. Хебб [2] предположил, что условный рефлекс, открытый И. П. Павловым, возникает вследствие способности отдельных нейронов к установлению ассоциаций, и сформулировал соответствующее правдоподобное правило обучения биологических нейронов.

Первое практическое использование ИНС приходится на конец 1950-х годов, оно связано с изобретением Ф. Розенблаттом персептрона [3]. Основная идея Розенблatta сводилась к замене жестких логических схем Мак-Каллока и Питтса системами со статистическими свойствами. Персепtron продемонстрировал способность ИНС к обучению и решению задач классификации. Этот успех вызвал всплеск интереса к исследованию подобных систем.

Примерно в то же время Б. Видров и М. Е. Хофф [4] предложили другой обучающий алгоритм настройки адаптивных линейных нейронных сетей. Однако оказалось, что однослойные сети Розенблатта и Видрова имеют сходные ограничения, сужающие область их применения. Это было доказано в работе М. Минского и С. Пайперта [5]. Авторы сформулировали и доказали ряд теорем, подтверждающих принципиальную ограниченность однослойных

ИИС и их неспособность решать многие простые задачи, в том числе реализовать функцию «исключающее ИЛИ».

После выхода в свет этой книги Розенблatt и Видров разработали многослойные сети, свободные от выявленных недостатков. Однако им не удалось модернизировать свои обучающие алгоритмы так, чтобы данные более сложные сети можно было настраивать автоматически.

Эти результаты вызвали некоторое разочарование в возможностях ИИС, которое продлилось до начала 80-х годов, хотя в 70-е годы были опубликованы важные работы по таким сетям Т. Кохонена [6] и С. Гроссберга [7].

В 1980-е годы появились мощные персональные компьютеры и рабочие станции, что позволило выполнять сложные эксперименты с ИИС.

В 1982 г. Дж. Хопфилд успешно применил методы механики для описания работы однослойных полносвязных динамических ИИС [8]. Кохонен предложил новый класс искусственных нейронных сетей для решения задач векторной классификации [9].

Однако наибольшее значение для возрождения интереса к ИИС имело появление алгоритма обратного распространения ошибки, позволяющего обучать многослойные ИИС прямого распространения [10].

Алгоритм обратного распространения ошибки (существующий в многочисленных модификациях) показал очень хорошие результаты по обучению ИИС при решении многих прикладных задач, связанных с распознаванием текста, символов и т. п. Вместе с тем этот алгоритм является локальным, т. е. гарантированно работает только тогда, когда минимизируемая при обучении функция ошибки является унимодальной. Во многих задачах функция ошибки мультимодальная, поэтому в последние годы наряду с алгоритмом обратного распространения для обучения ИИС используются такие универсальные алгоритмы глобальной оптимизации, как алгоритм «отжига металла» [11], генетический алгоритм [12], «роевой интеллект» [13], метод муравьиных колоний [14].

Подробное описание истории развития ИИС можно найти в [15]. Немало работ по теории и практике применения ИИС ([16–24] и др.), а также переводов [25–28] было издано на русском языке.

В результате можно выделить следующие направления использования искусственных нейронных сетей:

- *классификация образов.* Задача состоит в указании принадлежности входного образа, представленного вектором признаков,

к одному или нескольким предварительно определенным классам. К подобным задачам относятся распознавание символов, речи, классификация электрокардиограмм, клеток крови и т. д. В робототехнике одним из основных приложений является распознавание объектов из видеинформации, полученной от системы технического зрения;

- *аппроксимация функций.* Предположим, что имеется обучающая выборка, заданная парами вход-выход: $((x_1, y_1), (x_2, y_2), \dots, (x_n, y_n))$, полученными от системы, описываемой неизвестной функцией f . Задача аппроксимации состоит в нахождении такой ИНС, поведение которой соответствует данной функции;

- *кластеризация.* При решении задачи кластеризации описание классов заранее не известно. Имеется большое множество объектов, которое требуется разбить на группы близкородственных элементов (кластеры). Кластеризация применяется при обработке изображений, извлечении знаний, сжатии данных;

- *прогнозирование.* Пусть заданы n дискретных отсчетов $\{y(t_1), y(t_2), \dots, y(t_n)\}$ в последовательные моменты времени t_1, t_2, \dots, t_n . Задача состоит в предсказании значения $y(t_{n+1})$. Прогнозирование имеет большое значение при принятии решений в разных областях человеческой деятельности;

- *ассоциативная память.* В современных вычислительных машинах обращение к памяти доступно путем указания адреса, который не зависит от ее содержания. Ассоциативная же память должна быть доступна по указанию заданного содержания, которое может быть искаженным или неполным. Такое свойство характерно для человеческой памяти, когда небольшая деталь объекта позволяет представить его полное описание. Реализация ассоциативной памяти особенно важна при создании мультимедийных приложений и баз знаний;

- *оптимизация.* Многие проблемы в науке, технике, медицине и экономике могут рассматриваться как задачи оптимизации. Под задачей оптимизации понимается нахождение такого решения, которое удовлетворяет системе ограничений и обеспечивает экстремум заданной целевой функции;

- *управление динамической системой* предполагает перевод ее из некоторого начального состояния в заданное целевое. Использование ИНС способно обогатить аппарат классической теории управления.

Эти общие направления в разных сочетаниях находят использование в практических приложениях, к которым относятся обработка

ка изображений, управление антропоморфными роботами и манипуляторами, техника связи, управление финансовыми рынками, активная реклама в Интернете и т. д.

Широкое применение ИНС находят и в следующих областях авиации [23]:

- управление динамикой полета летательного аппарата (ЛА);
- управление силовой установкой ЛА;
- управление взлетом, посадкой и маневрированием ЛА;
- обеспечение отказоустойчивости управления полетом ЛА;
- контроль и диагностика силовой установки ЛА;
- идентификация аэродинамических параметров ЛА;
- системы авионики, бортовые навигационные экспертные системы;
- системы управления воздушным и наземным движением ЛА;
- диагностика и прогнозирование усталостного разрушения авиационных конструкций.

Нейронные сети могут быть эффективно использованы для решения задач обработки бортовых данных космической техники, в которых присутствуют элементы комбинаторной оптимизации и распознавания образов. К ним относятся [22]:

- адаптивное управление переориентацией и угловой стабилизацией космического аппарата (КА) в условиях непредвиденных изменений его динамической схемы и априорной неопределенности возмущающих моментов;
- минимизация потерь бортовой системы электроснабжения (СЭС) вследствие оптимального токораспределения;
- диагностирование бортового оборудования КА на основе автономного анализа формируемой на борту телеметрической информации;
- повышение качества функционирования системы управления сближением КА путем оптимизации состава селективных признаков и оперативного определения параметров относительного состояния.

Существует множество пакетов прикладных программ для работы с нейронными сетями, таких, как NeuroSolutions фирмы NeuroDimension Inc., NeuralWorks Professional U/Plus с модулем UDND фирмы Neural Ware, Inc., Process Advisor фирмы AlWare, Inc., NeuroShell2 фирмы Ward Systems Group, BrainMaker Pro фирмы California Scientific Software и т. д.

В учебном пособии используются примеры работы с нейросетями, ориентированные на использование системы MatLab Simulink с расширением Neural Network toolbox. Это обусловлено рядом причин:

- работа с MatLab является необходимым инструментом на протяжении всего периода подготовки бакалавров и магистров по направлению 220400 «Управление в технических системах»;
- пакет Neural Network toolbox позволяет легко встраивать ИНС в модели, полученные с помощью других компонентов MatLab;
- система MatLab отличается открытостью, позволяющей легко создавать новые и модифицировать существующие программные модули.

Приводимые в учебном пособии примеры позволяют понять основные принципы проектирования ИНС разной топологии.

Изучение принципов использования искусственных нейронных сетей является необходимым компонентом подготовки магистров по направлению «Автоматизация и управление».

1. НЕЙРОННЫЕ СЕТИ. БАЗОВЫЕ ПОНЯТИЯ

1.1. Некоторые сведения о мозге человека

Принято считать, что понятие «интеллект» ввел в русский язык Н. В. Гоголь для обозначения способности человека к познанию [29].

Интеллектуальные способности человека неразрывно связаны с деятельностью мозга, но так считали не всегда. Древние египтяне полагали, что вместе с разумом является не мозг, а сердце человека. Мозг древнеегипетских мумий безжалостно удалялся из черепа.

Механизмы мыслительной деятельности человека во многом еще остаются не познанными, однако их изучение непрерывно продолжается.

Нормальная масса мозга человека составляет от 1000 до 2000 г. Так, мозг великого писателя И. С. Тургенева весил около 2000 г., а мозг другого великого писателя – А. Франса – всего 1100 г.

Было установлено [30], что у немцев средняя масса мозга составляет 1291 г, у швейцарцев – 1374 г, у русских и украинцев – по 1377 г, а у бурят – 1508 г. Интересно, что средний размер мозга неандертальца несколько превышал средний размер мозга современного человека и достигал 1610 г.

Выявлено также, что средний размер мозга древних египтян на протяжении 18 династий фараонов уменьшался на один грамм каждые 10 лет, что за 3000 лет составило около 300 г.

По сравнению с другими млекопитающими, мозг которых подобен человеческому, человек хоть и выделяется размерами головного мозга, но не является чемпионом. Так, мозг синего кита весит до 6800 г, а мозг слона – около 5000 г. Впрочем, удельная масса мозга человека составляет 1/50 по отношению к массе тела, в то время как у кита этот показатель всего 1/10000.

Таким образом, между размером мозга человека и уровнем его интеллекта нет прямой зависимости, важнее его внутреннее строение и особенности функционирования. От животных человек отличается, по-видимому, лишь тем, что у него лучше развиты отдельные участки мозга. Косвенным свидетельством этого служит тот факт, что 97% генома человека и шимпанзе совпадают. В 3% отличий укладываются все внешние признаки человека – строение руки, стопы, черепа и т. д., а также внутренние признаки – способность к членораздельной речи, абстрактному и логическому мышлению.

Головной мозг человека представляет собой очень сложную структуру, содержащую множество отделов и слоев.

Мозг состоит из серого и белого вещества. Серое вещество представляет собой сеть дендритов, аксонов и тел нервных клеток. Миелинизированные (изолированные) волокна, соединяющие разные области мозга друг с другом, с органами чувств и мускулами, образуют белое вещество.

Мозг состоит из двух полушарий, каждое из которых в свою очередь включает в себя лобную, теменную, височную и затылочную части.

В лобной части находятся отдел эмоций и центры управления движениями: правое полушарие отвечает за движение левой руки и ноги, а левое – за движение правой руки и ноги), в теменной части – зона телесных ощущений и осязаний. К ней примыкает височная зона, в которой расположены центр речи, слуха, вкуса. В затылочной части находится зрительный отдел.

В мозге существуют структурно обособленные отделы, такие, как кора, гиппокамп, таламус, мозжечок, миндалина, полосатое тело и т. д. (рис. 1.1).

Низшие отделы мозга отвечают за базовые функции: продолговатый мозг контролирует дыхание, пищеварение и сердцебиение, мозжечок управляет органами движения и т. д.

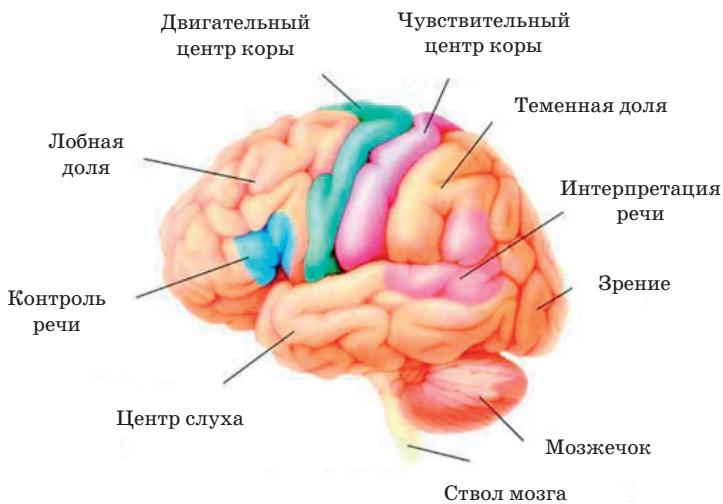


Рис. 1.1. Карта головного мозга

Наиболее поздно возникший отдел головного мозга – это *кора*, или *неокортекс*. В отличие от многих млекопитающих у человека кора составляет значительную часть мозга. Она представляет собой достаточно тонкий слой (2–3 мм) нервной ткани, площадь которого составляет примерно 2500 см². Эта поверхность «скомкана» в чепральной коробке и образует многочисленные извилины и борозды.

Лобные доли мозга имеют особое значение: считается, что они отвечают за принятие решений, а также за личностные качества человека. Префронтальная кора мозга человека (передняя часть лобных долей) занимает 29% всей коры, в то время как у шимпанзе – только 17% [31].

Кора головного мозга содержит от 10 до 20 млрд нейронов (по другим оценкам – до 100 млрд). Каждый нейрон (нервная клетка) может иметь связи примерно с 10000 других нейронов. На обеспечение работы мозга приходится примерно 1/4 всех энергетических затрат организма и потребляется 1/5 кислорода тела.

Нейроны мозга в процессе жизни человека постепенно отмирают, но это слабо отражается на его функционировании, поскольку человек использует, как полагают, не более 10% объема мозга, так что уменьшение числа нейронов в мозгу пожилого человека от 100 до 70% может и не быть особенно важным. Обработка огромных объемов информации мозгом осуществляется очень быстро, за доли секунды, несмотря на то, что сам нейрон является медленнодействующим элементом со временем реакции несколько миллисекунд.

Мозг отделен от остального тела специальной фильтрующей системой (*гематоэнцефалический барьер*), которая защищает его от проникания вредных веществ. Защита обеспечивается низкой проницаемостью кровеносных сосудов мозга, а также глиальными клетками, окружающими нейроны. Однако некоторые вещества (пары бензина, наркотики и т. д.) легко преодолевают этот барьер и могут вызывать необратимую деградацию коры головного мозга.

Если у человека доминирует левое полушарие мозга, то он – «правша», если правое – то он левша. Левое полушарие отвечает за абстрактное логическое мышление, правое – за образное. Известно, что у детей младшего школьного возраста доминирует правополушарное (образное) восприятие, а примерно к 14 годам может произойти переход к левополушарному (логическому) доминированию.

Связь между полушариями осуществляется мозолистое тело мозга.

Информация о функционировании полушарий была получена при наблюдении больных с так называемым «расщепленным мозгом», у которых рассечено мозолистое тело (данная операция выпол-

няется при лечении эпилепсии [32]). У этих людей информация от каждого глаза обрабатывается только одним полушарием: от левого глаза – правым, от правого – левым. Если такому человеку предлагать рассматривать разные предметы только левым глазом, то он не сможет их назвать, так как правое полушарие не отвечает за речевые функции, оно способно обрабатывать только самые простые речевые команды. Соответственно если демонстрировать предметы правому глазу, то возникают трудности с их перемещением. Интересно, что больные с «расщепленным мозгом» могут играть в шахматы как два разных игрока, наблюдая доску правым и левым глазом поочередно, так что один «игрок» не знает о замыслах другого.

В то же время одно полушарие может принимать на себя функции другого. Известны случаи, когда люди, утратившие одно из полушарий мозга, постепенно восстанавливали практически все потерянные функции (например, речь).

Достаточно хорошо изучена специализация отдельных участков головного мозга. Так, в височной его доле находятся первичные слуховые, а в затылочной – первичные зрительные поля.

Зрение дает человеку самый большой объем информации. Сетчатка человеческого глаза содержит около 110 млн светочувствительных клеток. Глаз соединен с мозгом 10 млн нервных волокон, которые связаны со зрителной корой головного мозга [33].

Центральная нервная система состоит из головного и спинного мозга.

Спинной мозг отвечает в основном за прием сенсорных сигналов от тела к мозгу и за передачу нервных сигналов от мозга к мышцам. Скорость распространения возбуждения измеряется десятками метров в секунду.

Пройдя путь через нейронные структуры мозга от рецепторов (слуховых, зрительных и других) до исполнительных органов, входная информация преобразуется в набор управляющих воздействий, адекватных ситуации.

По местоположению и функциям в нервной системе различают сенсорные (*рецепторные*), вставочные (*интернейроны*) и эффекторные (*мотонейроны*) нейроны.

Рецепторные нейроны воспринимают энергетические воздействия и сигналы среды. Взаимодействующие друг с другом интернейроны осуществляют внутреннюю обработку информации, а мотонейроны передают результаты этой обработки непосредственно на исполнительные системы организма, в качестве которых выступают мышцы и железы внутренней секреции.

Нервные волокна, передающие в головной мозг возбуждение из его периферических отделов и спинного мозга, называются *афферентными* (восходящими) путями. Возбуждение, направленное из головного мозга, называется *эфферентными* (ниходящими) путями.

Отдельные нейроны, соединяясь между собой, приобретают новое качество, которое в зависимости от характера межнейронных соединений имеет разный уровень биологического моделирования:

- группа нейронов;
- нейронная сеть;
- нервная система;
- мыслительная деятельность;
- мозг.

На протяжении жизни человека каждая молекула тела многократно обновляется (с периодом примерно семь лет), между тем память хранит события жизни человека на протяжении многих лет.

Считается, что способность к забыванию необходима для нормальной жизнедеятельности, поскольку она связана со способностью к обобщению и накоплению опыта. Некоторые люди практически лишены способности к забыванию. Такое явление называется *эйдемизмом*, или *фотографической памятью*.

Память человека делится на *долговременную* и *кратковременную*.

В долговременной памяти хранятся объекты и связи между ними, т. е. символная информация. Время извлечения информации из долговременной памяти составляет обычно до 70 мс (за это время человек узнает тот или иной образ или ситуацию и реагирует соответствующим образом). Человек способен распознавать около 20 образов в секунду.

В кратковременной памяти находятся данные, получаемые с помощью органов чувств.

Перемещение данных из кратковременной памяти в долговременную занимает 15–20 мин. При этом запись одного образа и установление связей между ним и другими записями долговременной памяти требует около 7 с.

Человек эффективно воспринимает и запоминает около 7 ± 2 градаций свойств какого-либо объекта (именно поэтому мы считаем, что у радуги семь цветов). Это же относится и к наборам фактов и связей между ними, которые извлекаются как единое целое. В англоязычной литературе такие наборы называют *чанками*. Специалист в конкретной предметной области помнит от 50000 до 100000 чанков. Накопление такого объема знаний занимает от 10 до 20 лет [34].

Как же отображается чанк в мозгу человека? В соответствии с доминирующими представлениями о работе мозга каждому чанку соответствует группа нейронов, между которыми сформированы устойчивые связи. Иначе говоря, эти нейроны возбуждаются одновременно, реагируя на определенную входную информацию.

В мозгу человека ежедневно возникает примерно 50000 мыслей.

При решении интеллектуальных задач человек использует три свои основные способности [35]:

- к перебору,
- к абстракции,
- к математической индукции.

Способность человека к перебору связана с возможностью последовательного переключения внимания с одного предмета на другой с узнаванием искомого предмета. Эта способность весьма ограничена – в среднем человек может уверенно (не сбиваясь) перебирать в пределах 1000 предметов (элементов). Средством преодоления этой ограниченности является его *способность к абстракции*, благодаря которой человек может объединять разные предметы или экземпляры в одно понятие, заменять множество элементов одним элементом (другого рода). *Способность человека к математической индукции* позволяет ему справляться с бесконечными последовательностями.

Еще одно важное свойство человеческой памяти заключается в ее *ассоциативности*. Человек может восстанавливать правильный образ по его искаженной копии, или, что то же, воссоздавать полную информацию по ее части.

Ограниченнность способности человека к перебору позволяет ввести понятие простой и сложной системы. Под *простой* будем понимать такую систему, в которой человек может уверенно перебрать все пути взаимодействия между ее элементами, а под *сложной* – такую, в которой он этого сделать не в состоянии. Между простыми и сложными системами нет четкой границы, но условно можно считать, что простая система имеет не более 7 ± 2 элементов.

1.2. Биологические представления о нейроне

Нервная клетка, или нейрон, живого организма представляет собой элемент, который может находиться в двух состояниях – возбуждения и торможения. Однако нейрон имеет более сложную организацию, чем триггер, – элемент с двумя устойчивыми состояниями. Рассмотрим основные представления о работе нейрона [36].

Схематическое изображение нейрона приведено на рис. 1.2. Он состоит из трех основных частей: тела клетки (сомы), дендритов и аксона (нейрита).

Размеры нейронов колеблются в широких пределах в зависимости от уровня организации животных, местоположения, функционального их назначения и других факторов. Например, клетки мозжечка имеют размер около 5 мкм, моторные клетки головного и спинного мозга – 70 мкм.

Сома нейрона отвечает за управление расходом энергии, питание, обновление ресурсов и другие процессы.

Дендриты (древовидные отростки) – это входы, по которым к телу клетки подводятся импульсы раздражения. Число отростков дендритов может достигать сотен, а их длина колеблется от долей миллиметра до десятков сантиметров. Они образуют густо ветвящееся дендритное дерево разной формы.

Аксон является выходом, по которому проводится возбуждение из клетки. Он может иметь длину от долей миллиметра до 1,5 м. На конце аксона разветвляется на множество ветвей. Он передает выходные сигналы нейрона на дендриты других нейронов.

Области контакта нервных клеток друг с другом называются синапсами. Синапс проводит возбуждение только в одном направлении – с окончаний аксона одного нейрона на дендриты и сому другого. Каждый нейрон может возбуждаться через множество синаптических контактов, расположенных вдоль дендритов и тела нейрона. Число синапсов крупного нейрона может достигать тысяч.

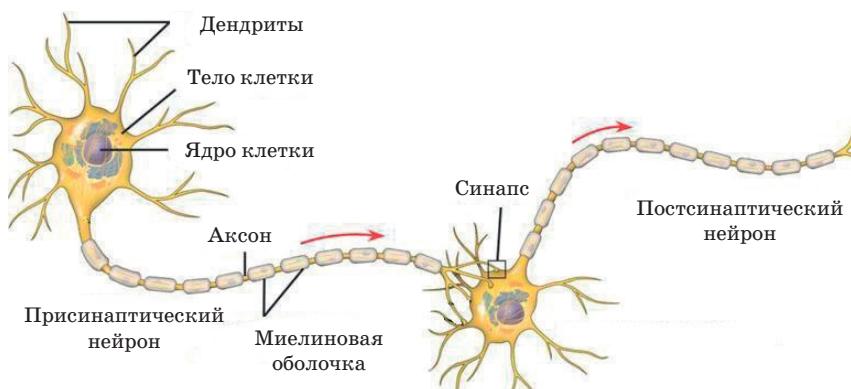


Рис. 1.2. Нейрон и межнейронное взаимодействие

Синапсы делятся на *химические* и *электрические*. Подавляющее большинство синапсов человека и животных относится к химическим синапсам.

Синаптический контакт химического синапса состоит из двух контактирующих частей: передающей – *пресинаптической*, представленной окончанием аксона, и приемной (рецепторной) – *постсинаптической*, представленной участком дендрита, сомы или аксона (в зависимости от типа взаимодействия). Обе части контакта разделены узким пространством шириной около 20 нм, называемым *синаптической щелью*. Пресинаптическая часть аксона содержит небольшие утолщения, включающие в себя сферические структуры, называемые *синаптическими пузырьками*, или *везикулами*, размером от 60 до 200 нм [36].

Когда нервный импульс приходит в аксон, некоторые из этих пузырьков высвобождают свое содержимое в синаптическую щель, тем самым инициируя процесс взаимодействия нейронов. Химические вещества, выбрасываемые в синаптическую щель, по функциональному назначению, а не химическому содержанию называются *медиаторами*, или *нейротрансмиттерами*. Медиаторы затем улавливаются на постсинаптической части специальными рецепторами дендрита и внедряются в тело клетки.

Определено более 30 видов медиаторов. Некоторые из них стремятся вызвать возбуждение клетки и называются *возбуждающими*, другие, наоборот, подавляют возбуждение и называются *тормозными*.

Электрические синапсы встречаются значительно реже, чем химические. Они описаны в нейронных сетях (НС) беспозвоночных, низших позвоночных и птиц. Характерной чертой этих синапсов является почти полное слияние пре- и постсинаптических полюсов контакта. Синаптическая щель в них сужена до 2 нм. Химическая передача в электрических синапсах полностью отсутствует, и возбуждение от нейрона к нейрону передается непосредственно на электрическом уровне.

У некоторых видов животных также обнаружены редкие типы синапсов со смешанной химической и электрической передачей [36].

Связи от аксонов к дендритам в нейронах реализуют основной тип аксодендритического взаимодействия. Однако в НС наблюдаются и другие его типы.

Важное свойство нейрона – способность к пространственному и временному суммированию раздражений. Это обуславливает сложность и гибкость их логических возможностей.

Пространственное суммирование заключается в том, что уровни отдельных раздражений, недостаточные для возбуждения нейрона, могут быть приложены к нему одновременно через несколько синапсов, что вызывает в результате возбуждение нейрона.

Временное суммирование состоит в том, что слабые раздражения, следующие одно за другим через достаточно короткие промежутки времени, также приводят к возбуждению нейрона, так как происходит не мгновенное, а экспоненциальное затухание действия отдельного импульса.

В состоянии покоя протоплазма нейрона заряжена отрицательно с потенциалом около 70 мВ. По мере поступления сигналов от дендритов заряд деполяризуется до значения потенциала 60 мВ, происходит диффузия в сому положительно заряженных ионов натрия Na^+ , нейрон срабатывает – его заряд резко повышается до положительного, и затем возбуждение распространяется через аксон на другие нейроны. Нейрон генерирует электрический импульс длительностью 1 мс, который проходит по аксону до синапсов. Затем ядро постепенно возвращается в исходное состояние.

Нейроны «срабатывают» в произвольные моменты времени с частотой от 1 до 100 Гц. Скорость распространения нервного импульса составляет приблизительно 100 м/с, что в миллион раз меньше скорости распространения электрического сигнала в медной проволоке. Если какой-то нейрон возбуждается чаще остальных, то сила его синаптической связи возрастает.

1.3. Понятие нейрокомпьютера

Рассмотренные количественные показатели, характеризующие работу мозга, не слишком впечатляющие. Тем не менее мозг человека пока еще качественно превосходит по возможностям современные компьютеры.

Большое влияние на разработку теории ИНС оказали идеи *коннекционизма* – раздела искусственного интеллекта, связанного с созданием моделей мозга и мышлением человека. С точки зрения коннекционизма (от англ. connection – связь), отдельные нейроны можно моделировать простыми автоматами, а вся сложность мозга определяется связями между нейронами. Это обеспечивает следующие свойства нейросетевой модели:

- однородность системы (элементы нейронной сети одинаковы и просты, функция определяется структурой связей);

- надежность системы, построенной из ненадежных элементов, благодаря избыточному числу связей;
- «голографичность», обеспечивающая сохранение свойств системы при разрушении ее части.

В задачах распознавания, например, несмотря на низкую скорость выполнения операций отдельным нейроном (миллисекунды), которая существенно меньше тактовой частоты процессоров современных компьютеров, мозг легко выигрывает у данных устройств. Это является следствием высокой степени параллелизма, обеспечиваемой огромным числом параллельно функционирующих нейронов и межнейронных соединений, что определяет высокую скорость работы мозга в целом.

Результаты сравнения характеристик компьютера и мозга человека в первом приближении приведены в табл. 1.1.

Компьютер работают безошибочно только при отсутствии аппаратно-программных сбоев, следуя жестко определенному алгоритму. Мозг ориентирован на обработку качественных данных в условиях неопределенности или неполноты, а также при отказах и повреждениях его участков. Это также можно считать следствием высокой степени параллелизма (избыточности) межнейронных связей.

Таблица 1.1

Качественное сравнение типов вычислителей

Параметр	Компьютер	Мозг
Процессорный элемент	Сложный, высокоскоростной, малое количество	Простой, низкоскоростной, большое количество
Память	Отделенная от процессора, локализованная, не ассоциативная	Интегрированная с процессором, распределенная, ассоциативная
Вычисления	Централизованные, последовательные, строго детерминированные	Распределенные, параллельные, адаптивные
Надежность	Сильная чувствительность к сбоям	Робастность (живучесть)
Способ решения задачи	Цифровая или символьная обработка	Обработка образов и знаний
Область действия	Алгоритмизированные задачи	Неограниченная, работа в условиях неопределенности

Необходимо отметить, что не только человеческий мозг, но и мозг примитивных живых существ обладает мощными способностями, обеспечивающими их выживание. Поэтому не совсем правильно утверждать, что искусственная нейронная сеть моделирует мозг человека. Скорее, это модель мозга живого существа, потенциальная мощность которой определяется числом нейронов.

В [28] дано следующее определение ИНС: *нейронная сеть* – это распределенный параллельный процессор, состоящий из элементарных единиц обработки информации, накапливающих экспериментальные знания и предоставляющих их для последующей обработки.

Такая сеть сходна с мозгом по двум признакам:

- знания поступают в нейронную сеть из окружающей среды и используются в процессе обучения;
- для накопления знаний применяются связи между нейронами, называемые *синаптическими весами*.

В литературе аппаратно-программные компьютерные системы, в основу функционирования которых положены ИНС, часто называют *нейрокомпьютерами*. Научная дисциплина, связанная с разработкой и исследованием методов использования ИНС в различных практических областях, называется *нейрокомпьютингом*.

Термин «нейрокомпьютер» подчеркивает принципиальное отличие вычислений в нейронных сетях от вычислений в обычном компьютере, хотя компьютер в силу своей универсальности может быть использован для моделирования ИНС.

Традиционный компьютер состоит из четырех основных блоков: центрального процессора (ЦП), состоящего из арифметико-логического устройства (АЛУ) и устройства управления, памяти, устройств ввода и вывода. Программа выполняется компьютером последовательно, команда за командой, следуя заранее предписанному алгоритму (хотя это уже не совсем верно для современных многопроцессорных и мультискэлярных ЭВМ).

В нейрокомпьютере АЛУ реализовано на базе ИНС, с которой связан блок обучения (рис. 1.3).

Памятью нейрокомпьютера можно считать набор весов межнейронных связей, который формируется в процессе обучения.

В работе нейрокомпьютера принципиально присутствуют два режима: обучения и рабочий.

Нейронная сеть должна пройти обучение для решения конкретной задачи. Задача обучения заключается в такой настройке коэффициентов межнейронных связей, при которой обеспечивается



Рис. 1.3. Обобщенная структура нейрокомпьютера

минимизация ошибки представления по всему обучающему множеству из совокупности обучающих пар, в которых каждому эталонному значению входного образа соответствует желаемое (эталонное) значение выходного образа. С математической точки зрения, процесс обучения представляет собой решение задачи оптимизации.

В рабочем режиме блок обучения отключен, и на вход нейрокомпьютера подаются произвольные сигналы (не входившие или входившие в обучающую выборку). На эти сигналы (входные образы) может быть наложен шум. Задача нейрокомпьютера заключается в выработке правильной реакции, наиболее соответствующей его «программе», под которой можно понимать топологию ИНС и набор весов ее межнейронных связей.

Таким образом, принципиальное отличие использования нейрокомпьютера состоит в отсутствии этапа алгоритмизации, который заменяется этапом обучения. Для понимания преимуществ, которые дает такая замена, следует напомнить понятия формализуемой и неформализуемой задач.

Формализуемая задача имеет алгоритм решения. Примером подобных задач являются традиционные вычислительные задачи: решение алгебраических, дифференциальных, интегральных и других уравнений, сортировка данных и т. п. Обычные ЭВМ ориентированы именно на формализуемые задачи.

Неформализуемая задача не имеет описанного алгоритма решения либо этот алгоритм требует чрезмерных вычислительных ресурсов.

В процессе развития науки и техники многие задачи могут переходить из класса неформализуемых в класс формализуемых, однако задач, алгоритм решения которых не известен, все еще намного больше.

На практике многие задачи можно назвать трудно формализуемыми, поскольку для них имеются частные алгоритмы, а универсальный алгоритм не известен. К этому классу задач относятся такие традиционные задачи искусственного интеллекта, как задачи управления сложными системами, распознавания образов, кластеризации данных, предсказания, аппроксимации функций и т. п.

Нейрокомпьютер является эффективным инструментом для решения трудно формализуемых задач.

1.4. Классификация нейронных сетей

Наиболее общая классификация ИНС делит их на два класса в зависимости от наличия обратных связей. Если ИНС не имеет обратных связей, то она называется *статической*, а если обратные связи существуют, то сеть *динамическая* (рекуррентная). На рис. 1.4 показаны наиболее распространенные типы ИНС, относящиеся к этим двум классам.

Еще один принцип классификации ИНС основан на их топологии. Соответственно можно выделить полносвязные, многослойные и слабосвязанные, а также модульные ИНС.

В *полносвязных* ИНС каждый нейрон передает выходной сигнал остальным нейронам, в том числе и самому себе. Все входные сигналы подаются всем нейронам. Выходными сигналами сети могут быть все или некоторые выходные сигналы нейронов после не-

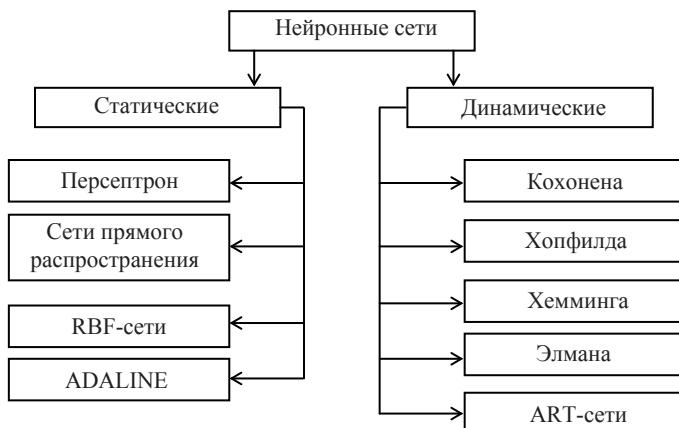


Рис. 1.4. Классификация нейронных сетей

скольких тактов функционирования сети. К таким ИНС относятся сети Хопфилда.

В *многослойных* ИНС нейроны объединяются в слои. Слой содержит совокупность нейронов с единными входными сигналами. Число нейронов в слое может быть любым и не зависит от их числа в других слоях. Всего сеть состоит из N слоев, пронумерованных слева направо.

В *слабосвязных* ИНС нейроны располагаются в узлах прямоугольной, или гексагональной, решетки. Каждый нейрон связан с четырьмя, шестью или восемью своими ближайшими соседями.

В *модульных* (или ядерных) ИНС, которые относятся к классу сетей прямого распространения, каждый нейрон последующего слоя получает сигналы только от части нейронов предыдущего. Так возникают нейронные ядра.

По типу структур нейронов ИНС делятся на *гомогенные* и *гетерогенные*. Гомогенные сети состоят из нейронов одного типа с единой функцией активации, а в гетерогенную сеть входят нейроны с различными функциями активации.

Нейронные сети можно также разделить на два класса в зависимости от наличия или отсутствия *латеральных* (от лат. *lateralis* – боковой) связей. Введение в скрытые слои ИНС латеральных (боковых) связей позволяет моделировать эффекты взаимного ослабления сигнала между соседними нейронами и усиления собственного сигнала нейрона. Это явление усиливает «контрастность» при решении задач распознавания.

Гибридные ИНС могут сочетать в себе признаки двух, а то и трех основных их видов. Как правило, эти сети многослойные, каждый слой которых представляется различной топологией и обучается по определенному алгоритму.

Наконец, существуют *бинарные* и *аналоговые*, *синхронные* или *асинхронные* ИНС.

Выбор топологии ИНС диктуется решаемой задачей, а также опытом разработчика.

При конструировании ИНС разработчик имеет, как правило, следующие исходные данные: размерность вектора входного сигнала, размерность вектора выходного сигнала, формулировку решаемой задачи, требования по точности ее решения.

Помимо выбора топологии разработчик должен назначить общее число нейронов в сети и число нейронов по слоям, вид функции активации нейронов, способ задания коэффициентов синаптической связи, метод проверки работоспособности новой сети.

1.5. Задача распознавания и линейная машина

Рассмотрим классическую постановку задачи распознавания, т. е. отнесения некоторого объекта к одному из известных классов [37].

Будем считать, что образ задан точкой n -мерного евклидова пространства R^n , т. е. является n -мерным вектором, компоненты которого действительные числа. При этом множество точек, соответствующих одному классу образов, группируется в некоторой области пространства измерений. Каждая ось такого пространства соотносится с одним из n входов или с одним из n рецепторов распознавающей системы. Каждый рецептор может находиться в одном из m состояний, если они дискретны, или иметь бесконечно большое число состояний, если рецепторы непрерывны. В зависимости от вида рецепторов рассматривается дискретное, непрерывное или смешанное (непрерывно-дискретное) пространство.

В пространстве образов вводится метрика – функция, которая каждой упорядоченной паре точек x и y пространства ставит в соответствие действительное число. Метрика должна обладать следующими свойствами:

1. $d(x, y) \geq 0$, ($d(x, y) = 0$ при $x = y$).
2. $d(x, y) = d(y, x)$.
3. $d(x, y) \leq d(x, z) + d(z, y)$.

Введение метрики позволяет говорить о степени близости точек пространства, а соответственно о мере сходства или различия образов.

Рассмотрим, например, задачу в пространстве R^2 . Пусть имеется четыре класса образов: A , B , V и Γ . Экземпляр каждого класса определяется точкой двумерного пространства, заданного координатами x_1 и x_2 (рис. 1.5).

Допустим, что существуют два объекта: M_1 и M_2 , которые требуется классифицировать. Очевидно, что классификация в такой постановке сводится к вычислению расстояния (метрики) от объекта до каждого из имеющихся классов. Объект принадлежит к тому классу, для которого это расстояние минимально.

В соответствии с рис. 1.5 M_2 классифицируется однозначно, в то время как относительно принадлежности M_1 возникает неопределенность, поскольку невозможно провести прямую, по одну сторону которой точки находились бы ближе к классу A , а по другую – ближе к классу B (иначе говоря, классы A и B – *несепарабельные*). Для остальных пар ($A - \Gamma$, $A - B$, $B - \Gamma$, $B - V$, $\Gamma - V$) такая прямая (ее называют *разделяющей* прямой) может быть проведена. Ее уравнение имеет вид

$$s(X) = w_1x_1 + w_2x_2 + w_3 = 0. \quad (1.1)$$

Здесь w_1, w_2, w_3 – параметры, а x_1, x_2 – координаты на плоскости. Для точек, которые лежат выше этой линии, $s(X) > 0$, а для тех, которые лежат ниже нее, $s(X) < 0$.

Рассмотрим некоторые два класса, например, A и B (рис. 1.6).

В случае гиперпространства разделяющая прямая преобразуется в гиперповерхность

$$s(X) = w_1x_1 + w_2x_2 + \dots + w_nx_n + w_{n+1} = WX + w_{n+1} = 0. \quad (1.2)$$

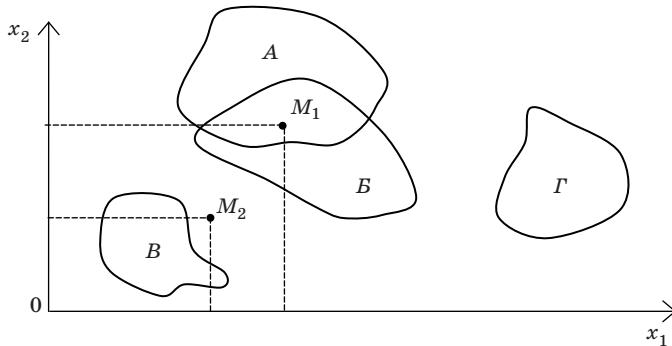


Рис. 1.5. Задача классификации

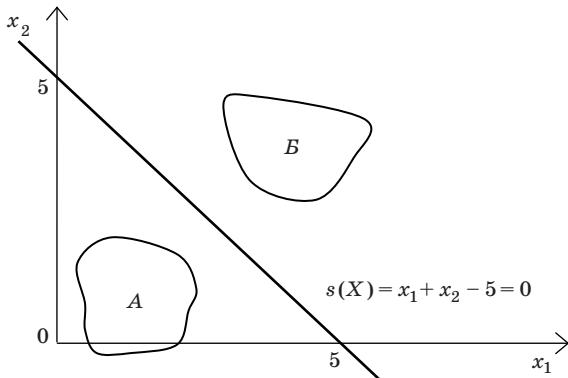


Рис. 1.6. Использование для классификации разделяющей прямой

Такая гиперповерхность также разделяет гиперпространство на две части. Как будет показано далее, формулы (1.1) и (1.2) являются частными случаями описания искусственного нейрона. Выражение (1.2) иногда называют *линейной машиной* (ЛМ).

Рассмотрим, каким образом ЛМ выполняет классификацию векторов по минимуму дистанции до центра кластера (под кластером здесь понимается группа точек, которые расположены друг к другу ближе, чем к точкам других кластеров).

Пусть в n -мерном евклидовом пространстве расстояние между двумя точками измеряется по формуле

$$\|X_i - X_j\| = \sqrt{[(X_i - X_j)^T(X_i - X_j)]^2}.$$

Обозначим через P_i центр тяжести i -го кластера ($i = 1, 2, \dots, N$). В процессе классификации требуется вычислить расстояние от входного вектора X до центра каждого кластера и выбрать минимальное расстояние.

Рассмотрим квадрат расстояния до центра i -го кластера:

$$\|X - P_i\|^2 = (X - P_i)^T(X - P_i) = X^T X - 2X^T P_i + P_i^T P_i \quad (1.3)$$

(так как $(A + B)^T = A^T + B^T$ и $X^T P = P^T X$).

Первое слагаемое (1.3) не зависит от i , следовательно, минимизация (1.3) означает максимизацию обратной функции:

$$d_i(X) = X^T P_i - \frac{1}{2} P_i^T P_i. \quad (1.4)$$

Выражение (1.4) можно переписать в виде

$$d_i(X) = W_i^T X + w_{i, n+1}, \quad (1.5)$$

где

$$w_{i, j} = p_{i, j}; \quad w_{i, n+1} = -\frac{1}{2} P_i^T P_i.$$

Рассмотрим пример. Пусть дано двумерное пространство с центрами кластеров, заданными векторами

$$P_1 = \begin{bmatrix} 3 \\ 3 \end{bmatrix}, \quad P_2 = \begin{bmatrix} 6 \\ -5 \end{bmatrix}, \quad P_3 = \begin{bmatrix} -8 \\ 10 \end{bmatrix}.$$

На основании (1.5) можем записать

$$W_1 = \begin{bmatrix} 3 \\ 3 \\ -9 \end{bmatrix}, \quad W_2 = \begin{bmatrix} 6 \\ -5 \\ -30,5 \end{bmatrix}, \quad W_3 = \begin{bmatrix} -8 \\ 10 \\ -82 \end{bmatrix},$$

$$\begin{aligned} d_1(X) &= 3x_1 + 3x_2 - 9, \\ d_2(X) &= 6x_1 - 5x_2 - 30,5, \\ d_3(X) &= -8x_1 + 10x_2 - 82. \end{aligned}$$

Структуру классифицирующей системы иллюстрирует рис. 1.7.

Таким образом, к первому классу принадлежат те точки плоскости, для которых выполняются неравенства ($d_1(X) > d_2(X)$) и ($d_1(X) > d_3(X)$) или

$$\begin{cases} -3x_1 + 8x_2 + 21,5 > 0, \\ 11x_1 - 7x_2 + 73 > 0. \end{cases}$$

Аналогично можно сформулировать условия принадлежности ко второму и третьему классам.

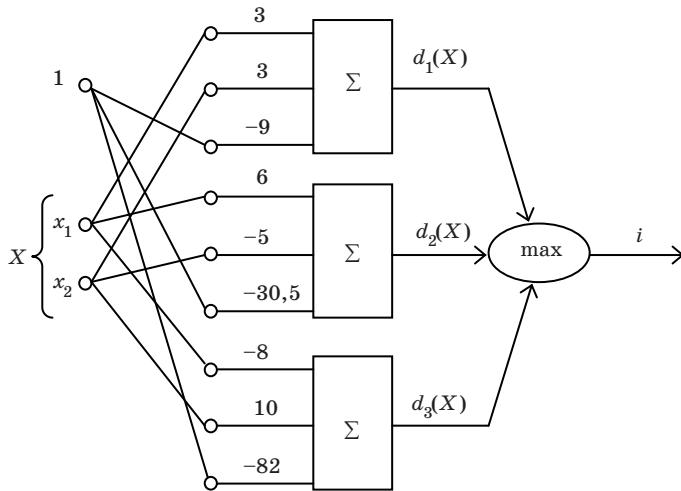


Рис. 1.7. Пример использования для классификации линейной машины

1.6. Искусственный нейрон

Искусственный нейрон (ИН) – это простейший аналоговый преобразующий элемент, моделирующий базовые представления о работе живого нейрона.

На вход ИН поступает некоторое множество сигналов. Каждый вход взвешивается – умножается на определенный коэффициент (**синаптическую силу**). Сумма всех произведений определяет уровень активации нейрона. Суммирующий блок соответствует соме живого нейрона (рис. 1.8).

Активационная функция F должна быть монотонной. Обычно $F(y)$ принадлежит к интервалу $[0,1]$ или $[-1,1]$. Чаще используют множество вариантов активационных функций (табл. 1.2).

Таким образом, ИН выполняет две операции. Сначала вычисляется сумма скалярного произведения вектора весов W и входного вектора X :

$$y = X^T W + b.$$

Затем срабатывает активационная функция, определяющая значение выходного сигнала:

$$z = f(y).$$

Например, если использовать знаковую активационную функцию (см. табл. 1.2), то выход ИН можно описать формулой

$$f(y) = \text{sgn}(X^T W + b).$$

Таким образом, ИН реализует разделяющую прямую вида (1.2).

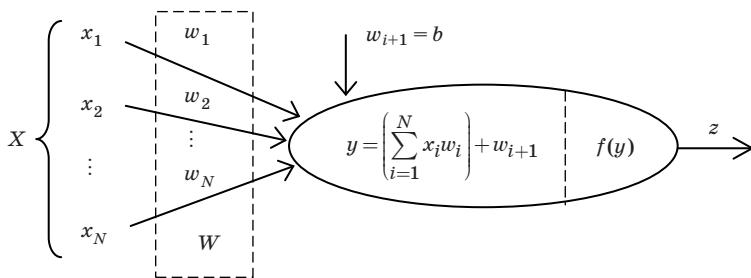


Рис. 1.8. Искусственный нейрон

Таблица 1.2

Основные варианты описания активационной функции

Название функции	Описание	Графическое представление
Линейная	$F(y) = ky$, $k > 0$, k – коэффициент активации	
Линейная с насыщением	$F(y) = \begin{cases} +1, & y > P, \\ ky, & y < 1, \\ -1, & y < -P \end{cases}$	
Определение знака	$F(y) = \text{sgn}(y)$	
Униполярная сигмоидальная (<i>s</i> -образная)	$F(y) = \frac{1}{1 + \exp(-ky)},$ $k > 0$	
Биполярная сигмоидальная (гиперболический тангенс)	$F(y) = \text{th}(ky),$ $k > 0$	
Пороговая	$F(y) = \begin{cases} 1, & y \geq P, \\ 0, & y < P \end{cases}$	

1.7. Проблема линейной разделимости

Рассмотрим ИН с двумя входами и пороговой активационной функцией (рис. 1.9).

Все возможные комбинации $x_1w_1 + x_2w_2$ определяют плоскость (x_1x_2) , а уравнение (1.1) – некоторую прямую в этой плоскости.

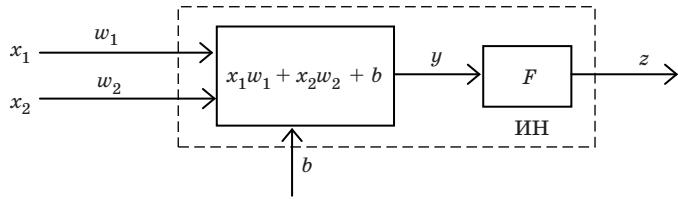


Рис. 1.9. ИН с двумя входами

Рассмотрим пример. Пусть необходимо реализовать логические функции, заданные в табл. 1.3.

Положение точек $A_1 - A_4$ на плоскости выглядит следующим образом (рис. 1.10).

Очевидно, что для реализации функции OR следует отделить точку A_1 от остальных точек, а для реализации функции AND необходимо отделить точку A_4 .

Таблица 1.3

Базовые логические функции

Точка	x_1	x_2	$OR(x_1, x_2)$	$AND(x_1, x_2)$	$XOR(x_1, x_2)$
A_1	0	0	0	0	0
A_2	0	1	1	0	1
A_3	1	0	1	0	1
A_4	1	1	1	1	0

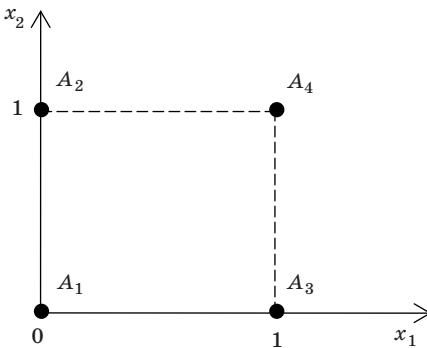


Рис. 1.10. Входные комбинации логических функций

Пусть входы x_1 и x_2 имеют единичные веса: $w_1 = w_2 = 1$, а в качестве F выберем пороговую функцию с порогом $P = 0$. Тогда для реализации конкретной логической функции следует выбрать смещение b . Например, при $b = -1,5$ получается логическая функция AND (рис. 1.11).

При выборе $b = -0,5$ получаем логическую функцию OR.

На рис. 1.12 показаны соответствующие разделяющие прямые на плоскости.

Иная ситуация возникает при попытке реализовать функцию XOR.

В этом случае необходимо провести разделяющую прямую таким образом, чтобы точки A_2 и A_3 лежали в одной полуплоскости, а точки A_1 и A_4 – в другой. Это очевидно невозможно, т. е. функция XOR является нереализуемой для единичного нейрона с двумя входами.

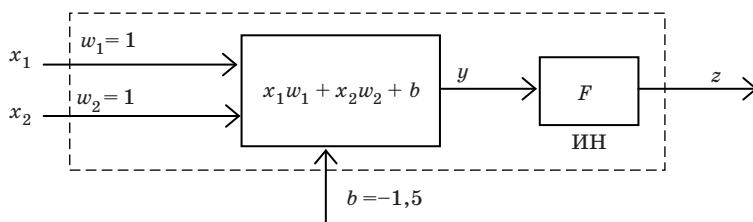


Рис. 1.11. Входные комбинации логических функций

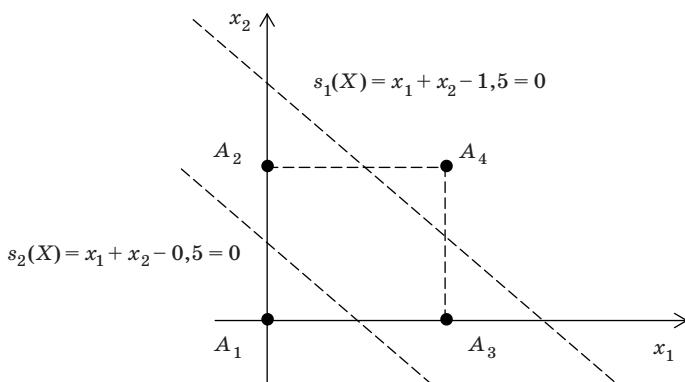


Рис. 1.12. Графическое представление работы нейронов AND и OR

Однако условие линейной разделимости легко обойти, если усложнить нейронную структуру. Проблема функции XOR легко решается, если добавить второй нейрон (рис. 1.13).

Очевидно, что нейрон AND вносит свой вклад только в ситуации $x_1 = x_2 = 1$, при этом выход нейрона XOR

$$z = F(1 + 1 - 0,5 - 2) = 0.$$

Реализация функции XOR связана с выделением на плоскости области, получающейся в результате комбинации двух разделяющих прямых, поэтому структуру на рис. 1.13 правильнее изобразить в виде трех нейронов (рис. 1.14).

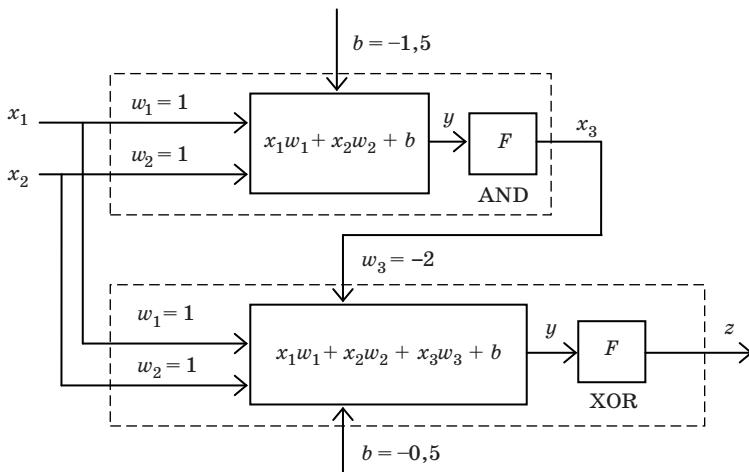


Рис. 1.13. Реализация функции XOR с помощью двух нейронов

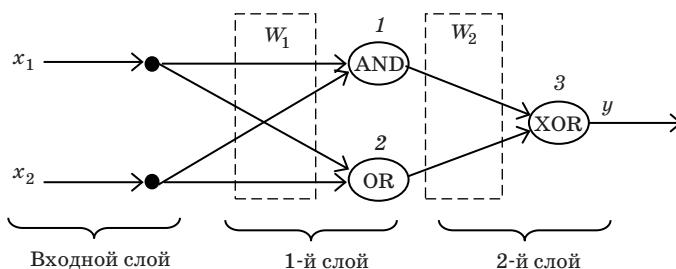


Рис. 1.14. ИНС для описания функции XOR

Нейрон OR обеспечивает разделяющую прямую $s_1(X) = 0$, а нейрон AND – разделяющую прямую $s_2(X) = 0$, как показано на рис. 1.15.

Для реализации функции XOR нейрон 3 должен выдать единичный сигнал только при выполнении условия

$$(s_1(X) > 0) \text{ AND } (s_2(X) < 0),$$

т. е. $\text{XOR}(X, Y) = \text{OR}(X, Y) \text{ AND } (\text{NOT}(\text{AND}(X, Y)))$.

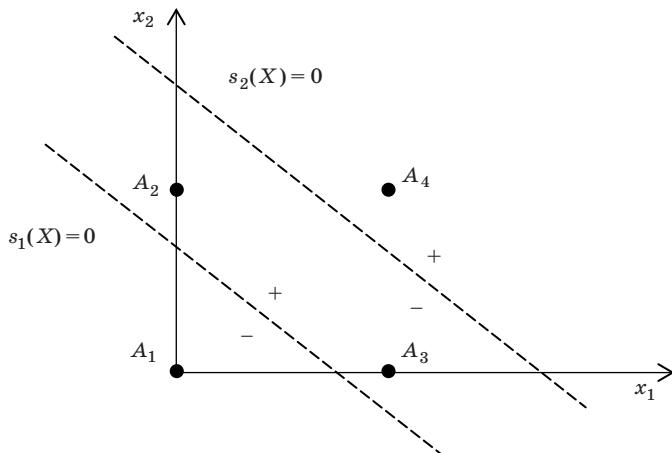


Рис. 1.15. Нейронная реализация функции XOR

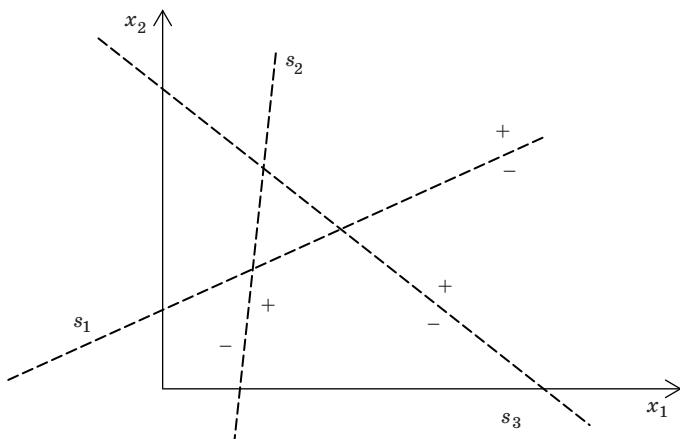


Рис. 1.16. Выделение выпуклой области на плоскости

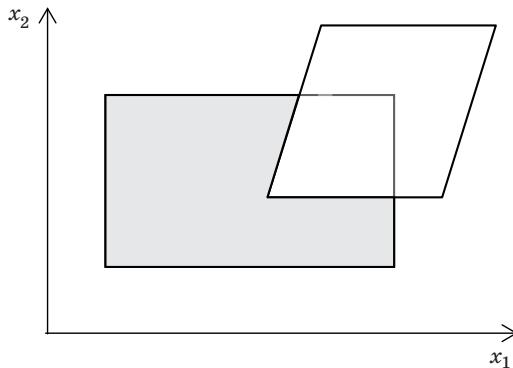


Рис. 1.17. Выделение на плоскости невыпуклой области

Структура на рис. 1.14 представляет собой простейшую двухслойную ИНС.

Рис. 1.16 показывает, что, комбинируя разные разделяющие прямые, на плоскости можно выделить область выпуклой формы. Число разделяющих прямых соответствует числу нейронов 1-го слоя.

Трехслойная сеть с двумя входами позволяет комбинировать различные выпуклые фигуры, получая невыпуклую фигуру (рис. 1.17).

При трех входах ИНС способна выделять заданные области трехмерного пространства, при четырех и более входах речь идет о гиперпространстве.

Заметим, что для двух логических переменных можно описать 16 логических функций, из которых 14 являются линейно разделимыми, т. е. могут быть реализованы на базе ИН с двумя входами. Для логической функции трех переменных доля линейно разделимых функций уменьшается (104 из 256) и с дальнейшим ростом числа переменных резко уменьшается [25].

Однако использование многослойных ИНС позволяет решать проблему линейной разделимости. При этом задача выбора весов и смещений уже не может быть решена так же легко, как для функции XOR. В этом случае требуется реализация процедуры обучения.

1.8. Правило обучения Хебба

Д. Хебб предложил формальное правило, в соответствии с которым вес w_{ij} связи нейрона i с нейроном j изменяется пропорцио-

нально уровням их возбуждения, т. е. произведению их выходных сигналов:

$$\Delta w_{ij} = \eta y_i y_j,$$

где $\eta \in [0,1]$ – коэффициент обучения.

При обучении с учителем вместо выходного сигнала y_j используется заданный выходной сигнал z_j .

В каждом цикле обучения происходит суммирование текущего значения веса и его приращения Δw_{ij} :

$$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}.$$

В результате применения правила Хебба веса нейрона могут принимать произвольно большие значения. Один из способов стабилизации обучения по правилу Хебба состоит в учете последнего значения w_{ij} , уменьшенного на коэффициент забывания γ . При этом правило Хебба представляется в виде

$$w_{ij}(t+1) = w_{ij}(t)(1 - \gamma) + \Delta w_{ij}.$$

Значение γ чаще всего составляет некоторый процент от коэффициента обучения η .

Модификацией правила обучения Хебба является дельта-правило, которое базируется на идее непрерывного изменения синаптических весов для уменьшения разности (дельта) между значением желаемого и текущего выходного сигнала:

$$\Delta w_{ij} = \eta(z_j - y_j).$$

1.9. Концепция входной и выходной звезды

Понятия входной и выходной звезды были введены С. Гроссбергом для анализа процесса обучения нейронов.

Входная звезда (*instar*) состоит из нейрона, на который подается группа входов, умноженных на синаптические веса (рис. 1.18).

Входная звезда должна выполнять задачу распознавания, т. е. ее реакция должна быть тем сильнее, чем больше входной вектор X похож на запомненный образ W . Если входной вектор и вектор весов нормализованы (имеют единичную длину), то максимальное значение скалярного произведения y вырабатывается при условии $X = W$.

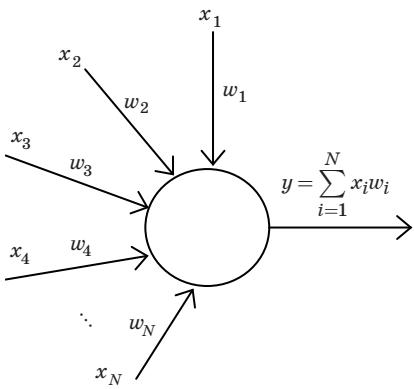


Рис.1.18. Входная звезда
Гроссберга

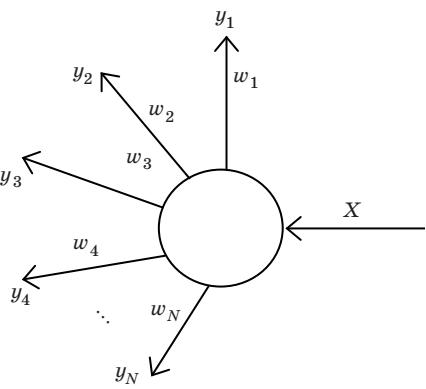


Рис.1.19. Выходная звезда
Гроссберга

Для того чтобы входная звезда реагировала не только на сам образ X , но и на его искаженные и зашумленные варианты, формируется обучающее множество векторов, незначительно отличающихся от эталона.

Для каждого вектора из обучающего набора выполняется коррекция весов

$$w_i^{t+1} = w_i^t + \alpha_t(x_i - w_i^t),$$

где t – момент времени; α_t – коэффициент скорости обучения, который имеет начальное значение порядка 0,1 и постепенно уменьшается.

Таким образом, веса входной звезды постепенно усредняются по набору обучающих векторов, приближаясь к центру кластера, который они образуют.

Выходная звезда Гроссберга (outstar) представляет собой нейрон, управляющий группой весов (рис. 1.19).

Выходная звезда должна выдавать требуемый вектор Y .

Процесс обучения выходной звезды напоминает процесс обучения входной. Он предполагает формирование обучающего множества выходных векторов с последующей коррекцией весов:

$$w_i^{t+1} = w_i^t + \alpha_t(y_i - w_i^t).$$

Входные и выходные звезды могут соединяться друг с другом в различных конфигурациях.

1.10. Парадигмы обучения

При обучении нейронной сети (неявном построении алгоритма решения задачи) топология сети обычно считается неизменной, а настройке подлежат веса связей между нейронами.

По способу использования учителя выделяют три основных метода обучения.

Обучение с учителем (*supervised learning*). В таком случае НС предоставляются входные данные и желаемые выходные и виде набора обучающих пар. Поскольку для каждого входного вектора задан выходной вектор, НС может корректировать веса, сравнивая текущие и эталонные выходы. В процессе обучения должны быть минимизированы ошибки по всем парам. Разумеется, обучающая выборка содержит только часть возможных вариантов входных сигналов, а цель обучения заключается в получении правильной реакции на любой допустимый входной сигнал. Поэтому данные обучающей выборки лишь задают опорные точки в пространстве входов-выходов сети, и цель обучения состоит в выполнении правильной интерполяции (экстраполяции) между этими опорными точками. Таким образом, правильное задание обучающей выборки также оказывает влияние на успех процесса обучения.

Обучение без учителя (*unsupervised learning*). В этом случае НС сама анализирует особенности входных данных с целью поиска структур и закономерностей. Веса подстраиваются так, чтобы получались согласованные выходные векторы, т. е. чтобы при получении близких входов получались близкие выходы, и наоборот. В процессе обучения ИНС выполняет кластеризацию (группирование) входной информации, а после обучения сеть работает в режиме векторного классификатора. Такой подход характерен для НС Кохонена. Процедура обучения без учителя во многом соответствует процессам, протекающим в живой природе. Живые существа в процессе развития собирают сходные раздражители в группы, и с каждой группой связывается положительная или отрицательная реакция, т. е. обучение можно понимать как обобщение и сжатие информации.

Обучение с подкреплением (*reinforcement learning*). Такой подход занимает промежуточное положение между двумя предыдущими. В этом случае вместо эталонного значения выхода НС используется оценка, формируемая внешней средой. Данная оценка имеет невысокую информативность, поэтому можно сказать, что обучающаяся система получает намного меньше информации, чем при обучении с учителем, и немного больше, чем при обучении без

учителя. Обучение с подкреплением является сложным процессом, основанным на методе проб и ошибок. Основной сферой применения данной парадигмы являются задачи, в которых необходимо найти оптимальную последовательность действий. Такие задачи характерны при нейросетевом управлении автономными агентами, например, в робототехнике.

По существу, задача обучения ИНС представляет собой задачу оптимизации, поэтому возможна еще одна классификация, которая делит все методы обучения ИНС на две группы:

- *детерминированные методы обучения*. Этими методами процедура коррекции весов НС осуществляется шаг за шагом на основании информации о входе сети, ее текущем и желаемом выходе. Такие методы обучения являются локальными, т. е. предполагают, что функция ошибки унимодальная;

- *стохастические методы обучения*. Такой подход соответствует глобальной оптимизации, при которой предполагается, что функция ошибки может иметь множество экстремумов (мультимодальная функция). Для того чтобы попасть в окрестность глобального экстремума, требуется совершать движения, временно ухудшающие целевую функцию. С этой целью в формулу коррекции весов вводится случайный фактор.

Таким образом, существуют две подгруппы методов обучения с учителем. К первой подгруппе относятся методы поградиентного спуска: метод наименьших квадратов и метод обратного распространения ошибки. Оба этих подхода локальные, т. е. адекватны при унимодальном характере функции ошибки. Ко второй подгруппе относятся стохастические методы, такие, как генетический алгоритм, роевой алгоритм, метод отжига металла и т. д. Эти методы работают и при мультимодальном характере функции ошибки, т. е. являются глобальными.

1.11. Предварительная обработка информации и оценка качества работы нейросети

На вход ИНС поступает числовая информация, которая может иметь разный смысл. С помощью кодирования можно описывать текстовую информацию, изображения и любые параметры исследуемых объектов для ИНС.

Качественные переменные (признаки) принимают конечное число значений.

Если значений всего два, то такая переменная кодируется одним битом. Если значений n , то потребуется N бит:

$$N = \log_2 n.$$

Для описания качественных признаков может быть использована теория нечетких множеств, в соответствии с которой для одного состояния объекта могут быть одновременно справедливы два или больше лингвистических значений, имеющих разную степень принадлежности [38].

Успех обучения ИНС во многом зависит от подготовки количественных обучающих данных, которая обычно содержит исключение аномальных наблюдений, заполнение пропущенных данных, а также нормирование входных переменных.

Так, в предыдущем примере рассматривалась схема, в которой производная берется от ступенчатой функции. В точке скачка эта производная получает бесконечное значение, что может сделать обучение невозможным. Для исправления ситуации достаточно заменить аномальное «бесконечное» значение на «большое».

Восстановление пропущенных данных может быть выполнено с помощью интерполяции (экстраполяции), моделирования или экспертизы оценок.

Нормирование означает приведение каждой компоненты входного вектора к интервалу $[0, 1]$ или $[-1, 1]$. При известном диапазоне изменения входной переменной $[x_{\min}, x_{\max}]$ нормирование выполняется по формуле

$$x_N = \frac{x - x_{\min}}{x_{\max} - x_{\min}}.$$

Если значения входной переменной требуется привести к заданному интервалу $[a, b]$, то применяется формула

$$x_N = \frac{(x - x_{\min})(b - a)}{x_{\max} - x_{\min}} + a.$$

Иногда значение x_{\max} является очень большим: $x_{\max} \rightarrow \infty$. В такой ситуации можно использовать формулу

$$x_N = \frac{1}{1 + e^{-x}}.$$

Если требуется учитывать малые значения больших величин, то можно использовать так называемую *модулярную обработку* дан-

ных, при которой число x заменяется вектором Z , каждую компоненту которого находят по формуле

$$z_i = \frac{((x \bmod y_i) + y_i)(b - a)}{2y_i} + a,$$

где z_i – i -я компонента вектора Z , а y_i – положительные целые числа из заданного набора.

В зависимости от особенности решаемой задачи полезными могут оказаться разнообразные преобразования как отдельных переменных (возвведение в степень, извлечение корня, взятие обратных величин и т. д.), так и их комбинаций (суммы, произведения, частные).

При оценке качества работы ИНС обычно требуется рассматривать среднюю квадратическую ошибку, определяемую как усредненная на n примерах сумма квадратов разностей между желаемой величиной выхода z_i и реально полученными на сети значениями y_i для каждого i -го примера:

$$E = \frac{1}{N} \sum_{i=1}^N (z_i - y_i)^2.$$

В задачах классификации разные группы примеров могут иметь разный вес либо сами примеры могут иметь разную значимость. Тогда в последнюю формулу целесообразно ввести весовые коэффициенты:

$$E = \frac{1}{N} \sum_{i=1}^N k_i (z_i - y_i)^2.$$

При аппаратной реализации ИНС часто применяют упрощенную оценку:

$$E = \sum_{i=1}^N |z_i - y_i| \text{ или } E = \max_i |z_i - y_i|.$$

Вопросы для самопроверки

1. С каким органом тела связаны интеллектуальные возможности человека? Менялись ли представления об этом на протяжении истории?

2. Является ли человек чемпионом по размеру мозга среди млекопитающих? Какова нормальная масса мозга человека?

3. Каков примерный процент совпадений генома человека и шимпанзе?
4. Какие признаки отличают человека от высших приматов?
5. Каковы общие представления о строении мозга человека?
6. Какой отдел мозга человека возник позднее других?
7. Какова примерная площадь коры головного мозга человека?
8. Каковы количественные оценки нервных клеток в коре головного мозга человека?
9. Сколько связей имеет нейрон с другими нейронами?
10. Какой процент всех энергозатрат организма приходится на работу мозга?
11. Сколько процентов кислорода тела потребляет мозг?
12. Что такое гематоэнцефалический барьер?
13. Какие функции выполняет мозолистое тело мозга?
14. Из каких частей состоит центральная нервная система?
15. Сколько светочувствительных клеток содержит глаз человека?
16. Сколько нервных волокон соединяет глаз со зрительной корой головного мозга?
17. Какие функции выполняет спинной мозг?
18. С какой скоростью распространяется возбуждение по нервным волокнам?
19. Как различаются нейроны по местоположению и функциям?
20. Какие уровни можно выделить при рассмотрении межнейронального взаимодействия?
21. Как количественно оценивается время извлечения информации из долговременной памяти человека?
22. Сколько градаций свойств какого-либо объекта может эффективно запомнить человек?
23. Какова примерная оценка количества мыслей, возникающих в мозгу человека в течение дня?
24. Какие способности использует человек при решении интеллектуальных задач?
25. Из каких частей состоит нейрон живого организма?
26. Какие размеры имеет нейрон?
27. Что такое дендриты? Каково их число?
28. Что такое аксон? Какова его длина?
29. Как называются области контакта нервных клеток друг с другом?
30. Что такое химические и электрические синапсы?
31. Чем пространственное суммирование раздражений в нейронах отличается от временного суммирования?

32. Каковы основные отличия описания работы мозга и традиционного компьютера? Назовите их.
33. Что такое искусственная нейронная сеть?
34. Что такое нейрокомпьютер и нейрокомпьютинг?
35. Как можно описать структуру нейрокомпьютера?
36. Какие режимы можно выделить при использовании нейрокомпьютера?
37. Как классифицируются задачи, решаемые человеком по признаку формализуемости?
38. Какие принципы используются при классификации нейронных сетей?
39. В чем заключается смысл задачи распознавания?
40. Что такое метрика?
41. Что такое разделяющая прямая? Каким уравнением она описывается?
42. Каким уравнением описывается разделяющая гиперплоскость?
43. Как выполняется классификация с помощью линейной машины?
44. Каково определение искусственного нейрона?
45. Из каких частей состоит искусственный нейрон?
46. Какие варианты активационной функции могут быть использованы?
47. Как выглядит искусственный нейрон для реализации функции AND?
48. Каков искусственный нейрон для реализации функции OR?
49. В чем заключается проблема линейной разделимости и как она решается?
50. Какова нейронная реализация функции XOR?
51. Как связаны число слоев нейронов и сложность решаемых задач при двух входных переменных?
52. Как формулируется правило обучения Хебба?
53. В чем особенности способа обучения входной звезды Гроссберга?
54. В чем особенности способа обучения выходной звезды Гроссберга?
55. Каковы три основные парадигмы обучения нейронных сетей?
56. Чем отличается детерминированное обучение от стохастического?
57. Какие операции могут выполняться при предварительной обработке обучающих данных для нейросети?
58. Как оценить качество обучения нейросети?

2. ОДНОСЛОЙНЫЕ НЕЙРОННЫЕ СЕТИ

2.1. Описание искусственного нейрона в MatLab

На рис. 2.1 представлен искусственный нейрон в системе MatLab.

При описании работы с пакетом Neural Net toolbox MatLab используется ряд общих обозначений. Так, скаляры обозначаются курсивными строчными буквами (a, b, c, \dots), векторы – прямыми строчными полужирными буквами ($\mathbf{a}, \mathbf{b}, \mathbf{c}, \dots$), матрицы – прямыми прописными полужирными буквами ($\mathbf{A}, \mathbf{B}, \mathbf{C}, \dots$).

На рис. 2.1 \mathbf{p} – вектор-столбец входа, \mathbf{W} – матрица весов (для одного нейрона это вектор-строка), f – активационная функция, a – выход нейрона.

Простейшая ИНС из одного нейрона с линейной активационной функцией может быть создана командой

```
>> net = newlin([0 10; 0 10],1);
```

Здесь матрица $[0 \ 10; 0 \ 10]$ описывает диапазоны входных значений для каждого входа ИНС (т. е. число строк соответствует числу входов), цифра указывает число нейронов; net – произвольное название создаваемой сети.

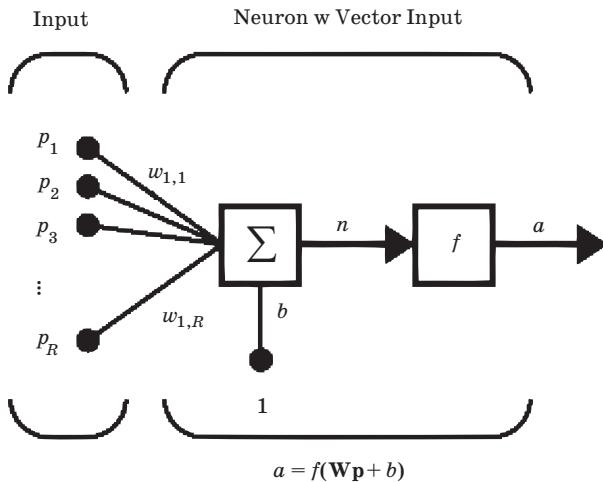


Рис. 2.1. Структура нейрона в MatLab

Веса и смещения нейрона могут быть заданы командами

```
>> net.IW{1,1}=[3 4]
>> net.b{1}=1
```

Здесь IW – матрица весов входного слоя (он же является первым слоем). Выражение в скобках {n, m} указывает на то, что веса соответствуют связи от слоя *m* к слою *n*. Выражение b{n} означает смещение *n*-го слоя.

Для моделирования работы сети следует описать некоторый входной вектор-столбец, длина которого равна числу входов:

```
>> P=[1; 3]
```

P =

```
1
3
```

Затем выполняется моделирование работы сети:

```
>> X=sim(net,P)
```

X =

```
16
```

Можно описать последовательность входных векторов:

```
>> P=[[1; 3],[2;4],[3;8]]
```

P =

```
1 2 3
3 4 8
```

Тогда выход сети будет представлять собой вектор

```
>> X=sim(net,P)
```

X =

```
16 23 42
```

Варианты активационных функций приведены в табл. 2.1.

Таблица 2.1

Варианты описания активационной функции в MatLab

Функция активации	Описание функции
hardlim(<i>x</i>)	Пороговая с порогом 0
hardlims(<i>x</i>)	Биполярная пороговая с порогом 0
purelin(<i>x</i>)	Линейная
logsig(<i>x</i>)	Логистическая (сигмоидная)
poslin(<i>x</i>)	Положительная линейная
satlin(<i>x</i>)	Положительная линейная с насыщением
satlins(<i>x</i>)	Биполярная линейная с насыщением
radbas(<i>x</i>)	Радиальная базисная
tribas(<i>x</i>)	Треугольная базисная
tansig(<i>x</i>)	Гиперболический тангенс

2.2. Персептрон

Персептрон (от лат. *perceptio* – восприятие) – простейшая искусственная нейронная сеть, выявленная в результате многолетних исследований мозга человека и животных [3]. Каждый из пяти видов информации о внешней среде (зрительная, слуховая, осязательная, обонятельная, вкусовая) воспринимается своими специализированными сенсорными нейронами и передается по отдельным сенсорным трактам в центральную нервную систему. Рассмотрим модель зрительной системы (рис. 2.2).

Модель включает в себя три последовательно соединенных множества нейронов: чувствительные *S*-элементы, ассоциирующие *A*-элементы и реагирующие *R*-элементы.

Чувствительным элементам соответствуют сенсорные нейроны, ассоциирующим – локальные специализированные зрительные центры в коре мозга и реагирующими – моторные нейроны, передающие управляющие сигналы мышцам и железам организма. Можно считать, что *R*-элемент реагирует на определенный класс изображений.

Сенсорные нейроны возбуждаются от воздействия энергии света при превышении некоторого порога. В свою очередь *A*-элементы возбуждаются только тогда, когда возбуждено достаточное число *S*-элементов, связанных с ними. *R*-элемент возбуждается, если на входе возникает определенная комбинация *A*-элементов.

Заметим, что под персептроном понимается обычно однослойная нейронная сеть. Многослойный персептрон называется *нейронной сетью прямого распространения*.

Сетчатка глаза из *S*-элементов

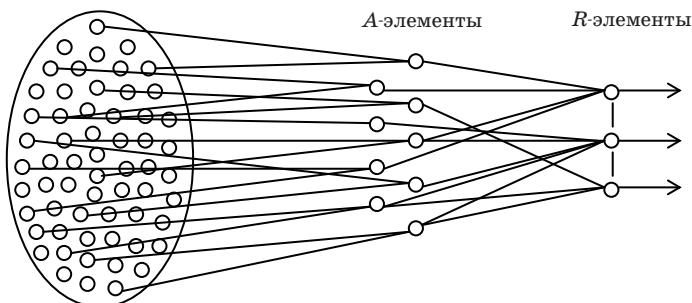


Рис. 2.2. Структура модели зрительной системы

Рассмотрим персепtron с одним выходом, т. е. фактически один нейрон, имеющий множество входных сигналов. Задача обучения персептрона заключается в возбуждении его только при представлении определенных входных векторов и отсутствии реакции на другие входные векторы.

Таким образом, при обучении персептрона решается задача бинарной классификации – путем настройки весов персептрона строится гиперплоскость, разделяющая все входные векторы на два класса.

Например, пусть требуется обучить персептрон распознавать четные и нечетные цифры по их изображениям. Эталонное изображение каждой цифры разбивается на сегменты с заданной степенью подробности. Если в пределы сегмента попадает часть изображения, то ему соответствует единичный сигнал, в противном случае – сигнал нулевой. Так, каждая цифра получает соответствующий бинарный код. Пример для цифры «5» приведен на рис. 2.3.

Таким образом, получаем десять обучающих пар, описывающих вход и желаемый выход персептрона (табл. 2.2). Поскольку желаемое решение заранее известно, ошибку работы ИНС можно вычислять в явном виде, и речь идет об обучении с учителем.

Смысл использования персептрона заключается в том, что после обучения на его вход могут поступать уже не эталонные, а искаженные и зашумленные векторы. Реакция ИНС должна быть правильной до достижения определенного максимального уровня шумов. Как следует из табл. 2.2, некоторые образы различаются только значением одного бита, так что задачу распознавания не удастся решить уже при минимальном шуме, поэтому в реальности описание должно быть более подробным.

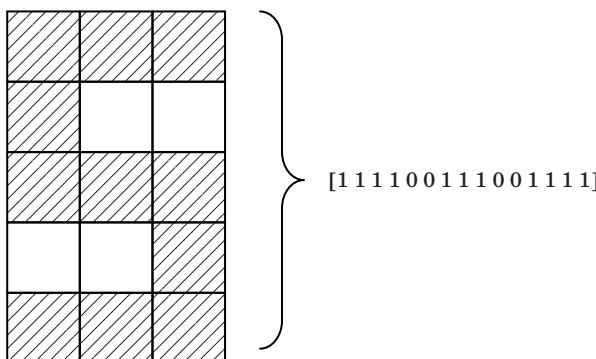


Рис. 2.3. Кодирование изображений цифр

Таблица 2.2

Обучающие пары

Цифра	Входной вектор X	Признак четности z	Двоичный код цифры Z
0	[1 1 1 1 0 1 1 0 1 1 0 1 1 1 1]	1	0000
1	[0 1 0 0 1 0 0 1 0 0 1 0 0 1 0]	0	0001
2	[1 1 1 0 0 1 1 1 1 1 0 0 1 1 1]	1	0010
3	[1 1 1 0 0 1 1 1 1 0 0 1 1 1 1]	0	0011
4	[1 0 1 1 0 1 1 1 1 0 0 1 0 0 1]	1	0100
5	[1 1 1 1 0 0 1 1 1 1 0 0 1 1 1]	0	0101
6	[1 1 1 1 0 0 1 1 1 1 0 1 1 1 1]	1	0110
7	[1 1 1 0 0 1 0 1 0 1 0 0 1 0 0]	0	0111
8	[1 1 1 1 0 1 1 1 1 1 0 1 1 1 1]	1	1000
9	[1 1 1 1 0 1 1 1 1 0 0 1 1 1 1]	0	1001

При обучении персептрон получает поочередно входные векторы в соответствии с рис. 2.4.

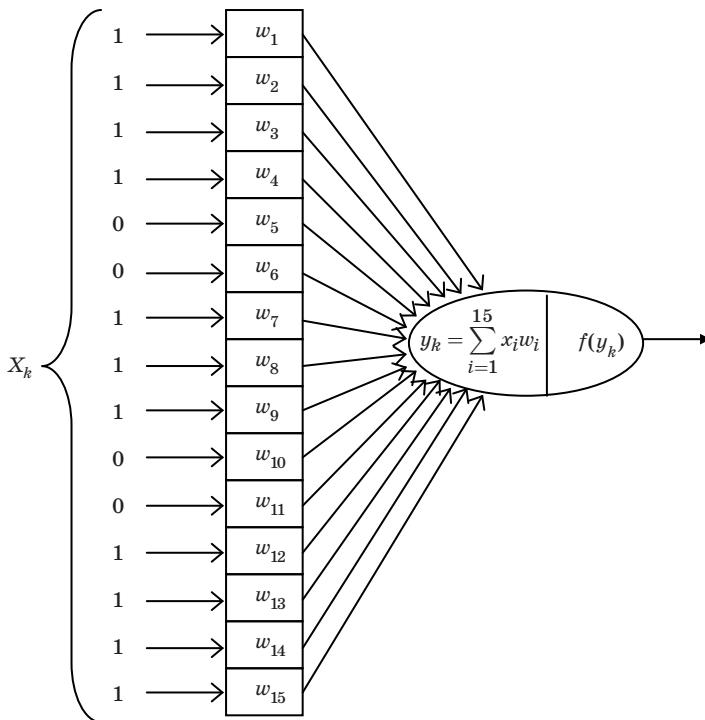


Рис. 2.4. Обучение персептрана

Ошибку обучения в рассматриваемой задаче можно описать формулой

$$E = \sum_{k=0}^9 |e_k| = \sum_{k=0}^9 |f(y_k) - z_k|, \quad (2.1)$$

где $f(y_k)$ и z_k – реальный и заданный выход персептрана для k -го обучающего входа.

Активационная функция выбирается пороговой. Первоначально матрица весовых коэффициентов W получает случайные значения.

Алгоритм обучения персептрана сводится к следующим действиям:

1. На вход подается k -й образ из обучающего набора.
2. Вычисляется ошибка $e_k = f(y_k) - z_k$.
3. Если $e_k = 0$, то весовые коэффициенты W не изменяются.
4. Если $e_k \neq 0$, то изменяются те весовые коэффициенты, которые усиливают ошибку (w_i , которым соответствуют ненулевые значения x_i):

$$w_i \leftarrow (w_i - \text{sgn}(e_k) x_i)$$

(при таком описании шаг обучения равен единице, и порог нейрона должен быть в несколько раз больше).

5. Ошибка e_k запоминается. Шаги 1–6 повторяются по всем обучающим парам. Вычисляется ошибка обучения (2.1). Если $E > 0$, то происходит переход на шаг 1.

Если задача относится к классу реализуемых на персептране, то описанный алгоритм позволит настроить весовые коэффициенты желаемым образом.

Последовательность подачи на вход ИНС всех обучающих примеров называется *эпохой*.

Персептрон может иметь более одного выхода. Например, если мы хотим распознать конкретную цифру, то на выходе персептрана должен появляться двоичный код этой цифры, состоящий из четырех битов (см. табл. 2.2). Соответственно потребуется четыре ИН (рис. 2.5).

Формула ошибки обучения будет иметь вид

$$E = \sum_{k=0}^9 \Delta_k = \sum_{k=0}^9 \left(\sum_{j=1}^4 \delta_{kj} \right) = \sum_{k=0}^9 \left(\sum_{j=1}^4 |f(y_{kj}) - z_{kj}| \right). \quad (2.2)$$

Активационная функция также выбирается пороговой, а матрица W получает первоначально случайные значения.

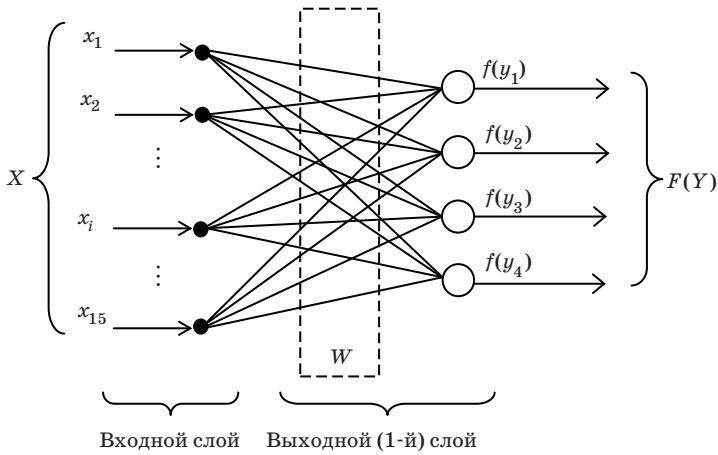


Рис. 2.5. Персепtron с множеством выходных нейронов

Алгоритм обучения персептрана несколько изменяется:

1. На вход подается k -й образ из обучающего набора.
2. Вычисляется ошибка для первого нейрона:

$$\delta_{k1} = f(y_{k1}) - z_{k1}.$$

3. Корректируются весовые коэффициенты нейрона, усиливающие ошибку выхода.

4. Шаги 2 – 3 повторяются для остальных нейронов.

5. Δ_k запоминается. Шаги 1–4 повторяются по всем обучающим парам. Вычисляется ошибка обучения (2.2). Если $E > 0$, то происходит переход на шаг 1.

Заметим, что способ кодирования изображения, показанный на рис. 2.3, на практике неприменим, так искажение всего одного бита приводит к неправильному распознаванию. В реальных задачах рассматриваются векторы большей размерности.

Входы и выходы персептрана могут быть действительными числами. В этом случае закон обучения персептрана обобщается в виде так называемого *дельта-правила*, в соответствии с которым алгоритм коррекции веса связи между i -м входом и j -м выходом записывается в следующем виде:

$$w_{ij}(k+1) = w_{ij}(k) + \eta \delta_{kj} x_i, \quad (2.3)$$

где η – коэффициент скорости обучения (обычно $\eta \in [0.01, 0.1]$), который постепенно уменьшается.

Выражение (2.3) показывает, что величина коррекции веса пропорциональна соответствующей компоненте входного сигнала и ошибке выхода.

На рис. 2.6 приведено описание структуры персептрона, используемое в MatLab.

Создание персептрона происходит по команде

```
>> net = newp(PR, S)
```

Здесь PR – матрица $R \times 2$, описывающая минимальное и максимальное значения по каждому из R входов; S – число нейронов.

Можно использовать более подробное описание:

```
>> net = newp(PR, S, tf, lf)
```

где tf – передаточная функция из списка {hardlim, hardlims}, при чем по умолчанию задается hardlim; lf – обучающая функция из списка {learnp, learnpn}, по умолчанию – learnp.

При создании персептрона матрица весов и вектор смещений инициализируются нулями с помощью функций initzero. Можно также проинициализировать значения весов и смещений командами

```
>> net.IW{1,1} = [-1 1];
```

```
>> net.b{1} = [1]
```

Обучение персептрона производится с помощью функции адаптации adapt, которая корректирует веса и смещения по результатам обработки каждой пары входных и выходных значений (обучение с учителем):

```
>> adapt(net,P,T)
```

где P – входные векторы; T – целевые значения.

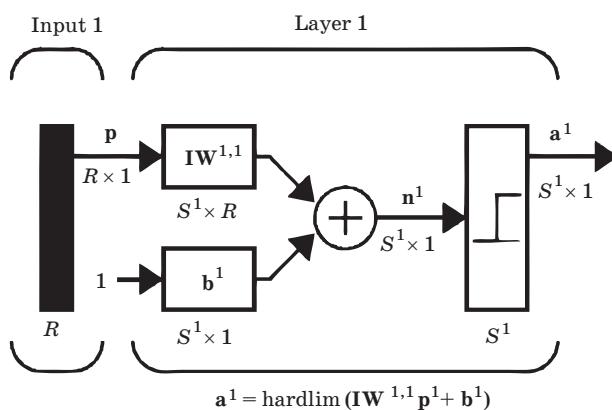


Рис. 2.6. Графическое изображение персептрона

Применение функции `adapt` гарантирует, что любая задача классификации с линейно отдельмыми векторами будет решена за конечное число циклов настройки.

Функция обучения `train` также может быть использована для обучения персептрона, но здесь настройка выполняется не после каждого прохода, а в результате всех проходов обучающего множества. Каждый пересчет для набора входных векторов является эпохой.

Рассмотрим пример бинарной классификации четырех векторов:

```
>> p = [[2;2] [1;-2] [-2;2] [-1;-0.5]];
>> t =[0 1 0 1];
```

С помощью функции

```
>> plotpv(p,t)
```

входной и целевой векторы можно отобразить графически. Это помогает понять, является ли задача линейно разделимой (рис. 2.7):

```
>> net = newp([-2 2;-2 2],1);
>> net = train(net,p,t);
TRAINC, Epoch 0/100
TRAINC, Epoch 2/100
TRAINC, Performance goal met.
```

Появившийся на экране график показывает, что задача решена за три эпохи (рис. 2.8).

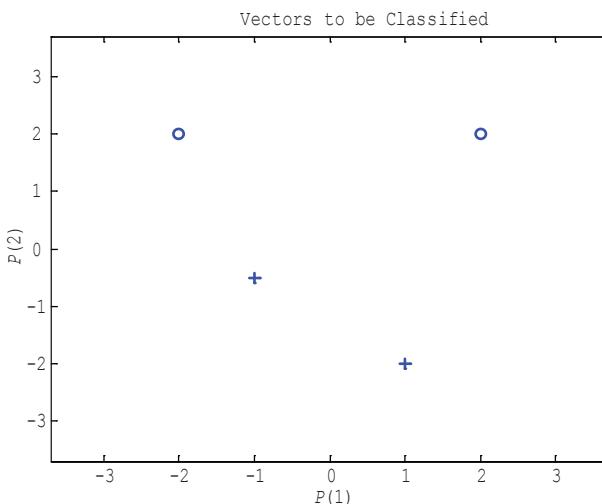


Рис. 2.7. Исходные данные для классификации

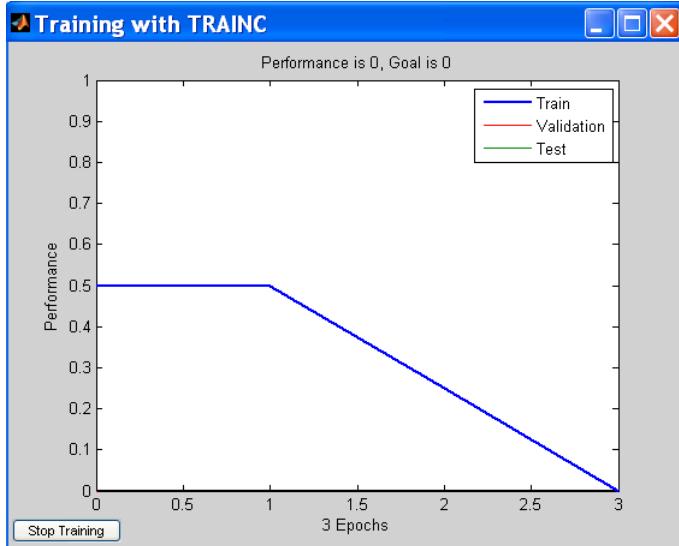


Рис. 2.8. Графический протокол обучения персептрона

С помощью функции plotpc можно проанализировать положение разделяющей прямой после обучения (рис. 2.9):

```
>> plotpc(net.IW{1,1},net.b{1});
```

Проверку качества работы сети можно выполнить с помощью команды sim:

```
>> a = sim(net,p)
a =
0 1 0 1
```

Выход сети совпадает с целевым вектором t .

Следующий пример описывает обучение персептрона реализации функции OR:

```
>> percOR=newp([0 1; 0 1],1);
>> input =[0 0 1 1; 0 1 0 1]; d=[0 1 1 1];
>> plotpv(input,d);
>> percOR.adaptParam.passes=20;
>> percORa=adapt(percOR,input,d);
>> plotpc(percORa.IW{1},percORa.b{1});
>> rez=sim(percORa,[0 0 1 1; 0 1 0 1])
rez =
0 1 1 1
```

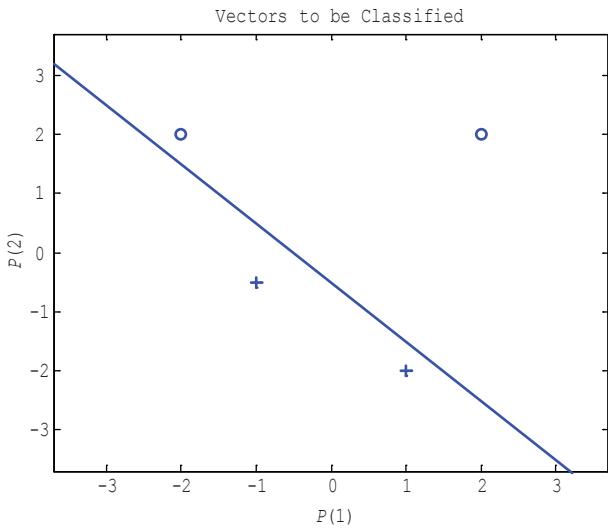


Рис. 2.9. Разделяющая прямая персептрана

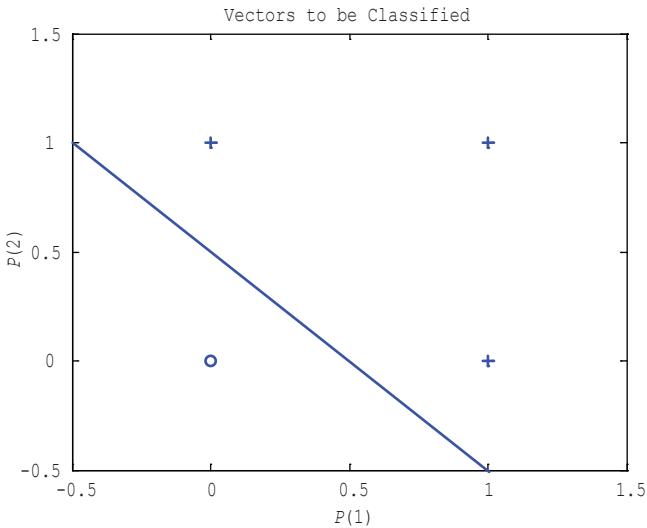


Рис. 2.10. Персептронная реализация функции OR

На рис. 2.10 приведена получившаяся разделяющая прямая.

Рассмотрим более сложный пример. Зададим на плоскости набор из четырех классов A, B, C, D . Каждый класс представляет со-

бой набор точек на плоскости. Классы смещены относительно друг друга для обеспечения сепарабельности:

```
>> A = [rand(1,20) - 0.6; rand(1,20) + 0.6];
>> B = [rand(1,20) + 0.6; rand(1,20) + 0.6];
>> C = [rand(1,20) + 0.6; rand(1,20) - 0.6];
>> D = [rand(1, 20) - 0.6; rand(1,20) - 0.6];
>> plot(A(1,:),A(2,:),'bs')
>> hold on
>> plot(B(1,:),B(2,:),'r+')
>> plot(C(1,:),C(2,:),'go')
>> plot(D(1,:),D(2,:),'m*')
>> grid on
>> text(-.1,1.7,'Class A')
>> text(1.1,1.7,'Class B')
>> text(-0.1,-0.7,'Class C')
>> text(1.1,-0.7,'Class D')
```

Получившиеся классы показаны на рис. 2.11.

Закодируем каждый класс двухбитовым кодом:

```
>> a = [0 1];
>> b = [1 1];
>> c = [1 0];
>> d = [0 0];
```

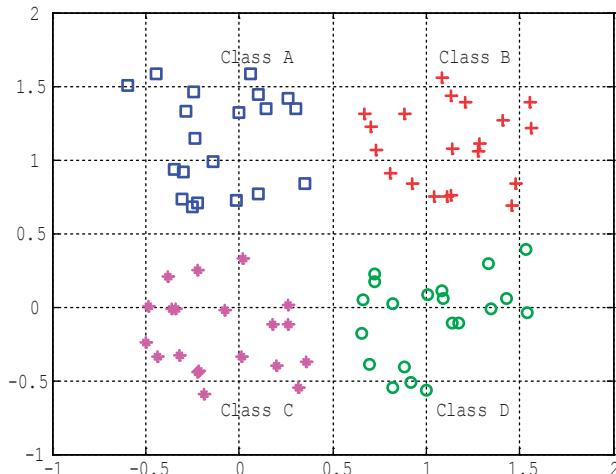


Рис. 2.11. Четыре класса объектов на плоскости

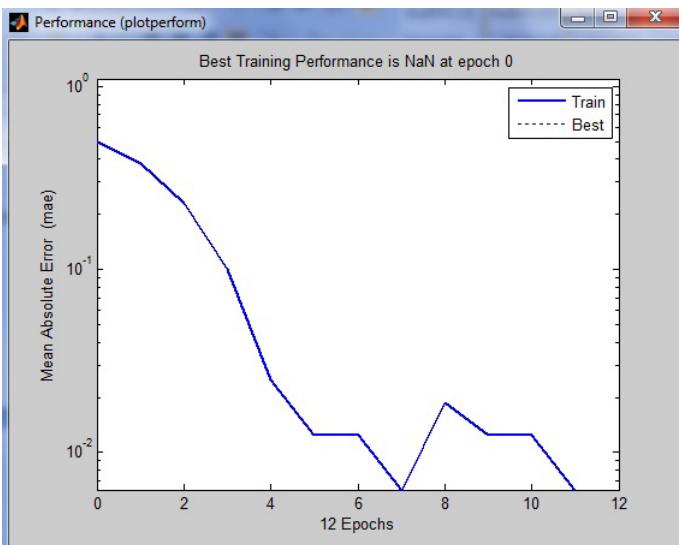


Рис. 2.12. Изменение ошибки в процессе обучения

Зададим вектор входа и целевой вектор:

```
>> P = [A B C D];
>> T = [repmat(a,1,length(A)) repmat(b,1,length(B)) ...
    repmat(c,1,length(C)) repmat(d,1,length(D))];
```

С помощью последней команды каждой точке плоскости ставим в соответствие вектор. Затем создаем и обучаем персептрон:

```
>> net = newp([-1 2;-1 2],2);
>> net = train(net,P,T);
```

Процесс обучения персептрана иллюстрирует рис. 2.12.

Таким образом, в рассмотренном примере персептрон имел два бинарных выхода.

2.3. Линейная нейронная сеть

Линейные нейронные сети, или ADALIN (ADaptive LInear Neuron networks), по структуре аналогичны персептрану и отличаются лишь функцией активации и алгоритмом обучения. Активационная функция здесь линейная, а алгоритм обучения – метод наименьших квадратов (МНК), который является более эффективным, чем правило обучения персептрана. Линейный нейрон с двумя входами приведен на рис. 2.13.

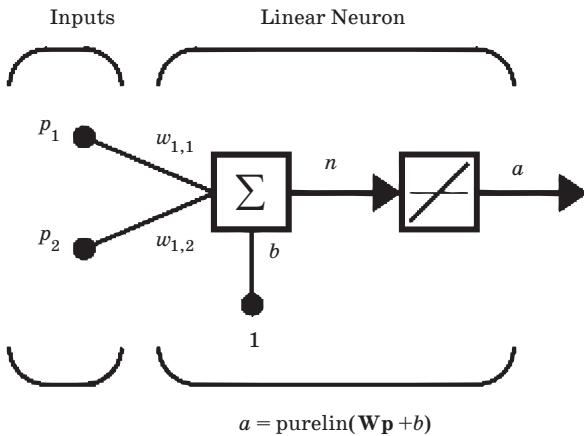


Рис. 2.13. Линейный нейрон в MatLab

Как следует из рис. 2.13, помимо основных входов нейроны линейной сети имеют вход для постоянного смещения, равного единице.

Линейные сети, как и персептроны, способны решать только линейно отделимые задачи классификации, разделяющая прямая определяется уравнением

$$WP + b = 0.$$

Можно записать

$$n = WP + b = w_{11}p_1 + w_{12}p_2 + b = 0.$$

Пересечения с осями координат:

$$p_1 = 0: p_2 = -b/w_{12},$$

$$p_2 = 0: p_1 = -b/w_{11}.$$

Таким образом, вектор весов здесь определяет наклон прямой, а константа b – ее смещение относительно начала координат.

При $b = 0$ выполняется $WP = 0$, поэтому вектор весов всегда перпендикулярен разделяющей прямой.

Рассмотрим пример.

Пусть $w_{1,1} = 1$, $w_{1,2} = 0,5$, $b = -1$. Тогда, например, для точки с координатами $p_1 = 2$, $p_2 = 2$ имеем

$$a = \text{purelin}(WP + b) = [1 \quad 0,5] \begin{bmatrix} 2 \\ 2 \end{bmatrix} - 1 = 2 > 0.$$

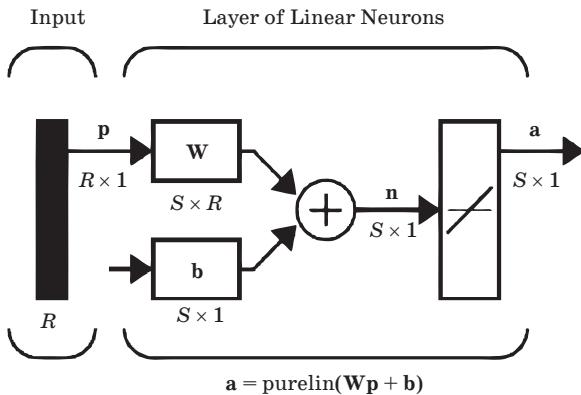


Рис. 2.14. Линейная нейронная сеть в MatLab

Линейные нейроны могут образовывать однослойную линейную нейронную сеть (рис. 2.14).

Для создания линейной НС можно использовать следующие варианты команд:

```
>> net=newlin(PR, S, id, lr),
>> net=newlin(PR, S, 0, P),
>> net=newlind(P, T),
```

где PR – массив размером $R \times 2$ минимальных и максимальных значений R векторов входа; S – число нейронов; id – описание линии задержки на выходе сети (по умолчанию нуль); lr – параметр скорости настройки (по умолчанию 0,01); P – обучающие последовательности входов размером $R \times Q$, где Q – число последовательностей; T – последовательность целей для P размером $S \times Q$;

Рассмотрим пример. Линейная нейронная сеть создается командой

```
>> net = newlin([-1 1; -1 1],1);
```

Здесь матрица описывает пределы изменения двух скалярных входов, цифра «1» соответствует числу выходов, т. е. числу нейронов.

Задать значения весов и смещения можно командами

```
>> net.IW{1,1}=[1 0.5];
>> net.b{1}=[-1];
```

Входной вектор

```
>> p=[2; 2];
```

Моделирование работы сети

```
>> a = sim(net,p)
```

```
a = 2
```

2.4. Рекуррентный метод наименьших квадратов

Рассмотрим использование метода наименьших квадратов для обучения линейных сетей.

Как и для персептрана, для линейной сети применяется процедура обучения с учителем, которая использует обучающее множество из N пар векторов:

$$\{P_1, Z_1\}, \{P_2, Z_2\}, \dots, \{P_N, Z_N\},$$

где P_i – входной вектор; Z_i – заданный выходной вектор.

Пусть A_i – выход сети для входа P_i . Тогда функцию ошибки на k -й итерации можно представить в виде

$$E(k) = \frac{1}{N} \sum_{i=1}^N (Z_i - A_i)^2.$$

Представим линейный нейрон на k -й итерации в виде, приведенном на рис. 2.15.

В процессе обучения сети требуется найти такие значения весов и смещений, чтобы значение e было минимальным. Эта задача разрешима, так как для линейной сети поверхность ошибки как функция входов имеет единственный минимум. Поэтому функция ошибки уменьшается в направлении, обратном ее градиенту:

$$\Delta w_{1,j} = w_{1,j}(k+1) - w_{1,j}(k) = -\alpha \frac{\partial(e^2(k))}{\partial w_{1,j}}, \quad j = \overline{1, R},$$

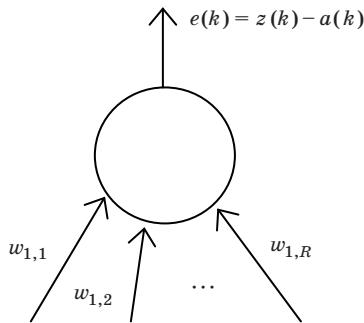


Рис. 2.15. Обучение линейного нейрона

$$\begin{aligned} \frac{\partial(e^2(k))}{\partial w_{1,j}} &= 2e(k) \frac{\partial e(k)}{\partial w_{1,j}} = 2e(k) \frac{\partial(z(k) - a(k))}{\partial w_{1,j}} = \\ &= 2e(k) \frac{\partial \left(z(k) - \left(\sum_{j=1}^R w_{1,j} p_j(k) + b \right) \right)}{\partial w_{1,j}} = -2e(k) p_j(k), \quad j = \overline{1, R}. \end{aligned}$$

Следовательно,

$$\begin{aligned} \Delta w_{1,j} &= w_{1,j}(k+1) - w_{1,j}(k) = -\alpha(-2e(k)p_j(k)) = \\ &= 2\alpha e(k)p_j(k) = \eta e(k)p_j(k), \quad j = \overline{1, R}, \end{aligned}$$

$$w_{1,j}(k+1) = w_{1,j}(k) + \eta e(k)p_j(k), \quad j = \overline{1, R}.$$

Аналогично можно получить

$$b_j(k+1) = b_j(k) + \eta e(k), \quad j = \overline{1, R},$$

где η – константа скорости обучения.

Таким образом, вес и смещение изменяются в направлении антиградиента пропорционально значению ошибки.

Для линейной НС из множества нейронов обучающие правила обобщаются:

$$W(k+1) = W(k) + \eta E(k) P^T(k),$$

$$B(k+1) = B(k) + \eta E(k).$$

В целом алгоритм обучения предполагает следующие шаги:

1. Выбирается первая пара из обучающего множества $\{P_1, Z_1\}$ (или случайная пара).

2. Вычисляется ошибка выхода сети.

3. Корректируются веса и смещения сети.

4. Проверяются критерии остановки. Если они не выполняются, то выбирается следующая пара и повторяются шаги 2 и 3.

В качестве критериев окончания процесса обучения может быть выбран порог ошибки e_{\min}^2 (по всему обучающему множеству), или достижение максимального числа итераций.

Рассмотрим пример. Пусть задано обучающее множество ($N = 2$):

$$\{p_1 = [-1 \ 1 \ -1]^T, t_1 = [-1]\}, \{p_2 = [1 \ 1 \ -1]^T, t_1 = [1]\}.$$

Параметры алгоритма МНК: $\eta = 0,4$, $e_{\min}^2 = 0,1$.

На первой итерации веса равны нулю, и для первой обучающей пары

$$a(1) = W(1)P_1 = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix} = 0,$$

$$e(1) = z_1 - a(1) = -1 - 0 = -1,$$

$$\begin{aligned} W(2) &= W(1) + \eta e(1)P^T(1) = \\ &= [0 \ 0 \ 0] + 0,4(-1) \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix}^T = [0,4 \ -0,4 \ 0,4]. \end{aligned}$$

На второй итерации рассматривается вторая обучающая пара:

$$a(2) = W(2)P(2) = [0,4 \ -0,4 \ 0,4] \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix} = -0,4,$$

$$e(2) = z_2 - a(2) = 1 - (-0,4) = 1,4,$$

$$\begin{aligned} W(3) &= W(2) + \eta e(1)P^T(2) = \\ &= [0,4 \ -0,4 \ 0,4] + 0,4(1,4) \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix}^T = [0,96 \ 0,16 \ -0,16]. \end{aligned}$$

Вычисляемая ошибка по всему обучающему множеству равна

$$W(3)p_1 = [0,96 \ 0,16 \ -0,16] \cdot \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix} = -0,64,$$

$$W(3)p_2 = [0,96 \ 0,16 \ -0,16] \cdot \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix} = 1,28,$$

$$e = \frac{1}{2}((-0,36)^2 + (0,28)^2) = 0,2.$$

Так как $e^2 > e_{\min}^2$, итерации должны быть продолжены.
Рассмотрим реализацию этого простого примера в MatLab:

```
>> p=[-1 1; 1 1; -1 -1];
>> t=[-1 1];
>> net=newlin([-1 1; -1 1; -1 1],1);
>> net.trainParam.goal=0.1;
>> [net,tr]=train(net,p,t);
TRAINB, Epoch 0/100, MSE 1/0.
TRAINB, Epoch 25/100, MSE 0.36417/0.
TRAINB, Epoch 50/100, MSE 0.13262/0.
TRAINB, Epoch 75/100, MSE 0.048296/0.
TRAINB, Epoch 100/100, MSE 0.0175879/0.
TRAINB, Maximum epoch reached.
```

Работа закончена по достижении заданного числа эпох (100), хотя требуемое качество обучения еще не достигнуто.

Заметим, что для линейной нейронной сети справедливы те же ограничения, что и для персептрона, т. е. обучающие данные должны быть линейно разделимы.

При обучении ИНС с помощью процедуры `train` можно указывать точность (ошибку) обучения и число эпох обучения. Например,

```
>> p = [2 1 -2 -1; 2 -2 2 1]; t = [0 1 0 1];
>> net = newlin([-2 2; -2 2],1);
>> net.trainParam.goal= 0.1;
>> net.trainParam.epochs = 60;
>> net = train(net,p,t);
```

На рис. 2.16 показано, как меняется среднеквадратическая ошибка в процессе обучения, приближаясь к заданному порогу.

Многослойные линейные сети не имеют смысла, так как подобная сеть всегда преобразуется в однослойную. Для доказательства этого тезиса рассмотрим пример линейной нейронной сети, приведенный на рис. 2.17.

Можно записать

$$\begin{aligned}y_1 &= x_1 w_{11} + x_2 w_{21} + b_1, \\y_2 &= x_1 w_{12} + x_2 w_{22} + b_2, \\z &= y_1 v_1 + y_2 v_2 + b_3 = (x_1 w_{11} + x_2 w_{21} + b_1) v_1 + \\&+ (x_1 w_{12} + x_2 w_{22} + b_2) v_2 + b_3 = x_1 (w_{11} v_1 + w_{12} v_2) + \\&+ x_2 (w_{21} v_1 + w_{22} v_2) + b_1 v_1 + b_2 v_2 + b_3 = x_1 q_{11} + x_2 q_{21} + b,\end{aligned}$$

где $q_{11} = w_{11} v_1 + w_{12} v_2$; $q_{21} = w_{21} v_1 + w_{22} v_2$; $b = b_1 v_1 + b_2 v_2 + b_3$.

При нулевых смещениях можно использовать матричную запись

$$Z = VW^T X = QX.$$

Комбинация из нескольких нейронов ADALIN называется MADALINE (Many ADaptive LInear NEurons). Пример приведен на рис. 2.18.

Такая ИНС может выполнять, например, распознавание символов независимо от их ориентации на плоскости. Отдельные модули

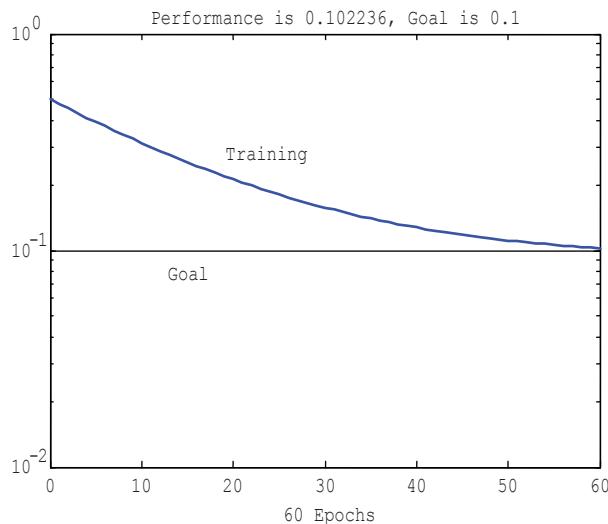


Рис. 2.16. График процесса обучения

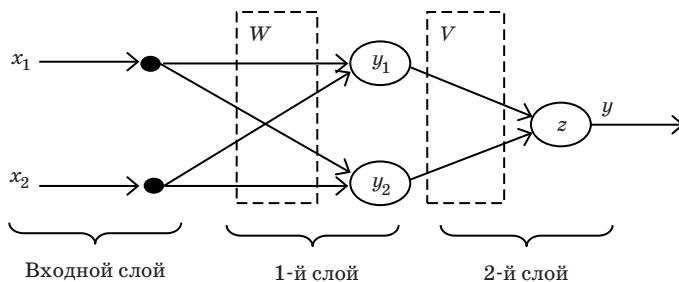


Рис. 2.17. Двухслойная линейная нейронная сеть

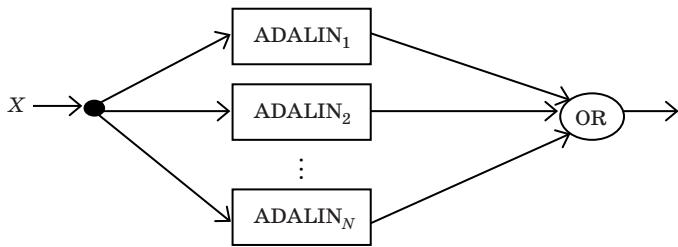


Рис. 2.18. Пример нейронной сети MADALINE

ADALIN отвечают за повороты символа на определенный угол, а узел OR обеспечивает инвариантность распознавания.

2.5. Линейная сеть с линией задержки

Линейные сети с задержкой входного сигнала позволяют создавать динамические нейронные сети ADALIN, которые применяются при решении задач обработки сигналов и в системах управления.

Последовательность значений входного сигнала $\{p(k)\}$ поступает на линию задержки, состоящую из $N-1$ блока запаздывания.

Рассмотрим пример. Определим линейную нейронную сеть:

```
>> net = newlin([0,11],1);
```

Опишем две линии задержки на входе НС (рис. 2.19):

```
>> net.inputWeights{1,1}.delays = [0 1 2];
```

Зададим веса и смещения нейронной сети

```
>> net.IW{1,1} = [2 4 11];
```

```
>> net.b{1} = [3];
```

и начальные значения выходов линии задержки

```
>> pi = {1 2}
```

Выход сети будем описывать формулой

$$a(k) = \text{purelin}(Wp + b) = \sum_{i=1}^3 w_{1,i} p(k-i+1) + b.$$

Рассмотрим реакцию сети при подаче некоторой входной последовательности:

```
>> p = {1 2 5 9}
```

```
>> [a, pf] = sim(net, p, pi);
```

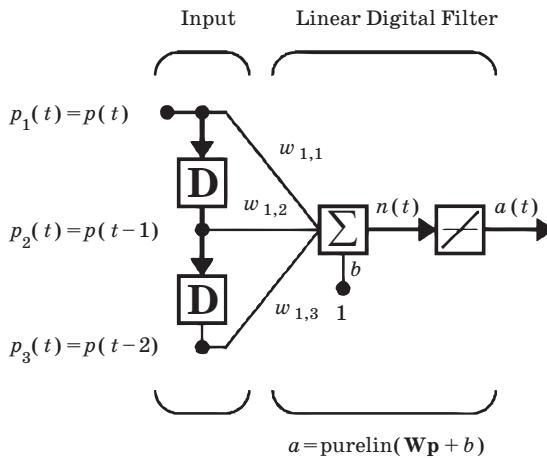


Рис. 2.19. Линейная нейронная сеть с двумя линиями задержки

Выходная последовательность имеет вид

$$a = [24] \quad [33] \quad [22] \quad [63]$$

Проверка:

$$a(1) = 1 \cdot 2 + 2 \cdot 4 + 1 \cdot 11 + 3 = 24,$$

$$a(2) = 2 \cdot 2 + 1 \cdot 4 + 2 \cdot 11 + 3 = 33,$$

$$a(3) = 5 \cdot 2 + 2 \cdot 4 + 1 \cdot 11 + 3 = 32,$$

$$a(4) = 9 \cdot 2 + 5 \cdot 4 + 2 \cdot 11 + 3 = 63.$$

Допустим, что нейронная сеть должна выдавать в ответ на входную последовательность 4, 5, 6, 7 заданную выходную последовательность 20, 25, 30, 35. Для этого требуется адаптация весов:

```

>> net = newlin([0,10],1);
>> net.inputWeights{1,1}.delays = [0 1 2];
>> net.IW{1,1} = [7 8 9];
>> net.b{1} = [0];
>> pi = {1 2};
>> p = {4 5 6 7};
>> T = {20 25 30 35}
>> net.adaptParam.passes = 200;
>> [net,y,E,pf,af] = adapt(net,p,T,pi);
>> T =
[20] [25] [30] [35]

```

Следующий пример описывает использование линейной НС с задержкой для моделирования динамического звена:

```
>> sys = ss(tf(1, [1 1 1])); % – описание колебательного звена  
>> t = 0:0.2:10; % – интервал и шаг моделирования  
>> [Y, t] = step(sys, t) % – реакция звена на скачок  
>> plot(t,Y)  
>> grid  
>> xlabel('t')  
>> ylabel('Y(t)')
```

Полученный переходный процесс приведен на рис. 2.20.

Следующие команды подготавливают исходные данные и выполняют адаптацию весов ИНС:

```
>> P = Y(1: length(t)-2);  
>> T = Y(3: length(t));  
>> Z=[1 2];  
>> net = newlin([-1 1], 1, Z);  
>> pi ={0 0};  
>> net.IW{1,1} =[1 1];  
>> net.adaptParam.passes = 200;  
>> P1 = num2cell(P);  
>> T1 = num2cell(T);  
>> [net,y,E pf,af] = adapt(net,P1,T1,pi);
```

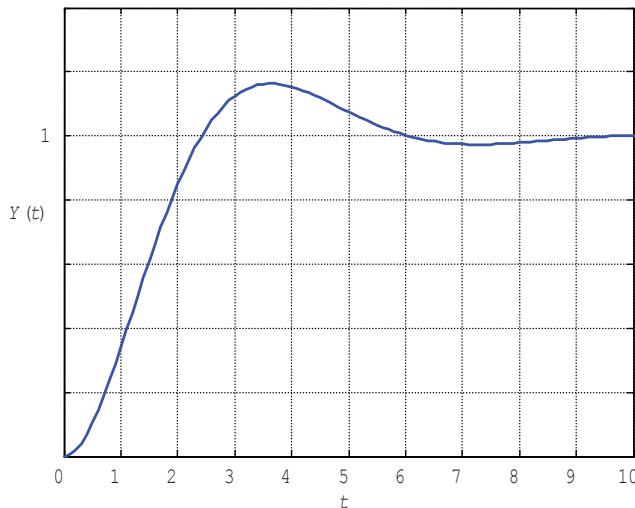


Рис. 2.20. Переходный процесс колебательного звена

Выполним моделирование ИНС и сравним заданный и полученный процессы (рис. 2.21):

```
>> x = sim(net,P1)
>> x1 = cat(1,x{:});
>> t1=t(1: length(t)-2)';
>> plot(t1,x1,'b:+', t1,P,'r-o')
```

Ошибка в начале переходного процесса вызвана методом формирования обучающей выборки.

При обучении можно было бы использовать команду

```
>> [net,y,E pf,af] = adapt(net,P1,P1,pi);
```

Ошибка в начале переходного процесса уменьшилась (рис. 2.22).

Рассмотрим следующий пример. Пусть требуется, чтобы линейная ИНС воспроизводила сигнал, приведенный на рис. 2.23. Ему соответствует набор команд

```
>> t = 0: 0.01: 6;
>> y = [sin(4.1*pi*0.2.*t.*abs(0.7*t - 5))];
>> plot(t,y)
>> ylim([-1.2 1.2])
>> xlabel('Time [sec]');
>> ylabel('Target Signal');
>> grid on
```

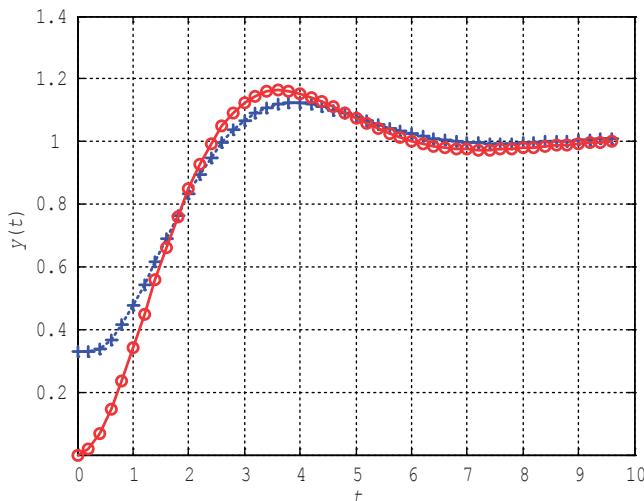
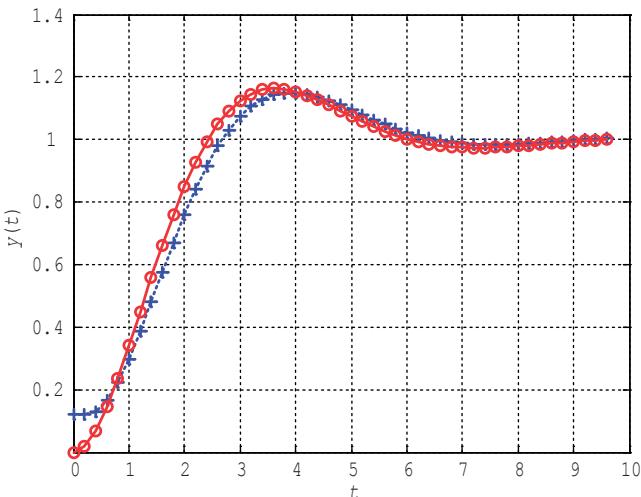
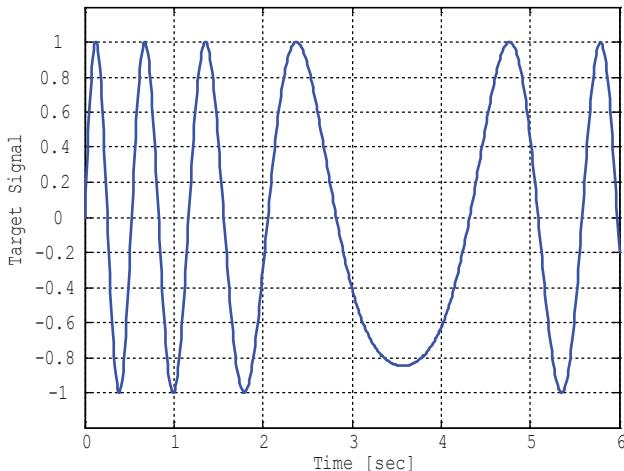


Рис. 2.21. Сравнение переходных процессов



*Рис. 2.22. Переходные процессы
после изменения обучающей выборки*



*Рис. 2.23. Целевой сигнал для линейной ИНС
с задержкой*

Определим конфигурацию ИНС и обучим ее:

```
>> p = con2seq(y);
>> net = newlin([-1,1],1);
>> net.inputWeights{1,1}.delays = [0 1 2 3 4 5];
>> [net,Y,E] = adapt(net,p,p);
```

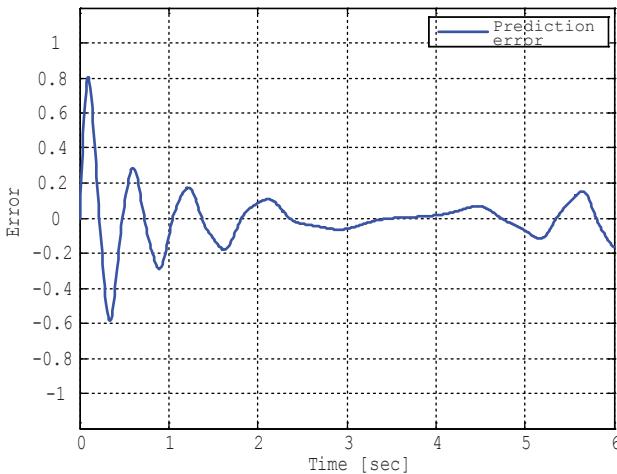


Рис. 2.24. Ошибка предсказания линейной ИНС с задержкой

Рассмотрим ошибку предсказания ИНС:

```
>> E = seq2con(E); E = E{1};
>> plot(t,E);
>> grid on
>> legend('Prediction error')
>> xlabel('Time [sec]');
>> ylabel('Error');
>> ylim([-1.2 1.2])
```

График ошибки представлен на рис. 2.24, ее наибольшие значения приходятся на начало переходного процесса.

После обучения были получены следующие веса ИНС:

```
>> net.IW{1}
ans =
0.2330  0.2045  0.1735  0.1405  0.1059  0.0702
```

Вопросы для самопроверки

1. Как описывается искусственный нейрон в среде MatLab?
2. Какими командами задаются параметры нейрона в MatLab?
3. Как описать входные данные для нейронной сети в MatLab?
4. Что такое персептрон?

5. Как описать функцию ошибки при обучении персептрона?
6. Каким методом можно обучить персептрон?
7. Какие параметры имеет команда создания персептрона в MatLab?
8. Чем отличаются команды `train` и `adapt`, применяемые для обучения персептрона?
9. Существует 16 логических функций от двух переменных. Выяснить, какие функции являются линейно разделимыми и могут быть реализованы с помощью персептрона?
10. Как обучить персептрон распознаванию четных и нечетных цифр?
 11. Как обучить персептрон выдавать код предъявленной цифры?
 12. Чем отличается линейная нейронная сеть от персептрона?
 13. Какие задачи могут решать линейные нейронные сети?
 14. Какие преимущества дает использование многослойных линейных ИНС?
15. Как используется МНК при обучении линейной нейронной сети?
16. Как оценить качество обучения линейной ИНС при использовании МНК?
17. Возможно ли использование МНК при обучении персептрона?
18. Какие параметры имеет команда создания линейной ИНС в MatLab?
19. Как можно использовать нейронные сети ADALIN?
20. Какие существуют ограничения при использовании ADALIN?
21. Что представляют собой нейронные сети MADALINE?

3. НЕЙРОННЫЕ СЕТИ ПРЯМОГО РАСПРОСТРАНЕНИЯ

3.1. Топология и свойства

Искусственная нейронная сеть прямого распространения (ПР) (англ. feed-forward network) содержит один или множество слоев нейронов. Такие сети иногда называют *многослойным перцептроном* (multi-layer perceptron).

Можно выделить следующие свойства ИНС ПР:

- между нейронами внутри одного слоя отсутствуют связи;
- отсутствуют обратные связи между слоями;
- нейрон последующего слоя получает сигналы от всех нейронов предыдущего слоя (полносвязность);
- число входов, выходов, а также число нейронов во внутренних слоях не обязательно одинаково.

Сети прямого распространения не имеют обратных связей (т. е. не имеют памяти). Этим они напоминают комбинационные логические схемы, поскольку сигнал на выходе полностью определяется текущими входами, весами связей и активационными функциями нейронов.

Большинство используемых на практике нейронных сетей относится к классу ИНС ПР.

На рис. 3.1 показан вариант простейшей однослойной ИНС ПР с m нейронами и входным вектором длиной n .

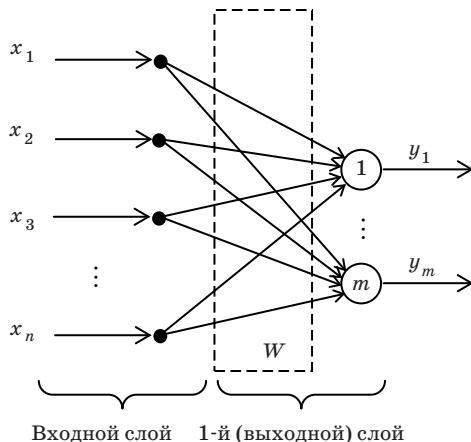


Рис. 3.1. Однослойная ИНС ПР

Многослойная НС состоит из чередующихся множеств нейронов и весов. При этом каждый слой ИНС может иметь произвольное число нейронов. На рис. 3.2 приведена трехслойная сеть (входной слой просто распределяет сигналы).

Многослойные ИНС ПР являются универсальными аппроксиматорами – с их помощью можно описать любую функцию от одной или множества переменных [39]. Но для этого активационные функции нейронов сети должны быть нелинейными. В двухслойных сетях часто выбирается сигмоидная активационная функция для нейронов 1-го слоя, и линейная – для нейронов 2-го слоя.

Следует подчеркнуть, что в общем случае нельзя заранее предсказать, сколько слоев и сколько нейронов должна иметь ИНС ПР для решения конкретной проблемы. Этот вопрос решается методом проб и ошибок с учетом имеющегося опыта.

Если ИНС ПР имеет два входа, то с ее помощью можно выполнять сложные классификации, выделяя на плоскости выпуклые или невыпуклые, ограниченные или неограниченные области. Если ИНС ПР имеет три входа, то она выполняет задачи классификации в пространстве, если более трех входов – то в гиперпространстве.

В системе MatLab ИНС ПР создается командой:

`NEWFF(PR,[S1 S2...SN],{TF1 TF2...TFN},BTB,BLF,PF),`

где PR – матрица $R \times 2$ минимальных и максимальных значений для каждого из R входных элементов; S_1, S_2, \dots, S_N – число нейронов

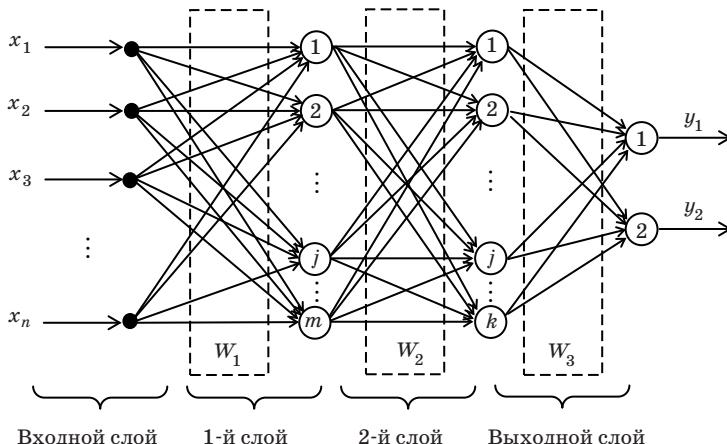


Рис. 3.2. Пример трехслойной ИНС ПР

в каждом слое (от 1-го до N -го); $\text{TF}_1, \text{TF}_2, \dots, \text{TF}_N$ – передаточные функции каждого слоя (по умолчанию 'tansig'); BTF, BLF – функции обучения (по умолчанию 'trainlm' и 'learngdm'), PF – функция оценки качества работы (по умолчанию 'mse').

Пример 3.1. Создать двухслойную ИНС ПР с одним входом и одним выходом:

```
>> P = [0 1 2 3 4 5 6 7 8 9 10];
>> net = newff(minmax(P),[5 1],{'tansig' 'purelin'});
```

Пример 3.2. Создать ИНС ПР с двумя входами и тремя нейронами в каждом из двух слоев:

```
>> net11 = newff([-1 2; 0 5], [3, 3]);
```

3.2. Алгоритм обратного распространения ошибки

Дельта-правило, которое применяется при обучении персептрона, использует величину ошибки выходного слоя. Если же сеть имеет два или больше слоев, то для промежуточных слоев значения ошибки в явном виде не существует, и использовать дельта-правило нельзя.

Основная идея обратного распространения состоит в том, как получить оценку ошибки для нейронов скрытых слоев. Заметим, что *известные ошибки*, допускаемые нейронами выходного слоя, возникают вследствие *неизвестных ошибок* нейронов скрытых слоев. Чем больше значение синаптической связи между нейроном скрытого слоя и выходным нейроном, тем сильнее ошибка первого влияет на ошибку второго. Следовательно, оценку ошибки элементов скрытых слоев можно получить как взвешенную сумму ошибок последующих слоев.

Алгоритм обратного распространения ошибки (АОРО), являющийся обобщением дельта-правила, позволяет обучать ИНС ПР с любым числом слоев. Можно сказать, что АОРО фактически использует разновидность градиентного спуска, перестраивая веса в направлении минимума ошибки.

При использовании АОРО предполагается, что в качестве активационной используется сигмоидная функция. Эта функция позволяет экономить вычислительные затраты, поскольку имеет простую производную:

$$\frac{dF(y)}{dy} = F(y)(1 - F(y)).$$

Сигмоидная функция ограничивает значением единица сильные сигналы и усиливает слабые.

Смысл алгоритма обратного распространения ошибки состоит в том, что при обучении сети сначала предъявляется образ, для которого вычисляется ошибка выхода. Далее эта ошибка распространяется по сети в обратном направлении, изменения веса межнейронных связей.

Алгоритм включает в себя такую же последовательность действий, как и при обучении персептрона. Сначала веса межнейронных связей получают случайные значения, затем выполняются следующие шаги:

- 1) выбирается обучающая пара (X, Z^*) , X подается на вход;
- 2) вычисляется выход сети $Z = F(Y)$;
- 3) рассчитывается ошибка выхода E ;
- 4) веса сети корректируются с целью минимизации ошибки;
- 5) возврат к п. 1 и т. д., пока не будет минимизирована ошибка по всем обучающим парам.

Шаги 1 и 2 – это прямое распространение по сети, а шаги 3 и 4 – обратное.

Перед обучением необходимо разбить имеющиеся пары вход-выход на две части: *обучающие* и *тестовые*.

Тестовые пары используются для проверки качества обучения: НС хорошо обучена, если при заданной тестовой паре выдает на выходе выход, близкий к тестовому.

При обучении возможна ситуация, когда НС показывает хорошие результаты для обучающих данных, и плохие – для тестовых. Это может быть обусловлено двумя причинами:

1. Тестовые данные сильно отличаются от обучающих, т. е. обучающие пары охватывали не все области входного пространства.
2. Возникло явление «переобучения» (*overfitting*), при котором поведение НС оказывается более сложным, чем решаемая задача.

Последний случай для задачи аппроксимации функции по точкам иллюстрирует рис. 3.3, на котором светлые кружки соответствуют тестовым данным, а темные – обучающим.

На рис. 3.3 сплошная линия обозначает выход НС для произвольного входа из области определения X . График на рис. 3.3,*a* соответствует хорошо обученной НС, а график на рис. 3.3,*b* – переобученной.

Рассмотрим для простоты изложения двухслойную ИНС ПР, в которой веса скрытого слоя описываются вектором W , а веса выходного слоя – вектором V (рис. 3.4).

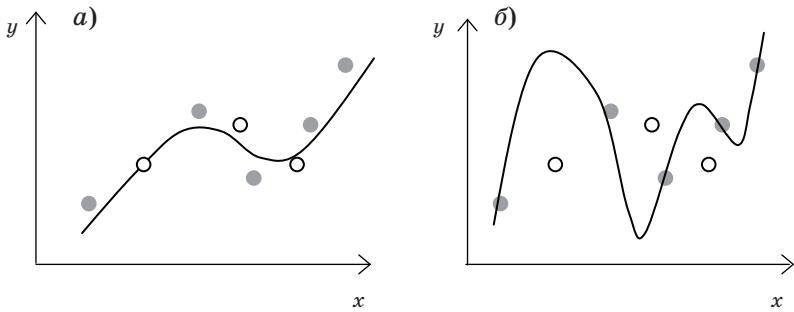


Рис. 3.3. Иллюстрация явления переобучения

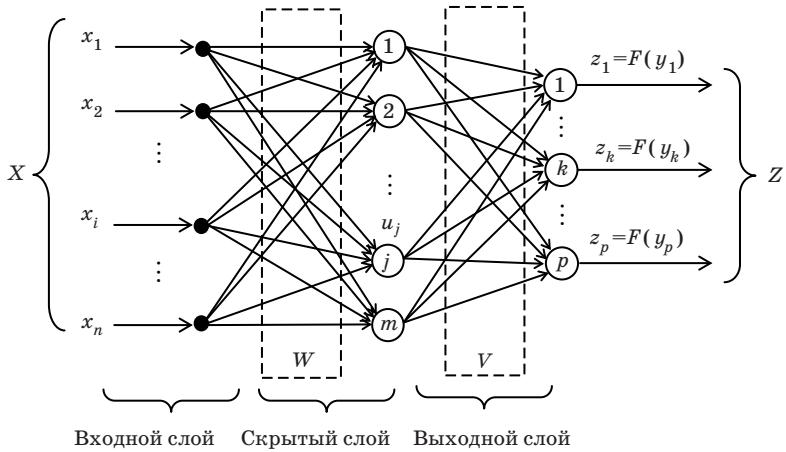


Рис. 3.4. Двухслойная ИНС ПР

Классический градиентный метод поиска минимума функции $f(X)$, где X – вектор, состоит в изменении аргумента в направлении антиградиента:

$$X(t+1) = X(t) - \eta \frac{\partial f(X)}{\partial X}.$$

Для выходного слоя можно записать

$$V(t+1) = V(t) - \eta \frac{\partial E(V)}{\partial V}.$$

Таким образом, вес v_{jk} , связывающий j -й нейрон скрытого слоя и k -й нейрон выходного, корректируется по формуле

$$v_{jk}(t+1) = v_{jk}(t) - \eta \frac{\partial E}{\partial v_{jk}}.$$

Ошибка выхода может быть описана следующим образом:

$$E = \frac{1}{2} \sum_{k=1}^p (z_k - z_k^*)^2.$$

Очевидно, что E явно не зависит от V , но для получения производной можно использовать правила дифференцирования сложной функции

$$\frac{\partial E}{\partial z_k} = z_k - z_k^* = \Delta_k,$$

$$\frac{\partial E}{\partial y_k} = \frac{\partial E}{\partial z_k} \frac{\partial z_k}{\partial y_k} = \Delta_k z_k (1 - z_k),$$

$$\frac{\partial E}{\partial v_{jk}} = \frac{\partial E}{\partial z_k} \frac{\partial z_k}{\partial y_k} \frac{\partial y_k}{\partial v_{jk}} = \Delta_k z_k (1 - z_k) u_j,$$

где u_j – выход j -го нейрона скрытого слоя $\left(y_k = \sum_{j=1}^m v_{jk} u_j \right)$.

Таким образом, формула для коррекции весов выходного слоя приобретает вид

$$v_{jk}(t+1) = v_{jk}(t) - \eta \Delta_k z_k (1 - z_k) u_j.$$

Рассмотрим далее коррекцию весов скрытого слоя. Здесь используется формула

$$w_{ij}(t+1) = w_{ij}(t) - \eta \frac{\partial E}{\partial w_{ij}}.$$

Выход нейрона скрытого слоя описывается формулой

$$a_j = \sum_{i=1}^n w_{ij} x_i, \quad u_j = F(a_j).$$

По аналогии с формулой для выходного слоя можно записать

$$\frac{\partial E}{\partial a_j} = \frac{\partial E}{\partial u_j} \frac{\partial u_j}{\partial a_j} = \Delta_j u_j (1 - u_j).$$

Однако величина ошибки для скрытого слоя Δ_j не задана, т. е. не ясно, каким должен быть эталонный выход скрытого слоя. В то же время очевидно, что ошибки выходного слоя зависят от ошибки скрытого слоя ИНС, и это влияние тем больше, чем больше вес связи между нейроном скрытого слоя и выходным нейроном. Таким образом, оценку ошибки нейрона скрытого слоя можно получить как взвешенную сумму ошибок выходного слоя (рис. 3.5):

$$\frac{\partial E}{\partial u_j} = \Delta_j = \sum_{k=1}^p \frac{\partial E}{\partial z_k} \frac{\partial z_k}{\partial u_j} = \sum_{k=1}^p \Delta_k z_k (1 - z_k) v_{jk},$$

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial u_j} \frac{\partial u_j}{\partial a_j} \frac{\partial a_j}{\partial w_{ij}} = \left(\sum_{k=1}^p \Delta_k z_k (1 - z_k) v_{jk} \right) u_j (1 - u_j) x_i.$$

Тогда формула для коррекции весов скрытого слоя окончательно приобретает следующий вид:

$$w_{ij}(t+1) = w_{ij}(t) - \eta \left(\sum_{k=1}^p \Delta_k z_k (1 - z_k) v_{jk} \right) u_j (1 - u_j) x_i.$$

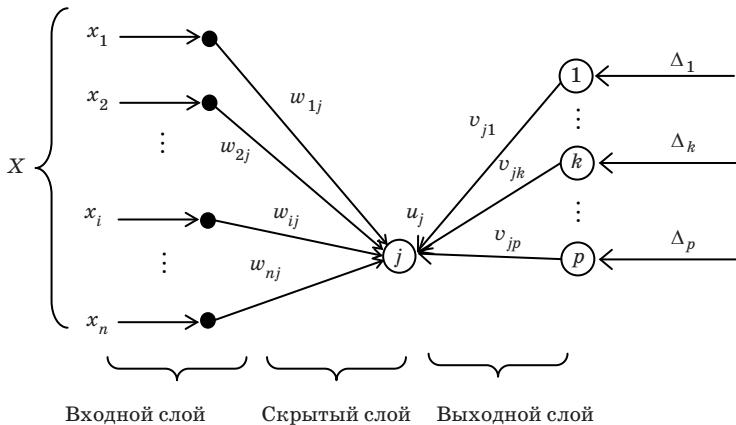


Рис. 3.5. Обратное распространение ошибки

Достоинство АОРО заключается в том, что он обобщается для ИНС ПР с любым числом слоев.

Алгоритм обратного распространения ошибки нашел применение в ряде прикладных разработок. Особенно впечатляющие успехи достигнуты при распознавании печатных букв, когда эффективность алгоритма близка к 100%. Однако применение АОРО в других задачах может быть затруднено ввиду причин, характерных для градиентных алгоритмов минимизации:

- при больших значениях сигмоидной функции ее производная становится весьма малой, и процесс обучения замедляется;
- поверхность ошибки сложной сети может иметь локальные минимумы, обнаружив которые сеть перестанет улучшать свое поведение.

Работу АОРО можно улучшить, учитывая вторые производные активационной функции.

Ускорения работы алгоритма можно добиться при использовании переменной скорости обучения η . В начале работы АОРО ее величина имеет значение, близкое к единице, а затем последовательно уменьшается примерно до 0,01. Это позволяет быстро подойти к окрестности минимума, а затем точно попасть в него.

Однако локальность АОРО является принципиальным ограничением его использования. Если функция ошибки имеет сложный характер, то требуется использовать методы случайного поиска. Тем не менее АОРО может быть успешно применен во многих практических задачах, некоторые из которых будут рассмотрены далее.

3.3. Реализация логических функций

Рассмотрим задачу классификации бинарных векторов длиной 4 бита: если вектор содержит нечетное число единичных битов, то на выходе сети должна быть 1, в противном случае – 0.

Пример 3.3. Входные векторы описываются матрицей

```
>> inp = [0 0 0 0 0 0 0 0 1 1 1 1 1 1 1;  
          0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1;  
          0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1;  
          0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1]
```

Выходной вектор:

```
>> out=[0 1 1 0 1 0 0 1 1 0 0 1 0 1 1 0]
```

Создадим сеть прямого распространения с помощью команды

```
>> network=newff([0 1;0 1; 0 1],[6 1],{'logsig','logsig'})
```

В этой команде первый массив содержит диапазоны изменения сигнала для каждого входа, второй массив – размер каждого слоя (число нейронов), затем задается тип передаточной функции для каждого слоя.

До начала обучения выполняется инициализация НС:

```
>> network=init(network);
```

Зададим число эпох, скорость и запустим обучение:

```
>> network.trainParam.epochs = 500;
```

```
>> network.trainParam.lr = 0.05;
```

```
>> network=train(network,inp,out);
```

Проверим работу сети:

```
>> y=sim(network,inp);
```

```
>> out=y
```

```
ans =
```

```
1.0e-004 *
```

```
-0.0191 0.0044 0.0204 -0.0237 0.0039 -0.0000 -0.0316
```

```
0.0041 0.0014 -0.0367 -0.0253 0.1118 -0.0299 0.0384
```

```
0.0000 -0.0000
```

Ошибки между заданным выходом НС и реальным выходом можно считать пренебрежимо малыми.

Рассмотрим далее простейший закон логического управления динамическим объектом.

Пример 3.4. Требуется, чтобы нейросетевой регулятор, получающий на входе ошибку управления e , реализовывал релейный закон управления:

$$u = \begin{cases} 1, & e \geq 0, \\ -1, & e < 0. \end{cases}$$

MatLab программа:

```
>> w=2*pi;
>> for i=1:300 % формирование обучающей выборки
    time(i)=0.01*i;
    e(i)=(exp(-time(i)))*sin(w*time(i));
    if (e(i)>0.0) t(i)=1;
    elseif (e(i)<0.0) t(i)=-1;
    end
end
```

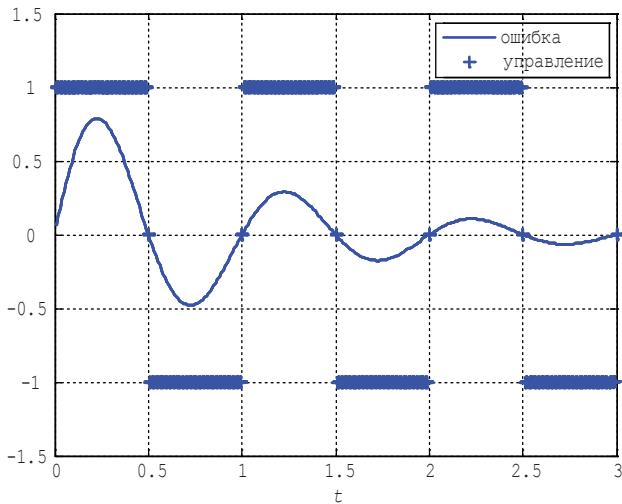


Рис. 3.6. Нейросетевой релейный регулятор

```
>> plot(time,e); grid on; hold on;
>> net = newff([min(e) max(e)],[10 1],{'tansig','purelin'});
>> net.trainParam.epochs = 1000;
>> net = train(net,e,t);
>> u=sim(net,e); plot(time,u,'+'); legend('ошибка','управление');
xlabel('t');
```

Работу релейного регулятора иллюстрирует рис. 3.6.

3.4. Аппроксимация функций

В задачах аппроксимации функций требуется восстановить функциональную зависимость на основании ограниченного набора известных точек. Эту проблему иллюстрируют следующие примеры.

Пример 3.5. Аппроксимация функций:

```
>> P = [0 1 2 3 4 5 6 7 8 9 10];
>> T = [0 1 2 3 4 3 2 1 2 3 4];
>> net = newff([0 10],[5 1],{'tansig', 'purelin'});
>> net.trainParam.goal=0.01;
>> net.trainParam.epochs = 50;
>> net = train(net,P,T);
>> X = linspace(0,10);
>> Y = sim(net,X);
```

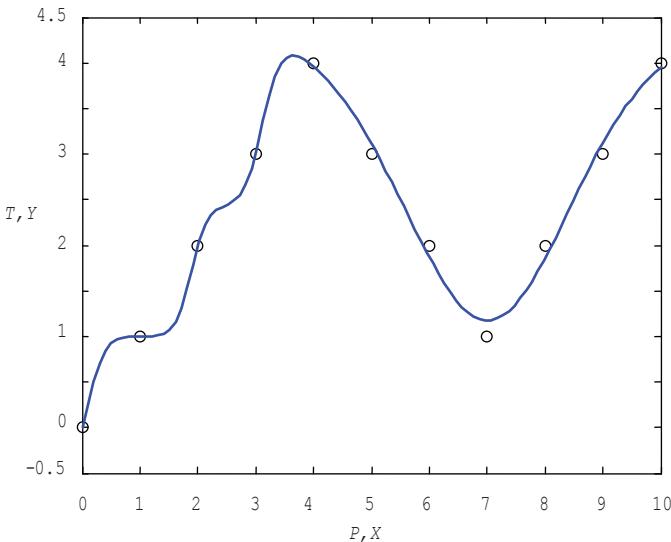


Рис. 3.7. Аппроксимация функции, заданной набором точек

```
>> figure(1)
>> plot(P,T,'ko',X,Y)
```

В этом примере векторы P и T задают 11 опорных точек, по которым требуется построить непрерывную функцию. Нейронная сеть имеет два слоя, скрытый слой содержит пять нейронов. Задано 50 эпох обучения при требуемой ошибке 0,01. Из рис. 3.7 следует, что задача решена достаточно качественно.

Пример 3.6. Пусть имеется большой массив точек:

```
>> x = 0:.05:2;
>> y = 1 ./ ((x-.3).^2 + .01) + 1 ./ ((x-.9).^2 + .04) - 6;
```

Требуется построить ИНС ПР для аппроксимации этого массива:

```
>> net=newff([0 2], [15,1], {'tansig','purelin'},'trainlm');
>> net.trainParam.show = 50;
>> net.trainParam.epochs =100;
>> net.trainParam.goal = 0.001;
>> P=x;
>> T=y;
>> net1 = train(net, P, T);
```

График обучения представлен на рис. 3.8. Заданная ошибка оказалась достигнута уже на 55-й итерации.

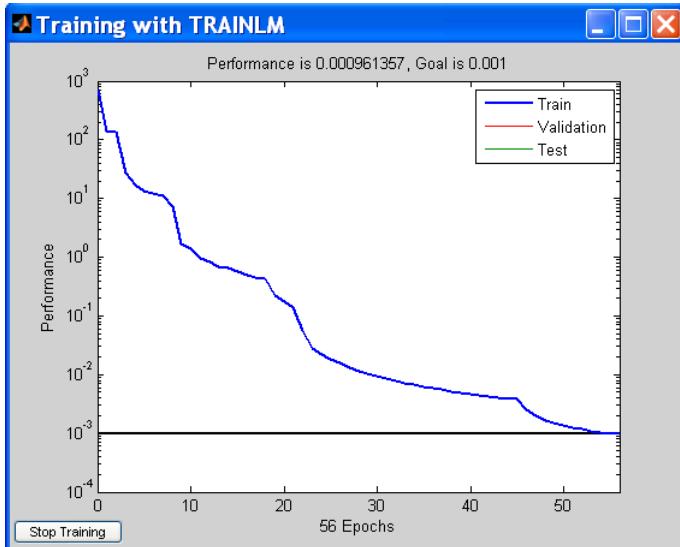


Рис. 3.8. Изменение функции ошибки в процессе обучения

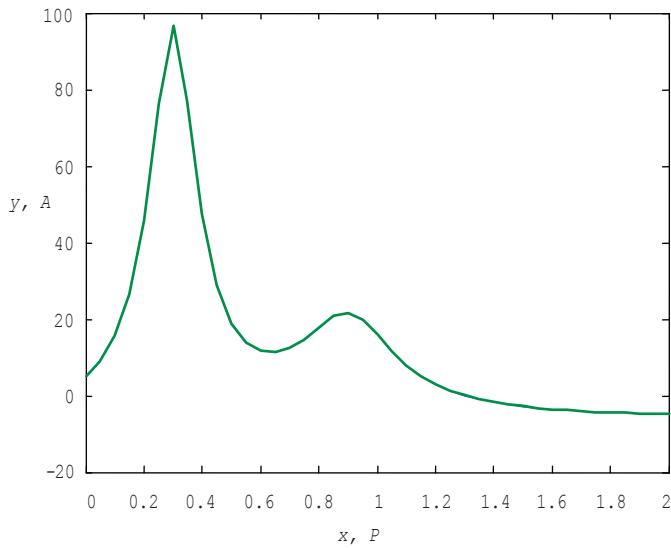


Рис. 3.9. Сравнение функции и ее аппроксимации

Заметим, что в этом примере использовался алгоритм обучения trainlm (Levenberg–Marquardt).

Проверим полученный результат:

```
>> A = sim(net1,P);
>> plot(x,y,P,A)
```

Графики функций полностью совпадают (рис. 3.9).

Пример 3.7. Рассмотрим аппроксимацию функции двух переменных:

$$z=\sin(x)\cos(y).$$

С помощью стандартных функций MatLab получаем поверхность, приведенную на рис. 3.10:

```
>> x = -2:0.25:2; y = -2:0.25:2;
>> z = cos(x)'.*sin(y);
>> mesh(x,y,z)
```

Опишем обучающее множество для нейронной сети:

```
>> P = [x;y]; T = z;
```

При таком способе формирования обучающей выборки каждой паре $\{x; y\}$ соответствует столбец целевого множества. Поэтому нейронная сеть должна иметь два входа и 17 нейронов на выходе:

```
>> net=newff([-2 2; -2 2], [25 17], {'tansig' 'purelin'}, 'trainlm');
>> net.trainParam.show = 50;
net.trainParam.lr = 0.05;
net.trainParam.epochs = 300;
```

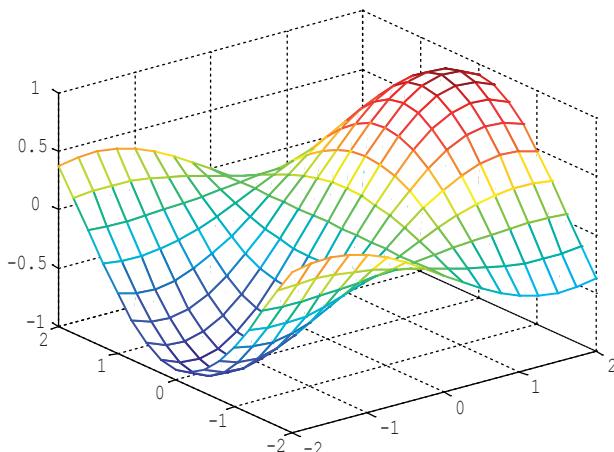


Рис. 3.10. Исходная функция двух переменных

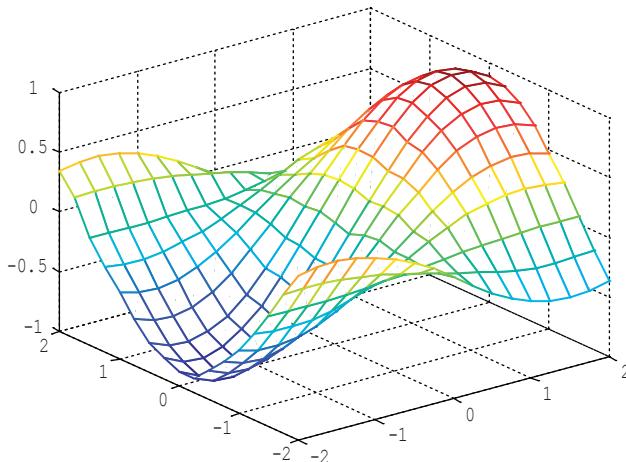


Рис. 3.11. Выход нейронной сети после обучения

```
net.trainParam.goal = 0.001;
>> net1 = train(net, P, T);
```

Выполним проверку (рис. 3.11):

```
>> A=sim(net1,P)
>> figure(1); mesh(x,y,A)
```

Сравнение рис. 3.10 и 3.11 позволяет говорить о достаточно высоком качестве аппроксимации.

Пример 3.8. Аппроксимация функции трех переменных.

Пусть имеется модель, с помощью которой можно генерировать вход-выходные зависимости системы

$$y = 3a + 4ab + 2c + f,$$

где a, b, c – входные переменные; f – сигнал шума. Опишем эти величины как массивы случайных чисел:

```
>> a = rand(1,100);
>> b = rand(1,100);
>> c = rand(1,100);
>> f = rand(1,100)*0.025;
>> y = 3*a + 4*a.*b + 2*c + f;
```

Обучающее множество задается в виде

```
>> P = [a; b; c];
>> T = y;
```

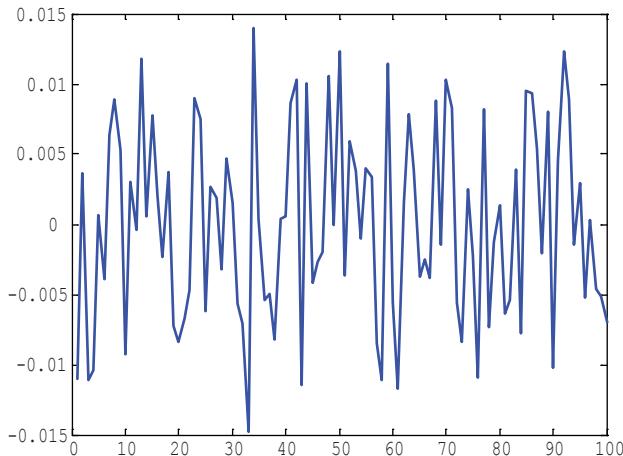


Рис. 3.12. Ошибка выхода нейронной сети

Опишем ИНС и запустим процесс обучения:

```
>> net = newff([0 1; 0 1; 0 1], [4 1], {'tansig', 'purelin'});
>> net = train(net, P,T)
```

Для оценки качества работы ИНС выполним команды:

```
>> out=sim(net,P);
>> t=1:100;
>> plot(t,y-out)
```

Результат приведен на рис. 3.12.

При нейросетевой аппроксимации функции по точкам может возникать явление переобучения. В следующем примере ИНС после обучения воспроизводит влияние шумов, наложенных на обучающую выборку.

Пример 3.9. Явление переобучения:

```
>> net = newff([-1 1],[30,1],{'tansig','purelin'}); % двухслойная
ИНС ПР
```

```
>> net.trainParam.epochs = 500;
>> p = [-1:0.05:1]; % область определения функции
>> t1 = sin(2*pi*p); % эталонная функция
>> t = sin(2*pi*p)+0.1*randn(size(p)); % функция с шумом
>> [net,tr] = train(net,p,t);
>> t2 = sim(net,p);
>> figure(1); plot(p,t,'+',p,t2,'-',p,t1,:')
>> legend('вход','выход','эталон'); grid on
```

Результат представлен на рис. 3.13. Выход обученной сети соответствует всем точкам зашумленной обучающей выборки, местами значительно отступая от эталонного процесса.

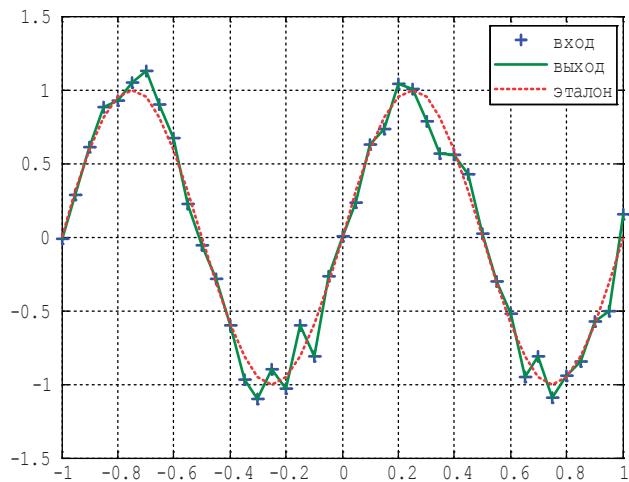


Рис. 3.13. Пример переобучения нейронной сети

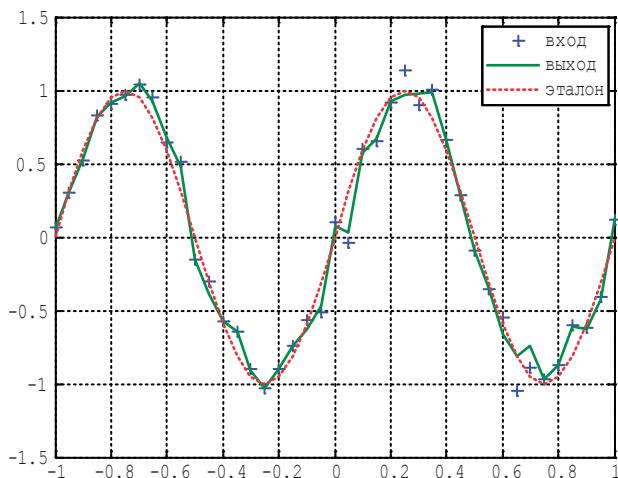


Рис. 3.14. Контроль переобучения нейронной сети

Для устранения эффекта переобучения можно использовать расширенный вариант команды `train`, в котором помимо обучающей выборки указывается также проверочная выборка:

```
>> val.P = [-0.975: 0.05: 0.975]; % формирование проверочной  
выборки  
>> val.T = sin(2*pi*val.P) + 0.1*randn(size(val.P));  
>> [net, tr] = train(net, p, t, [], [], val);
```

Результат обучения для этого варианта команды представлен на рис. 3.14. Обучение здесь прекращается раньше, как только контрольная ошибка начинает значительно превышать ошибку обучения. Благодаря этому выход нейронной сети не реагирует на отдельные аномальные точки обучающего множества (см. рис. 3.13).

3.5. Распознавание символов

Задача распознавания изображений является одной из наиболее массовых задач, успешно решаемых с помощью ИНС. Возможны самые разные постановки проблемы, одна из наиболее простых – распознавание фиксированного набора символов.

Пример 3.10. Распознавание букв.

В системе MatLab предусмотрена специальная функция

```
>> [alphabet, targets] = prprob;
```

Эта функция возвращает две двоичные матрицы: в матрице `alphabet` (размером 35×26) каждый столбец кодирует одну букву, а матрица `targets` (размером 26×26) является диагональной и служит для идентификации столбца.

Каждому столбцу `alphabet` соответствует матрица 7×5 , представляющая собой двоичное изображение буквы.

Следующая функция отображает все столбцы `alphabet` в виде букв (функцию требуется поместить в рабочий каталог MatLab):

```
function plotletters(alphabet)  
fprintf('plotletters is plotting the first 25 letters\n');  
[m,n]=size(alphabet);  
if m~=35  
error('plotletters needs columns 35 numbers long');  
end  
figure  
MM=colormap(gray);  
MM=MM(end:-1:1,:);
```

```

colormap(MM);
nn=min([n,25]);
for j=1:nn
subplot(5,5,j)
imagesc(reshape(alphabet(:,j),5,7)');
axis equal
axis off
end

```

Результат выполнения функции приведен на рис. 3.15:

```
>> plotletters(alphabet);
```

Исходя из структуры матрицы targets нейронная сеть должна иметь 26 выходных нейронов. Число нейронов скрытого слоя положим равным 10:

```
>> net = newff(minmax(alphabet),[10 26],{'logsig' 'logsig'},'traingdx');
```

```
>> P = alphabet; T = targets;
```

Зададим число эпох и запустим процесс обучения:

```
>> net.trainParam.epochs = 1000;
```

```
>> [net,tr] = train(net,P,T);
```

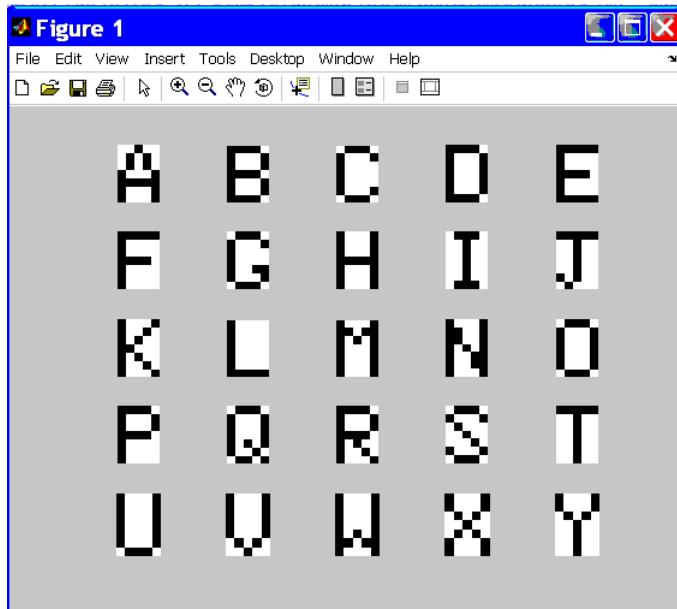


Рис. 3.15. Двоичное кодирование алфавита

Кривая обучения приведена на рис. 3.16.

Для проверки качества работы обученной сети рассмотрим зашумленные изображения букв (рис. 3.17):

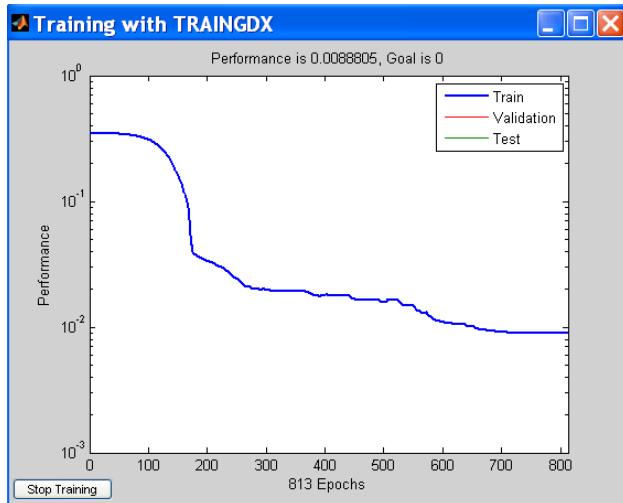


Рис. 3.16. Изменение ошибки в процессе обучения

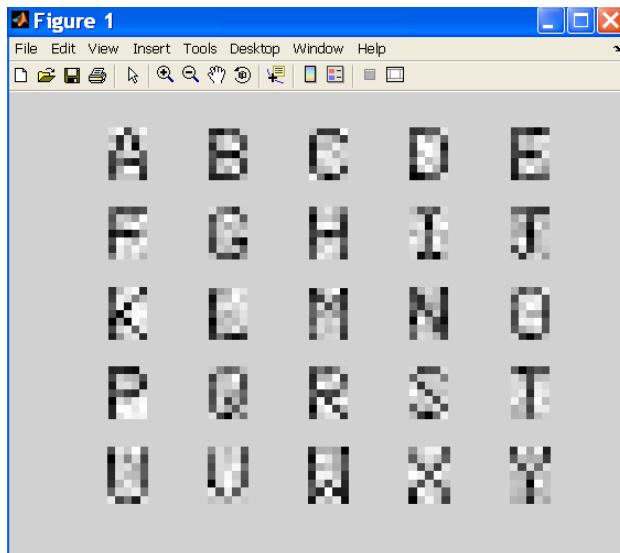


Рис. 3.17. Изображения букв в присутствии шума

```
>> noisyP = alphabet+randn(size(alphabet)) * 0.2;
>> plotletters(noisyP);
```

С помощью следующей команды выполняется запуск сети для зашумленного входного множества:

```
>> A2 = sim(net,noisyP);
```

Матрица A2 здесь содержит различные числа в интервале [0, 1]. С помощью функции compet в каждом столбце можно выделить максимальный элемент, затем присвоить ему значение «1», а остальные элементы столбца обнулить:

```
>> for j=1:26
A3 = compet(A2(:,j));
answer(j) = find(compet(A3) == 1);
end
```

Затем можно визуально оценить ответы сети для зашумленных входных векторов с помощью команд

```
>> NetLetters=alphabet(:,answer);
>> plotletters(NetLetters);
```

На рис. 3.18 приведен окончательный результат распознавания.

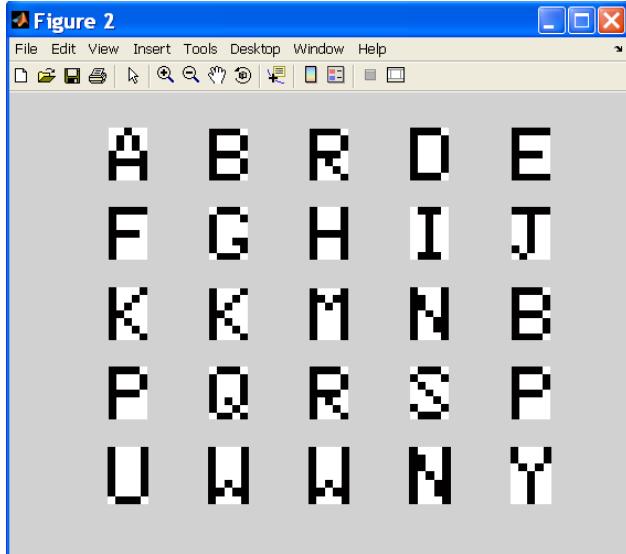


Рис. 3.18. Результат выполнения распознавания нейронной сетью

Очевидно, что некоторые буквы идентифицированы ошибочно. Это может быть либо следствием плохой обученности сети, либо слишком высокого уровня шума, либо неправильного выбора числа нейронов внутреннего слоя.

3.6. Моделирование статических зависимостей

Важной является возможность описания вход-выходных зависимостей динамических объектов с помощью ИНС ПР.

Пример 3.11. Рассмотрим моделирование с помощью ИНС ПР динамической системы, описываемой функцией Бесселя

$$t^2 \frac{d^2y}{dt^2} + t \frac{dy}{dt} + (t^2 - a^2)y = 0.$$

График функции Бесселя представлен на рис. 3.19:

```
>> t=0:0.1:20;
y=bessel(1,t);
plot(t,y)
grid
```

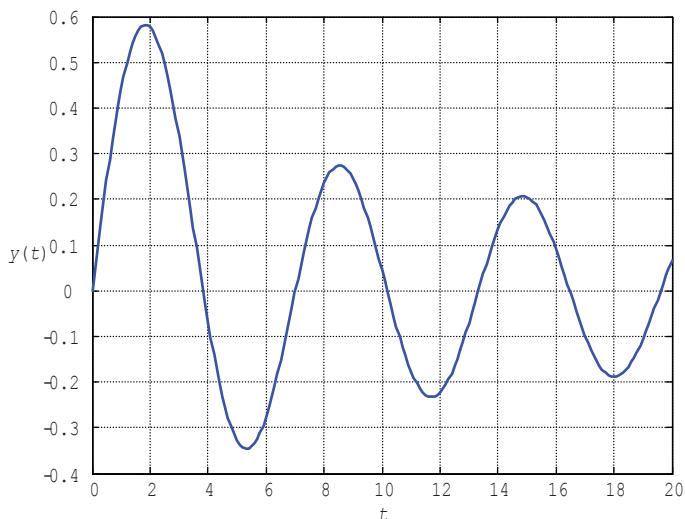


Рис. 3.19. Функция Бесселя

Для моделирования будем использовать двухслойную сеть с десятью нейронами скрытого слоя:

```
>> net=newff([0 20], [10,1], {'tansig','purelin'},'trainlm');
>> P=t; T=y;
>> net.trainParam.show = 50;
>> net.trainParam.lr = 0.05;
>> net.trainParam.epochs = 300;
>> net.trainParam.goal = 0.001;
>> net1 = train(net, P, T);
```

Для обучения потребовалось всего 11 эпох (рис. 3.20).

Выполним моделирование работы сети:

```
>> A = sim(net1,P);
```

Для проверки качества работы сети рассмотрим ошибку моделирования (см. рис. 3.18):

```
>> figure(1); plot(P,T-A)
```

Как следует из рис. 3.21, в конце переходного процесса ошибка недопустимо возрастает, поэтому параметры нейронной сети требуют коррекции.

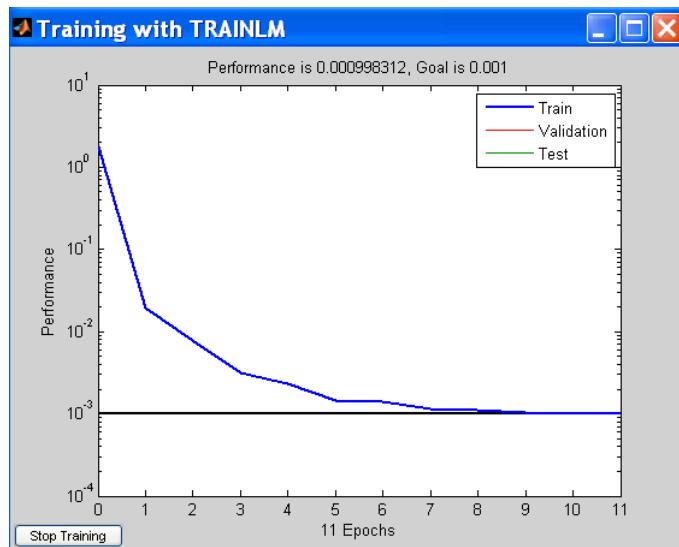


Рис. 3.20. Изменение ошибки в процессе обучения

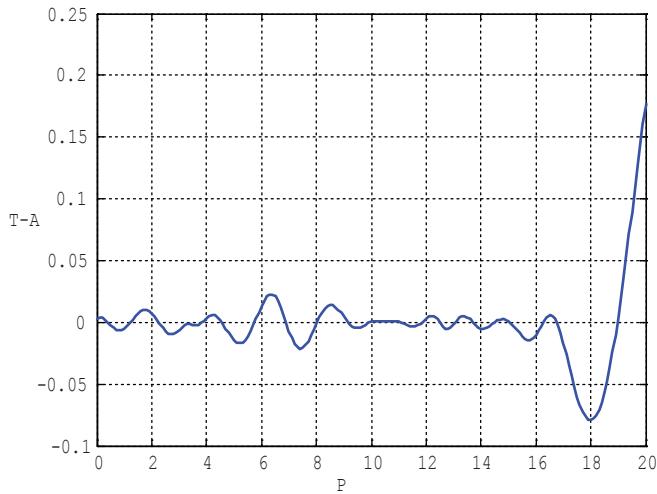


Рис. 3.21. Ошибка на разных участках переходного процесса

Пример 3.12. Пусть динамическая система описывается уравнением Ван дер Поля

$$\frac{d^2y}{dt^2} + (y^2 - 1) \frac{dy}{dt} + y = 0.$$

Этому уравнению соответствует схема, собранная в MatLab Simulink (рис. 3.22).

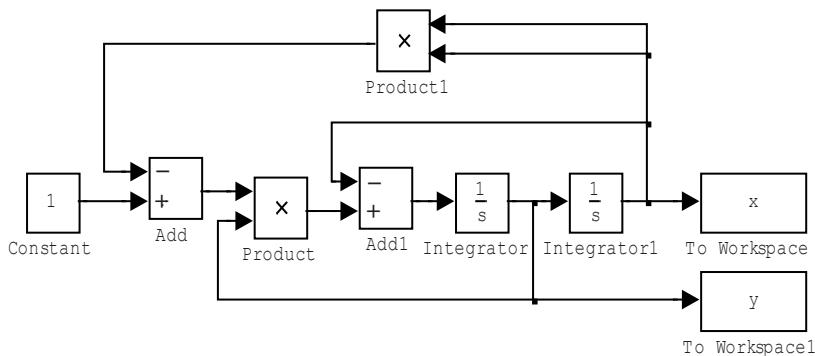


Рис. 3.22. Блок-схема, соответствующая уравнению Ван дер Поля

Переменные x и y описывают здесь состояние динамической системы (положим начальное состояние $x = 2$ и $y = 2$).

Моделирование со временем 10 с можно запустить командой (где 'Van_der_Pol' – название mdl-файла):

```
>> [t,z]=sim('Van_der_Pol',10);
```

На рис. 3.23 показан результат моделирования.

Зададим обучающую выборку, параметры нейронной сети и обучения:

```
>> P = t';  
>> T = z';  
>> net=newff([0 20], [20,2], {'tansig','purelin'}, 'trainlm');  
>> net.trainParam.show = 50;  
>> net.trainParam.lr = 0.05;  
>> net.trainParam.epochs = 300;  
>> net.trainParam.goal = 0.001;  
>> net1 = train(net, P, T);
```

Затем выполним моделирование:

```
>> A= sim(net1,P);  
>> figure(1); plot(P,A)  
>> grid
```

Сравнение рис. 3.23 и 3.24 позволяет судить о хорошем качестве моделирования.

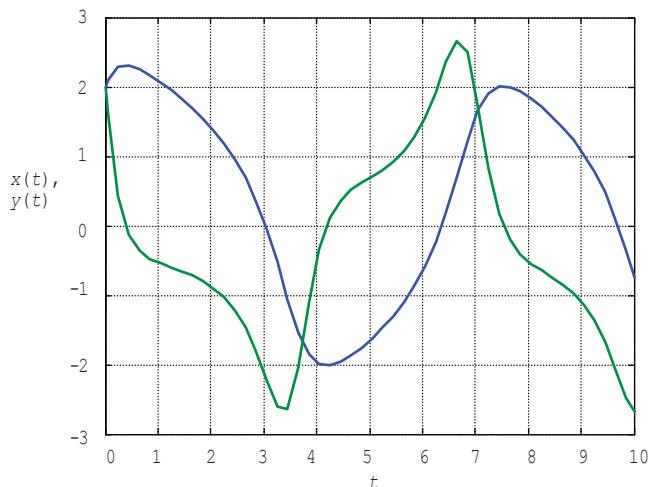


Рис. 3.23. Переходные процессы в системе Ван дер Поля

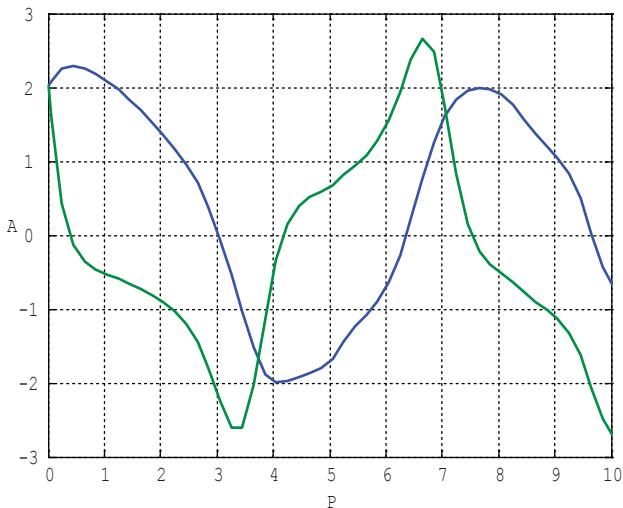


Рис. 3.24. Результат работы моделирующей нейронной сети

В рассмотренных примерах НС обучалась выдавать выходной сигнал, подобный выходу динамической системы. Однако поведение НС в данной ситуации качественно отличается от поведения динамического объекта. В подтверждение этого тезиса проанализируем еще один простой пример.

Пример 3.13. Пусть имеется линейное динамическое звено 2-го порядка. Его можно описать в виде дифференциального уравнения или в виде соответствующей передаточной функции

$$\frac{d^2y(t)}{dt^2} + 0,5 \frac{dy(t)}{dt} + y(t) = x(t) \Leftrightarrow W(s) = \frac{Y(s)}{X(s)} = \frac{1}{s^2 + 0,5s + 1}.$$

На рис. 3.25 представлена собранная в MatLab Simulink схема для получения реакции звена на единичный скачок.

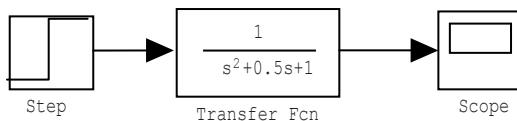


Рис. 3.25. Модель системы в MatLab Simulink

Выполним моделирование (здесь PF – имя файла модели):

```
>> [t,y]=sim('PF',20);
```

и построим график переходного процесса (рис. 3.26):

```
>> plot(t,y(:,2))
```

Зададим обучающую выборку, параметры нейронной сети и обучения:

```
>> P = t';  
>> T = y(:,2)';  
>> net=newff([0 20], [15,1], {'tansig','purelin'},'trainlm');  
>> net.trainParam.show = 50;  
>> net.trainParam.lr = 0.05;  
>> net.trainParam.epochs = 100;  
>> net.trainParam.goal = 0.001;  
>> net1 = train(net, P, T);
```

и выполним моделирование:

```
>> A= sim(net1,P);  
>> figure(1); plot(P,A)  
>> grid
```

Затем экспортим полученнную нейронную сеть в Simulink (здесь 0,01 – постоянный шаг интегрирования по времени):

```
>> gensim(net1,0.01)
```

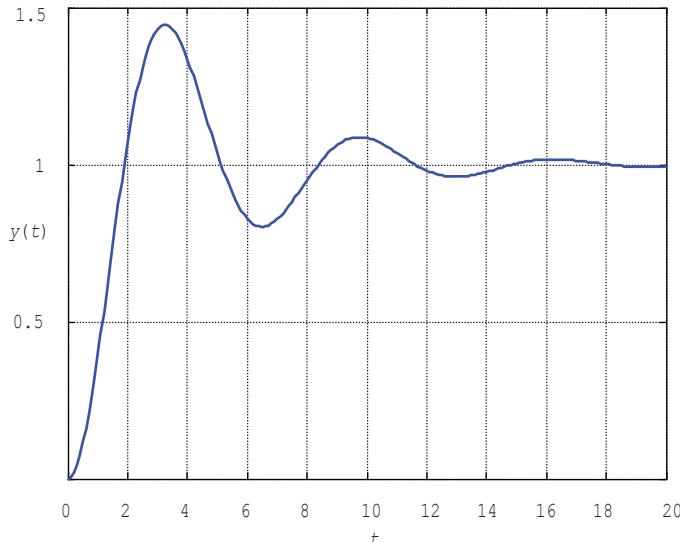


Рис. 3.26. Выходной сигнал динамического объекта

Схема, приведенная на рис. 3.27, позволяет сравнить выходные сигналы передаточной функции и нейронной сети.

Как следует из рис. 3.28, выходные сигналы достаточно близки.

Однако очевидно, что при изменении входного сигнала динамической системы (например, при замене step на sine wave) выходной сигнал также будет изменяться, и потребуется формирование новой обучающей выборки (рис. 3.29).

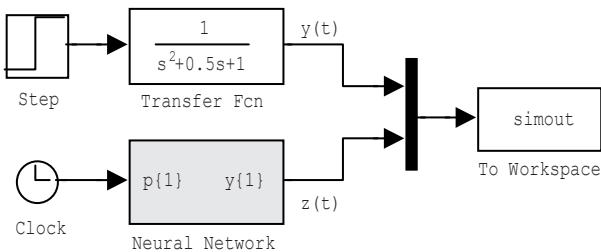


Рис. 3.27. Сравнение динамического звена и его аппроксимации

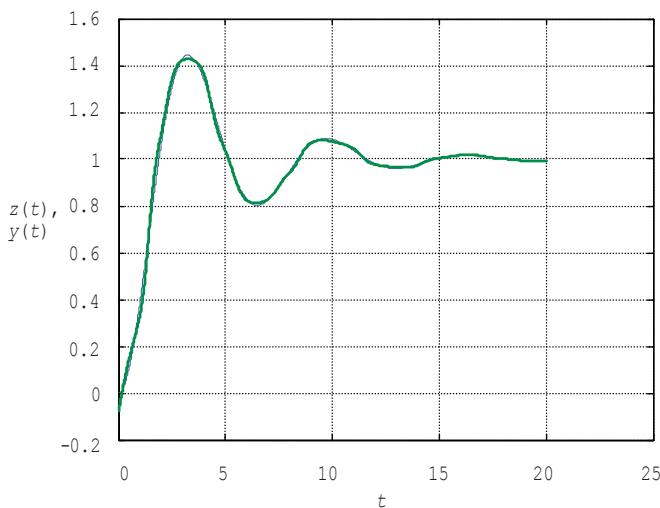


Рис. 3.28. Выходы динамического звена и нейронной сети

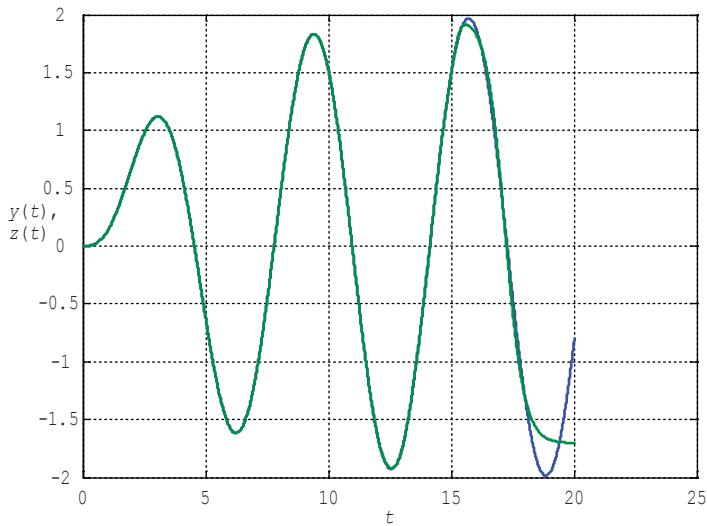


Рис. 3.29. Сравнение реакций на синусоидальный сигнал

Таким образом, при изменении входного сигнала требуется переобучение нейронной сети. В этом и заключается статичность ее поведения.

3.7. Масштабирование и восстановление данных

При работе с ИНС ПР часто возникает необходимость предобработки и постобработки информации. В MatLab предусмотрен набор команд для выполнения этих операций.

Пример 3.14. Пусть имеется файл data.txt с числовой информацией:

-10	0
-7,5	7,07
-5	-10
-2,5	-7,07
0	0
2,5	7,07
5	10
7,5	7,07
10	0

Данныечитываются командой

```
>> load data.txt;
>> P=data(1:9,1);
>> T=data(1:9,2);
```

Следующая команда выполняет нормализацию обучающих данных:

```
>> [pn,minp,maxp,tn,mint,maxt] = premnmx(P',T')
pn =
-1.0000 -0.7500 -0.5000 -0.2500 0 0.2500 0.5000 0.7500 1.0000
minp = -10
maxp = 10
tn =
0 0.7070 -1.0000 -0.7070 0 0.7070 1.0000 0.7070 0
mint = -10
maxt = 10
```

Пример 3.15. Сначала с помощью функции premnmx выполним масштабирование обучающей последовательности к диапазону $[-1 \ 1]$, затем создадим и обучим ИНС ПР, выполним ее моделирование и восстановление выхода с помощью функции postmnmx:

```
>> P = [-0.92 0.73 -0.47 0.74 0.29; -0.08 0.86 -0.67 -0.52 0.93];
>> t = [-0.08 3.40 -0.82 0.69 3.10];
>> [pn,minp,maxp,tn,mint,maxt] = premnmx(P,t);
>> net = newff(minmax(pn),[5 1],{'tansig' 'purelin'},'trainlm');
>> net = train(net,pn,tn); grid on
>> an = sim(net,pn)
an = -0.6493 1.0000 -1.0000 -0.2844 0.8578
>> a = postmnmx(an,mint,maxt)
a = -0.0800 3.4000 -0.8200 0.6900 3.1000
```

Аналогично используется функция prestd, которая выполняет масштабирование обучающей последовательности по нормальному закону распределения с параметрами $[0 \ 1]$, а восстановление выхода происходит с помощью функции poststd.

Вопросы для самопроверки

1. Каковы свойства искусственных нейронных сетей прямого распространения?
2. В чем сходство ИНС ПР и комбинационных логических схем?

3. От чего зависит число входов, выходов, а также нейронов во внутренних слоях ИНС ПР?
4. Каким свойством должна обладать ИНС ПР, чтобы быть универсальным аппроксиматором?
5. Какие параметры требуется указать при создании ИНС ПР в MatLab?
6. В чем заключается проблема обучения многослойной ИНС ПР?
7. Обобщением какого алгоритма является алгоритм обратного распространения ошибки?
8. Почему сигмоидная активационная функция удобна при использовании АОРО?
9. Какова последовательность действий при использовании АОРО?
10. В чем состоит явление переобучения ИНС?
11. Что собой представляет градиентный метод поиска минимума функции?
12. Какой вид имеет формула коррекции весов выходного слоя ИНС ПР при использовании АОРО?
13. Какой вид имеет формула коррекции весов скрытых слоев ИНС ПР при использовании АОРО?
14. Как выбирается константа скорости обучения в АОРО?
15. В чем заключается основной недостаток АОРО для обучения нейронных сетей?
16. В чем состоит задача аппроксимации функции с помощью нейронной сети?
17. В чем состоит задача распознавания символов с помощью нейронной сети?
18. Можно ли с помощью ИНС ПР моделировать динамическую систему?
19. В чем заключается статичность поведения ИНС ПР?

4. НЕЙРОУПРАВЛЕНИЕ

4.1. Идентификация динамических звеньев

В процессе идентификации требуется по известным вход-выходным зависимостям динамического объекта построить его описание, которое можно использовать для предсказания выходного сигнала при произвольном входном.

Однако, как уже отмечалось, ИНС ПР – это не динамическая сеть. Простой путь внесения динамики в поведение ИНС заключается в подаче на вход сети не только текущих, но и задержанных значений входа и выхода. Число задержанных сигналов и величина задержки зависят от конкретного объекта.

На рис. 4.1 приведен принцип использования нейросетевой модели в режиме идентификации. Число линий задержки Δ на входах ИНС должно примерно соответствовать порядку объекта. В этой схеме с помощью алгоритма обратного распространения должна минимизироваться ошибка между выходом объекта $y(t)$ и выходом модели $y^m(t)$.

Полученная нейросетевая модель является «черным ящиком». Она не позволяет судить о физических процессах, протекающих в объекте управления, но может быть эффективно использована для анализа и прогноза поведения объекта, а также для синтеза системы управления.



Рис. 4.1. Нейросетевая идентификация

Для линейных и слабонелинейных объектов управления классические методы идентификации могут не уступать нейросетевым методам. Однако ИНС являются универсальным инструментом и пригодны для идентификации существенно нелинейных объектов, о которых имеется малый объем априорной информации.

Пример 4.1. Рассмотрим линейный динамический объект из примера 3.13 (переходный процесс приведен на рис. 3.26). Это объект 2-го порядка, поэтому ему может соответствовать ИНС с двумя линиями задержки (рис. 4.2).

Заметим, что структурная избыточность является характерным свойством ИНС, поэтому в структуре, приведенной на рис. 4.2, можно было выбрать не три, а больше нейронов скрытого слоя.

Схема моделирования представлена на рис. 4.3 (используется метод интегрирования с постоянным шагом 0,01 с).

Далее опишем ИНС ПР с тремя входами, зададим параметры и запустим процесс обучения:

```
>> net=newff([0 1; -3 3; -3 3], [3,1], {'purelin','purelin'},'trainlm');
>> P = simout';
>> T = simout1';
>> net.trainParam.show = 50;
>> net.trainParam.lr = 0.05;
>> net.trainParam.epochs = 1000;
>> net.trainParam.goal = 0.001;
>> net1 = train(net, P, T);
```

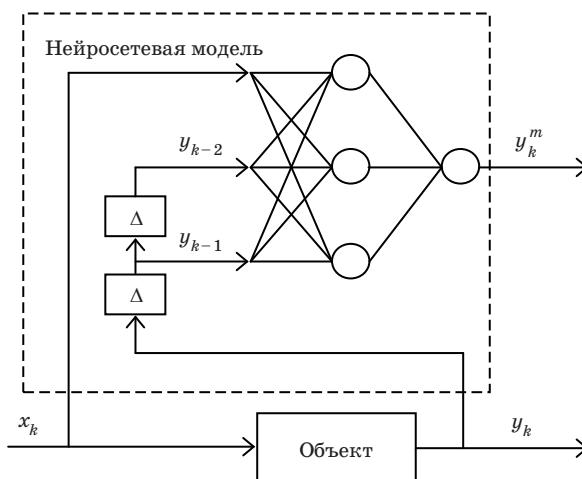


Рис. 4.2. Нейросетевая идентификация объекта 2-го порядка

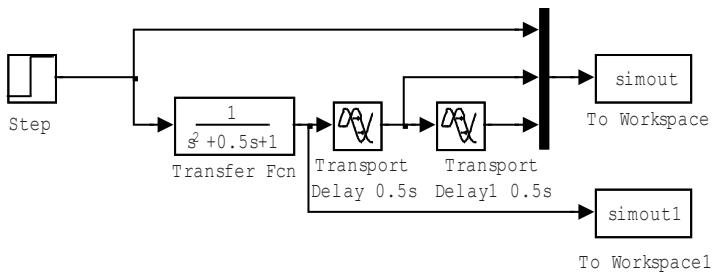


Рис. 4.3. Схема подготовки данных для идентификации

Для обучения потребовалось всего четыре эпохи (рис. 4.4).

Выполним проверку нейросетевой модели (схема на рис. 4.5). С помощью следующей команды создается блок Neural Network в Simulink-модели:

```
>> gensim(net1,0.01)
```

Здесь 0,01 – шаг интегрирования по времени.

Как следует из рис. 4.6, выходы объекта и модели достаточно близки. Повышения качества идентификации можно добиться, изменяя параметры нейросетевой модели.

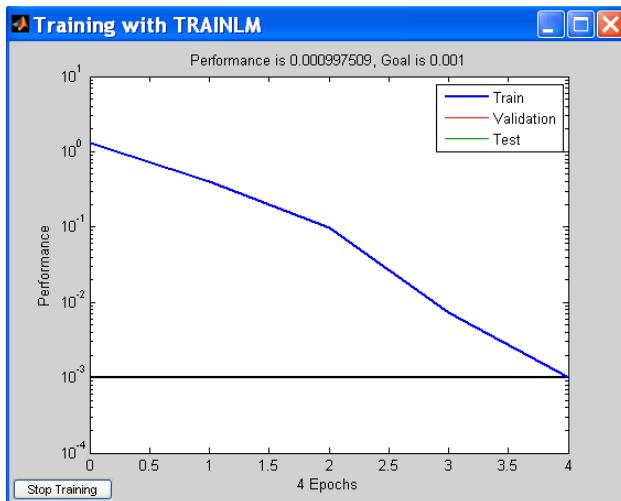


Рис. 4.4. Процесс обучения нейросетевого идентификатора

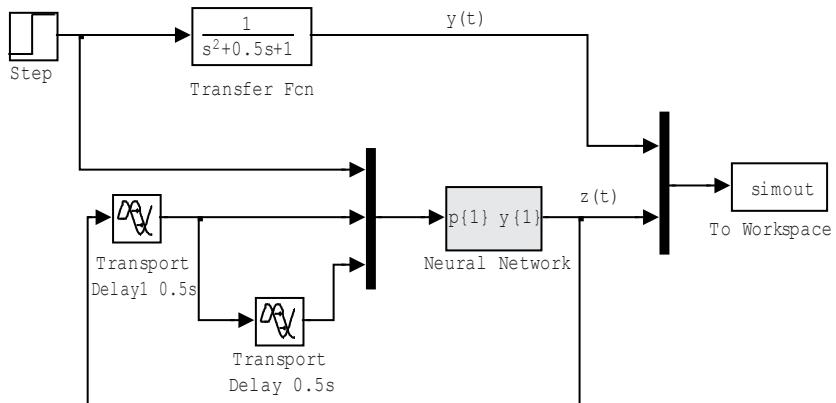


Рис. 4.5. Схема проверки качества нейросетевой модели

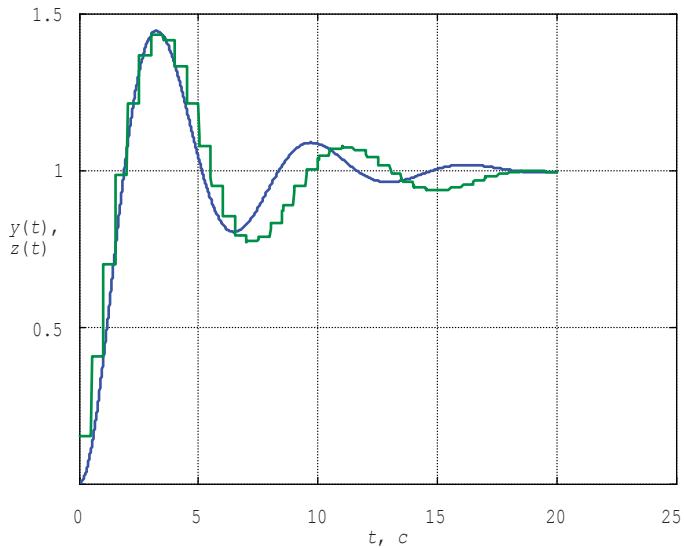


Рис. 4.6. Сравнение выхода объекта и его нейросетевой модели

Пример 4.2. Рассмотрим пример идентификации нелинейной системы:

$$y(k+1) = \frac{y(k)(y(k-1) + 2)(y(k) + 3)}{8 + y(k)^2 + y(k-1)^2} + u(k).$$

Перепишем уравнения системы в форме

$$y(k+1) = f(y(k), y(k-1)) + u(k).$$

Тогда задачу идентификации можно представить в виде, приведенном на рис. 4.7, где нейронная сеть реализует оператор f .

Опишем входной сигнал (система устойчива при $u \in [-2, 2]$):

```
>> u=rands(1, 301)*2;
```

Реакция системы на входной сигнал:

```
>> for k=2 : 301
```

$$\begin{aligned} y(k+1) &= y(k) * (y(k-1) + 2) * (y(k) + 3) / (8 + y(k)^2 + y(k-1)^2) + u(k); \\ \text{out}(k-1) &= (y(k+1) - u(k)) / 20; \\ \text{in}(k-1) &= y(k) / 20; \end{aligned}$$

```
end;
```

Последние две команды цикла нормируют входы и выходы ИНС. Затем создадим ИНС ПР с двумя входами и одним выходом:

```
>> net = newff([min(in) max(in); min(in) max(in)], [2 10 1], {'tansig' 'tansig'}, 'trainlm', 'learngdm', 'mse');
```

Сформируем обучающие данные:

```
>> plantin=[in(1:299); in(2:300)];  
>> plantout=out(1:299);
```

и обучим нейронную сеть:

```
>> net.trainParam.epochs = 500;  
>> net.trainParam.goal = 0.0005;  
>> net=train(net, plantin, plantout);
```

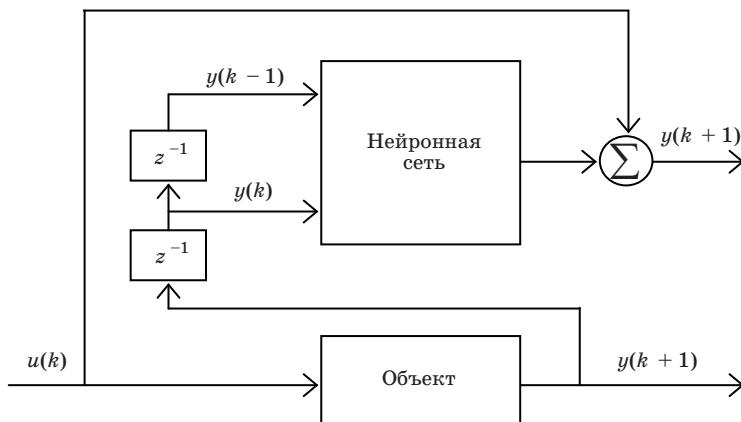


Рис. 4.7. Идентификация нелинейного объекта

Далее выполним проверку обученной нейронной сети, подав последовательно два синусоидальных сигнала:

```

>> yp(1)=0.0; yp(2)=0.0;out1(1)=0; out1(2)=0;
>> for k=2:500
    if (k<=200)u(k)=2.0*cos(2*pi*k*0.01);
    else
        u(k)=1.2*sin(2*pi*k*0.05);
    end;
    yp(k+1)=yp(k)*(yp(k-1) + 2)*(yp(k) + 2.5)/(8.5 + yp(k)^2 + yp(k-
1)^2) + u(k);
    out1(k)=yp(k)/20;
    out1(k-1)=yp(k-1)/20;
    nnout(k+1)=20*sim(net,[out1(k);out1(k-1)]) + u(k);
end;
>> plot(yp, 'b');
>> hold on;
>> plot(nnout, ':k');
>> grid;
>> axis([0, 500, -4.0, 10.0]);

```

На рис. 4.8 приведены графики выхода объекта (сплошная линия) и идентификационной модели (пунктир).

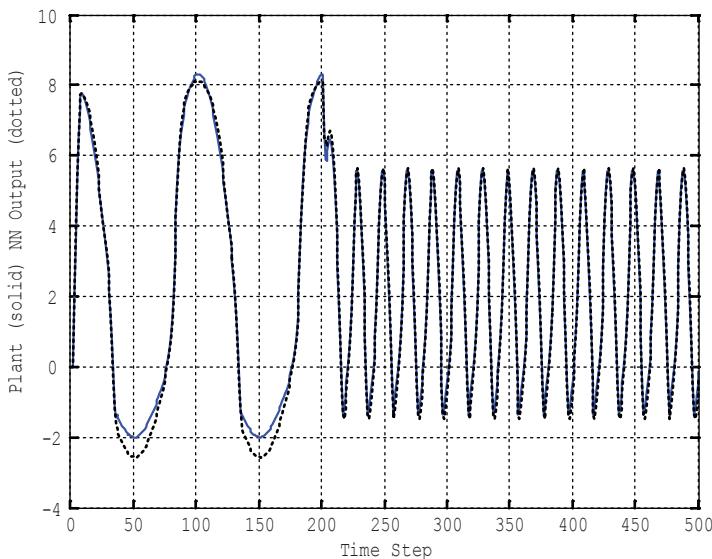


Рис. 4.8. Проверка качества идентификации

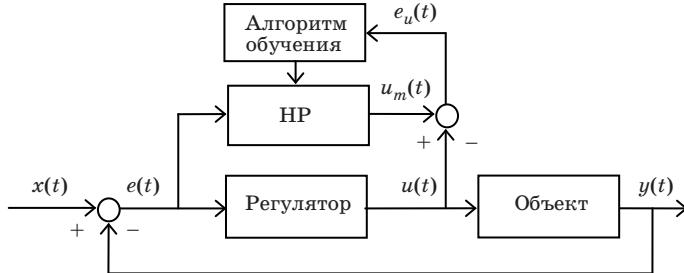


Рис. 4.9. Обучение с помощью существующего регулятора

С помощью ИНС можно идентифицировать не только объект управления, но и регулятор объекта, поскольку последний также является динамическим звеном. Искусственные нейронные сети могут быть использованы в различных конфигурациях в контексте управления. Такая сеть может заменять человека в контуре управления, когда работа происходит в опасных условиях или для исключения ошибок из-за усталости, неопытности и т. п.

Для замены существующего регулятора (или человека-оператора) нейросетевой моделью в режиме *on line* (рис. 4.9) следует в определенные моменты времени считывать с датчиков сигналы со входа регулятора (т. е. ошибка $e(t)$ или описание состояния объекта $x(t)$) и с его выхода – сигнал управления $u(t)$. Описание состояния подается на вход ИНС, а описание сигнала управления используется для расчета текущей ошибки выхода ИНС. Алгоритм обратного распространения ошибки позволяет минимизировать ее:

$$e_u(t) = u_m(t) - u(t).$$

При обучении в режиме *offline* с помощью существующего регулятора создается обучающая выборка, и ИНС обучается по этой выборке.

Пример 4.3. Пусть имеется система управления с ПД-регулятором. Требуется заменить его эквивалентным нейронным.

На рис. 4.10 приведена исходная схема, необходимая для получения обучающей выборки.

Поскольку ПД-регулятор является линейным, для его замены можно использовать ИНС ПР с линейной активационной функцией:

```
>> net=newff([-1 1; -5 5], [3,1], {'purelin','purelin'},'trainlm');
P = simout';
T = simout1';
net.trainParam.show = 50;
```

```

net.trainParam.lr = 0.05;
net.trainParam.epochs = 1000;
net.trainParam.goal = 0.001;
net1 = train(net, P, T);

```

Как следует из рис. 4.11, для обучения потребовалось всего шесть эпох.

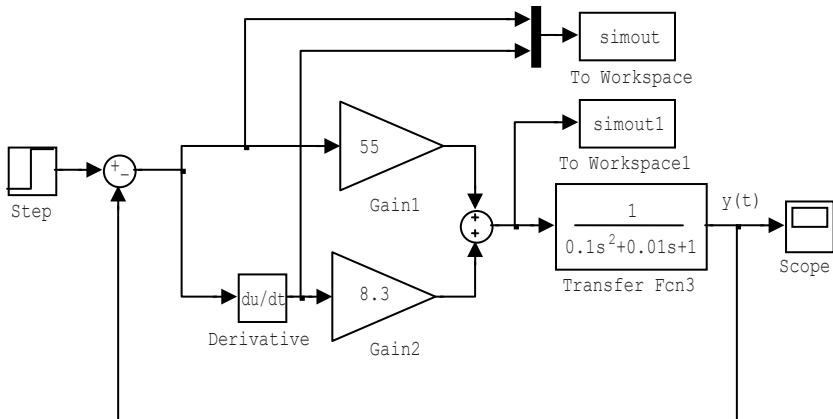


Рис. 4.10. Система с ПД-регулятором в MatLab Simulink

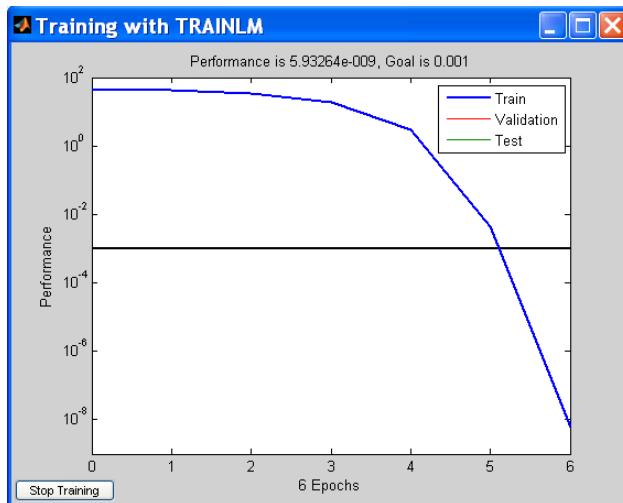


Рис. 4.11. Изменение ошибки в процессе обучения нейронного регулятора

На рис. 4.12 приведена эквивалентная схема с нейронным регулятором.

Переходные процессы исходного регулятора и его нейронной модели практически совпадают (рис. 4.13).

Полученный таким способом нейронный регулятор, естественно, не может работать лучше своего прототипа, использованного при обучении. Для повышения качества управления необходимо применять другие подходы.

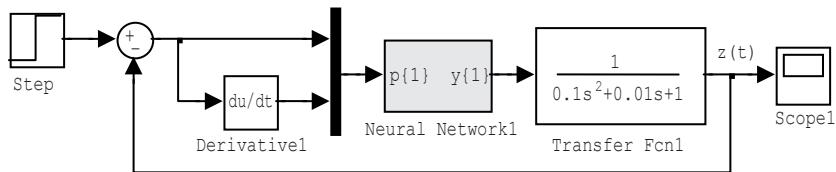


Рис. 4.12. Нейронный регулятор, эквивалентный ПД-регулятору

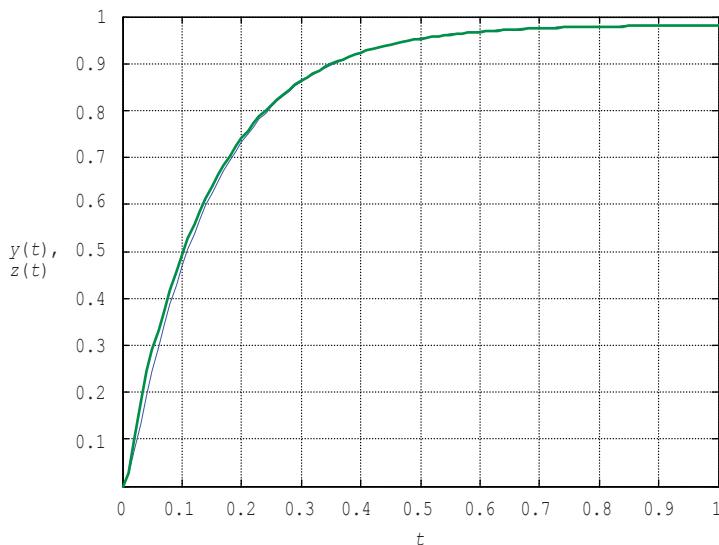


Рис. 4.13. Сравнение переходных процессов

4.2. Нейроэмуляторы и нейропредикторы

Представление динамического объекта на базе ИНС в корне отличается от описания динамического объекта с помощью дифференциальных уравнений.

Аналитическое описание с помощью дифференциальных уравнений использует «глубинные» знания, полученные на основе законов физики, химии, биологии и других фундаментальных наук.

Нейросетевое же моделирование рассматривает объект в качестве «черного ящика», у которого известны входы и выходы. Задача ИНС – наилучшим образом подражать поведению этого «черного ящика».

Вместе с тем при аналитическом описании используется некоторый шаблон – уравнения динамики, – в котором следует настроить значения параметров конкретного объекта. У нейросетевой модели шаблоном является структура ИНС, а настроить требуется коэффициенты межнейронных связей.

Для того чтобы подчеркнуть особенности использования нейросетевой модели, вводятся понятия *нейросетевого эмулятора* и *нейросетевого предиктора* [40].

Одношаговый предиктор получает информацию о состоянии объекта у самого объекта, что позволяет делать прогноз состояния на один шаг вперед (рис. 4.14).

Нейросетевой эмулятор вместо выхода объекта использует выход нейронной сети. Это позволяет делать прогноз динамики объекта на много шагов вперед, однако ошибка здесь накапливается с увеличением длительности прогноза (рис. 4.15).

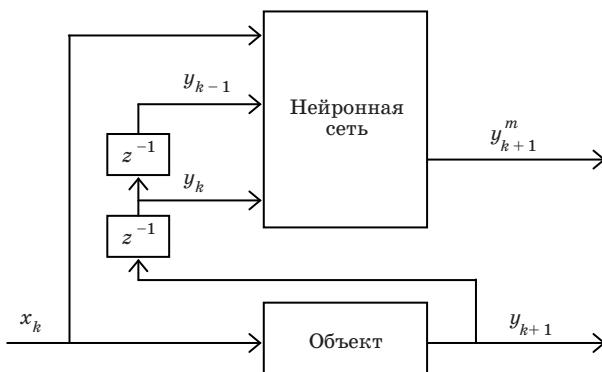


Рис. 4.14. Нейросетевой предиктор

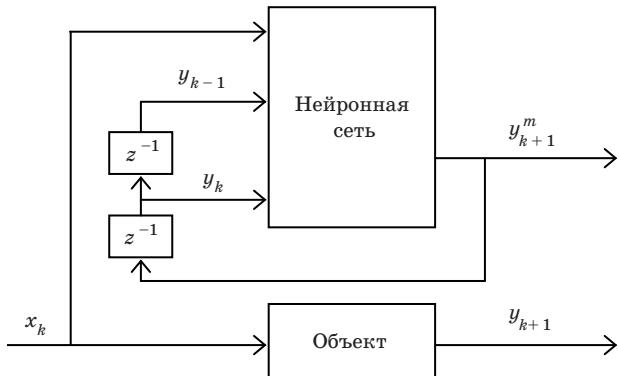


Рис 4.15. Нейросетевой эмулятор

Интуитивно понятно, что использование краткосрочного предиктора естественно в ситуации, когда структура объекта значительно сложнее структуры ИНС. Например, в задачах предсказания биржевых котировок или курсов валют неразумно пытаться получить предсказание на длительный срок ввиду множества факторов, которые влияют на поведение объекта прогноза. Реализация же нейроэмульятора возможна, если структура ИНС соответствует сложности объекта.

4.3. Концепция нейроуправления

Человеческий мозг решает самые разнообразные задачи по обеспечению жизнедеятельности организма. Например, перемещение из одной точки пространства в другую требует фиксации цели движения и определения ошибки положения, исходя из которой формируются сигналы управления двигательной системой (рис. 4.16).

Легко заметить, что рис. 4.16 соответствует традиционной схеме управления с обратной связью (рис. 4.17, где $u(t)$ – сигнал управления; $y(t)$ – реальный выходной сигнал; $g(t)$ – желаемый выходной сигнал; $e(t)$ – ошибка управления). В рамках этой структуры мозг выступает в качестве регулятора, костно-мускульная система является объектом управления, а органы чувств играют роль датчиков.

Задача синтеза регулятора в замкнутой системе управления может рассматриваться как задача синтеза обратной модели объекта (рис. 4.18, где $g(t)$ – задающее воздействие).

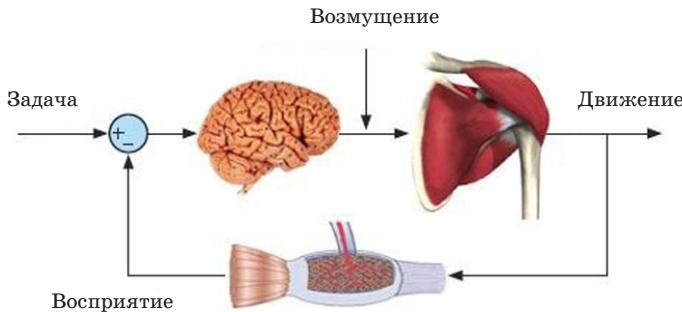


Рис. 4.16. Принципы управления с обратной связью

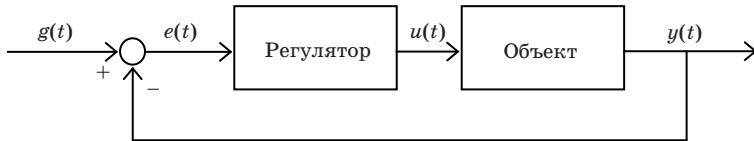


Рис. 4.17. Управление с обратной связью



Рис. 4.18. Идеализированная система управления

Если F – оператор объекта, а F^{-1} – оператор регулятора, то при любых условиях выполняется равенство

$$y(t) = g(t) F^{-1} F = g(t).$$

В реальных условиях точное получение обратной модели объекта обычно невозможно. Рассмотрим эту проблему на примерах.

Пусть F – статический оператор, реализующий монотонную функциональную зависимость $y = F(x)$ (рис. 4.19,а). Тогда можно построить обратную зависимость $x = F^{-1}(y)$ (рис. 4.19,б).

Если же оператор модели не является монотонным, то обратная зависимость не будет функциональной. Следовательно, модель необратима (рис. 4.20).

Для описания динамических моделей объектов часто применяют аппарат передаточных функций (ПФ). Если $W(s)$ – передаточ-

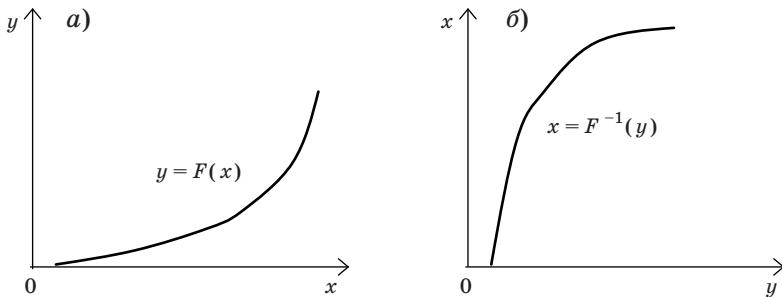


Рис. 4.19. Монотонные прямой и обратный оператор статической модели

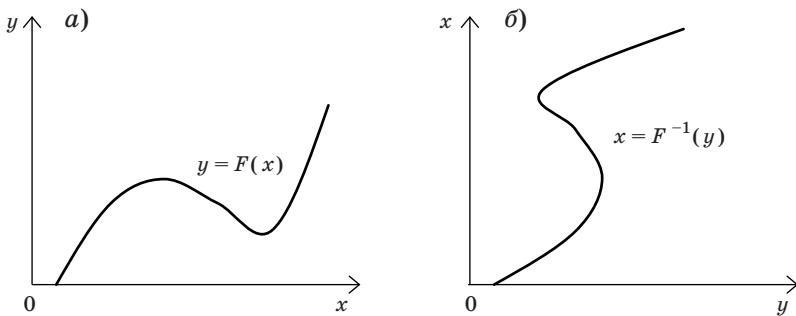


Рис. 4.20. Пример необратимой статической модели

ная функция объекта, то для обратной модели $W_{\text{inv}}(s)$ должно выполняться условие

$$W(s)W_{\text{inv}}(s) = 1. \quad (4.1)$$

При этом $W_{\text{inv}}(s)$ должна быть минимально-фазовой передаточной функцией.

Например, пусть $W(s)$ – апериодическое звено 1-го порядка:

$$W(s) = \frac{1}{T_1 s + 1}.$$

Тогда

$$W_{\text{inv}}(s) = T_1 s + 1.$$

Формально условие (4.1) выполняется, однако $W_{\text{inv}}(s)$ содержит операцию идеального дифференцирования, которая физически не реализуема. Для того чтобы избежать этого ограничения, вместо (4.1) можно рассмотреть условие

$$W(s)W_{\text{inv}}(s) = \frac{1}{T_2 s + 1}. \quad (4.2)$$

Тогда

$$W_{\text{inv}}(s) = \frac{T_1 s + 1}{T_2 s + 1}.$$

Чем меньше постоянная времени T_2 , тем ближе (4.1) и (4.2).

Пример 4.4. Пусть в задаче управления электромотором математическая модель задана передаточной функцией

$$W_{\text{дв}}(s) = \frac{K_{\text{дв}}}{T_{\vartheta} T_{\text{м}} s^2 + (T_{\text{м}} + T_{\vartheta}) s + 1},$$

где $K_{\text{дв}}$ – общий коэффициент усиления двигателя; $T_{\text{м}}$ – механическая постоянная времени; T_{ϑ} – электрическая постоянная времени.

Рассмотрим задачу синтеза ПИД-регулятора, передаточная функция которого имеет следующую форму:

$$W_p(s) = K_p + K_i \frac{1}{s} + K_d s, \quad (4.3)$$

где K_p , K_i , K_d – неизвестные коэффициенты.

Допустим, что требуется обеспечить апериодический переходный процесс в замкнутой системе, заданный передаточной функцией

$$W_{\text{ж}}(s) = \frac{1}{T_{\text{ж}} s + 1}.$$

Приравняем передаточную функцию желаемой системы к передаточной функции замкнутой системы

$$\frac{1}{T_{\text{ж}} s + 1} = \frac{W_p(s)W_{\text{дв}}(s)}{1 + W_p(s)W_{\text{дв}}(s)}$$

и найдем передаточную функцию регулятора

$$W_p(s) = \frac{1}{T_{\text{ж}} s W_{\text{дв}}(s)}.$$

После подстановки в последнюю формулу значения $W_{\text{дв}}(s)$ получим

$$W_p(s) = \frac{T_{\text{м}} + T_{\vartheta}}{T_{\text{ж}} K_{\text{дв}}} + \left(\frac{1}{T_{\text{ж}} K_{\text{дв}}} \right) \frac{1}{s} + \left(\frac{T_{\text{м}} T_{\vartheta}}{T_{\text{ж}} K_{\text{дв}}} \right) s.$$

В этом примере оказалось возможным получить аналитическое решение задачи синтеза регулятора, который является обратной моделью объекта.

Однако на практике аналитическое решение обычно невозмож но. Применение нейронных сетей позволяет находить приближенные решения этой задачи.

Передаточной функции ПИД-регулятора (4.3) соответствует формула

$$u(t) = K_p e(t) + K_d \frac{de(t)}{dt} + K_i \int e(t) dt.$$

Такому регулятору можно поставить в соответствие единственный нейрон с линейной активационной функцией (рис. 4.21).

Искусственная нейронная сеть как универсальный настраиваемый элемент может быть использована в самых разных постановках задач при анализе и синтезе систем автоматического управления.

В системах управления наиболее часто применяется ИНС ПР, которая может выступать в качестве идентификационной модели объекта управления или регулятора [26].

В теории автоматического управления динамические звенья (объекты управления и регуляторы) часто описываются передаточной функцией ([41] и др.):

$$W(s) = \frac{y(s)}{x(s)} = \frac{s^m b_m + s^{m-1} b_{m-1} + \dots + s b_1 + b_0}{s^n a_n + s^{n-1} a_{n-1} + \dots + s a_1 + a_0},$$

где y – выходной сигнал звена; x – входной сигнал; a_i и b_j – постоянные коэффициенты; $u(s)$ и $e(s)$ – изображения по Лапласу выхода и входа регулятора, $m \leq n$.

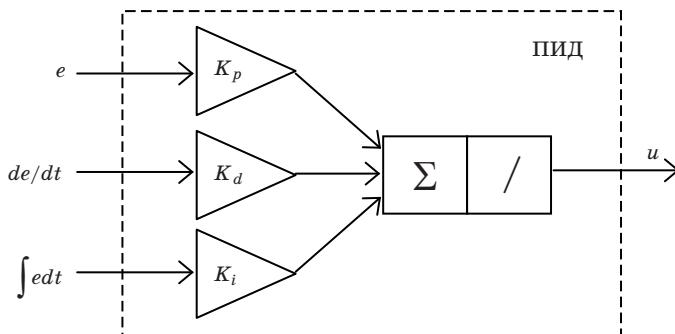


Рис. 4.21. ПИД-регулятор как нейроноподобная структура

Это выражение можно записать в виде

$$\begin{aligned} y(s)(s^n a_n + s^{n-1} a_{n-1} + \dots + s a_1 + a_0) &= \\ = x(s)(s^m b_m + s^{m-1} b_{m-1} + \dots + s b_1 + b_0). \end{aligned}$$

При цифровом моделировании рассматривается дискретное представление передаточной функции. Процесс дискретизации ПФ заключается в замене операторов дифференцирования отношениями конечных разностей

$$sx = \frac{x_k - x_{k-1}}{\Delta t}, \quad s^2x = \frac{x_k - 2x_{k-1} + x_{k-2}}{\Delta t^2}, \dots$$

где Δt – шаг дискретизации по времени; k – номер момента времени.

После всех преобразований и упрощений получаем формулу вида

$$y_n = \sum_{i=1}^n a'_i y_{n-i} + \sum_{j=1}^m b'_j y_{m-j},$$

где a'_i и b'_j – постоянные коэффициенты, зависящие от шага дискретизации по времени.

Этого же результата можно добиться с использованием z -преобразования, при котором получается дискретная ПФ, имеющая следующий стандартный вид:

$$W(z) = \frac{y(z)}{x(z)} = \frac{b_0 + b_1 z^{-1} + \dots + b_m z^{-m}}{a_0 + a_1 z^{-1} + \dots + a_n z^{-n}},$$

где оператор z^{-i} означает задержку на i тактов.

Заметим, что из непрерывной ПФ можно получить бесконечное число вариантов дискретной ПФ при разных периодах дискретизации.

Дискретной ПФ соответствует структура, показанная на рис. 4.22. Часто ее называют *цифровым фильтром*.

В реальных системах число линий задержки обычно невелико. Например, при цифровой реализации ПИД-регулятора можно использовать формулу вида

$$y = z^{-1}y + b_1x + b_2z^{-1}x + b_3z^{-2}x.$$

Структура на рис. 4.22 очевидно напоминает нейрон с обратными связями и линейной активационной функцией. Ограниченные возможности такой системы были проанализированы в предыдущих разделах.

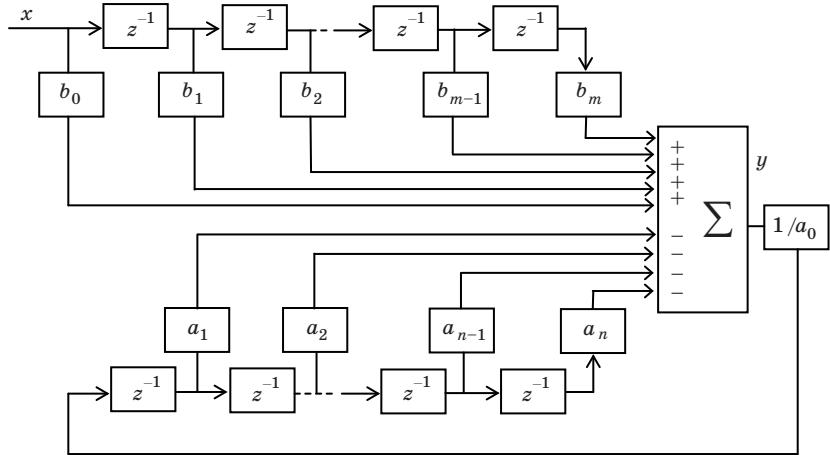


Рис. 4.22. Структура цифрового фильтра

Использование многослойных ИНС ПР дает более широкие возможности моделирования, поскольку их поведение может быть нелинейным (рис. 4.23).

Существует два подхода к применению нейронных регуляторов (или нейроконтроллеров (НК)) ([42 – 45] и др.):

– *прямые методы*, основанные на непосредственном управлении объектом с помощью НР;

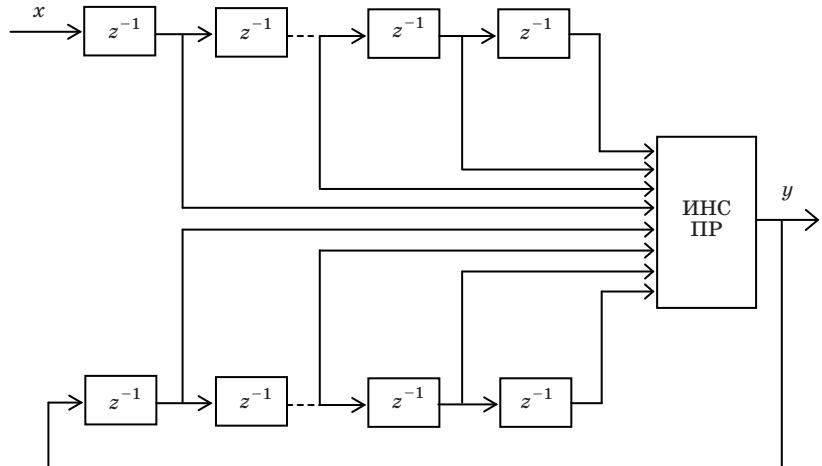


Рис. 4.23. Общая структура нейронного регулятора

– *непрямые методы*, когда ИНС используется для выполнения вспомогательных функций управления, таких, как фильтрация шума или идентификация динамического объекта.

В зависимости от числа нейронных сетей НК могут быть *одномодульными* или *многомодульными*.

Схемы нейроуправления, в которых НК применяются совместно с традиционными контроллерами, называются *гибридными*.

В задачах нейроуправления для представления объекта управления используют модель типа «черный ящик», в котором наблюдаемыми являются текущие значения входа и выхода. Состояние объекта считается недоступным для внешнего наблюдения, хотя размерность вектора состояний обычно принимается фиксированной. Динамику поведения объекта управления можно представить в дискретном виде:

$$\begin{cases} X(k+1) = F(X(k), U(k)), \\ Y(k+1) = G(X(k)), \end{cases}$$

где $X(k) \in R^n$ – значение вектора состояния на такте k ; $U(k) \in R^m$ – значение вектора входа (управления); $Y(k+1) \in R^p$ – значение вектора выхода на такте $k+1$.

Для оценки вектора состояния динамического объекта порядка n может быть использована модель нелинейной авторегрессии с дополнительными входными сигналами (NARX), которая для одномерного объекта принимает вид

$$X(k) = [y(k), y(k-1), \dots, y(k-n), u(k), u(k-1), \dots, u(k-m)]^T.$$

Эту модель часто упрощают, приводя к модели вида

$$X(k) = [y(k), y(k-1), \dots, y(k-n)]^T.$$

Таким образом, на основании этого выражения и рис. 4.23 структуру системы управления можно изобразить так, как показано на рис. 4.24 (Δ – обозначение линии задержки).

Будем считать, что в приводимых далее схемах ИНС имеет на выходе необходимые линии задержки.

4.4. Инверсное нейроуправление

При инверсном нейроуправлении формирование инверсной модели объекта управления осуществляется путем обучения нейронной сети. Известно несколько видов такого нейроуправления, ори-

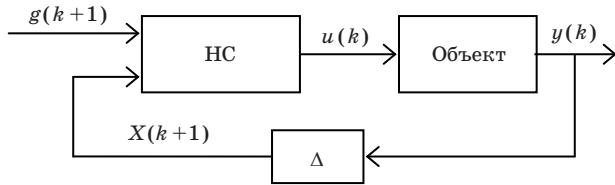


Рис. 4.24. Нейроуправление по состоянию

ентированных на использование при обучении алгоритма обратного распространения ошибки [26, 42, 43].

В схеме *прямого инверсного управления* (Direct Inverse Neuro-control или Generalized Inverse Neurocontrol) ИНС рассматривается как инверсная модель динамики системы (рис. 4.25).

Обучение ИНС происходит в режиме offline, причем сигнал $u(t)$ здесь выступает в качестве тестового, поэтому он может представлять собой некоторый случайный процесс.

В ходе обучения ИНС должна аппроксимировать зависимость значений управляющего сигнала $u(k)$ от предыдущего состояния $X(k-1)$. Таким образом, обучающую выборку формирует множество пар вида

$$\{X(k-1), u(k)\}_i, i = 1, 2, \dots, N.$$

Основная трудность использования алгоритма обратного распространения ошибки в рамках этой схемы заключается в том, что при этом необходимо находиться в окрестности глобального минимума ошибки, а это требует достаточно точного знания структуры системы. Поэтому более эффективным может быть использование алгоритмов глобальной оптимизации, таких, как *генетический алгоритм*.

После обучения ИНС используется для непосредственного управления объектом.

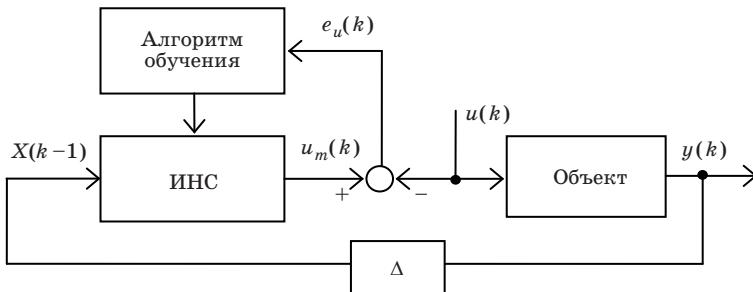


Рис. 4.25. Прямое инверсное нейроуправление

Достоинством прямого инверсного нейроуправления являются возможность обучения в режиме offline и отсутствие необходимости в точной математической модели объекта управления. К недостаткам метода следует отнести сложность формирования обучающей выборки из-за необходимости тщательного подбора идентифицирующего случайного процесса, подаваемого на вход системы, а также низкое качество работы в тех случаях, когда инверсия объекта управления оказывается неоднозначной функцией. Неоднозначность приводит к появлению противоречий в обучающей выборке, заводящих в тупик процесс обучения нейронной сети.

Для решения этой проблемы предлагается ряд методов.

В схеме *непрямого (косвенного) инверсного управления*, приведенной на рис. 4.26, используются две копии ИНС как инверсной модели [26].

Входной сигнал поступает на вход первой копии ИНС, которая вырабатывает сигнал управления. Вторая копия ИНС использует аналогично схеме прямого инверсного обучения.

В общем случае регулятор должен компенсировать отклонение реального выхода объекта управления (ОУ) $y(t)$ от желаемого сигнала $g(t)$:

$$e_y(t) = y(t) - g(t).$$

Проблема использования схемы непрямого управления заключается в том, что она иногда не работает, потому что в процессе обучения ИНС может отображать большое число различных входных сигналов на одно и то же значение управления (вырожденное отображение), так что

$$e_u(t) = 0 \text{ при } e_y(t) \neq 0.$$

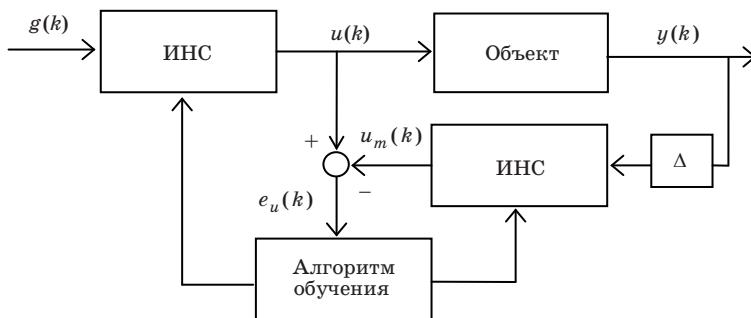


Рис. 4.26. Непрямое инверсное управление

Поэтому при обучении целесообразно использовать ошибку выхода объекта $e_y(t)$, а не ошибку управления $e_u(t)$.

В методе *специализированного инверсного нейроуправления* (рис. 4.27) проблема обучения инверсной динамике решается путем аппроксимации аналитической модели управляемого объекта и вычисления локальных значений якобиана для различных областей пространства состояний.

Коррекция весовых коэффициентов нейронной сети выполняется методом наискорейшего спуска:

$$w(k+1) = w(k) - \Delta w(k),$$

$$\Delta w(k) = -\alpha e(k) \frac{\partial y(k+1)}{\partial u(k)} \frac{\partial u(k)}{\partial w(k)},$$

где α – коэффициент скорости обучения.

Величина производной $\frac{\partial u(k)}{\partial w(k)}$ вычисляется методом обратного распространения ошибки, а производная $\frac{\partial y(k+1)}{\partial u(k)}$ представляется собой якобиан объекта управления, значение которого можно найти аналитически по заданной математической модели объекта управления. На практике для получения приемлемого качества управления часто бывает достаточно вычислить лишь знак якобиана [26]. Коррекция значений весовых коэффициентов продолжается до достижения приемлемого качества управления.

Нейронная сеть обучается с целью минимизации ошибки управления $e_y(t)$. Метод может обеспечить оптимальное решение, если ошибка измеряется надежно.

Проблема специализированного инверсного обучения при использовании АОРО заключается в том, что в этой схеме $e_y(t)$ оказы-

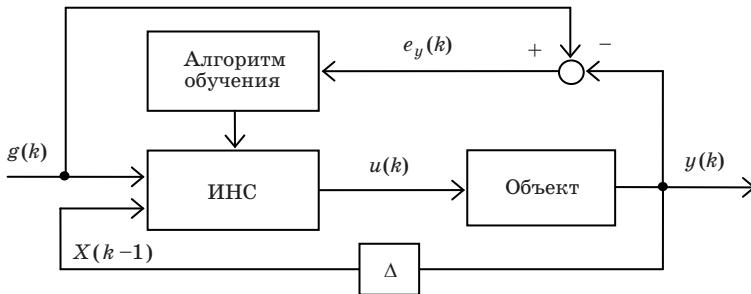


Рис. 4.27. Специализированное инверсное управление

вается нескоррелированной с ошибкой обучения инверсного контроллера $e_u(t)$, т. е. она может иметь другие знак и амплитуду [26].

В то же время любая достаточно грубая аппроксимация $e_u(t)$ с помощью $e_y(t)$ позволяет процессу обучения сходиться.

Для того чтобы избежать проблем специализированного инверсного обучения была разработана *схема обратного распространения ошибки во времени* [26, 43].

Этот алгоритм основан на идее применения двух нейронных сетей, одна из которых выполняет функцию нейроконтроллера, а вторая – функцию нейроэмулатора, который обучается моделировать динамику объекта управления.

На первом этапе происходит обучение нейроэмулатора в режиме offline. Нейроэмулатор считается обученным, если при одинаковых значениях на его входах и входах реального объекта различие их значений на выходах становится незначительным. После завершения обучения нейроэмулатора проводится обучение нейроконтроллера. Обучение выполняется в режиме on line по такой же схеме, как и в случае инверсного специализированного нейроуправления (рис. 4.28).

В процессе обучения НК текущая ошибка управления пропускается через нейроэмулатор в обратном направлении.

Алгоритм позволяет эмулировать $e_u(t)$ посредством использования ошибки обратного распространения $e_y(t)$ через предсказывающую модель (forward model). Здесь ошибка $e_y(t)$, распространяясь обратно к входному слою, будет соответствовать ошибке $e_u(t)$. Это делает схему обучения пригодной для использования в большинстве линейных систем.

Механизм обратного прохождения ошибки через нейроэмулатор реализует локальную инверсную модель в текущей точке про-

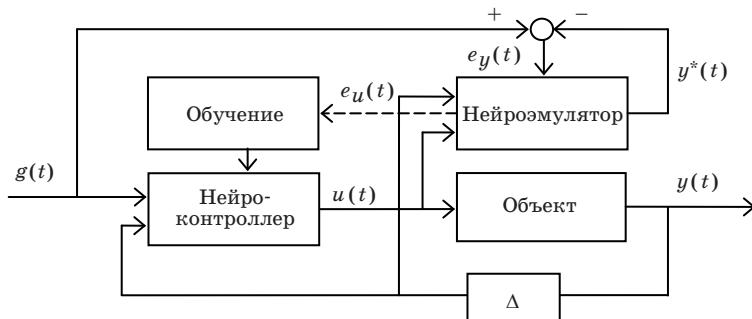


Рис. 4.28. Обратное распространение во времени

странства состояний объекта управления. Пройдя через нейроэмulator, ошибка распространяется далее через нейроконтроллер, но теперь ее прохождение сопровождается коррекцией весовых коэффициентов НК. Нейроэмulator при этом выполняет функции дополнительных слоев нейронной сети НК, в которых веса связей не корректируются.

4.5. Нейроконтроллеры в MatLab

Нейронные сети успешно применяются при синтезе систем управления динамическими процессами. Универсальные возможности аппроксимации многослойных ИНС прямого распространения позволяют решать задачи идентификации, проектирования и моделирования нелинейных систем управления.

В пакете прикладных программ Neural Network toolbox реализованы три варианта контроллеров:

1. Контроллер с предсказанием (NN Predictive Controller).
2. Контроллер на основе авторегрессии со скользящим средним (NARMA – L2 Controller).
3. Контроллер на основе эталонной модели (Model Reference Controller).

Все эти варианты требуют выполнения двух этапов проектирования: идентификации управляемого процесса и синтеза закона управления.

На этапе идентификации разрабатывается модель управляемого процесса в виде ИНС, которая затем используется для синтеза регулятора.

Схема идентификации управляемого процесса приведена на рис. 4.29. Она включает в себя модель управления в виде ИНС, которая должна быть обучена в автономном режиме так, чтобы минимизировать ошибку между выходами объекта и модели: $e = y_p - y_m$ для последовательности пробных входных сигналов u .

Для идентификации применяется двухслойная ИНС с линиями задержки (рис. 4.30). Модель использует предыдущий вход и предыдущий выход объекта для предсказания его будущего выхода.

Модель обучается в режиме offline, используя накопленную ранее информацию о поведении объекта.

Настройка ИНС производится по данным испытаний реального объекта. Метод обучения может быть любым из имеющегося набора.

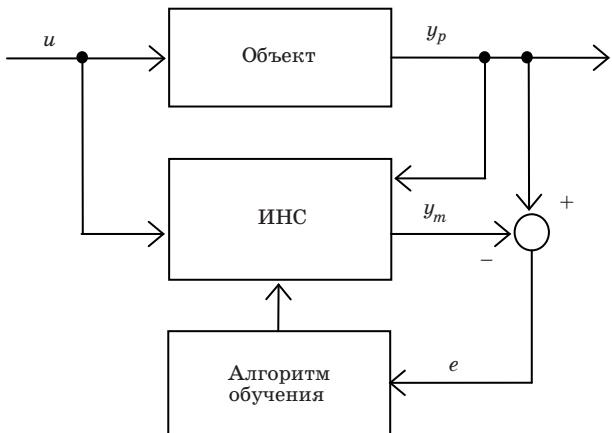


Рис. 4.29. Общая схема нейросетевой идентификации в MatLab

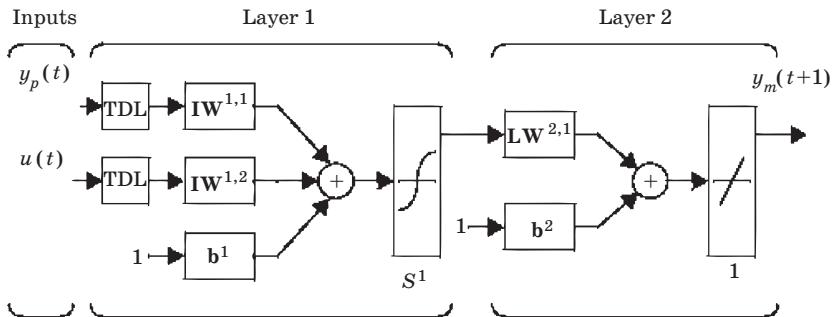


Рис. 4.30. Идентификационная нейронная сеть

Хотя для каждой из трех архитектур используется одна и та же процедура идентификации, этапы синтеза существенно различны.

При управлении с предсказанием модель управляемого процесса в виде ИНС используется для предсказания его будущего поведения, а алгоритм оптимизации применяется для расчета такого управления, которое минимизирует разность между желаемым и действительными изменениями выхода модели. Контроллер, реализующий такой регулятор, требует значительного объема вычислений, поскольку для расчета оптимального закона управления оптимизация выполняется на каждом такте управления.

Регулятор с предсказанием использует модель нелинейного управляемого процесса в виде ИНС для предсказания его будущего поведения. Кроме того, регулятор вычисляет сигнал управления для оптимизации поведения объекта на заданном интервале времени.

Управление с предсказанием строится на принципе удаляющегося горизонта, согласно которому нейросетевая модель управляемого процесса предсказывает реакцию объекта управления на определенном интервале времени в будущем. Предсказание используется программой оптимизации для вычисления управляющего сигнала, который минимизирует критерий качества управления [46]:

$$J = \sum_{j=N_1}^{N_2} (y_r(t+j) - y_m(t+j))^2 + \rho \sum_{j=1}^{N_u} (u'(t+j-1) - u'(t+j-2))^2,$$

где константы N_1 , N_2 и N_u задают пределы, внутри которых вычисляются ошибка слежения и мощность управляющего сигнала. Переменная u' описывает пробный управляющий сигнал, y_r и y_m – желаемая и реальная реакции модели. Величина ρ определяет вклад мощности управления.

Структурная схема, приведенная на рис. 4.31, иллюстрирует процесс управления с предсказанием.

Регулятор состоит из нейросетевой модели и блока оптимизации. Блок оптимизации вычисляет значения u' , которые минимизируют критерий качества управления, а соответствующий сигнал управляет процессом.

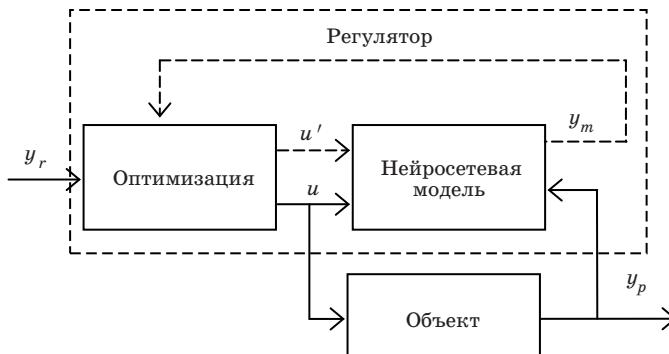


Рис. 4.31. Структурная схема регулятора с предсказанием

Блок NN Predictive Controller находится в библиотеке инструментов MatLab в папке Neural Network Blockset/Control Systems (см. Приложение 1).

При управлении на основе *авторегрессии со скользящим средним* регулятор представляет собой достаточно простую реконструкцию модели управляемого процесса. Такой регулятор требует наименьшего объема вычислений. Вычисления в реальном времени связаны только с реализацией нейронной сети. Недостаток метода состоит в том, что модель процесса должна быть задана в канонической форме пространства состояния, что может приводить к вычислительным погрешностям.

Принципиальная идея NARMA-L2 (Nonlinear Autoregressive – Moving Average (нелинейная авторегрессия со скользящим средним)) заключается в представлении нелинейной системы в виде линейной модели [47].

Модель NARMA имеет вид

$$y(k+d) = N[y(k), y(k-1), \dots, y(k-n+1), u(k), u(k-1), \dots, u(k-n+1)],$$

где $y(k)$ – выход модели; d – число тактов предсказания; $u(k)$ – вход модели.

На стадии идентификации необходимо обучить ИНС аппроксимировать нелинейную функцию N . Эта идентификационная процедура использует NN Predictive Controller.

Если требуется обеспечить соответствие выхода системы эталонному выходу $y(k+d) = y_r(k+d)$, то следующий шаг заключается в разработке нелинейного контроллера в форме

$$u(k) = G[y(k), y(k-1), \dots, y(k-n+1), y_r(k+d), u(k-1), \dots, u(k-m+1)].$$

Для того чтобы ИНС аппроксимировала функцию G с минимальной среднеквадратической ошибкой, необходимо использовать динамический алгоритм ее обратного распространения, что приводит к необходимости чрезмерного объема вычислений.

Решение, предложенное в [48], предполагает использование приближенной модели с выделенной составляющей управления. Такая модель, именуемая NARMA-L2, имеет вид

$$\begin{aligned} \hat{y}(k+d) &= f[y(k), y(k-1), \dots, y(k-n+1), u(k-1), \dots, u(k-m+1)] + \\ &+ g[y(k), y(k-1), \dots, y(k-n+1), u(k-1), \dots, u(k-m+1)]u(k). \end{aligned}$$

Преимущество этой формы состоит в том, что теперь текущее управление можно вычислить непосредственно, если известны же-

ляемая траектория y_r , предыстория управления $\{u(k-1), \dots, u(k-m+1)\}$, а также предшествующие и текущее значения выхода $\{y(k), \dots, y(k-n+1)\}$:

$$u(k) = \frac{y_r(k+d) - f[y(k), y(k-1), \dots, y(k-n+1), u(k-1), \dots, u(k-m+1)]}{g[y(k), y(k-1), \dots, y(k-n+1), u(k-1), \dots, u(k-m+1)]}.$$

В этом уравнении $u(k)$ зависит от текущего значения $y(k)$, поэтому непосредственное его использование затруднительно. Однако уравнение можно модифицировать следующим образом:

$$u(k+1) = \frac{y_r(k+d) - f[y(k), y(k-1), \dots, y(k-n+1), u(k-1), \dots, u(k-m+1)]}{g[y(k), y(k-1), \dots, y(k-n+1), u(k-1), \dots, u(k-m+1)]},$$

где параметр предсказания d должен удовлетворять условию $d \geq 2$.

Общая структура системы с регулятором NARMA-L2 имеет вид, приведенный на рис. 4.32.

На рис. 4.33 показана реализация регулятора NARMA-L2 в MatLab.

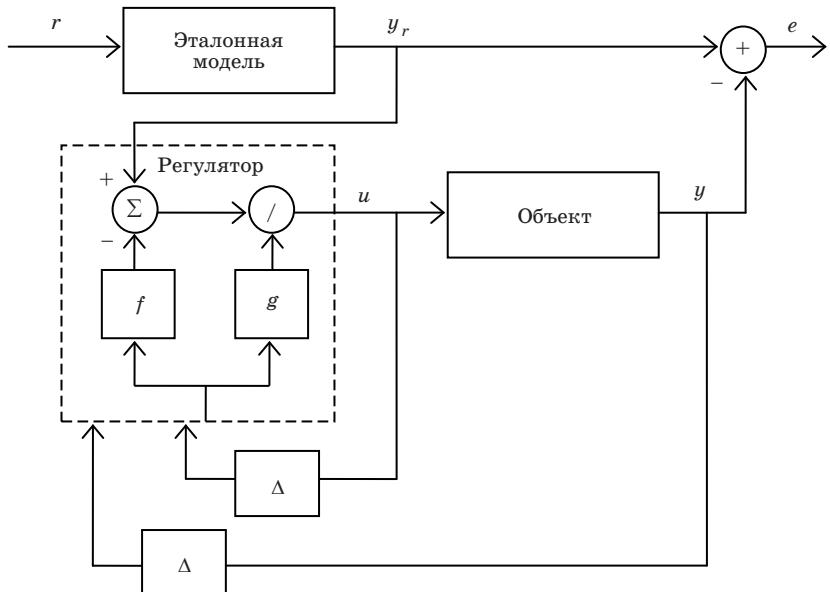


Рис. 4.32. Структура регулятора на основе авторегрессии

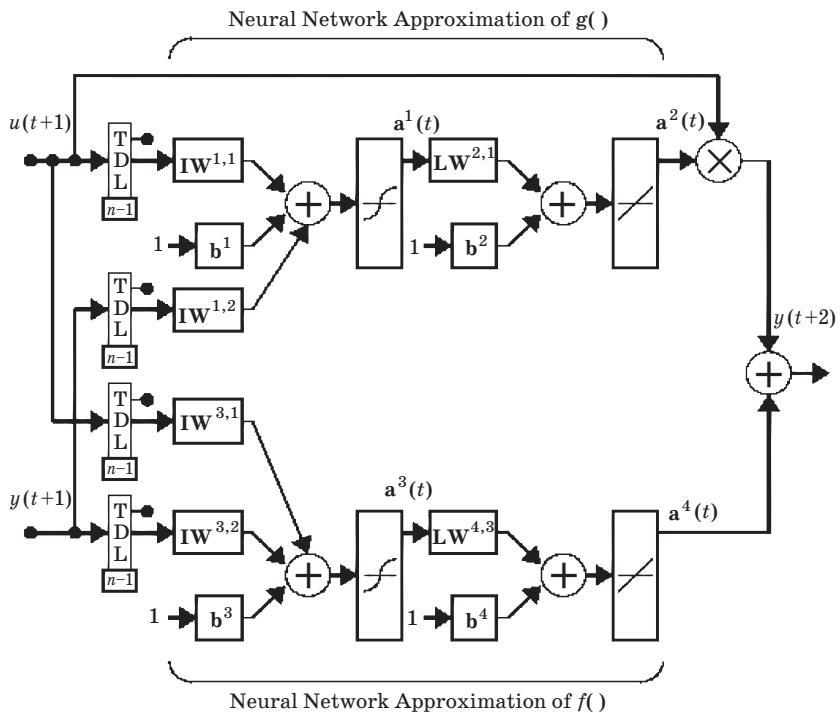


Рис. 4.33. NARMA-L2 регулятор в MatLab

Блок NARMA-L2 Controller находится в библиотеке инструментов MatLab в папке Neural Network Blockset/Control Systems (см. Приложение 1).

При управлении на основе эталонной модели регулятор – это нейронная сеть, которая должна обеспечивать отслеживание объектом заданного эталонного процесса. При этом модель управляемого процесса активно используется на этапе настройки параметров регулятора. Требуемый объем вычислений сравним с предыдущим вариантом. Однако здесь обучение основано на динамическом варианте обратного распространения ошибки и является достаточно сложным. Достоинством регуляторов на основе эталонной модели является их применимость к разным классам управляемых объектов.

Проектирование нейронных регуляторов с эталонной моделью осуществляется в два этапа (рис. 4.34).

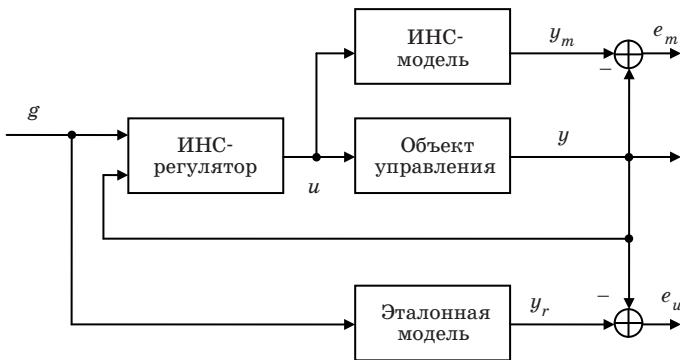


Рис. 4.34. Нейроуправление с эталонной моделью

На первом этапе производятся идентификация динамической системы с помощью наборов входных и соответствующих им выходных величин, разработка архитектуры нейронной модели динамической системы и настройка ее параметров. Полученная ИНС с заданной точностью воспроизводит поведение динамической системы и используется затем для синтеза нейронного регулятора.

На втором этапе осуществляются поиск архитектуры нейрорегулятора и такая его настройка, чтобы поведение системы с заданной точностью соответствовало поведению эталонной модели.

Блок Model Reference Controller также находится в библиотеке инструментов MATLAB в папке Neural Network Blockset/Control Systems (см. Приложение 1).

Вопросы для самопроверки

1. В чем заключается задача идентификации динамического звена?
2. С какой целью вводятся линии задержки на входе ИНС ПР при решении задачи идентификации?
3. Как оценить необходимое число линий задержки?
4. В чем смысл утверждения, что нейросетевая модель является «черным ящиком»?
5. Может ли нейронная сеть заменить человека-оператора или существующий регулятор динамического объекта?
6. Чем нейросетевой эмулятор отличается от краткосрочного предиктора?
7. В чем заключается задача синтеза обратной модели объекта?

8. Почему задача синтеза обратной модели обычно не может быть решена точно?
9. Каковы особенности варианта аналитического расчета параметров регулятора для электромотора?
10. Какой нейронной сети соответствует ПИД-регулятор?
11. Какая структура соответствует дискретной передаточной функции динамического объекта?
12. Что такое прямые и непрямые методы использования нейронных регуляторов?
13. Какие нейроконтроллеры называются гибридными?
14. Какой вид имеет модель нелинейной авторегрессии для одномерного объекта?
15. Каковы принципы прямого инверсного нейроуправления?
16. Каковы принципы непрямого инверсного нейроуправления?
17. Что представляет собой специализированное инверсное нейроуправление?
18. Как работает схема обратного распространения ошибки во времени?
19. Какие модели нейроуправления реализованы в MatLab?
20. Какие этапы проектирования нейрорегуляторов используются в MatLab?
21. Как работает схема нейроуправления с предсказанием?
22. Как работает схема нейроуправления на основе авторегрессии со скользящим средним?
23. Какую структуру имеет регулятор NARMA-L2 в MatLab?
24. Как происходит нейроуправление на основе эталонной модели?

5. РАДИАЛЬНЫЕ НЕЙРОННЫЕ СЕТИ

5.1. Структура радиальной нейронной сети

Радиальные нейронные сети (Radial Basis Function – RBF) были предложены в работе [49]. Это двухслойные сети (без учета распределительного входного слоя) прямого распространения (рис. 5.1).

Радиальная нейронная сеть подобно многослойной сети прямого распространения является универсальным аппроксиматором.

Радиально-базисные функции – специальный класс функций, значение которых монотонно уменьшается (увеличивается) с увеличением расстояния от центра.

Все веса радиально-базисного слоя полагаются равными единице, и работу i -го нейрона RBF-слоя можно описать формулой

$$f_i(X) = \varphi(\|X - C_i\|),$$

где C_i – вектор центра активационной RBF-функции нейрона: $X, C \in R^n$. Таким образом, входной вектор и вектор центра имеют одинаковую размерность.

В качестве радиальной базисной функции φ обычно используется гауссова функция

$$\varphi(\|X - C_i\|) = \exp\left(-\frac{\|X - C_i\|^2}{2\sigma_i^2}\right),$$

где σ – ширина «окна» активационной функции.

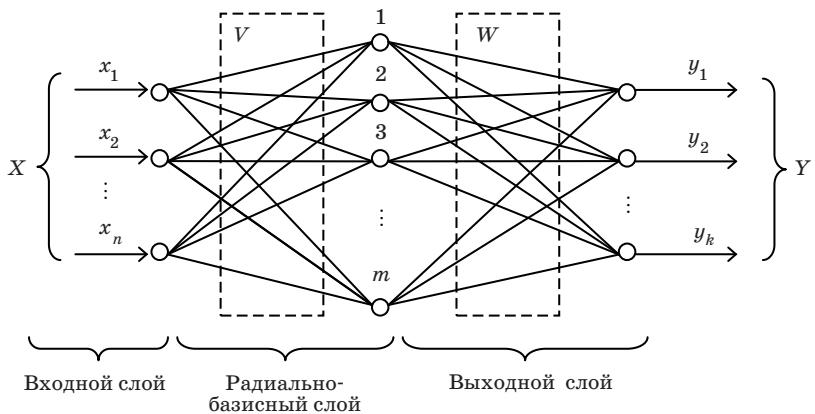


Рис. 5.1. Структура RBF-сети

В качестве метрики обычно используется евклидово расстояние

$$\|X - C_i\| = \sqrt{(x_1 - c_{i1})^2 + (x_2 - c_{i2})^2 + \dots + (x_n - c_{in})^2}.$$

Таким образом, i -й нейрон скрытого слоя определяет сходство между входным вектором X и эталонным вектором C_i .

На рис. 5.2 приведена гауссова функция одной переменной при $c = 0$ и $\sigma = 1$.

Как следует из рис. 5.2, функция активации RBF-нейрона принимает большие значения лишь в тех случаях, когда входной образ находится вблизи центра нейрона. Вне области, «покрытой» обозами обучающей последовательности, сеть формирует лишь малые значения на своих выходах.

Возможны и другие варианты активационной функции. Например,

$$\varphi(\|X - C_i\|) = \|X - C_i\| - \text{линейная функция},$$

$$\varphi(\|X - C_i\|) = \|X - C_i\|^3 - \text{кубическая функция},$$

$$\varphi(\|X - C_i\|) = (\|X - C_i\|^2 + \sigma^2)^{1/2} - \text{мультиквадратическая функция}.$$

График мультиквадратической функции одной переменной приведен на рис. 5.3.

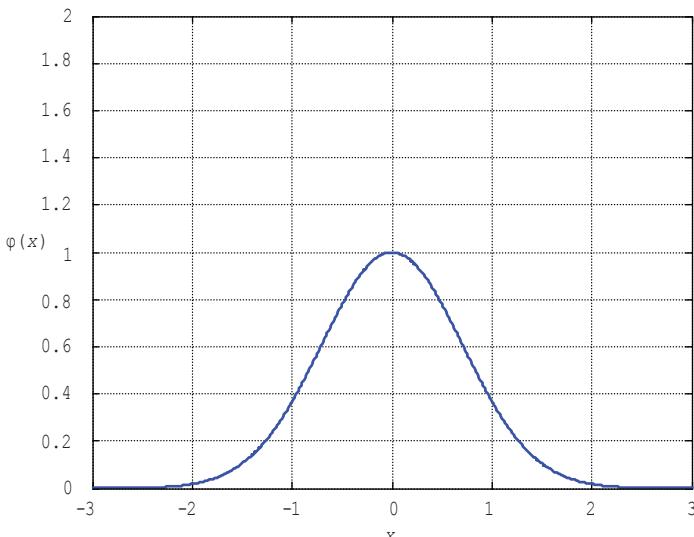


Рис. 5.2. График гауссовой функции

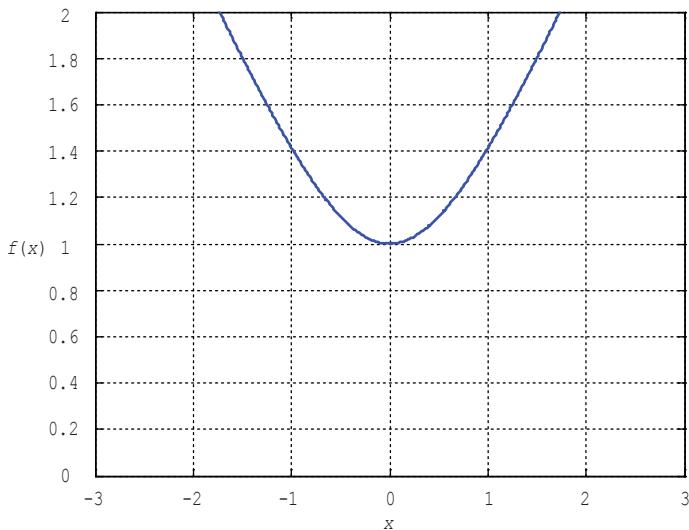


Рис. 5.3. График мультиквадратической функции

Главное отличие RBF-сетей от обычных многослойных сетей прямого распространения состоит в функции нейронов скрытого слоя. В обычной многослойной сети каждый нейрон рабочего слоя реализует в многомерном пространстве гиперплоскость, а RBF-нейрон – гиперсферу (рис. 5.4).

В проблемах, где данные близки к образованию круговой симметрии, это позволяет уменьшить число нейронов.

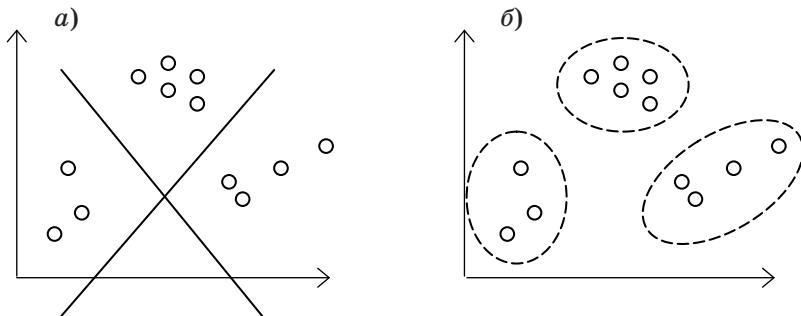


Рис. 5.4. Принципы классификации с помощью многослойного персептрана (а) и RBF-сети (б)

Нейроны выходного слоя имеют линейную активационную функцию. Их роль сводится исключительно к взвешенному суммированию сигналов, генерируемых нейронами рабочего слоя:

$$y_j = \sum_{i=1}^m w_{ij} f_i(X), \quad j = \overline{1, k}.$$

Число нейронов выходного слоя определяется характером представления выходных данных.

5.2. Расчет параметров радиальной нейронной сети

Рассмотрим простой вариант определения весов RBF-сети.

Пусть RBF-сеть имеет k входов и один выход (рис. 5.5). Выберем число рабочих нейронов $m = n$, где n – число обучающих пар, заданных набором $\{(X_1, y_1), (X_2, y_2), \dots, (X_n, y_n)\}$, где $X_i = [x_{i1}, x_{i2}, \dots, x_{ik}]^T$.

Для того чтобы каждый нейрон реагировал на «свой» вектор из обучающего набора, полагаем

$$C_i = X_i.$$

Окна активационной функции σ выбирают достаточно большими, но так, чтобы они не перекрывались в пространстве входных сигналов.

Требуется найти такие весовые коэффициенты W , чтобы для каждого входного вектора из обучающего набора выполнялось

$$h(X_i) = y_i.$$

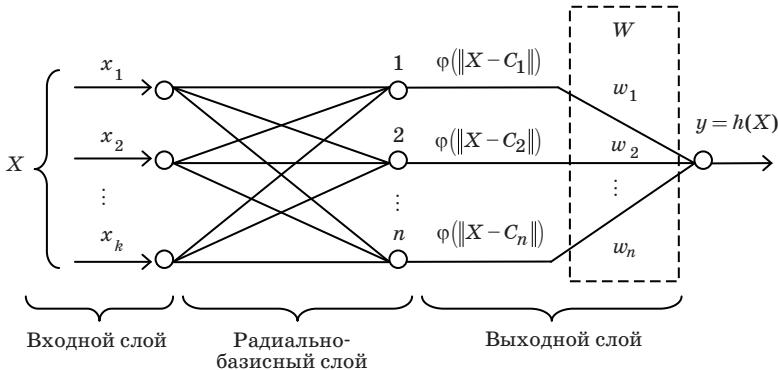


Рис. 5.5. Структура RBF-сети со скалярным выходом

Для первого входного вектора из обучающего набора можно записать

$$h(X_1) = \sum_{i=1}^n w_i f_i(X_1) = w_1 f_1(X_1) + w_2 f_2(X_1) + \dots + w_n f_n(X_1).$$

Для всех n входных векторов при правильном выборе W должно выполняться

$$\begin{bmatrix} f_1(X_1) & f_2(X_1) & \dots & f_n(X_1) \\ f_1(X_2) & f_2(X_2) & \dots & f_n(X_2) \\ \vdots & \vdots & \vdots & \vdots \\ f_1(X_n) & f_2(X_n) & \dots & f_n(X_n) \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}.$$

Вводим обозначения для матриц:

$$FW = Y.$$

Тогда

$$W = F^{-1}Y. \quad (5.1)$$

Последняя формула позволяет рассчитать веса RBF-сети с одним выходом при числе нейронов скрытого слоя, равном числу обучающих пар.

Далее будет показано, что аналогичный результат может быть получен при произвольном числе нейронов выходного слоя RBF-сети, если число обучающих пар равно числу нейронов скрытого слоя.

Пусть выходной слой содержит p нейронов, так что вектор выхода имеет вид

$$Y_i = [y_{i1}, y_{i2}, \dots, y_{ip}]^T.$$

Определим веса нейронов выходного слоя w_{ij} , $i = \overline{1, n}$, $j = \overline{1, p}$. Для этого сети предъявляется весь набор шаблонов, так что для всех n входных векторов можно записать

$$\begin{bmatrix} f_1(X_1) & f_2(X_1) & \dots & f_n(X_1) \\ f_1(X_2) & f_2(X_2) & \dots & f_n(X_2) \\ \vdots & \vdots & \vdots & \vdots \\ f_1(X_n) & f_2(X_n) & \dots & f_n(X_n) \end{bmatrix} \begin{bmatrix} w_{11} & w_{21} & \dots & w_{1p} \\ w_{21} & w_{22} & \dots & w_{2p} \\ \vdots & \vdots & \vdots & \vdots \\ w_{n1} & w_{n2} & \dots & w_{np} \end{bmatrix} = \begin{bmatrix} y_{11} & y_{12} & \dots & y_{1p} \\ y_{21} & y_{22} & \dots & y_{2p} \\ \vdots & \vdots & \vdots & \vdots \\ y_{n1} & y_{n2} & \dots & y_{np} \end{bmatrix}.$$

Строки матрицы F соответствуют выходам нейронов скрытого слоя для каждого входного шаблона. Столбцы матрицы W соответствуют весовым коэффициентам нейронов выходного слоя. Строки матрицы Y описывают выходы нейронов второго (выходного) слоя для каждого входного вектора.

Матрица весов W может быть рассчитана согласно (5.1).

Таким образом, веса RBF-сети могут быть рассчитаны по тренировочным шаблонам. Если обучающие пары выбраны удачно, то сеть будет успешно выполнять интерполяцию и порождать близкие выходные сигналы для близких входных сигналов.

Однако в практических задачах условие $m = n$ обычно непримлемо, поскольку требует использования очень большого числа нейронов. Кроме того, сеть становится чрезмерно чувствительной к шумам в обучающей выборке. Таким образом, обычно $m \ll n$ (число нейронов скрытого слоя m меньше числа обучающих пар n), и требуется найти приближенное решение задачи аппроксимации.

Процесс подбора приближенного значения весов может рассматриваться как задача минимизации целевой функции, описывающей ошибку выхода сети.

Для оптимального выбора коэффициентов RBF-сети может быть использован метод наименьших квадратов.

Рассмотрим RBF-сеть с одним выходным и m скрытыми нейронами:

$$y = h(X) = \sum_{i=1}^m w_i f_i(X). \quad (5.2)$$

Пусть необходимо аппроксимировать зависимость, заданную множеством вход-выходных данных (обучающая выборка):

$$\{(X_1, y_1), (X_2, y_2), \dots, (X_n, y_n)\}.$$

Для качественной аппроксимации требуется минимизировать ошибку выхода сети, заданную формулой

$$E = \sum_{i=1}^n (h(X_i) - y_i)^2. \quad (5.3)$$

Рассмотрим производную (5.3)

$$\frac{\partial E}{\partial w_j} = 2 \sum_{i=1}^n (h(X_i) - y_i) \frac{\partial h(X_i)}{\partial w_j}.$$

В соответствии с (5.2)

$$\frac{\partial h(X_i)}{\partial w_j} = f_j(X_i),$$

следовательно,

$$\frac{\partial E}{\partial w_j} = 2 \sum_{i=1}^n (h(X_i) - y_i) f_j(X_i).$$

В точке оптимума

$$\frac{\partial E}{\partial w_j} = \sum_{i=1}^n (h(X_i) - y_i) f_j(X_i) = 0,$$

или

$$\sum_{i=1}^n f_j(X_i) h(X_i) = \sum_{i=1}^n f_j(X_i) y_i.$$

Обозначим

$$F_j = \begin{bmatrix} f_j(X_1) \\ f_j(X_2) \\ \vdots \\ f_j(X_n) \end{bmatrix}, \quad H = \begin{bmatrix} h(X_1) \\ h(X_2) \\ \vdots \\ h(X_n) \end{bmatrix}, \quad Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}.$$

Тогда

$$F_j^T H = F_j^T Y, \quad j = \overline{1, m},$$

$$\begin{bmatrix} F_1^T H \\ F_2^T H \\ \vdots \\ F_m^T H \end{bmatrix} = \begin{bmatrix} F_1^T Y \\ F_2^T Y \\ \vdots \\ F_m^T Y \end{bmatrix},$$

$$F^T H = F^T Y,$$

где $F = [F_1 \ F_2 \ \cdots \ F_m].$

Поскольку

$$H = \begin{bmatrix} h(X_1) \\ h(X_2) \\ \vdots \\ h(X_n) \end{bmatrix} = \begin{bmatrix} \sum_{j=1}^m w_j f_j(X_1) \\ \sum_{j=1}^m w_j f_j(X_2) \\ \vdots \\ \sum_{j=1}^m w_j f_j(X_n) \end{bmatrix} = \begin{bmatrix} f_1(X_1) & f_2(X_1) & \cdots & f_m(X_1) \\ f_1(X_2) & f_2(X_2) & \cdots & f_m(X_2) \\ \vdots & \vdots & \ddots & \vdots \\ f_1(X_n) & f_2(X_n) & \cdots & f_m(X_n) \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_m \end{bmatrix} = FW,$$

можно записать

$$F^T FW = F^T Y,$$

и окончательно

$$W = (F^T F)^{-1} F^T H = F^+ H,$$

где F^+ – псевдообратная матрица.

Пример 5.1. Пусть задан набор из трех пар точек ($p = 3$):

$$\{(0, 9; 1), (2, 1; 1, 9), (3, 1; 3)\}.$$

Требуется аппроксимировать эту зависимость функцией

$$y(x) = w_1 h_1(x) + w_2 h_2(x),$$

где $h_1(x)$ и $h_2(x)$ – выходы радиально-базисных нейронов, заданных в виде

$$h_1(x) = \exp(-(x - 1,5)^2), \quad h_2(x) = \exp(-(x - 2,5)^2).$$

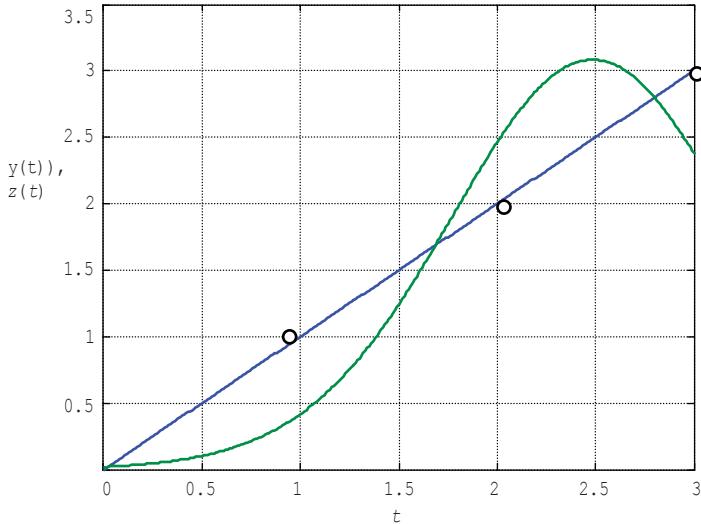


Рис. 5.6. Аппроксимация функции, заданной тремя точками

Требуется найти неизвестные коэффициенты w_1 и w_2 :

$$F = \begin{bmatrix} h_1(x_1) & h_2(x_1) \\ h_1(x_2) & h_2(x_2) \\ h_1(x_3) & h_2(x_3) \end{bmatrix} = \begin{bmatrix} 0,6977 & 0,0773 \\ 0,6977 & 0,8521 \\ 0,0773 & 0,6977 \end{bmatrix}, \quad Y = \begin{bmatrix} 1 \\ 1,9 \\ 3 \end{bmatrix},$$

$$F^T F = \begin{bmatrix} 0,9795 & 0,7024 \\ 0,7024 & 1,2188 \end{bmatrix},$$

$$(F^T F)^{-1} = \begin{bmatrix} 1,7398 & -1,0026 \\ -1,0026 & 1,3982 \end{bmatrix},$$

$$W = (F^T F)^{-1} F^T Y = \begin{bmatrix} 0,1244 \\ 3,0373 \end{bmatrix}.$$

На рис. 5.6 приведен результат аппроксимации. Полученное качество очевидно невысоко по сравнению с линейной функцией $z(x)$, которая обеспечивает в данном случае наилучшую аппроксимацию исходных данных.

Таким образом, если параметры гауссовой функции (центр и радиус) заданы, то задача нахождения весов выходного слоя RBF-сети может быть решена методами линейной алгебры – методом псевдообратных матриц.

5.3. Обучение радиальной нейронной сети

В большинстве случаев выбор центров и радиусов активационных функций представляет собой сложную задачу, требующую анализа обучающей выборки. При этом можно выделить три взаимозависимые проблемы, связанные с выбором:

- числа нейронов скрытого слоя;
- центров и радиусов нейронов;
- весов выходного слоя.

Для решения первой задачи может быть использована, например, вершинная кластеризация [50]. Полученное число кластеров примерно соответствует необходимому числу нейронов скрытого слоя. Центры кластеров можно использовать для первоначального задания центров активационных функций нейронов.

Для решения второй и третьей задач можно использовать один из методов глобальной оптимизации, например, генетический алгоритм (ГА) [51].

Использование ГА предполагает кодирование параметров строек действительных чисел – хромосомой. Множество хромосом образуют популяцию. Каждая хромосома снабжается оценкой пригодности, т. е. соответствия критерию, который может описываться, например, формулой (5.3).



Рис. 5.7. Генетическая оптимизация параметров RBF-сети

Алгоритм оптимизации параметров сети с помощью ГА приведен на рис. 5.7. Работа алгоритма заканчивается, когда значение ошибки достигает приемлемой величины.

5.4. Радиальные нейронные сети в MatLab

Структура радиального базисного нейрона с R входами, используемая в MatLab, приведена на рис. 5.8.

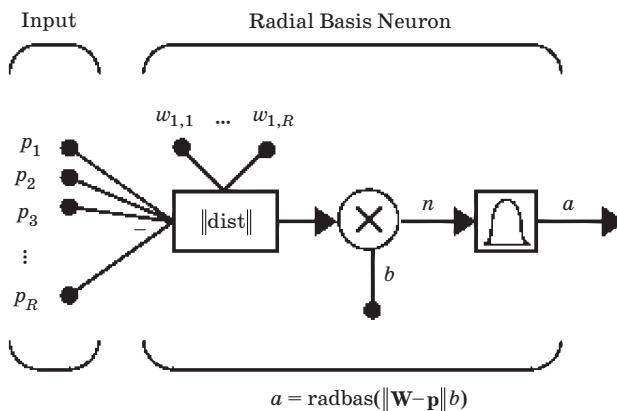


Рис. 5.8. Радиальный базисный нейрон

Функция dist вычисляет евклидово расстояние между вектором входа P и вектором весов W нейрона. Смещение b служит для коррекции чувствительности нейрона.

Описание его блока активации представлено на рис. 5.9.

Выход нейрона описывается выражением

$$a = \text{radbas}(n) = e^{-n^2}.$$

Таким образом, выход нейрона равняется точно единице, если векторы P и W идентичны.

Структура RBF-сети представлена на рис. 5.10. Она содержит S^1 нейронов рабочего слоя и S^2 нейронов выходного, имеющих линейные активационные функции.

Операции создания RBF-сети в MatLab оформлены в виде функций `newrbe` и `newrb`. Первая позволяет построить радиальную базисную сеть с нулевой ошибкой, вторая – управлять числом нейронов в скрытом слое.

Функция `newrb` создает RBF-сеть, используя итеративную процедуру, которая добавляет по одному нейрону на каждом шаге. Нейроны добавляются к скрытому слою до тех пор, пока сумма ква-

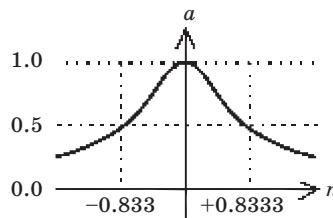


Рис. 5.9. Активационная функция нейрона

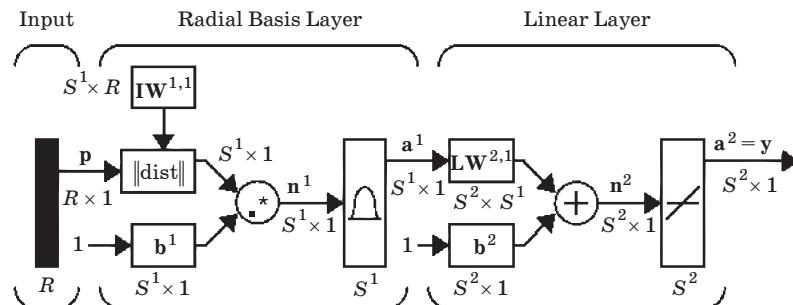


Рис. 5.10. Структура RBF-сети в MatLab

драгов ошибок не станет меньше заданного значения или не будет использовано максимальное число нейронов.

Рассмотрим пример использования функции newrb:

```
>> P = [0 1 2 3 4 5 6 7 8 9 10]; % определение обучающих данных  
>> T = [0 1 2 3 4 3 2 1 2 3 4]; % на входе и выходе  
>> GOAL = 0.01; % допустимый предел ошибки  
>> SPREAD = 1; % ширина «окна» RBF-функций  
>> net = newrb(P,T,GOAL,SPREAD); % создание RBF-сети  
>> figure(1), clf, % создание и очистка окна графика  
>> plot(P,T,'sr','MarkerSize',8,'MarkerFaceColor','y')  
>> hold on; % хранение изображения  
>> X = 0:0.01:10; % задание непрерывного входа RBF-сети.  
>> Y = sim(net,X); % моделирование работы сети  
>> plot(X,Y,'LineWidth',2), grid on % график работы RBF-сети.
```

На рис. 5.11 приведены графики моделирования RBF-сети. Узнать, сколько образовалось радиальных нейронов, можно с помощью команды

```
>> net.layers{1}.size  
ans = 10
```

При использовании функции newrb получившееся число нейронов зависит от заданного предела ошибки и ширины «окна» ак-

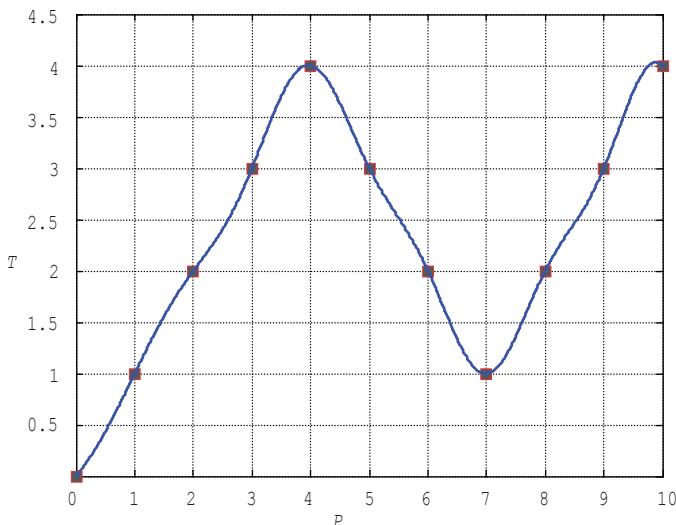


Рис. 5.11. Пример моделирования RBF-сети в MatLab

тивационной функции. На рис. 5.12 приведен вариант, когда параметр $GOAL = 0,1$. В этом случае сеть содержит семь нейронов, и качество аппроксимации ниже. На рис. 5.13 показан вариант моделирования при уменьшении зоны взаимодействия RBF-нейронов ($SPREAD = 0,5$).

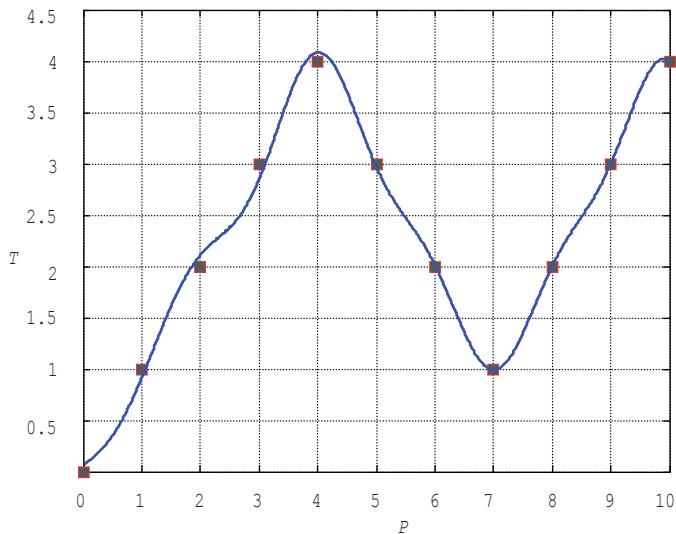


Рис. 5.12. RBF-сеть при снижении требований к точности

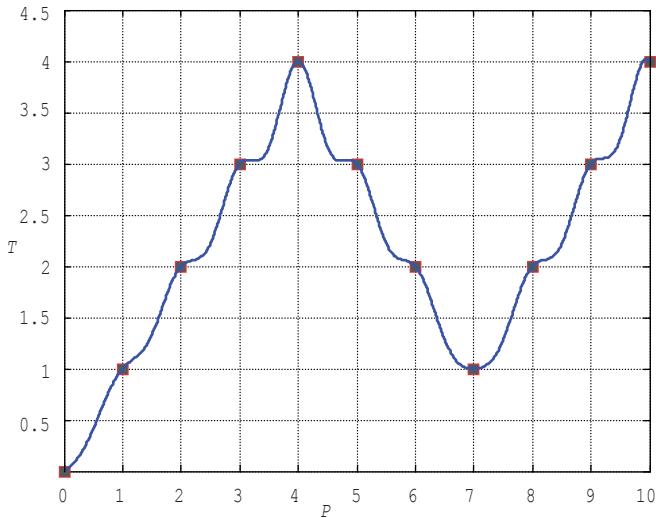


Рис. 5.13. RBF-сеть при уменьшении «окна» активационной функции

Рассмотрим еще один пример использования функции newrb.
Пусть требуется аппроксимировать по точкам функцию humps из библиотеки MatLab:

```
>> x = 0:.05:2; y=humps(x);
>> P=x; T=y;
>> goal=0.02; spread= 0.1;
>> net1 = newrb(P,T,goal,spread);
>> A= sim(net1,P);
>> plot(x,y,P,A)
```

Графики исходной функции и ее аппроксимации практически совпадают.

В команде создания RBF-сети с нулевой ошибкой параметр GOAL не указывается. Например (рис. 5.14),

```
>> P =[0 1 2 3 4 5 6 7 8 9 10];
>> T =[2 2 2 2 4 4 4 2 2 2 2];
>> SPREAD = 0.5;
>> net = newrbe(P,T,SPREAD);
>> figure(1), clf,
>> plot(P,T,'sr','MarkerSize',8,'MarkerFaceColor','y')
>> hold on;
>> X = 0:0.01:10; Y = sim(net,X); plot(X,Y,'LineWidth',2), grid on
>>
```

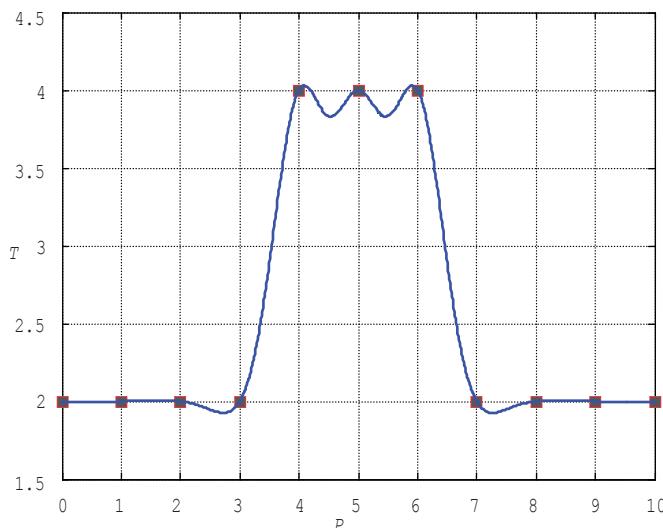


Рис. 5.14. Пример использования RBF-сети с нулевой ошибкой

В RBF-сети с нулевой ошибкой число радиальных нейронов выбирается равным числу элементов обучающего множества, в этом заключается трудность использования подобной сети при большой обучающей выборке.

В заключение рассмотрим моделирование системы Ван дер Поля (см. рис. 3.19). В данном случае потребуется такая последовательность команд:

```
>> [t,z]=sim('Van_der_Pol',10);
>> P = t';
>> T = z';
>> net2=newrb(P,T,0.01);
>> A= sim(net1,P);
>> figure(1); plot(P,A)
>> grid
```

Полученный результат совпадает с результатом, приведенным на рис. 3.21.

В MatLab описаны также разновидности RBF-сетей – сети GRNN и PNN.

Нейронная сеть GRNN (Generalized Regression Neural Network) предназначена для решения задач обобщенной регрессии, анализа временных рядов и аппроксимации функций. Она имеет радиально-базисный и специальный линейный слой (рис. 5.15). Характерной особенностью этих сетей является высокая скорость обучения.

Радиально-базисный слой функционирует так же, как у RBF-сети с нулевой ошибкой. Число нейронов этого слоя равно числу элементов Q обучающего множества. В качестве начального приближения для матрицы весов выбирается массив входов P , так что

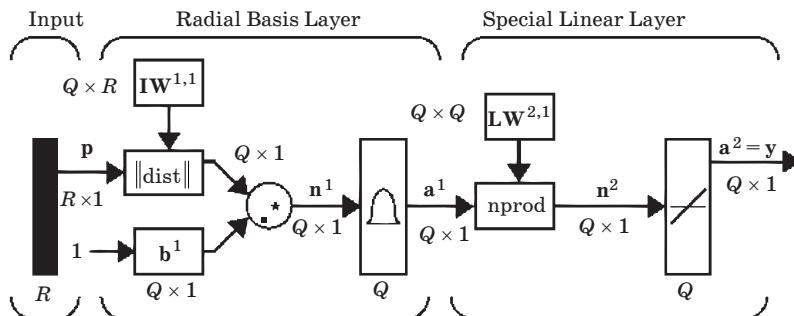


Рис. 5.15. Структура GRNN-сети

если вектор веса нейрона равен транспонированному вектору входа, то выход функции активации равен единице.

Второй слой – это линейный слой с числом нейронов, также равным R , причем в качестве начального приближения для матрицы весов $LW\{2,1\}$ выбирается массив выходов T .

Блок normprod служит для вычисления нормированного скалярного произведения строки массива весов выходного слоя и вектора выхода радиально-базисного слоя.

Таким образом, если вектор входа близок к одному из векторов входа P_i из обучающего множества, то значение i -го выхода радиально-базисного слоя будет близко к единице, и i -й выход второго слоя будет близок к T_i .

Если параметр влияния SPREAD мал, то диапазон входных значений, на который реагирует нейрон скрытого слоя, оказывается малым. С увеличением параметра SPREAD уже несколько нейронов реагирует на значения вектора входа. Тогда на выходе сети формируется вектор, соответствующий среднему нескольких целевых векторов, соответствующих входным векторам обучающего множества, близких к данному вектору входа.

Пусть

```
>> P = [0 1 2 3 4 5 6 7 8 9 10];
T = [2 2 2 2 4 4 4 2 2 2 2];
SPREAD = 0.5;
net = newgrnn(P,T);
figure(1), clf,
plot(P,T,'sr','MarkerSize',8,'MarkerFaceColor','y')
hold on;
X = 0:0.01:10; Y = sim(net,X); plot(X,Y,'LineWidth',2), grid on
>> net.layers{1}.size
ans = 11
```

Результат аппроксимации (рис. 5.16) существенно отличается от приведенного на рис. 5.14 (при одинаковом обучающем множестве).

Нейронные сети PNN (Probabilistic Neural Network) предназначены для решения вероятностных задач, в том числе и задач классификации.

В такой сети первый слой содержит радиально-базисные функции. Второй слой называется *слоем конкуренции* (рис. 5.17).

Радиально-базисный слой функционирует аналогично соответствующему слою GRNN-сети.

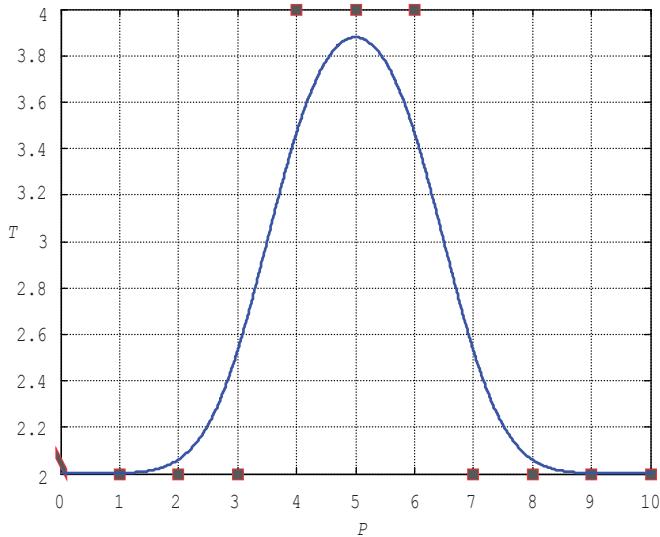


Рис. 5.16. Пример аппроксимации с помощью GRNN-сети

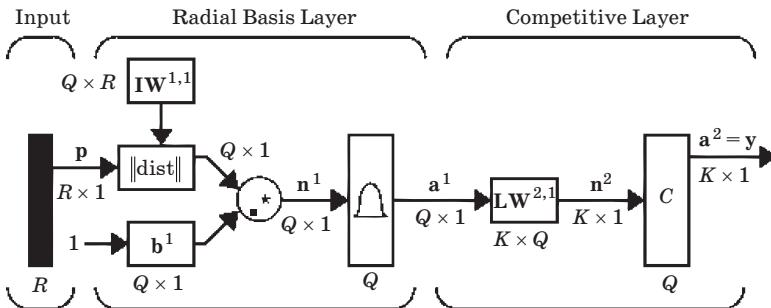


Рис. 5.17. Структура PNN-сети

Слой конкуренции вычисляет вероятность принадлежности вектора к тому или иному классу, а затем выбирает класс с наибольшей вероятностью.

Обучающее множество содержит \$Q\$ пар векторов вход-цель. Имеется \$K\$ классов, к которым может принадлежать входной вектор. Пусть, например, рассматриваются семь векторов и три класса принадлежности:

`>> P = [0 0;1 1;0 3;1 4;3 1;4 1;4 3]`

`Tc = [1 1 2 2 3 3 3]`

```

P =
  0   1   0   1   3   4   4
  0   1   3   4   1   1   3
Tc =
  1   1   2   2   3   3   3

```

На основании этих данных может быть образована матрица связности T размером $K \times Q$, состоящая из нулей и единиц, строки которой соответствуют классам принадлежности, а столбцы – векторам входа. Таким образом, если элемент $T(i, j)$ матрицы связности равен единице, то это означает, что j -й входной вектор принадлежит к классу i .

В разреженной форме эта матрица имеет вид

```
>> T = ind2vec(Tc)
```

```

T =
  (1,1)    1
  (1,2)    1
  (2,3)    1
  (2,4)    1
  (3,5)    1
  (3,6)    1
  (3,7)    1

```

В полной форме

```
>> T=full(T)
```

```

T =
  1   1   0   0   0   0   0
  0   0   1   1   0   0   0
  0   0   0   0   1   1   1

```

Массивы P и T задают обучающее множество, что позволяет выполнить формирование сети

```

>> net = newpnn(P,T);
Y = sim(net,P);

```

Для проверки работы сети можно подавать на ее вход произвольные векторы. Например,

```

>> P = [0.1 0.5;1.2 1.3;4 4]
P =
  0.1000  1.2000  4.0000
  0.5000  1.3000  4.0000
>> Y = sim(net,P)

```

```

Y =
(1,1)    1
(1,2)    1
(3,3)    1
>> Yc = vec2ind(Y)
Yc =
1   1   3

```

Таким образом, два первых вектора принадлежат к классу 1, а третий вектор – к классу 3.

Рассмотренные варианты радиально-базисных сетей используют фундаментальный принцип нелинейного преобразования входного пространства в скрытое пространство более высокой размерности. В пространстве более высокой размерности с большей вероятностью возможна линейная разделимость входного пространства, а также повышается возможность гладкой аппроксимации входных данных.

Простая структура RBF-сетей, содержащих только один слой скрытых нейронов, делает возможным прямой расчет весов сети. В этом заключается преимущество данных сетей по сравнению с другими типами НС, которые используют трудоемкие алгоритмы обучения. Впрочем, RBF-сеть также может дообучиваться.

5.5. Радиальные нейронные сети и нечеткие системы

Можно показать, что работа RBF-сети функционально соответствует работе системы нечеткого логического вывода. Описания математических основ нечетких логических систем и нечетких регуляторов можно найти, например, в [38]. Здесь лишь приведем типовую структуру системы нечеткого логического вывода с n входными переменными (посылками), m правилами и одним заключением. Эту структуру можно представить в виде ИНС прямого распространения, как показано на рис. 5.18 [52].

Узлы входного слоя просто передают входные сигналы узлам 1-го (скрытого) слоя, которые вычисляют максимальное отклонение входного сигнала от соответствующей посылки правила.

Веса V описывают посылки правил. Каждый i -й нейрон имеет свой набор весов $\{v_{ji}\}$, $j = \{1, 2, \dots, n\}$, которые соответствуют значениям посылок правила. Векторы X и V обычно нормализуются.

Каждый нейрон 1-го слоя вычисляет сходство входного вектора и своего вектора посылок, в результате чего определяется степень запуска соответствующего правила.

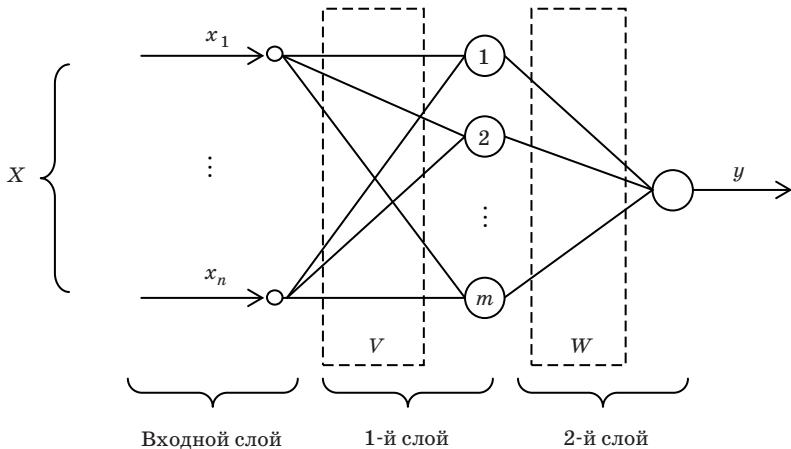


Рис. 5.18. Система нечетких правил как двухслойная ИНС

Для измерения сходства X и V может быть использовано вычисление:

- скалярного произведения векторов:

$$R = XV = \sum_{i=1}^n x_i v_i;$$

- максимального отклонения

$$R = \max(|x_1 - v_2|, |x_2 - v_2|, \dots, |x_n - v_n|);$$

- евклидова расстояния между векторами

$$R = \sqrt{(x_1 - v_1)^2 + (x_2 - v_2)^2 + \dots + (x_n - v_n)^2}.$$

Последний вариант соответствует операции вычисления сходства, выполняемой RBF-нейроном.

Затем полученная величина используется как аргумент функции принадлежности, которая может иметь вид, приведенный на рис. 5.19.

Таким образом, работа 1-го слоя структуры на рис. 5.18, описывающей нечеткую систему вывода, практически соответствует работе радиально-базисного слоя. Все отличия касаются деталей реализации.

Нейрон 2-го слоя структуры, приведенной на рис. 5.18, выполняет операцию дефазификации. Каждый вес вектора W соответ-

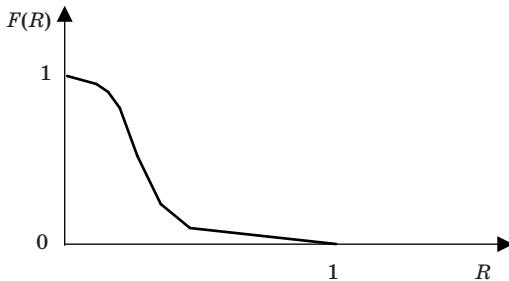


Рис. 5.19. Вариант описания функции принадлежности

ствует заключению правила, и выходной сигнал представляет собой взвешенную сумму выходов нейронов 1-го слоя:

$$y = w_1 \frac{F(R_1)}{\sum_{i=1}^m F(R_i)} + w_2 \frac{F(R_2)}{\sum_{i=1}^m F(R_i)} + \dots + w_m \frac{F(R_m)}{\sum_{i=1}^m F(R_i)} = \frac{w_i \sum_{i=1}^m F(R_i)}{\sum_{i=1}^m F(R_i)}.$$

Если нормализовать выход RBF-сети, то получится точно такой же результат.

Таким образом, RBF-сеть функционально эквивалентна системе нечетких правил.

Вопросы для самопроверки

1. Сколько слоев содержит RBF-сеть?
2. Что представляет собой радиально-базисная функция?
3. Является ли RBF-сеть универсальным аппроксиматором?
4. Каковы возможные варианты радиально-базисных функций?
5. В каких ситуациях нейрон радиально-базисного слоя выдает максимальный сигнал?
6. Как назначаются значения весов радиально-базисного слоя?
7. В чем состоит главное отличие RBF-сетей от обычных многослойных сетей прямого распространения?
8. Какую активационную функцию имеют нейроны выходного слоя RBF-сети?
9. При каком условии можно аналитически рассчитать веса выходного слоя RBF-сети?

10. В каком соотношении находится обычно число нейронов скрытого слоя RBF-сети и число обучающих пар?
11. Какой метод используется для приближенного выбора коэффициентов RBF-сети?
12. В чем суть основных принципов и расчетных формул метода наименьших квадратов для подбора весов RBF-сети?
13. Какие параметры RBF-сети полагаются заданными при использовании метода наименьших квадратов?
14. С какой целью используется кластеризация при обучении RBF-сети?
15. Какие задачи можно решить с помощью генетического алгоритма при обучении RBF-сети?
16. Каковы принципы использования генетического алгоритма при обучении RBF-сети?
17. Какие команды можно использовать в системе MatLab для создания RBF-сетей? Чем отличаются эти команды?
18. Что представляют собой сети GRNN?
19. Что представляют собой сети PNN?
20. В чем заключается сходство функционирования системы нечеткого логического вывода и RBF-сети?

6. МОДЕЛИ АССОЦИАТИВНОЙ ПАМЯТИ

6.1. Нейронная сеть Элмана

Нейронная сеть Элмана [53] относится к классу частично рекуррентных ИНС (рис. 6.1).

Многие практические задачи можно преобразовать в задачи распознавания временных последовательностей. В качестве простого примера рассмотрим функцию XOR. Как уже было показано, для ее реализации требуется использовать двухслойную сеть прямого распространения. Альтернативой является описание функции XOR во временной области как последовательности битов. Например,

1 0 1 0 0 0 0 1 1 1 1 0 1 0 1 ...

Здесь каждый третий бит является логической функцией первых двух битов.

Таким образом, поведение ИНС, реализующей функцию XOR, должно заключаться в предсказании следующего бита последовательности:

Вход: 1 0 1 0 0 0 0 1 1 1 1 0 1 0 1

Выход: 0 1 0 0 0 0 1 1 1 1 0 1 0 1 ?

Для обработки временных последовательностей в работе [53] была предложена структура ИНС, приведенная на рис. 6.2.

Нейронная сеть Элмана состоит из N входов, K нейронов скрытого слоя, охваченных обратными связями через элементы задержки z^{-1} , и M нейронов выходного слоя.

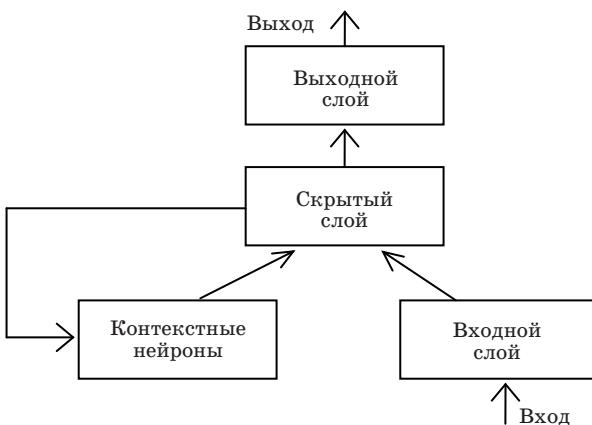


Рис. 6.1. Структура ИНС Элмана

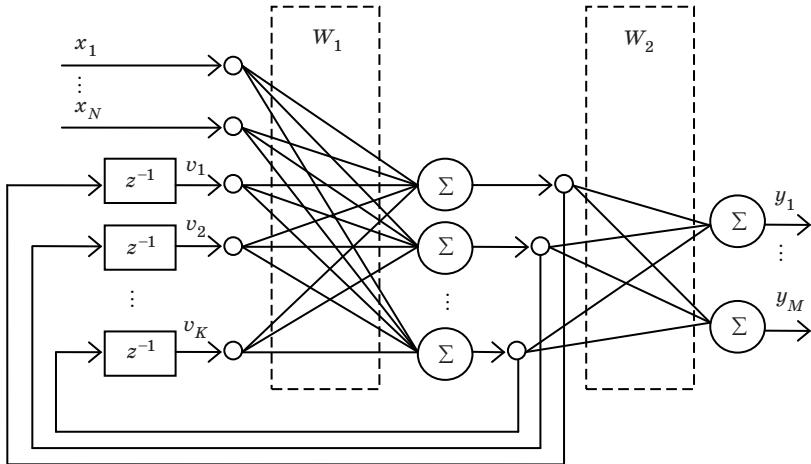


Рис. 6.2. Нейронная сеть Элмана

Вектор состояния входного слоя сети в момент времени kT_0 (T_0 – шаг дискретизации по времени) имеет вид

$$X(k) = [x_0(k), x_1(k), \dots, x_N(k), v_1(k-1), \dots, v_K(k-1)],$$

где $x_0(k), x_1(k), \dots, x_N(k)$ – вектор входного сигнала; $v_1(k-1), \dots, v_K(k-1)$ – состояние нейронов скрытого слоя.

Состояние нейронов скрытого слоя определяется выражением

$$v_i(k) = f_i^1(u_i(k)), \quad u_i(k) = \sum_{j=0}^{N+K} w_{ij}^1 v_j(k),$$

где w_{ij}^1 – синаптические весовые коэффициенты; $f_i^1(u_i(k)) = h(u_i(k))$ – функции активации нейронов скрытого слоя.

Аналогично для выходного слоя

$$y_i(k) = f_i^2(g_i(k)), \quad g_i(k) = \sum_{j=0}^M w_{ij}^2 v_j(k),$$

где $y_0(k), y_1(k), \dots, y_M(k)$ – вектор выходного сигнала сети.

Во входном слое двухслойной сети Элмана используется активационная функция гиперболического тангенса tansig , в выходном слое – линейная функция purelin . Такое сочетание передаточных функций позволяет максимально точно аппроксимировать функции с конечным числом точек разрыва. Для этих целей необходимо

также, чтобы выходной слой имел достаточно большое число нейронов. Все эти слои Элмана имеют смещения.

При обучении сети Элмана веса выходного слоя подвергаются точно такой же коррекции, как и веса персептрона. Формулы уточнения весов скрытого слоя сети Элмана более сложны из-за наличия обратных связей между скрытым и контекстным слоями [54].

В MatLab сети Элмана создаются функцией

```
net = newelm(PR, [S1, S2, ..., SN], {TF1,TF2, ..., TFN}, BTF, BLF, PF),
```

где PR – массив размером $R \times 2$ минимальных и максимальных значений для R векторов входа; S_1, S_2, \dots, S_N – число нейронов в слоях; TF_1, TF_2, \dots, TF_N – функции активации в слоях (по умолчанию `tansig`); BTF – обучающая функция, реализующая метод обратного распространения (по умолчанию `traingdx`); BLF – функция настройки, реализующая метод обратного распространения (по умолчанию `learngdm`); PF – критерий качества обучения (по умолчанию `mse`).

Сети Элмана ориентированы на моделирование временных рядов.

Пример 6.1. Пусть нейронная сеть должна в ответ выдавать последовательность битов, в которой появляется единичный бит, если в исходной последовательности встретилось две единицы подряд.

Зададим случайную последовательность из 20 битов командой

```
>> P = round (rand (1, 20))  
P =  
0 0 1 1 0 0 0 1 1 1 0 1 1 0 0 0 1 0 1 0
```

Тогда целевой вектор можно описать командой

```
>> T = [0 (P(1:end-1)+P(2:end) == 2)]  
T =  
0 0 0 1 0 0 0 0 1 1 0 0 1 0 0 0 0 0 0 0
```

Для обучения создадим массивы ячеек:

```
>> Pseq = con2seq(P);  
>> Tseq = con2seq(T);
```

Сеть Элмана создается командой

```
>> net = newelm ([0 1], [10, 1], {'tansig', 'logsig'});
```

Задание параметров и запуск обучения:

```
>> net.trainParam.goal = 0.001;  
>> net.trainParam.epochs = 1000;  
>> net = train(net, Pseq, Tseq);
```

Проверка:

```
>> Y = sim(net, Pseq)
```

```
Y =
[2.6513e-004] [4.0384e-004] [0.2860] [0.9994] [0.0366]
[4.4019e-004] [3.0681e-005] [4.4659e-004] [0.7891] [0.9995]
[0.0356] [9.9000e-004] [0.8228] [0.0218] [4.0585e-004] [2.9824e-
005] [4.4890e-004] [7.5032e-005] [4.7443e-004] [5.7692e-005]
```

Степень соответствия достаточно высокая.

Проверка для случайной последовательности:

```
>> P = round (rand (1, 20))
```

```
P =
```

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

```
>> Pseq = con2seq(P);
```

```
>> Y = sim(net, Pseq)
```

```
Y =
```

```
[0.0992] [0.0554] [0.0020] [2.3346e-004] [4.6559e-004] [3.8692e-
005] [3.6769e-004] [0.3296] [0.9994] [0.0365] [0.0010] [0.8307]
[0.0216] [4.0568e-004] [2.9802e-005] [3.5165e-004] [0.3219]
[0.9994] [0.9993] [0.9994]
```

Пример 6.2. Использование сети Элмана для детектирования амплитуды гармонического сигнала.

Определяются две синусоиды, амплитуды которых различаются в два раза:

```
>> p1 = sin(1:20);
```

```
>> p2 = sin(1:20)*2;
```

Целевые векторы заполняются известным значением амплитуды каждой синусоиды:

```
>> t1 = ones(1,20);
```

```
>> t2 = ones(1,20)*2;
```

Формируется комбинация синусоид и соответствующих целевых векторов:

```
>> p = [p1 p2 t1 t2];
```

```
>> t = [t1 t2 t1 t2];
```

Формируются массивы ячеек:

```
>> Pseq = con2seq(p);
```

```
>> Tseq = con2seq(t);
```

Создается и обучается сеть Элмана.

Проверка:

```
>> a = sim(net,Pseq);
```

```
>> a1 = seq2con(a);
```

```
>> x = 1:80;  
>> plot(x,a1{1,1})
```

Результат приведен на рис. 6.3.

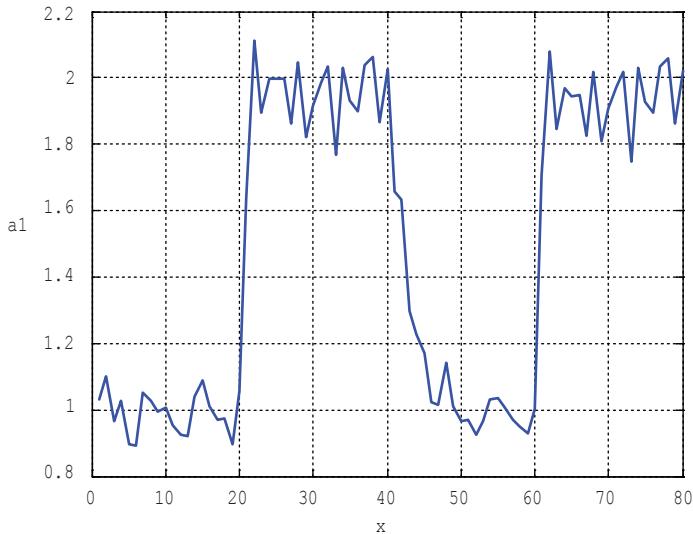


Рис. 6.3. Проверка на воспроизведение обучающих данных

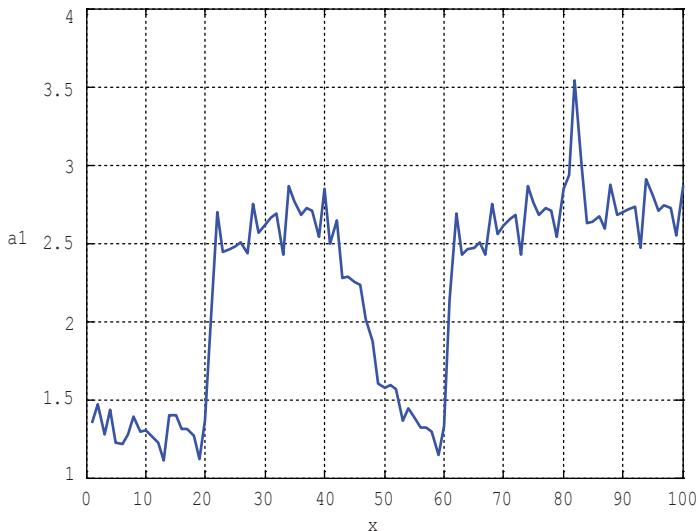


Рис. 6.4. Проверка для измененной амплитуды сигнала

Проверим работу сети при других амплитудах сигналов:

```
>> p1 = sin(1:20)*1.5;
>> p2 = sin(1:20)*2.5;
>> p = [p1 p2 p1 p2 p2];
>> Pseq = con2seq(p);
>> a = sim(net,Pseq);
>> a1 = seq2con(a);
>> x = 1:100;
>> plot(x,a1{1,1})
```

Результат проверки приведен на рис. 6.4.

6.2. Сети Хопфилда

Нейронная сеть Хопфилда – однослойная сеть, в которой каждый нейрон имеет связи со всеми другими нейронами (рис. 6.5).

Сети Хопфилда имеют обратные связи и являются динамическими (рекуррентными), поскольку после получения каждого нового входного сигнала начинается переходный процесс, который заканчивается установлением постоянного выхода или продолжается бесконечно долго. Поэтому проблема устойчивости сети с обратными связями может иметь большое значение при решении прикладных задач.

При подаче на вход устойчивой сети нового входного вектора ИНС переходит от состояния к состоянию, пока не стабилизирует-

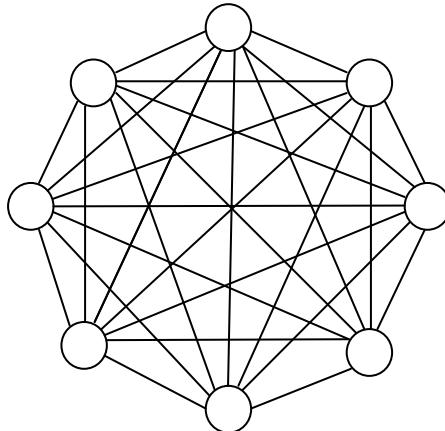


Рис. 6.5. Сеть Хопфилда из восьми нейронов

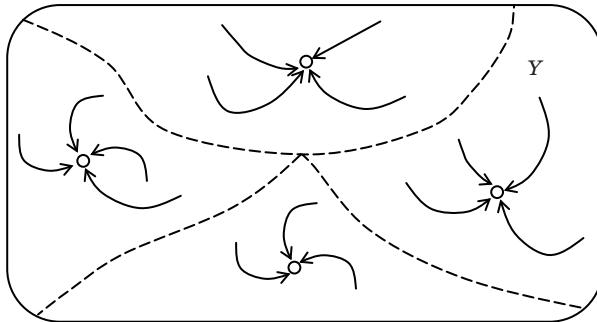


Рис. 6.6. Зоны притяжения и аттракторы в пространстве состояний

ся. Можно сказать, что устойчивые точки (аттракторы) образуют зоны притяжения (рис. 6.6). Помимо целевых аттракторов в сети могут иметь место ложные аттракторы, которым не соответствует никакой образ.

При подаче на вход сети частично неправильного входного вектора сеть стабилизируется в состоянии, ближайшем к желаемому. Для этого веса сети следует выбирать так, чтобы образовывать состояния устойчивого равновесия для каждого входного образа.

Каждый нейрон сети Хопфилда получает входной сигнал, а также сигналы с выходов всех остальных нейронов (рис. 6.7).

Выход нейрона изменяется по формуле

$$y_i^{t+1} = F \left(\sum_i w_{ij} y_i + x_i \right).$$

где x_i – входной сигнал.

В качестве активационной используется пороговая функция с порогом T :

$$y^{t+1} = \begin{cases} y^t, & \text{если } \sum_i w_{ij} y_i + x_i = T, \\ +1, & \text{если } \sum_i w_{ij} y_i + x_i > T, \\ -1, & \text{если } \sum_i w_{ij} y_i + x_i < T. \end{cases}$$

Обычно выбирают нулевой порог.

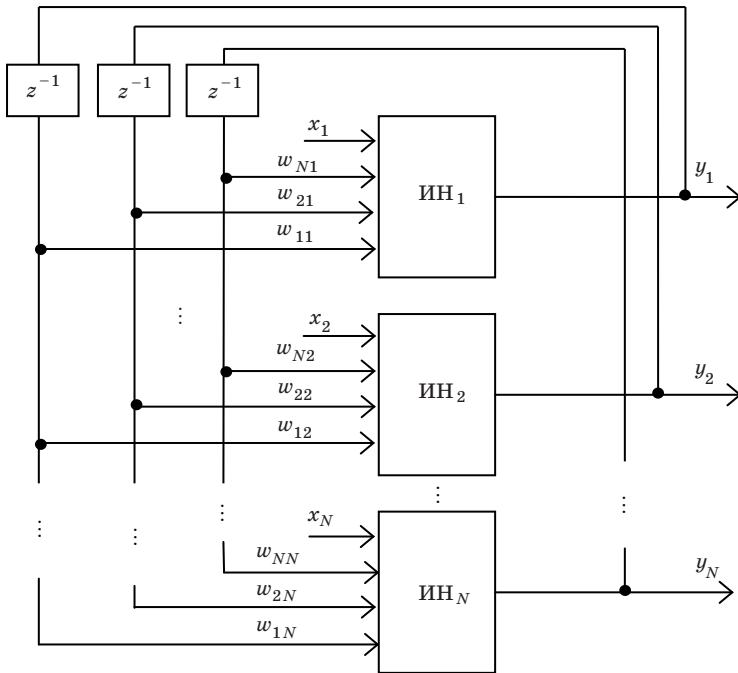


Рис. 6.7. Структура сети Хопфилда

Значение -1 здесь кодирует двоичный 0 .

Состояние сети – это двоичное число длиной N бит:

$$Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix},$$

и система с m нейронами будет иметь 2^m состояний.

Доказательства устойчивости ИНС с обратными связями произвольного вида в настоящее время не существует, но имеется теорема об устойчивости подмножества таких сетей.

В соответствии с этой теоремой сеть с обратными связями устойчива, если ее матрица весов W симметрична: $w_{ij} = w_{ji}$, а элементы главной диагонали нулевые: $w_{ii} = 0$.

В сети Хопфилда веса задаются, и обучение как таковое отсутствует. Вычисление весов производится по формуле

$$w_{ij} = \begin{cases} \sum_{k=1}^m x_j^k x_i^k, & i \neq j, \\ w_{ij} = 0, & i = j, \end{cases}$$

где x_j^k – j -я компонента запоминаемого k -го вектора; m – общее число запоминаемых образов.

Иначе говоря, весовой массив W может быть найден путем вычисления произведения каждого запоминаемого вектора с самим собой и суммированием получившихся матриц размером $n \times n$:

$$W = \sum_{k=1}^m X_k^T X_k - E.$$

Несложно рассчитать, что число весов сети Хопфилда равно $N^2 - N$. Так, например, при длине входного вектора 120 битов получаем $120^2 - 120 = 14280$.

При большой длине запоминаемых векторов целесообразно использовать формулу

$$W = \frac{1}{n} \sum_{k=1}^m X_k^T X_k - E.$$

При таком подходе реализуется правило обучения Хебба: если два нейрона одновременно возбуждены, то связь между ними усиливается.

Пример 6.3. Пусть требуется запомнить образ

$$X = [1 \ 1 \ 1 \ -1]^T.$$

Матрица весов

$$W = \begin{bmatrix} 1 \\ 1 \\ 1 \\ -1 \end{bmatrix} [1 \ 1 \ 1 \ -1] - \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 & -1 \\ 1 & 0 & 1 & -1 \\ 1 & 1 & 0 & -1 \\ -1 & -1 & -1 & 0 \end{bmatrix}.$$

Пусть на вход сети поступает зашумленный вектор $Y(0)$. Тогда

$$Y(1) = \text{sgn}[WY(0)] = \text{sgn} \begin{pmatrix} 0 & 1 & 1 & -1 \\ 1 & 0 & 1 & -1 \\ 1 & 1 & 0 & -1 \\ -1 & -1 & -1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ -1 \\ 1 \\ -1 \end{pmatrix} = \text{sgn} \begin{pmatrix} 1 \\ 3 \\ 1 \\ -1 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ -1 \end{pmatrix},$$

$$Y(2) = \text{sgn}[WY(1)] = \text{sgn} \begin{pmatrix} 0 & 1 & 1 & -1 \\ 1 & 0 & 1 & -1 \\ 1 & 1 & 0 & -1 \\ -1 & -1 & -1 & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \\ 1 \\ -1 \end{pmatrix} = \text{sgn} \begin{pmatrix} 3 \\ 3 \\ 3 \\ -3 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ -1 \end{pmatrix}.$$

Таким образом, сеть восстанавливает запомненный образ.

Алгоритм работы сети Хопфилда может быть описан так:

1. Задается входной образ (неполный, искаженный): Y при $t = 0$;
2. Выполняются асинхронные вычисления по нейронной сети (итерационное правило):

$$Y_i^{t+1} = \text{sgn} \left[\sum_{j=1}^n w_{ij} Y_j^t \right].$$

3. Вычисления выполняются до тех пор, пока не наступит равновесие, т. е. пока все выходы не станут постоянными. Этот вектор и будет образом, соответствующим входному вектору Y .

Для хорошей работы сети Хопфилда необходимо, чтобы запоминаемые образы были мало похожи друг на друга, т. е. слабо коррелированы (или ортогональны).

Мера коррелированности двоичных векторов X_j и X_k описывается формулой

$$K_{jk} = \left| \sum_{i=1}^n x_i^j x_i^k \right|.$$

Для всех m запоминаемых образов получаем формулу

$$K = \sum_{j=1}^m \sum_{k=1}^m K_{jk}.$$

Повышение качества работы сети Хопфилда может быть достигнуто благодаря использованию так называемого «алгоритма забывания». Суть этого подхода заключается в том, что на этапе формирования матрицы весов наряду с «истинными» образами сеть запоминает некоторое число «ложных» образов. Затем сеть получает некоторое входное значение и постепенно достигает точки притяжения, соответствующей некоторому образу. Если этот образ X_F оказался ложным, то веса сети корректируются по формуле

$$W(t+1) = W(t) - \lambda X_F^T X_F.$$

Многократное повторение этой процедуры позволяет исправить области притяжения запомненных образов.

Для описания работы сети Хопфилда часто используется понятие *энергетической функции*, которую можно выбрать для пары нейронов в следующем виде:

$$e_{ij} = -x_i w_{ij} x_j.$$

Тогда энергия всей сети может быть описана формулой

$$E = -\sum_i \sum_j x_i w_{ij} x_j = -X^T W X.$$

Каждому образу, хранящемуся в ассоциативной памяти, соответствует свой минимум энергии E .

Для выполнения условия $E \rightarrow \min$ требуется, чтобы

$$\sum_i \sum_j x_i w_{ij} x_j \rightarrow \max.$$

Выходы сети $Y \in \{-1, 1\}$, поэтому при выборе $w_{ij} = x_i x_j$ все слагаемые становятся положительными, и получаем минимум энергии

$$E = -\sum_i \sum_j (x_i)^2 (x_j)^2.$$

Можно также заметить, что при таком выборе весов обеспечивается

$$\sum_{j=1}^N w_{ij} x_j = \frac{1}{N} \sum_{j=1}^N x_i (x_j x_j) = x_i \frac{1}{N} \sum_{j=1}^N 1 = x_i.$$

Можно показать, что энергия сети Хопфилда при симметричной матрице весов либо убывает, либо не изменяется.

Рассмотрим энергию в момент времени t , выделив в ней составляющую, вносимую нейроном x_p :

$$E(t) = -\sum_i \sum_j x_i w_{ij} x_j = \sum_{i \neq p} \sum_{j \neq p} x_i w_{ij} x_j - \sum_j x_j w_{pj} x_p - \sum_i x_i w_{ip} x_p.$$

Пусть значение x_p в момент времени $t + 1$ меняется:

$$E(t+1) = -\sum_i \sum_j x_i w_{ij} x_j = \sum_{i \neq p} \sum_{j \neq p} x_i w_{ij} x_j - \sum_j x_j w_{pj} \hat{x}_p - \sum_i x_i w_{ip} \hat{x}_p.$$

Тогда

$$\Delta E = E(t+1) - E(t) = \\ = -\sum_j x_j w_{pj} x_p^* - \sum_i x_i w_{ip} x_p^* + \sum_j x_j w_{pj} x_p + \sum_i x_i w_{ip} x_p.$$

Поскольку весовая матрица симметрична, можно записать

$$\Delta E = 2 \sum_i x_i w_{pi} (x_p - x_p^*).$$

Возможны два варианта изменения энергии:

1. $x_p = -1, x_p^* = 1 \Rightarrow (x_p - x_p^*) = -2, \sum_i x_i w_{pi} > 0 \Rightarrow \Delta E < 0,$
2. $x_p = 1, x_p^* = -1 \Rightarrow (x_p - x_p^*) = 2, \sum_i x_i w_{pi} < 0 \Rightarrow \Delta E < 0.$

Таким образом, приращение энергии всегда оказывается отрицательным.

По существующим оценкам, информационная емкость сети Хопфилда сравнительно невелика – число случайных образов M , которые может запомнить сеть, связано с числом нейронов сети N формулой

$$M \leq 0,15N.$$

Рассмотрим примеры работы с сетью Хопфилда в MatLab.

Сеть Хопфилда имеет один нейронный слой с функциями взвешивания `dotprod`, накопления `netsum` и линейной ограниченной функцией активизации `satlins`. Слой охвачен динамической обратной связью с весами $LW\{1,1\}$ и имеет смещения (рис. 6.8).

Активационная функция нейронов приведена на рис. 6.9.

Функция для создания сети Хопфилда имеет вид

```
>> net = newhop(T)
```

где T – массив элементов размером $R \times Q$, объединяющий Q целевых векторов со значениями $+1$ или -1 ; R – число элементов вектора выхода.

Пример 6.4. Создать сеть Хопфилда с двумя устойчивыми точками в трехмерном пространстве:

```
>> T = [-1 -1 1; 1 -1 1]' % определение двух аттракторов  
T =  
-1 1  
-1 -1  
1 1
```

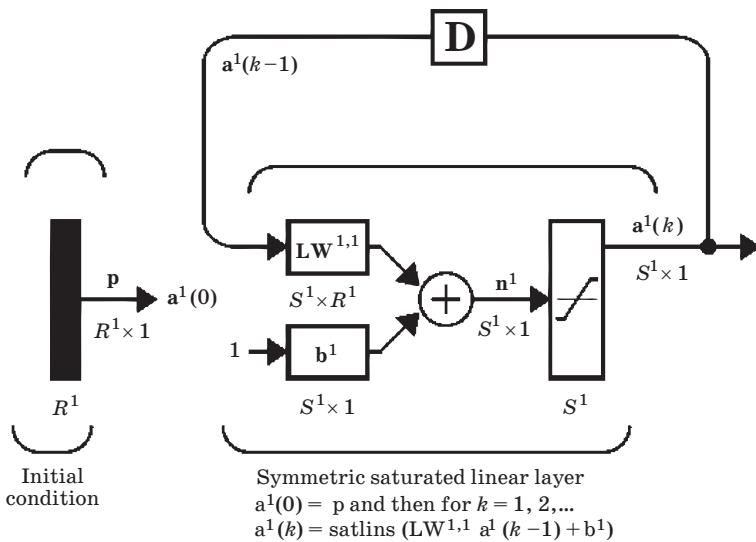


Рис. 6.8. Сеть Хопфилда в MatLab

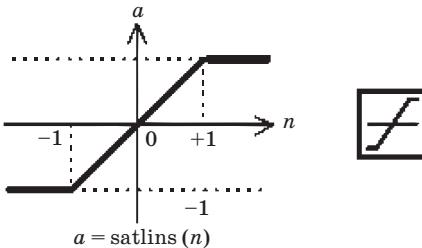


Рис. 6.9. Активационная функция сети Хопфилда

```

>> net = newhop(T); % создание сети Хопфилда
>> A = T; % проверка
>> Y = sim(net,2,[], A) % работы сети для эталонных образов
Y =
    -1    1
    -1   -1
     1    1
>> A = {[ -0.6; -0.6; 0.6]}


```

```

A =
[3x1 double]
>> [Y,Pf,Af] = sim(net,{1 5},{},A);

```

Можно посмотреть, как менялся выход сети после каждой итерации:

```

>> [Y{1} Y{2} Y{3} Y{4} Y{5}]
ans =
-0.6971 -0.8099 -0.9410 -1.0000 -1.0000
-0.9884 -1.0000 -1.0000 -1.0000 -1.0000
0.9884 1.0000 1.0000 1.0000 1.0000

```

Ошибка постепенно уменьшается, и сеть переходит в устойчивое состояние.

Пример 6.5. Рассмотрим четыре аттрактора в многомерном пространстве:

```

>> vectors = [-1 1 -1 -1 1 1 -1 -1 1 -1; -1 -1 -1 1 1 1 -1 -1 -1 -1; -1
-1 1 -1 1 -1 1 -1 -1; 1 -1 -1 -1 1 1 -1 -1 1];
>> net = newhop(vectors);
>> result = sim(net,4,[],vectors)
result =
-1 -1 -1 1
1 -1 -1 -1
-1 -1 1 -1
-1 1 -1 -1
1 1 1 1
-1 1 -1 -1
-1 -1 1 -1
1 -1 -1 -1
-1 -1 -1 1
>> test = {[0.1; 0.8; -1; -0.7; 0.5; -1; -0.9; 0.85; -1]};
>> result = sim(net,{1,5},{},test);
>> for i = 1:5,
    disp(sprintf('Network state after %d iterations:',i));
    disp(result{i});
end
Network state after 1 iterations:
-0.4930
0.8601
-1.0000
-1.0000
0.9661
-1.0000

```

-1.0000
0.8712
-0.7384

Network state after 2 iterations:

-0.7045
0.9879
-1.0000
-1.0000
1.0000
-1.0000
-1.0000
0.9904
-0.7593

Network state after 3 iterations:

-0.8625
1.0000
-1.0000
-1.0000
1.0000
-1.0000
-1.0000
1.0000
-0.8748

Network state after 4 iterations:

-0.9966
1.0000
-1.0000
-1.0000
1.0000
-1.0000
-1.0000
1.0000
-0.9993

Network state after 5 iterations:

-1
1
-1
-1
1
-1

-1
1
-1

Пример 6.6. Пусть точки притяжения сети Хопфилда заданы на плоскости в углах прямоугольника (рис. 6.10):

```
>> T = [+1 -1; -1 +1; +1 +1; -1 -1];  
>> T = T';  
>> plot(T(1,:),T(2,:),'ro','MarkerSize',13), hold on;  
>> axis([-1.2 1.2 -1.2 1.2]);  
>> title('Hopfield Network State Space');  
>> xlabel('x');  
>> ylabel('y');  
>> net = newhop(T);
```

Рассмотрим процесс изменения состояний сети Хопфилда для нескольких случайных точек:

```
>> for i = 1:5  
    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,:));
```

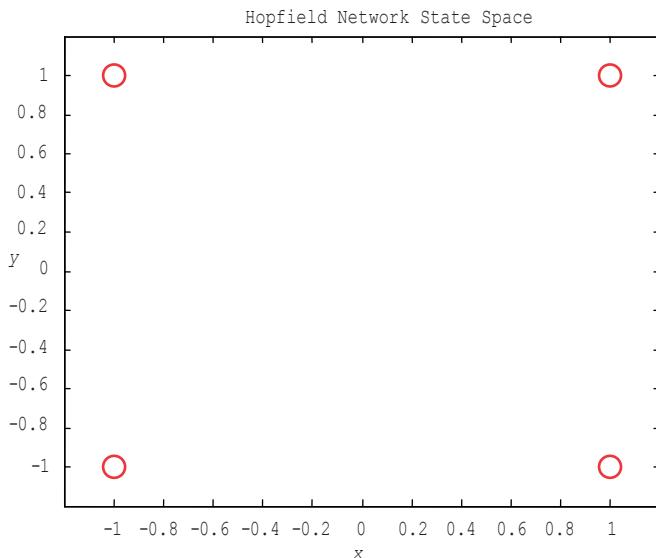


Рис. 6.10. Координаты точек притяжения сети Хопфилда

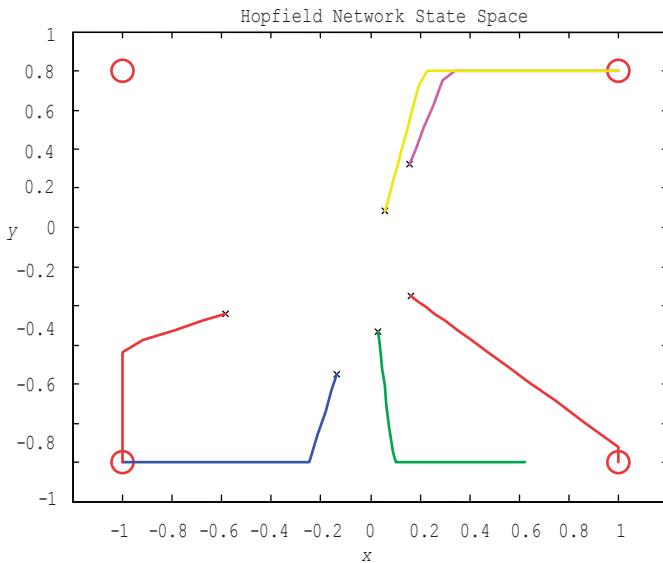


Рис. 6.11. Изменение состояний сети Хопфилда

```
record(2,:), color(rem(i,5)+1),'LineWidth',5)
end
```

Как следует из рис. 6.11, из каждого случайного состояния (обозначено крестиком) сеть Хопфилда попадает в ближайшее положение равновесия.

Пример 6.7. Рассмотрим задачу распознавания изображений цифр с помощью ИНС Хопфилда.

Закодируем цифры следующим образом:

```
>> zero = [ -1 -1 -1 -1 -1 -1 -1,
-1 +1 +1 +1 +1 +1 -1,
-1 +1 -1 -1 -1 +1 -1,
-1 +1 -1 -1 -1 +1 -1,
-1 +1 -1 -1 -1 +1 -1,
-1 +1 -1 -1 -1 +1 -1,
-1 +1 +1 +1 +1 +1 -1,
-1 -1 -1 -1 -1 -1 -1];
>> zero = reshape(zero',1,63); % преобразование в вектор-столбец
>> one = [ -1 -1 -1 -1 -1 -1 -1,
-1 -1 -1 +1 -1 -1 -1,
```

```

-1 -1 +1 +1 -1 -1 -1,
-1 -1 -1 +1 -1 -1 -1,
-1 -1 -1 +1 -1 -1 -1,
-1 -1 -1 +1 -1 -1 -1,
-1 -1 -1 +1 -1 -1 -1,
-1 +1 +1 +1 +1 +1 -1,
-1 -1 -1 -1 -1 -1 -1];
>> one = reshape(one',1,63);
>> two = [ -1 -1 -1 -1 -1 -1 -1,
-1 +1 +1 +1 +1 +1 -1,
-1 -1 -1 -1 -1 +1 -1,
-1 -1 -1 -1 -1 +1 -1,
-1 +1 +1 +1 +1 +1 -1,
-1 +1 -1 -1 -1 -1 -1,
-1 +1 -1 -1 -1 -1 -1,
-1 +1 +1 +1 +1 +1 -1,
-1 -1 -1 -1 -1 -1 -1];
>> two = reshape(two',1,63);
>> three = [ -1 -1 -1 -1 -1 -1 -1,
-1 +1 +1 +1 +1 +1 -1,
-1 -1 -1 -1 -1 +1 -1,
-1 -1 -1 -1 -1 +1 -1,
-1 -1 -1 +1 +1 +1 -1,
-1 -1 -1 -1 -1 -1 -1,
-1 -1 -1 -1 -1 +1 -1,
-1 +1 +1 +1 +1 +1 -1,
-1 -1 -1 -1 -1 -1 -1];
>> three = reshape(three',1,63);
>> four = [-1 -1 -1 -1 -1 -1 -1,
-1 +1 -1 -1 -1 -1 -1,
-1 +1 -1 -1 -1 -1 -1,
-1 +1 +1 +1 +1 -1 -1,
-1 -1 -1 +1 -1 -1 -1,
-1 -1 -1 +1 -1 -1 -1,
-1 -1 -1 +1 -1 -1 -1,
-1 -1 -1 -1 -1 -1 -1];
>> four = reshape(four',1,63);
>> five = [-1 -1 -1 -1 -1 -1 -1,
-1 +1 +1 +1 +1 +1 -1,

```

```

-1 +1 -1 -1 -1 -1 -1,
-1 +1 -1 -1 -1 -1 -1,
-1 +1 +1 +1 +1 +1 -1,
-1 -1 -1 -1 -1 +1 -1,
-1 -1 -1 -1 -1 +1 -1,
-1 +1 +1 +1 +1 +1 -1,
-1 -1 -1 -1 -1 -1 -1];
>> five = reshape(five',1,63);
>> six = [-1 -1 -1 -1 -1 -1 -1,
-1 +1 +1 +1 +1 +1 -1,
-1 +1 -1 -1 -1 -1 -1,
-1 +1 -1 -1 -1 -1 -1,
-1 +1 +1 +1 +1 +1 -1,
-1 +1 -1 -1 -1 +1 -1,
-1 +1 -1 -1 -1 +1 -1,
-1 +1 +1 +1 +1 +1 -1,
-1 +1 -1 -1 -1 -1 -1];
>> six = reshape(six',1,63);
>> seven = [-1 -1 -1 -1 -1 -1 -1,
-1 +1 +1 +1 +1 +1 -1,
-1 -1 -1 -1 -1 +1 -1,
-1 -1 -1 -1 +1 -1 -1,
-1 -1 -1 +1 -1 -1 -1,
-1 -1 +1 -1 -1 -1 -1,
-1 +1 -1 -1 -1 -1 -1,
-1 +1 -1 -1 -1 -1 -1,
-1 -1 -1 -1 -1 -1 -1];
>> seven = reshape(seven',1,63);
>> eight = [-1 -1 -1 -1 -1 -1 -1,
-1 +1 +1 +1 +1 +1 -1,
-1 +1 -1 -1 -1 +1 -1,
-1 +1 -1 -1 -1 +1 -1,
-1 +1 +1 +1 +1 +1 -1,
-1 +1 -1 -1 -1 +1 -1,
-1 +1 -1 -1 -1 +1 -1,
-1 +1 +1 +1 +1 +1 -1,
-1 -1 -1 -1 -1 -1 -1];
>> eight = reshape(eight',1,63);
>> nine = [-1 -1 -1 -1 -1 -1 -1,
-1 +1 +1 +1 +1 +1 -1,

```

```

-1 +1 -1 -1 -1 +1 -1,
-1 +1 -1 -1 -1 +1 -1,
-1 +1 +1 +1 +1 +1 -1,
-1 -1 -1 -1 -1 +1 -1,
-1 -1 -1 -1 -1 +1 -1,
-1 -1 -1 -1 -1 +1 -1,
-1 -1 -1 -1 -1 -1 -1];
>> nine = reshape(nine',1,63);

```

Зададим массив атTRACTоров сети Хопфилда (матрица 63×10):

```
>> digits1 = [zero; one; two; three; four; five; six; seven; eight; nine]';
```

Создадим сеть Хопфилда:

```
>> net = newhop(digits1);
```

Для визуального представления цифр можно использовать следующие команды:

```

>> digits = {zero, one, two, three, four, five, six, seven, eight, nine};
>> bnw = [1 1 1; 0 1 0];      % цветовая палитра
>> for P = 1:10              % вывод цифр на экран
    subplot(3,4,P);
    digit = digits{P};
    img = reshape(digit,7,9);
    image((img'+1)*255/2);
    axis image
    axis off
    colormap(bnw)
    title(sprintf('Number %d', P));
end

```

Результат использования функции show приведен на рис. 6.12.

Далее можно проверить работу созданной сети. Для этого опишем искаженное изображение. Например,

```

>> five1 = [-1 -1 -1 -1 -1 -1 -1,
            -1 +1 +1 +1 +1 +1 -1,
            -1 +1 -1 -1 -1 -1 -1,
            +1 +1 -1 -1 -1 -1 -1,
            -1 +1 +1 +1 +1 +1 -1,
            -1 -1 -1 +1 +1 +1 -1,
            -1 -1 -1 -1 -1 +1 -1,
            -1 +1 +1 +1 +1 +1 -1,
            -1 +1 -1 -1 -1 -1 -1];
>> five1 = reshape(five1',1,63);

```

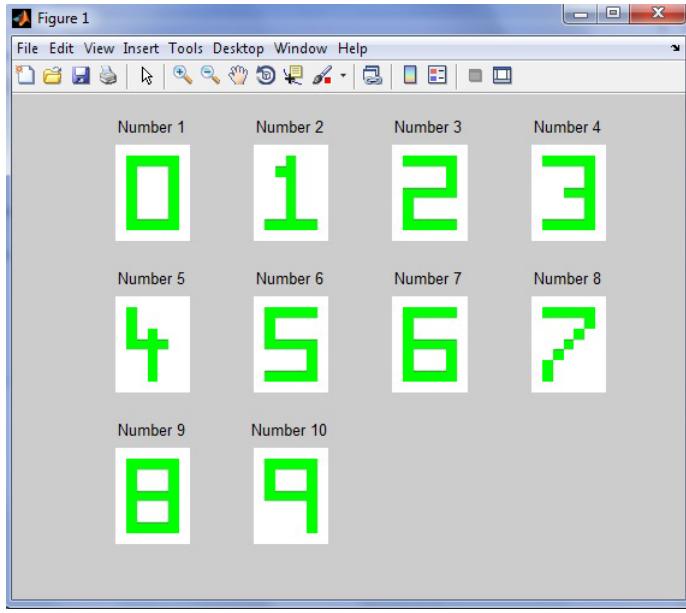


Рис. 6.12. Объекты, запомненные сетью Хопфилда

Отобразим этот зашумленный код с помощью команд

```
>> img = reshape(five1,7,9);
>> image((img'+1)*255/2);
>> axis image
>> axis off
>> colormap(bnw)
>> title(sprintf('Number 5 noise'));
```

Результат приведен на рис. 6.13.

Далее запустим сеть Хопфилда:

```
>> five2 = {five1'}      % преобразование входных данных в массив
>> [Y,Pf,Af] = sim(net,{1 10},{},five2); % запуск сети Хопфилда
после чего произведем вывод изображения на экран:
```

```
>> img = reshape(Y{10},7,9);
>> image((img'+1)*255/2);
>> axis image
>> axis off
>> colormap(bnw)
>> title(sprintf('Number 5 correct'));
```

Результат приведен на рис. 6.14.

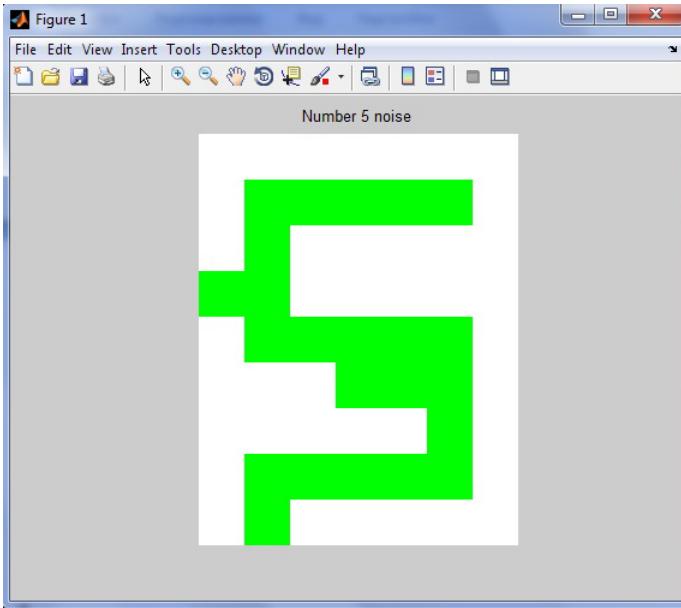


Рис. 6.13. Исажженный входной образ для сети Хопфилда

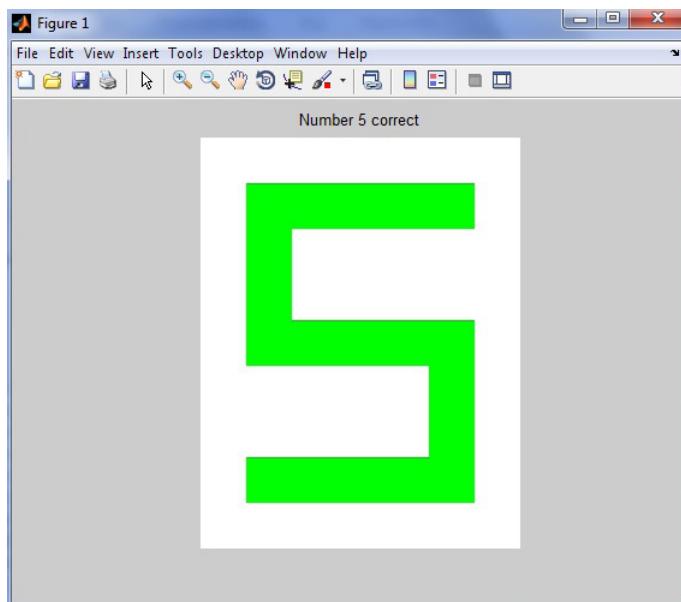


Рис. 6.14. Изображение, восстановленное сетью Хопфилда

6.3. Двунаправленная ассоциативная память

Предложенный Хопфилдом подход к организации ассоциативной памяти является автоассоциативным, т. е. можно восстановить неполный образ или исправить искаженный, но нельзя вызвать какой-то другой, поскольку сеть имеет только один слой.

Если использовать большее число слоев, то можно получить гетероассоциативную память, в которой размерность выходного вектора может быть больше размерности входного вектора.

Примером гетероассоциативной памяти является двунаправленная ассоциативная память (ДАП).

Как и сеть Хопфилда, ДАП позволяет вырабатывать правильные реакции на искаженные входы. Подобная сеть содержит два слоя, поэтому такая память может иметь входной вектор и вектор образа разной длины (рис. 6.15).

Входной вектор A кратковременно выставляется на выходе 1-го слоя, так что сеть вырабатывает выходной вектор B :

$$B = F(AW),$$

где F – активационная функция.

Затем вектор A снимается, и сеть вырабатывает новое значение выходного слоя:

$$A = F(BW^T).$$

Каждый цикл может уточнять векторы на выходе 1-го и 2-го слоя, пока не будет достигнут резонанс и векторы A и B не станут постоянными. Вектор B является ассоциированным образом.

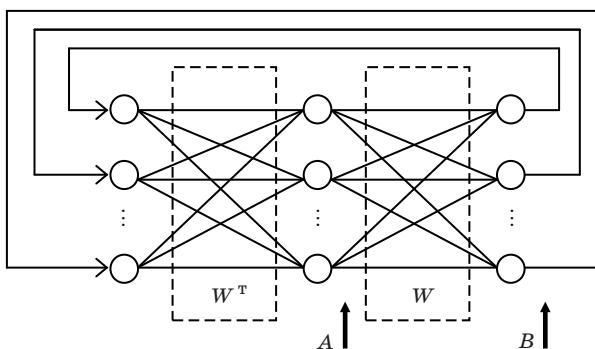


Рис. 6.15. Структура двунаправленной ассоциативной памяти

Векторы на выходе слоев можно назвать *кратковременной памятью*. Если приложить на выходе слоя один новый входной вектор, то состояние кратковременной памяти может измениться.

Система функционирует в направлении минимизации функции энергии, и каждый минимум этой функции соответствует одному запомненному образу.

Долговременная память реализуется в весах W и W^T .

Активационная функция F может быть пороговой с порогом 0.

Поведение сети изменяется по тактам. В промежутках между тактами выход нейрона фиксирован.

Векторы A и B задают обучающий набор, и весовая матрица вычисляется по формуле

$$W = \sum_i A_i^T B_i.$$

Пример 6.8. Пусть заданы обучающие пары

$$A_1 = (1 \ 0 \ 0), B_1 = (0 \ 0 \ 1 \ 0),$$

$$A_2 = (0 \ 1 \ 0), B_2 = (1 \ 0 \ 0 \ 1),$$

$$A_3 = (0 \ 0 \ 1), B_3 = (0 \ 1 \ 0 \ 0).$$

Нулевое значение кодируется с помощью значения -1 (это улучшает поведение сети):

$$A_1 = (1 \ -1 \ -1), B_1 = (-1 \ -1 \ 1 \ -1),$$

$$A_2 = (-1 \ 1 \ -1), B_2 = (1 \ -1 \ -1 \ 1),$$

$$A_3 = (-1 \ -1 \ 1), B_3 = (-1 \ 1 \ -1 \ -1).$$

Далее рассчитывается матрица весов:

$$W = A_1^T B_1 + A_2^T B_2 + A_3^T B_3,$$

$$A_1^T B_1 = \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix} \begin{bmatrix} -1 & -1 & 1 & -1 \\ 1 & 1 & -1 & 1 \\ 1 & 1 & -1 & 1 \end{bmatrix} = \begin{bmatrix} -1 & -1 & 1 & -1 \\ 1 & 1 & -1 & 1 \\ 1 & 1 & -1 & 1 \end{bmatrix},$$

$$A_2^T B_2 = \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix} \begin{bmatrix} -1 & 1 & 1 & -1 \\ 1 & -1 & -1 & 1 \\ -1 & 1 & 1 & -1 \end{bmatrix} = \begin{bmatrix} -1 & 1 & 1 & -1 \\ 1 & -1 & -1 & 1 \\ -1 & 1 & 1 & -1 \end{bmatrix},$$

$$A_3^T B_3 = \begin{bmatrix} -1 \\ -1 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & -1 & 1 & 1 \\ 1 & -1 & 1 & 1 \\ -1 & 1 & -1 & -1 \end{bmatrix} = \begin{bmatrix} 1 & -1 & 1 & 1 \\ 1 & -1 & 1 & 1 \\ -1 & 1 & -1 & -1 \end{bmatrix},$$

$$W = \begin{bmatrix} -1 & -1 & 3 & -1 \\ 3 & -1 & -1 & 3 \\ -1 & 3 & -1 & -1 \end{bmatrix}, \quad W^T = \begin{bmatrix} -1 & 3 & -1 \\ -1 & -1 & 3 \\ 3 & -1 & -1 \\ -1 & 3 & -1 \end{bmatrix}.$$

Пусть на вход сети подан искаженный вектор $A = (0,8 \ 0,2 \ 0)$. Тогда

$$B = F[AW] = F\left[\begin{bmatrix} 0,8 & 0,2 & 0 \end{bmatrix} \cdot \begin{bmatrix} -1 & -1 & 3 & -1 \\ 3 & -1 & -1 & 3 \\ -1 & 3 & -1 & -1 \end{bmatrix}\right] = \\ = F[-0,2 \ -1 \ 2,2 \ -1] = [0 \ 0 \ 1 \ 0],$$

$$A = F[BW^T] = F\left[\begin{bmatrix} -1 & -1 & 1 & -1 \end{bmatrix} \cdot \begin{bmatrix} -1 & 3 & -1 \\ -1 & -1 & 3 \\ 3 & -1 & -1 \\ -1 & 3 & -1 \end{bmatrix}\right] = \\ = F[6 \ -6 \ -2] = [1 \ 0 \ 0].$$

Таким образом, используя ДАП, можно связывать одни образы с другими, имеющими иную размерность. Кроме того, можно организовывать цепочки ассоциаций, как бы воссоздавая образ за образом.

Важное свойство ДАП заключается в ее устойчивости, которая сохраняется при любых значениях весов сети.

Описанный вариант ДАП является наиболее простым. В общем случае поведение нейронов может быть асинхронным, функция активации может быть сигмоидной, и соответственно выход нейронов может быть непрерывным, а не дискретным. Это характерно, например, при оптической реализации ДАП. Поэтому в общем случае веса ДАП не назначаются путем вычисления (как показано ранее), а получаются в результате процедуры обучения.

Рассмотрим, как можно использовать принцип машины Больцмана при обучении ДАП [25].

Первоначально весам ДАП присваиваются случайные значения. Вводятся искусственная температура T сети и искусственная энергия нейрона E . Искусственная температура получает высокое начальное значение.

Искусственную энергию нейрона рассчитывают по формуле

$$E_j = Y_j - \theta,$$

где θ – пороговое значение.

Для каждого нейрона рассчитывается величина

$$P_j = \frac{1}{\left[1 + \exp\left(-\frac{E_j}{T}\right)\right]}.$$

Состояние нейрона находят путем сравнения P_j и случайного числа $k \in [0,1]$. Если $P_j > k$, то $Y_j = 1$.

Процедура обучения включает в себя три циклически повторяющихся этапа:

1 этап

1.1. На вход и выход сети кратковременно подается обучающая пара векторов.

1.2. В сети происходит динамический процесс до достижения равновесия.

1.3. Записываются выходные значения всех нейронов.

1.4. Повторяются шаги 1.1 – 1.3 для всех обучающих пар.

1.5. Вычисляется так называемая закрепленная вероятность P_{ij}^+ как отношение числа опытов, когда выходы нейронов i и j одновременно равны единице, к общему числу опытов.

2 этап

2.1. Сети придается случайное состояние и рассматривается состояние после завершения переходного процесса. Выходы нейронов фиксируются.

2.2. Шаг 2.1 повторяется много раз.

2.3. Вычисляются так называемые незакрепленные вероятности P_{ij}^- как отношение числа опытов, когда выходы нейронов i и j одновременно равны единице, к общему числу опытов.

3 этап

3.1. Веса сети корректируются по формуле

$$w_{ij} = w_{ij} + \eta(P_{ij}^+ - P_{ij}^-),$$

где η – константа обучения.

6.4. Нейронная сеть Хэмминга

Нейронная сеть Хэмминга является моделью ассоциативной памяти. Ее можно рассматривать как развитие сети Хопфилда.

Основная идея функционирования этой сети состоит в параллельном вычислении расстояния Хэмминга между предъявляемыми

мым на входы сети входным вектором и образами, закодированными в структуре сети. При этом задача формирования эталонного образа на выходе сети не ставится, достаточно выдавать его номер.

Расстоянием Хэмминга между двумя двоичными векторами называется число компонент, в которых эти векторы различны.

В сети Хэмминга, как и в сети Хопфилда, входные сигналы $X = [x_1, x_2, \dots, x_n]^T$ могут быть только двоичными векторами, которые представляются с помощью биполярного кодирования: $x_i = +1$ (двоичная «1»), $x_i = -1$ (двоичный «0»).

Идея работы сети Хемминга заключается в нахождении расстояния от входного двоичного вектора длиной N до всех хранимых в сети L образов. Таким образом, размерность входа и выхода сети в общем случае не совпадает. После окончания переходного процесса в сети остается возбужденным один из нейронов выходного слоя.

Простейший вариант ИНС Хэмминга реализуется в виде однослоиной структуры (рис. 6.16).

Рабочий слой сети содержит m нейронов (по числу закодированных в сети образов) и вычисляет расстояния Хэмминга y_1, y_2, \dots, y_m между входным вектором X и каждым образом.

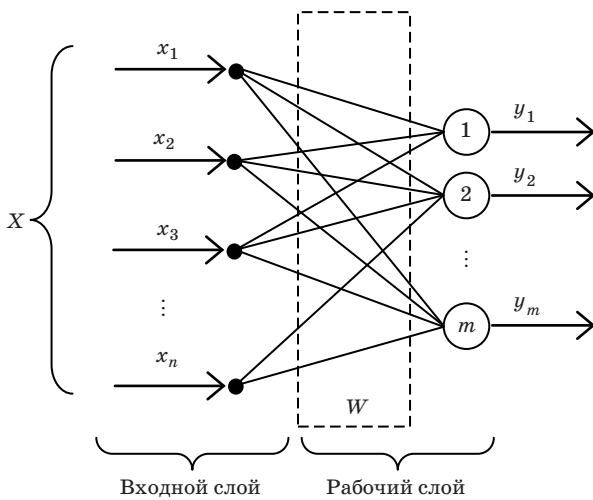


Рис. 6.16. Структура однослоиной ИНС Хэмминга

Рассмотрим два биполярных вектора $X = [x_1, x_2, \dots, x_n]$ и $W = [w_1, w_2, \dots, w_n]$. Их скалярное произведение можно описать следующим образом:

$$XW^T = \sum_{i=1}^n x_i w_i = a - b,$$

где a – число одинаковых компонент векторов; b – число разных его компонент.

Например,

$$n = 5, W = [1, -1, 1, -1, -1], \quad X = [-1, -1, -1, 1, 1].$$

Тогда

$$XW^T = 1 \cdot (-1) + (-1) \cdot (-1) + 1 \cdot (-1) + (-1) \cdot 1 + (-1) \cdot 1 = 1 - 4 = -3.$$

Поскольку $n = a + b$, можно записать

$$XW^T = 2a - n.$$

Следовательно,

$$a = \frac{n}{2} + \frac{XW^T}{2} = \frac{n}{2} + \frac{1}{2} \sum_{i=1}^n x_i w_i.$$

Этой формулой можно описать работу нейронов выходного слоя.

Матрица весов W рабочего слоя сети формируется на основе предъявленных обучающих данных:

$$w_{ij} = -x / 2, \quad i = \overline{1, n}, \quad j = \overline{1, m},$$

$$W_1 = \begin{bmatrix} w_{11} & w_{12} & \dots & w_{1m} \\ w_{21} & w_{22} & \dots & w_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n1} & w_{n2} & \dots & w_{nm} \end{bmatrix} = \begin{bmatrix} -x_{11} / 2 & -x_{12} / 2 & \dots & -x_{1m} / 2 \\ -x_{21} / 2 & -x_{22} / 2 & \dots & -x_{2m} / 2 \\ \vdots & \vdots & \ddots & \vdots \\ -x_{n1} / 2 & -x_{n2} / 2 & \dots & -x_{nm} / 2 \end{bmatrix}.$$

Нейроны рабочего слоя производят вычисления по обычной формуле

$$y_j = F \left(\sum_{i=1}^n w_{ij} x_i - P \right), \quad i = \overline{1, n}, \quad j = \overline{1, m},$$

где P – порог нейрона; F – его активационная функция.

Для того чтобы нейроны рабочего слоя могли реализовать вычисление расстояния Хэмминга между предъявленным вектором X и каждым из m образов, для них устанавливается порог $P = -n/2$, а активационная функция F выбирается как линейная с насыщением:

$$F(z) = \begin{cases} 0, & \text{если } z \leq 0, \\ z_i, & \text{если } 0 < z \leq n, \\ n, & \text{если } z > n. \end{cases}$$

График активационной функции нейронов рабочего слоя имеет вид, приведенный на рис. 6.17.

Например,

$$n = 5, X = [-1, -1, -1, 1, 1], W = [0,5; 0,5; 0,5; -0,5; -0,5].$$

Тогда

$$y = F(XW^T + 2,5) = F((-0,5) \cdot 3 + (-0,5) \cdot 2 + 2,5) = F(0) = 0.$$

Таким образом, входной и весовой векторы не имеют совпадающих битов (расстояние Хэмминга максимально).

Если же рассмотреть пару

$$X = [1, 1, 1, -1, -1], W = [0,5; 0,5; 0,5; -0,5; -0,5],$$

то

$$y = F(XW^T + 2,5) = F((0,5) \cdot 3 + (0,5) \cdot 2 + 2,5) = F(5) = 5.$$

Здесь расстояние Хэмминга минимально, и выходной сигнал максимальен.

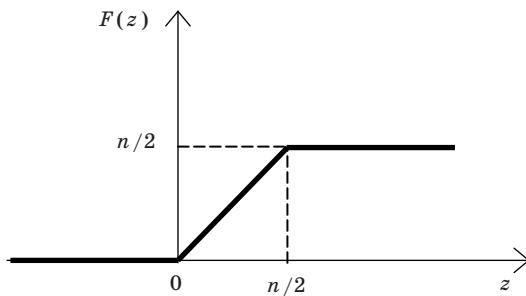


Рис. 6.17. График активационной функции нейронов рабочего слоя

Второй слой в сеть Хэмминга добавляется с целью выделения образа, в наибольшей степени соответствующего входному вектору. Этот слой (часто называемый maxnet) имеет обратные связи для организации конкуренции нейронов (рис. 6.18).

Второй слой, как и первый, содержит m нейронов. Все нейроны слоя связаны между собой обратными связями по принципу «каждый с каждым». Вместе с тем в отличие от сети Хопфилда все нейроны 2-го слоя имеют ненулевые обратные связи на самих себя. Веса всех связей постоянны, причем разноименные нейроны связаны отрицательной (тормозной) обратной связью с весом $-\varepsilon$, а на самих себя нейроны имеют положительную (возбуждающую) обратную связь с весом, равным $+1$.

Матрица весов 2-го слоя имеет вид

$$W_2 = \begin{bmatrix} w_{11} & w_{12} & \dots & w_{1m} \\ w_{21} & w_{22} & \dots & w_{2m} \\ \vdots & \vdots & \vdots & \vdots \\ w_{m1} & w_{m2} & \dots & w_{mm} \end{bmatrix} = \begin{bmatrix} 1 & -\varepsilon & \dots & -\varepsilon \\ -\varepsilon & 1 & \dots & -\varepsilon \\ \vdots & \vdots & \vdots & \vdots \\ -\varepsilon & -\varepsilon & \dots & 1 \end{bmatrix}.$$

Второй слой функционирует в режиме «победитель получает все», т. е. в установившемся режиме в слое остается активированным только один нейрон, а остальные находятся в невозбужденном состоянии. Отличное от нуля значение выходного сигнала активированного нейрона с номером k указывает на принадлежность входного образа к соответствующему классу.

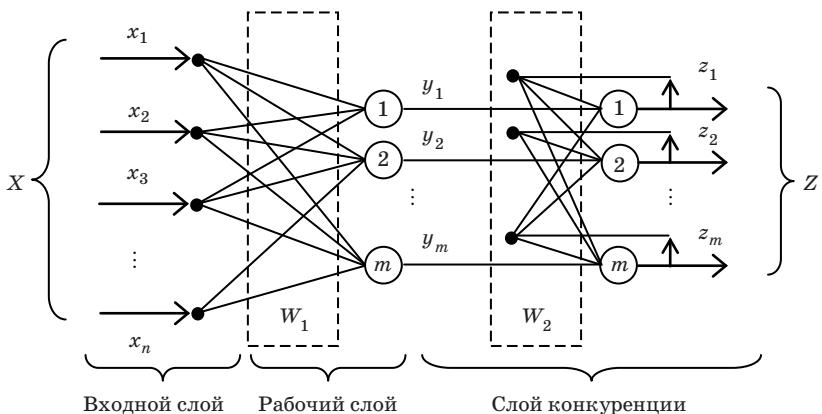


Рис. 6.18. Структура двухслойной ИНС Хэмминга

Двухслойная сеть Хэмминга имеет две фазы функционирования. В первой фазе на входы сети подается входной вектор X . На выходах 1-го рабочего слоя формируются выходные сигналы, которые задают начальные состояния $Z(0)$ нейронов 2-го слоя maxnet .

Во второй фазе инициировавшие слой maxnet сигналы удаляются, и из сформированного ими начального состояния запускается итерационный процесс внутри 2-го слоя. Итерационный процесс завершается в момент, когда все нейроны кроме одного перейдут в нулевое состояние. Нейрон-победитель становится представителем класса данных, к которому принадлежит входной вектор X . Процесс определения нейрона-победителя выполняется пошагово в соответствии с выражением

$$z_k(t) = F \left(\sum_{j=1}^m w_{jk} z_j(t-1) \right), \quad j, k = \overline{1, m}.$$

Активационная функция F здесь – линейная с насыщением, причем величина порога должна быть достаточно большой, чтобы любые возможные значения аргумента не приводили к насыщению.

Учитывая, что веса 2-го слоя выбираются исходя из условия

$$w_{ik} = \begin{cases} 1, & j = k, \\ -\varepsilon, & j \neq k, \end{cases}$$

процесс определения нейрона-победителя можно представить в виде

$$z_k(t) = F \left(z_k(p-1) - \varepsilon \sum_{j \neq k}^m z_j(p-1) \right), \quad k = \overline{1, m}.$$

Значение весов тормозных связей ε обычно выбирается в диапазоне

$$0 < \varepsilon < \frac{1}{m-1}.$$

Для обеспечения абсолютной сходимости процесса в слое maxnet веса тормозных связей должны отличаться друг от друга. Для удовлетворения этому условию при расчете весов тормозных связей добавляется малая случайная величина

$$w_{jk} = -\frac{1}{L-1} + \xi.$$

Сеть Хэмминга может быть модифицирована, чтобы выдавать на выходе не номер класса, к которому относится входной вектор, а ассоциированный с этим классом образ. При этом длина выходного вектора может не совпадать с длиной входного вектора (гетероассоциативная память). Для того чтобы добиться такого эффекта, к сети Хэмминга добавляется 3-й (выходной) слой (рис. 6.19).

В процессе функционирования трехслойной сети Хэмминга в отличие от двухслойной появляется третья фаза. В этой фазе нейрон-победитель во 2-м слое посредством весов, связывающих его с нейронами 3-го выходного слоя, формирует на выходе отклик сети в виде двоичного вектора P , соответствующий возбуждающему вектору X . Поскольку на выходе слоя конкуренции только один нейрон выдает единичный выходной сигнал, можно сформулировать простое правило назначения весов выходного слоя:

$$w_{il} = p_{lj}, \quad i, j = \overline{1, m}, \quad l = \overline{1, k},$$

$$W_3 = \begin{bmatrix} P_1 \\ P_2 \\ \vdots \\ P_m \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1k} \\ p_{21} & p_{22} & \cdots & p_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ p_{m1} & p_{m2} & \cdots & p_{mk} \end{bmatrix}.$$

Сеть Хэмминга имеет важные преимущества по сравнению с сетью Хопфилда:

1. Емкость данной сети не зависит от размерности входного сигнала и равна числу нейронов рабочего слоя.

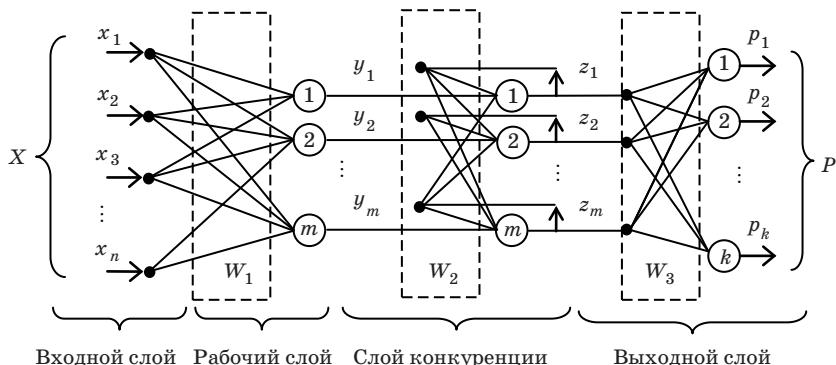


Рис. 6.19. Структура трехслойной ИНС Хэмминга

2. Число межнейронных связей в сети Хэмминга меньше, чем в сети Хопфилда. Например, сеть Хопфилда с размерностью входного сигнала 100 может запомнить примерно 13 случайных образов, при этом она будет содержать 10000 связей между нейронами. Однослойная сеть Хэмминга с такой же емкостью будет содержать 1000 взвешенных связей, а двухслойная – 1100, из которых 100 связей – в слое тахнет.

3. Однослойная сеть Хэмминга работает существенно быстрее сети Хопфилда, поскольку решение задачи формируется в результате однократного прохода через один слой нейронов.

4. Сеть Хэмминга имеет один из самых простых алгоритмов формирования весов и смещений.

5. Экспериментально доказано, что двухслойная сеть Хэмминга функционирует лучше, чем сеть Хопфилда при случайному наборе запоминаемых векторов.

Однако у сети Хэмминга есть и недостатки:

1. Плохая работа при сильно зашумленных входных сигналах. Если сигналы находятся на одинаковом расстоянии Хэмминга от двух или более эталонов, то выбор одного из эталонов становится совершенно случайным.

2. Сети Хэмминга рассчитаны на работу только с бинарными входными сигналами, что ограничивает их применение.

Однако в целом сети Хэмминга могут успешно использоваться для решения задач распознавания образов, классификации, реализации ассоциативной и гетероассоциативной памяти, а также передачи сигналов в условиях помех.

6.5. Адаптивные резонансные нейронные сети

Традиционные ИНС с трудом решают проблему стабильности и одновременной пластичности запоминания, т. е. проблему наращивания числа образов в памяти без искажения тех, которые там уже имеются.

Сеть прямого распространения обучается на фиксированном множестве примеров. Если после обучения число примеров увеличивается, то «дообучение» невозможно, требуется полное переобучение.

В сетях Кохонена также предполагается, что все входные образы принадлежат к одному из полученных при обучении классов. Выявление нового класса требует полного повторения процесса обучения.

Аналогичная ситуация наблюдается при использовании сетей Хопфилда и Хэмминга.

Сеть АРТ предназначена для решения проблемы стабильности – пластиичности. Существуют так называемые АРТ1-сети, обрабатывающие двоичные векторы, а также АРТ2-сети для обработки непрерывных векторов. Кроме того, следует назвать модели ARTMAP и FuzzyART.

Рассмотрим далее работу АРТ1.

Сеть АРТ1 представляет собой векторный классификатор. Входной вектор относится к одной из групп ранее запомненных векторов. Решение о классификации выдается в виде возбуждения одного из нейронов распознавающего слоя.

Отличие АРТ1 от других ИНС состоит в следующем:

- 1) если входной вектор похож на один из ранее запомненных векторов, то запомненный вектор будет изменяться, чтобы стать похожим на входной вектор;
- 2) возможен отказ от классификации, если входной вектор не похож ни на один из запомненных векторов. В этой ситуации образуется новый класс.

Упрощенная конфигурация сети АРТ1 [25] включает в себя слой сравнения, слой распознавания, сброс, приемник 1 и приемник 2 (рис. 6.20).

Введем обозначения: m – размер входного вектора; n – число запомненных образов.

Слой сравнения получает двоичный входной вектор X и первоначально пропускает его неизменным для формирования выходно-

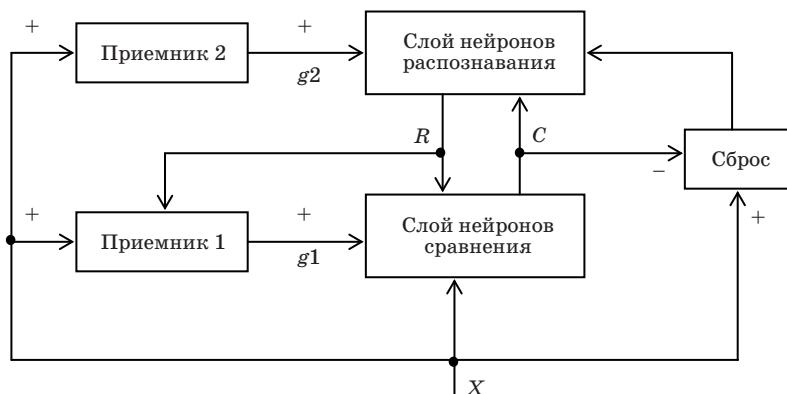


Рис. 6.20. Структура нейронной сети АРТ

го вектора C (на более поздней стадии компоненты вектора R модифицируют C).

Упрощенно слой сравнения имеет вид, приведенный на рис. 6.21.

Векторы T образуются весовыми коэффициентами t_{ij} , которые являются двоичными числами.

Каждый нейрон слоя сравнения ($\text{ИН}_1 - \text{ИН}_m$) получает три двоичных входа:

- 1) компонента входного вектора x_i ;
- 2) сигнал обратной связи p_j ;
- 3) вход от приемника $g1$.

Для того чтобы на выходе нейрона была единица, необходимо, чтобы не менее двух его входов были равны единице. Первоначально сигнал $g1$ установлен в «1», а сигналы p_j – в «0», поэтому исходно вектор C совпадает с X .

Слой распознавания служит для классификации входных векторов. Каждый нейрон слоя распознавания имеет вектор весов B . Только один нейрон в этом слое возбуждается, все остальные заторможены. Это достигается благодаря тому, что все нейроны слоя распознавания охвачены механизмом латерального возбуждения-торможения (рис. 6.22).

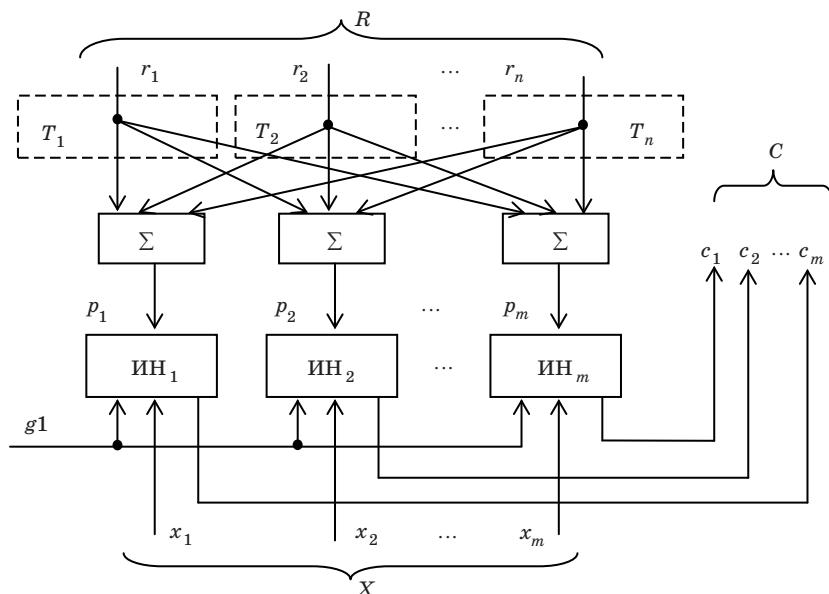


Рис. 6.21. Структура слоя сравнения

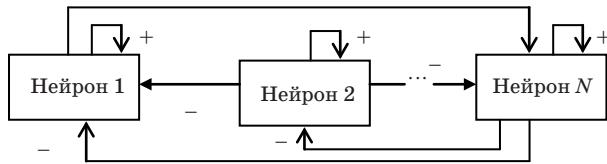


Рис. 6.22. Взаимодействие нейронов слоя распознавания

Таким образом, чем больше нейрон возбужден, тем больше он тормозит другие нейроны и одновременно поддерживает свой уровень.

Упрощенная версия слоя распознавания приведена на рис. 6.23.

На входы всех нейронов слоя распознавания поступают сигналы от всех нейронов слоя сравнения, и наоборот.

Каждый нейрон слоя распознавания имеет вектор весов B . Выход нейрона

$$Y_j = B_j C.$$

Активационная функция F пороговая:

$$F(Y_j) = \begin{cases} 1, & Y_j \geq T, \\ 0, & Y_j < T. \end{cases}$$

Таким образом, вектор R двоичный.

Нейрон j имеет максимальную реакцию, если вектор C (выход слоя сравнения) максимально похож на его весовой вектор B_j . Таким образом, веса нейрона представляют собой запомнивший образ для группы векторов.

Веса являются действительными числами. Двоичная версия этого образа запоминается в наборе весов слоя сравнения.

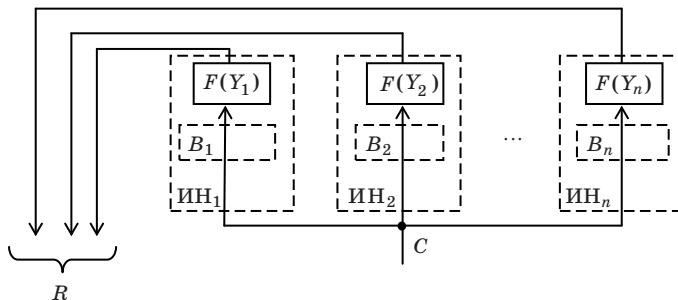


Рис. 6.23. Взаимодействие нейронов слоя распознавания

В процессе функционирования каждый нейрон слоя распознавания вычисляет свертку вектора собственных весов и входного вектора C . Чем ближе веса к вектору C , тем выше выход нейрона. Нейрон-победитель затормаживает остальные нейроны слоя распознавания.

Таким образом, только один нейрон слоя распознавания будет иметь на выходе «1», а остальные – «0». Следовательно, вектор R будет иметь только одну единичную компоненту.

Приемник 2. Его выход равен единице, если входной вектор X имеет хотя бы одну единичную компоненту. Иначе говоря, сигнал $g2$ представляет собой логическое ИЛИ компонент входного вектора:

$$g2 = x_1 \vee x_2 \vee x_3 \vee \dots \vee x_m.$$

Приемник 1. Работает так же, как приемник 2, но приход хотя бы одной единичной компоненты R сбрасывает $g1$ в «0»:

$$g1 = (x_1 \vee x_2 \vee \dots \vee x_m) \wedge \overline{(r_1 \vee r_2 \vee \dots \vee r_m)}.$$

Сброс. Модуль сброса измеряет сходство между векторами X и C . Мерой сходства служит отношение число единиц в X к их числу в C :

$$S = N / D,$$

где D – число единиц в векторе с наибольшим числом единиц.

Например,

$$\begin{aligned} X &= [1011101] \quad (D = 5), \\ C &= [0011101] \quad (N = 4). \end{aligned}$$

Таким образом, параметр сходства изменяется от 0 до 1.

Три основные операции АРТ заключаются в распознавании, сравнении и поиске.

Распознавание состоит в следующем. Пока входной вектор X отсутствует, все его компоненты можно считать нулевыми, следовательно, $g2 = 0$, и в «0» устанавливаются выходы всех нейронов слоя распознавания (так как $C = X$).

Затем подается входной вектор X , т. е. одна или больше компонент входного вектора становятся ненулевыми, а $g1$ и $g2$ – равными единице. Это создает условия для возбуждения нейронов слоя сравнения, и вектор C дублирует вектор X .

После этого в слое распознавания вычисляется свертка вектора весов B_j и C . Нейрон, веса которого наиболее похожи на входной вектор, побеждает и затормаживает остальные нейроны. Единственная компонента R становится единичной. Таким образом, один нейрон слоя распознавания соответствует одной из категорий классификации.

Сравнение. Компоненты вектора R поступают на все нейроны слоя сравнения через веса t_{ij} , которые принимают двоичные значения.

Взаимосвязь между векторами весов T и B заключается в том, что B является масштабированной версией вектора T .

Поскольку R больше не нулевой, сигнал $g1$ устанавливается в «0», поэтому в соответствии с правилом 2/3 возбудиться могут только нейроны, получающие на входе одновременно единицы от входного вектора X и вектора R .

Если X и P не имеют совпадающих компонент, то обратная связь от распознающего слоя сбросит компоненты C в «0» (рис. 6.24).

Если различий между векторами X и P много, то вектор C будет иметь нулей значительно больше, чем X , и вырабатывается сигнал сброса, который отключает возбужденный нейрон на время текущей классификации.

Поиск. Если сигнал сброса не вырабатывается, то сходство считается найденным, а классификация завершенной. Если же сброс произошел, то все компоненты R обнуляются, $g1$ устанавливается в «1», вектор C становится равным X , и происходит новая проверка, но уже без заторможенного нейрона. Эта процедура продолжается до тех пор, пока не будут заторможены все нейроны. В этом случае происходит запоминание нового образа, для чего выделяется новый нейрон, весовые коэффициенты которого B и T устанавливаются в соответствии с новым образом.

Если же все-таки будет найден один из нейронов, похожий на входной образ, то производится обучающий цикл с целью коррекции весов B и T .

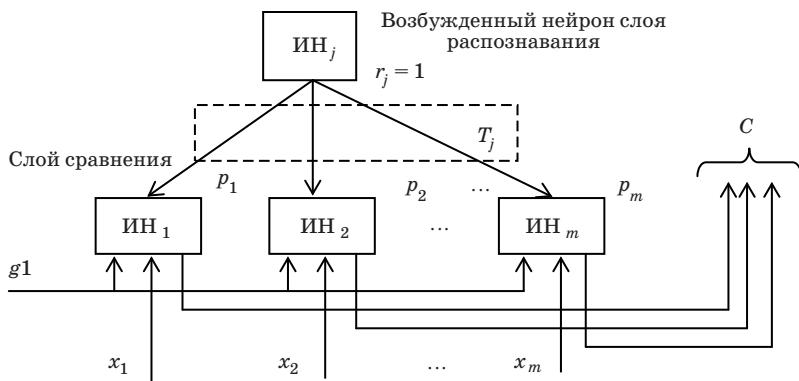


Рис. 6.24. Взаимодействие слоев распознавания и сравнения

При обучении на вход сети последовательно подаются векторы из обучающего множества и подстраиваются веса сети с тем, чтобы сходные векторы активизировали один и тот же нейрон.

Возможны разные алгоритмы обучения. Рассмотрим так называемый *быстрый алгоритм*.

Вес связи b_{ij} , связывающей i -й нейрон слоя сравнения и j -й нейрон слоя распознавания, может быть найден по формуле

$$b_{ij} = \frac{2C_i}{\sum_k C_k + 1},$$

где j – возбужденный нейрон; \sum – число единиц на выходе слоя сравнения.

Компоненты вектора весов T_j , связанного с новым запомненным вектором, изменяются таким образом, чтобы они стали равны соответствующим компонентам вектора C : $t_{ij} = c_i$ (t_{ij} – связь между выигравшим нейроном j в слое распознавания и нейроном i в слое сравнения).

Сумма может быть рассмотрена как «размер» вектора. Если вектор C «большой», то b_{ij} маленькое. Таким образом, оказывается возможным разделение векторов, если один из них является поднабором другого (т. е. входит в него).

Пример. Пусть имеются векторы $X_1 = [1 \ 0 \ 0 \ 0 \ 0]$, $X_2 = [1 \ 1 \ 1 \ 0 \ 0]$ (т. е. X_1 – поднабор X_2).

Если в формуле для b_{ij} убрать сумму, то получим

$$\begin{aligned} T_1 &= B_1 = [1 \ 0 \ 0 \ 0 \ 0], \\ T_2 &= B_2 = [1 \ 1 \ 1 \ 0 \ 0]. \end{aligned}$$

Если затем подать на вход сети вектор X_1 , то оба нейрона будут возбуждены одинаково. Если же в формуле для b_{ij} использовать сумму, то получим

$$\begin{aligned} B_1 &= [1 \ 0 \ 0 \ 0 \ 0], \\ B_2 &= [0,5 \ 0,5 \ 0,5 \ 0 \ 0], \end{aligned}$$

и при подаче вектора X_1 для нейрона 1 уровень возбуждения будет соответствовать «1», а для нейрона 2 – 0,5 (правильно).

При подаче X_2 получим соответственно 1 и 3/2 (тоже правильно).

Пусть далее подан вектор $X_3 = [1 \ 1 \ 0 \ 0 \ 0]$. Для нейрона 1 уровень возбуждения будет единица, а для нейрона 2 он составит 2/3. Нейрон 1 победит, вектор C приобретет значение $[1 \ 1 \ 0 \ 0 \ 0]$. Величина $S = 1/2$, и если уровень сходства $r = 2/3$, то нейрон 1 будет заторможен, и выиграет нейрон 2 ($C = [1 \ 1 \ 0 \ 0 \ 0]$), $S = 1$.

Таким образом, сеть АРТ может распознавать входные векторы, сильно похожие на эталонные образы, корректировать эталонные образы под модификации входных векторов или отслеживать медленные изменения эталона, а также наращивать число эталонных образов.

Разумеется, при моделировании сетей АРТ на традиционных компьютерах трудно добиться высокого быстродействия, в таких случаях особенно выгодно применять параллельно работающие вычислители.

Вопросы для самопроверки

1. Какова структура нейронной сети Элмана?
2. Как описать функцию XOR в виде временной последовательности?
3. Какие активационные функции используются в разных слоях сети Элмана?
4. Как корректируются весовые матрицы сети Элмана?
5. Какие параметры требуется задать при создании сети Элмана в MatLab?
6. Какую структуру имеет нейронная сеть Хопфилда?
7. Что такое атTRACTоры?
8. Может ли сеть Хопфилда быть неустойчивой?
9. Какую активационную функцию используют нейроны сети Хопфилда?
10. Сколько состояний может иметь нейронная сеть Хопфилда?
11. Какие условия должны выполняться для устойчивости сети Хопфилда?
12. Как рассчитываются веса межнейронных связей сети Хопфилда?
13. Как можно оценить число весов в сети Хопфилда?
14. Какие образы являются «хорошими», с точки зрения запоминания, в сети Хопфилда?
15. Сколько случайных образов может хранить сеть Хопфилда?
16. Что такое алгоритм забывания?
17. Как можно описать искусственную энергию сети Хопфилда?
18. Как с помощью искусственной энергии обосновать устойчивость сети Хопфилда?
19. Чем отличаются автоассоциативная и гетероассоциативная память?

20. Какую структуру имеет двунаправленная ассоциативная память?
21. Как вычисляется весовая матрица ДАП?
22. Что можно сказать об устойчивости ДАП?
23. В чем состоит принцип использования машины Больцмана при обучении ДАП?
24. Какова структура нейронной сети Хэмминга?
25. Как вычисляется расстояние Хэмминга?
26. Как оценить число межнейронных связей в сети Хэмминга?
27. Как определяются веса однослойной сети Хэмминга?
28. Какие активационные функции используются в однослойной сети Хэмминга?
29. Зачем добавляется второй слой в сети Хэмминга?
30. Как функционирует 2-й слой в сетях Хэмминга?
31. Как определяется матрица весов 2-слой в двухслойной сети Хэмминга?
32. Зачем может быть добавлен 3-й слой в сети Хэмминга?
33. Каковы преимущества сетей Хэмминга по сравнению с сетями Хопфилда?
34. Какие недостатки имеют сети Хэмминга?
35. В чем заключается проблема стабильности-пластичности для нейронных сетей?
36. Какие особенности имеет функционирование сетей ART в сравнении с другими нейронными сетями?
37. Какова структура нейронной сети ART?
38. Как функционирует слой сравнения сети ART?
39. Как функционирует слой распознавания сети ART?
40. Какие основные операции выполняет сеть ART?
41. Как происходит обучение сети ART?

7. НЕЙРОННЫЕ СЕТИ КОХОНЕНА

7.1. Структура сети Кохонена

Нейронные сети Кохонена [9] предназначены для решения задач векторной классификации. При этом до обучения не известно, сколько классов должно быть использовано для решения задачи, поэтому требуется использовать обучение без учителя.

Сеть Кохонена – это ИНС, содержащая входной слой и слой активных нейронов. Поскольку входной слой просто распределяет входные сигналы по нейронам активного слоя, такую сеть можно считать однослойной.

Обычно активный слой одномерный или двумерный. Это отражает представления нейробиологии, в соответствии с которыми сенсорная информация, представленная множеством сигналов, отображается в линейные или планарные структуры коры головного мозга.

Различают два варианта сетей Кохонена:

1. Слой Кохонена. В нем нейроны активного слоя не упорядочены. В процессе обучения подстраиваются веса только одного нейрона-победителя.

2. Карта Кохонена (Self-Organized Map – SOM). В данном случае нейроны активного слоя образуют регулярную структуру. Такая карта применяется обычно для кластеризации графических изображений и звуковых сигналов.

В процессе обучения SOM после предъявления достаточного числа образов все выходные нейроны сети разбиваются на подмножества, каждое из которых «откликается» на образы соответствующего класса.

Важное свойство SOM после обучения заключается в том, что чем более похожи объекты из обучающей выборки, тем ближе должны находиться нейроны активного слоя, ассоциированные с кластерами данных объектов. Расстояние между нейронами определяется *топологией* сети. На рис. 7.1 приведены варианты нейрона с восемью или шестью соседями.

Таким образом, нейроны с номерами i и j характеризуются своими позициями I и J , с помощью которых можно описать функцию близости (neighbourhood function), которая может иметь вид гауссовой функции

$$g(i, j) = \exp\left(-\frac{\|I - J\|^2}{2\sigma^2}\right),$$

где σ – уменьшающаяся во времени константа.

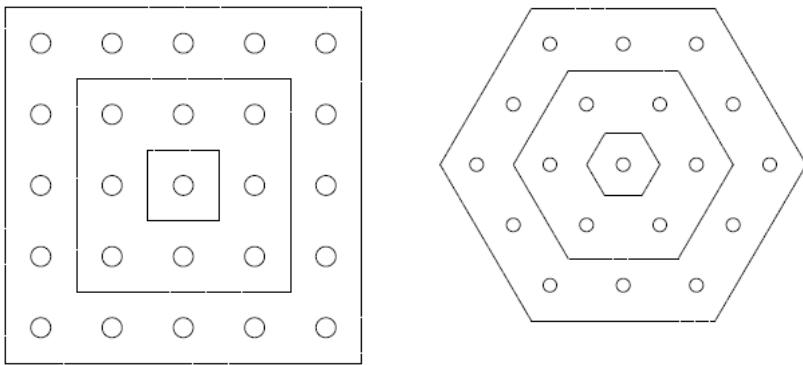


Рис. 7.1. Варианты топологии двумерной SOM

С помощью функции близости описывается область взаимодействия нейронов слоя Кохонена при обучении. На рис. 7.2 представлена одномерная сеть Кохонена из N нейронов.

С каждым нейроном Кохонена ассоциируется весовой вектор W , которому соответствует точка входного пространства.

Входной слой просто передает компоненты входного вектора X на каждый нейрон слоя Кохонена. Предполагается, что $N \gg m$.

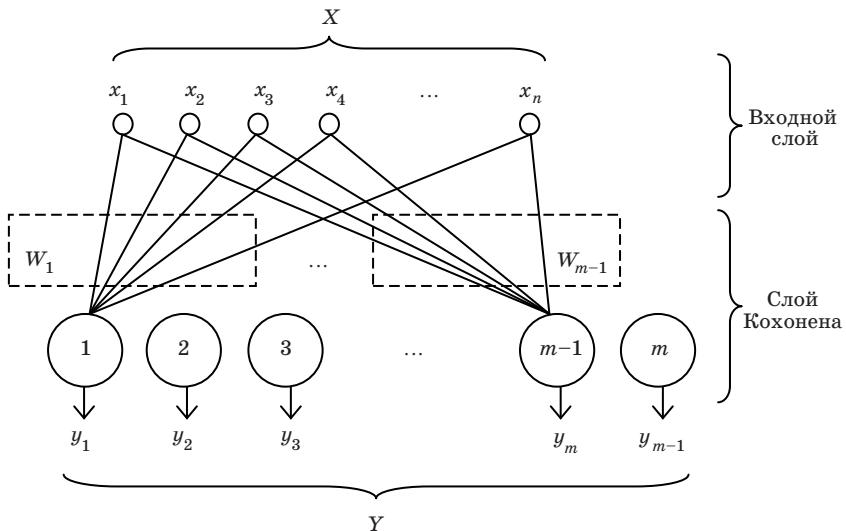


Рис. 7.2. Одномерная сеть Кохонена

Каждый i -й нейрон 2-го слоя имеет собственный вектор весов W_i , который сравнивается с входным вектором X . Сравнение предполагает вычисление расстояния между X и W_i , так что в слое Кохонена появляется нейрон-победитель с номером j , веса которого имеют наименьшее расстояние до входного вектора:

$$j = \arg \min_i \|X - W_i\|.$$

В качестве метрики здесь может выступать евклидово расстояние

$$\|X - W\| = \sqrt{\sum_{i=1}^n (x_i - w_i)^2}. \quad (7.1)$$

Если векторы X и W нормализованные, то в качестве меры близости можно использовать скалярное произведение, и тогда выход нейрона можно описать формулой

$$y_j = W_j X = \sum_{i=1}^n w_{ij} x_i, \quad (7.2)$$

и выход нейрона j оказывается максимальным при одинаковых X и W :

$$j = \arg \max_i \|W_i X\|.$$

Нормализация векторов выполняется по формулам

$$w'_i = \frac{w_i}{\sqrt{\sum_{j=1}^n w_j^2}}, \quad x'_i = \frac{x_i}{\sqrt{\sum_{j=1}^n x_j^2}}, \quad i = \overline{1, n}.$$

Таким образом, результатом работы слоя конкурирующих нейронов при подаче на входной слой некоторого вектора X является определение нейрона, который имеет наибольший выходной сигнал y_j (нейрон-победитель). Этот нейрон обладает весовым вектором W_j , который наиболее близок к входному вектору.

Нейроны слоя Кохонена работают не изолированно, между ними существуют соревновательные связи, с помощью которых близкие нейроны усиливают сигналы друг друга. Формула (7.2) дополняется вторым слагаемым:

$$y_j = \sum_{i=1}^n w_{ij} x_i + \sum_{l \neq j} g(j, l) y_l, \quad (7.3)$$

где $g(j, l)$ – сила связи между нейронами, которую часто называют *функцией соседства*.

Возможны разные варианты $g(j, l)$.

Простейший вариант функции соседства одномерного слоя

$$g(i, j) = \begin{cases} 1, & |i - j| \leq r, \\ 0, & |i - j| > r. \end{cases}$$

Гауссова функция соседства может быть описана в виде

$$g(i, j) = \exp\left(-\frac{R^2}{2\sigma^2}\right),$$

где σ – выбранная константа; R – расстояние между нейронами Кохонена, вычисляемое по формуле

$$R = \|r_i - r_j\|^2.$$

Здесь r_i и r_j – координаты нейронов.

Помимо гауссовой функции можно использовать функцию, которая имеет вид «мексиканской шляпы» (рис. 7.3).

Как следует из рис. 7.3, вокруг точки возбуждения имеется небольшая окрестность, в которой нейроны переходят в активное состояние. За пределами данной окрестности происходит торможение активности нейронов. Наблюдается также слабая связь активности между нейронами с места возбуждения и нейронами с удаленных участков.

На практике можно использовать упрощенный подход к описанию функции взаимодействия (рис. 7.4).

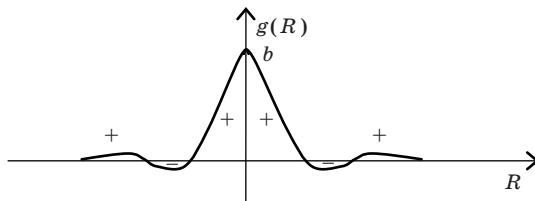


Рис. 7.3. Вариант описания взаимодействия нейронов

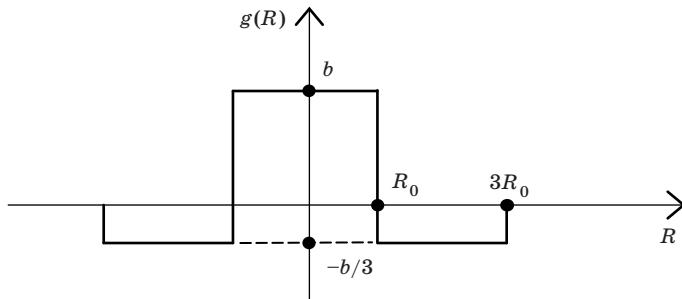


Рис. 7.4. Упрощенное представление силы связи слоя Кохонена

В соответствии с рис. 7.4 силу связи можно описать следующим образом:

$$g(R) = \begin{cases} b, & R \in [-R_0, R_0], \\ -\frac{b}{3}, & R \in [-R_0, -3R_0], \\ \frac{b}{3}, & R \in [R_0, 3R_0], \\ 0 & \text{в противном случае.} \end{cases}$$

При рассмотрении сети с двумерным слоем Кохонена функции взаимного влияния нейронов должны быть описаны для плоскости (рис. 7.5). Можно рассматривать также трехмерный слой Кохонена.

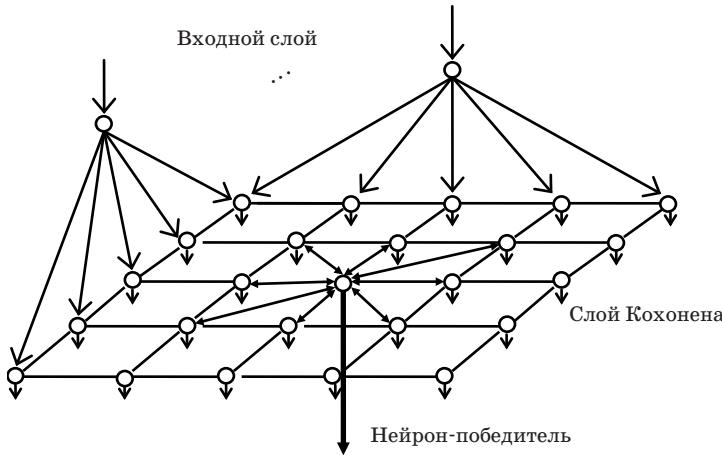


Рис. 7.5. ИНС с двумерным слоем Кохонена

7.2. Обучение сети Кохонена

Алгоритмы обучения сети Кохонена реализуют процесс обучения без учителя. При обучении n входных весов нейрона Кохонена рассматриваются как вектор в n -мерном пространстве. До обучения веса получают малые случайные значения. Затем каждый n -мерный вектор нормализуется в вектор единичной длины (с тем же направлением, что и исходный вектор).

При $n = 2$ все входные векторы можно изобразить на единичной окружности, при $n = 3$ – на единичной сфере, а для $n > 3$ следует рассматривать единичную гиперсферу.

Входные векторы обучающего набора также нормализуются, после чего сеть обучается по следующему алгоритму:

1. Подается входной вектор X .
2. Определяется нейрон-победитель с номером j по формуле (7.2).
3. Происходит подстройка весов нейронов.
 - а) если обучается слой конкурирующих нейронов, то обучается один нейрон:

$$W_i^T(t+1) = W_i^T(t) + \eta(X(t) - W_i^T(t));$$

б) если обучается SOM, то происходит подстройка весов нейронов Кохонена по формуле

$$W_i^T(t+1) = W_i^T(t) + \eta g(i,j)(X(t) - W_i^T(t)).$$

Здесь η – коэффициент скорости обучения.

4. Предъявляется новый входной вектор X , и шаги 1–4 повторяются.

Значение η должно постепенно уменьшаться.

В конце обучения влияние нейронов друг на друга может стать таким малым, чтобы обучался всего один ИН, т. е. при подаче на вход сети некоторого входного вектора будет возбуждаться только один нейрон.

Каждый вес, связанный с выигравшим нейроном Кохонена, изменяется пропорционально разности между его величиной и величиной входа, к которому он присоединен. Для случая двух входов это можно представить геометрически (рис. 7.6).

Если бы с каждым нейроном Кохонена ассоциировался только один вектор, то слой Кохонена мог бы быть обучен путем простого присвоения весам значений этого входного вектора. Но поскольку обучение происходит по всем векторам, веса нейрона получают в

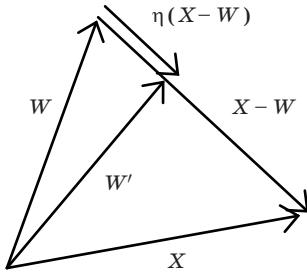


Рис.7.6. Сближение вектора весов
и входного вектора

конце концов усреднением по всем векторам, которые должны его активизировать.

Нормализованные входные и весовые векторы можно рассматривать как точки на поверхности единичной гиперсферы. Входные векторы группируются в классы (образы). Если переписать весовой вектор в область, соответствующую центру некоторого класса, то ИН будет возбуждаться при появлении на входе любого вектора этого класса.

Входные векторы, которым обучают слой Кохонена, обычно распределены по поверхности гиперсферы не равномерно, а занимают локальные ее участки. Если до обучения выбирать веса случайно, можно получить ситуацию, при которой большинство весов будет настолько удалено от входных векторов, что многие нейроны Кохонена всегда будут иметь нулевой выход. Поэтому необходимо, чтобы в окрестности каждого входного вектора находились весовые векторы. Для того чтобы добиться этого, применяют *метод выпуклой комбинации* [25].

Все веса приравниваются к одной и той же величине $1/\sqrt{n}$, где n – размерность входного и весового векторов, т. е. все компоненты весовых векторов совпадают и сами они имеют единичную длину. Каждой компоненте вектора входа X придается значение

$$\alpha x_i + \left(\frac{1}{\sqrt{n}} \right) (1 - \alpha).$$

Сначала α мало, и все входные компоненты близки к весовым компонентам. Потом α увеличивают, и входной вектор становится все более похож на самого себя. Весовые векторы отслеживают один вектор или небольшую их группу.

Кроме метода выпуклой комбинации существуют и другие методы. Например, к входным векторам можно добавлять шумы, чтобы каждый нейрон в конце концов захватил свой весовой вектор.

Еще один метод заставляет каждый нейрон Кохонена действовать «по справедливости», так, чтобы нейроны, которые возбуждаются чаще других, были подавлены, и могли обучаться все нейроны.

Пример 7.1. Рассмотрим простой пример обучения слоя конкурирующих нейронов. Пусть даны четыре входных вектора:

$$X_1 = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \quad X_2 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}, \quad X_3 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad X_4 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}.$$

Они должны быть кластеризованы сетью Кохонена, имеющей два нейрона. При обучении изменяются веса только нейрона-победителя.

Пусть веса нейронов получили некоторые случайные начальные значения (ненормализованные):

$$W_1^T = \begin{bmatrix} 0,2 \\ 0,6 \\ 0,5 \\ 0,9 \end{bmatrix}, \quad W_2^T = \begin{bmatrix} 0,8 \\ 0,4 \\ 0,7 \\ 0,3 \end{bmatrix}.$$

На первом шаге обучения подается входной вектор X_1 и вычисляется расстояние от него до весовых векторов (по формуле (7.1)):

$$d_1 = \|X_1 - W_1^T\| = \sqrt{\sum_{i=1}^4 (x_i - w_i)^2} \approx 1,36,$$

$$d_2 = \|X_2 - W_2^T\| = \sqrt{\sum_{i=1}^4 (x_i - w_i)^2} \approx 0,99.$$

Таким образом, победившим оказывается второй нейрон, и его веса подстраиваются ($\eta = 0,6$):

$$W_2^T = \begin{bmatrix} 0,8 \\ 0,4 \\ 0,7 \\ 0,3 \end{bmatrix} + 0,6 \cdot \left[\begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} - \begin{bmatrix} 0,8 \\ 0,4 \\ 0,7 \\ 0,3 \end{bmatrix} \right] = \begin{bmatrix} 0,92 \\ 0,76 \\ 0,28 \\ 0,12 \end{bmatrix}.$$

На втором шаге обучения подается входной вектор X_2 :

$$d_1 = \|X_2 - W_1^T\| = \sqrt{\sum_{i=1}^4 (x_i - w_i)^2} \approx 0,81,$$

$$d_2 = \|X_2 - W_2^T\| = \sqrt{\sum_{i=1}^4 (x_i - w_i)^2} \approx 1,5.$$

Победившим оказывается первый нейрон, и его веса подстраиваются:

$$W_1^T = \begin{bmatrix} 0,2 \\ 0,6 \\ 0,5 \\ 0,9 \end{bmatrix} + 0,6 \cdot \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} - \begin{bmatrix} 0,2 \\ 0,6 \\ 0,5 \\ 0,9 \end{bmatrix} = \begin{bmatrix} 0,08 \\ 0,24 \\ 0,20 \\ 0,96 \end{bmatrix}.$$

Аналогично подаются векторы X_3 и X_4 , корректируются веса, и первая эпоха обучения заканчивается.

На второй и последующих эпохах значение η постепенно уменьшается, происходит приближение весов к предельным значениям:

$$W_1^T = \begin{bmatrix} 0 \\ 0 \\ 0,5 \\ 1 \end{bmatrix}, \quad W_2^T = \begin{bmatrix} 1 \\ 0,5 \\ 0 \\ 0 \end{bmatrix}.$$

Таким образом, центр первого кластера (W_1) оказывается усреднением компонент векторов X_2 и X_4 , а центр второго кластера (W_2) – усреднением компонент векторов X_1 и X_3 .

7.3. Слой Кохонена

Архитектура соревновательного слоя приведена на рис. 7.7.

На рис. 7.7 R – размерность входного вектора \mathbf{p} и векторов весов; S_1 – число нейронов соревновательного слоя; блок C (competitive layers) обозначает вычисление выходов нейронов с учетом функции близости. Изменяющийся при обучении вектор смещений \mathbf{b} используется для охвата нейронами всего входного пространства.

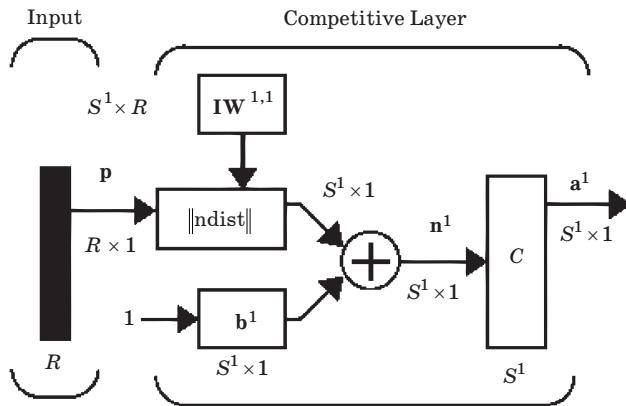


Рис. 7.7. Соревновательный слой Кохонена в MatLab

Слой Кохонена создается командой

```
net = newc(P, S, KLR, CLR),
```

где P – $(R \times Q)$ -матрица минимальных и максимальных значений Q входных векторов; S – число нейронов; KLR и CLR – константы скорости обучения весов и смещений.

Проанализируем примеры использования слоя Кохонена.

Пример 7.2. Рассмотрим создание слоя конкурирующих нейронов Кохонена для решения предыдущего примера. Сначала опишем входные векторы:

```
>> p=[1 0 1 0; 1 0 0 0; 0 0 0 1; 0 1 0 1]
p =
    1     0     1     0
    1     0     0     0
    0     0     0     1
    0     1     0     1
```

Затем создадим нейронную сеть:

```
>> net=newc([ 0 1; 0 1; 0 1; 0 1],2)
```

В этой команде массив указывает диапазоны для каждой входной переменной, а число – число нейронов соревновательного слоя. Начальные значения матрицы весов задаются в центре входных диапазонов.

Зададим число эпох обучения:

```
>> net.trainParam.epochs = 500
```

Обучение ИНС происходит по команде

```
>> net=train(net,p);
```

Проверим получившиеся значения весов:

```
>> ves=net.IW{1,1}
```

```
ves =
```

```
0.0000 0.0000 0.5135 1.0000
```

```
1.0000 0.4789 0.0000 0.0000
```

Значения оказались близкими к ожидаемым для этого примера.

Проверим работу сети:

```
>> p=[0 0 0 1];
```

```
>> Y=sim(net,p)
```

```
Y =
```

```
(1,1) 1
```

```
>> p=[1 0 0 0];
```

```
>> Y=sim(net,p)
```

```
Y =
```

```
(2,1) 1
```

Пример 7.3. Рассмотрим задачу кластеризации точек на плоскости. Разместим случайным образом 60 точек на плоскости с помощью команд:

```
>> A =[rand(1,20) - 0.7; rand(1,20) + 0.2];
>> B = [rand(1,20) + 0.7; rand(1,20) + 0.7];
>> C = [rand(1,20) + 0.2; rand(1,20) - 0.7];
>> plot(A(1,:),A(2,:),'bs')
>> hold on
>> plot(B(1,:),B(2,:),'r+')
>> plot(C(1,:),C(2,:),'go')
>> grid on
>> P = [A, B, C]
>> ncl = 3;
>> MN=[min(X(1,:)) max(X(1,:)); min(X(:,2)) max(X(:,2)) ]
>> net = newc(MN, ncl, 0.1, 0.0005);
>> net.trainParam.epochs=49;
>> net.trainParam.show=7;
>> net = train(net,P);
>> w = net.IW{1};
>> plot(w(:,1),w(:,2),'kp');
```

Результат обучения приведен на рис. 7.8, где звездочками обозначены центры трех кластеров.

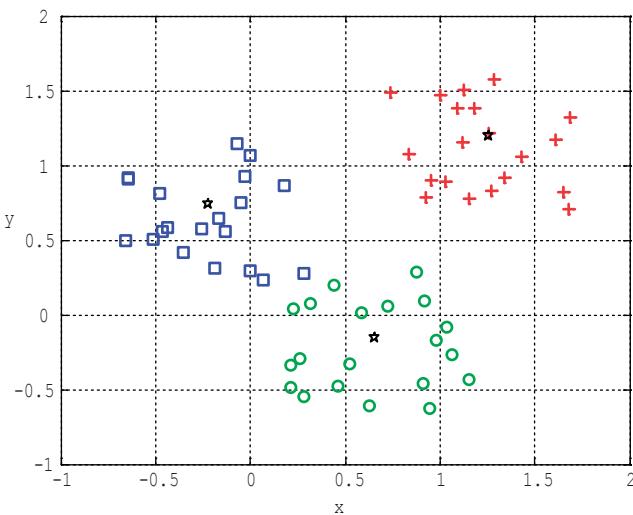


Рис. 7.8. Определение трех кластеров на плоскости

Пример 7.4. В рассматриваемом примере для создания кластеров используется специальная функция.

```
>> X=[0 1; 0 1];
>> clusters=8;
>> points=10;
>> std=0.05;
>> P=nngenc(X,clusters,points,std);
>> plot(P(1,:),P(2,:),'r');
>> hold on;
>> h=newc([0 1;0 1],8,.1);
>> h.trainParam.epochs=500;
>> h=init(h);
>> h=train(h,P);
>> w=h.IW{1};
>> plot(w(:,1),w(:,2),'ob');
>> xlabel('p(1)'); ylabel('p(2)');
```

Результат приведен на рис. 7.9.

Пример 7.5. В анализируемом примере кластеризация выполняется в пространстве.

```
>> d1 = randn(3,20); % кластер с центром в начале координат
```

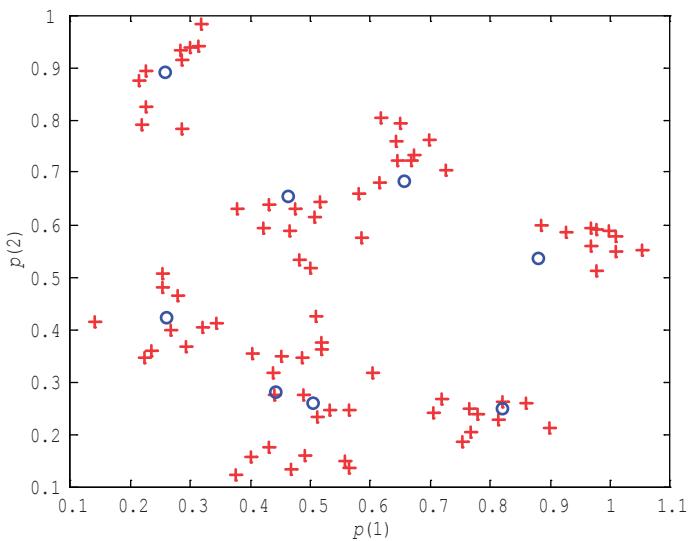


Рис. 7.9. Определение восьми кластеров на плоскости

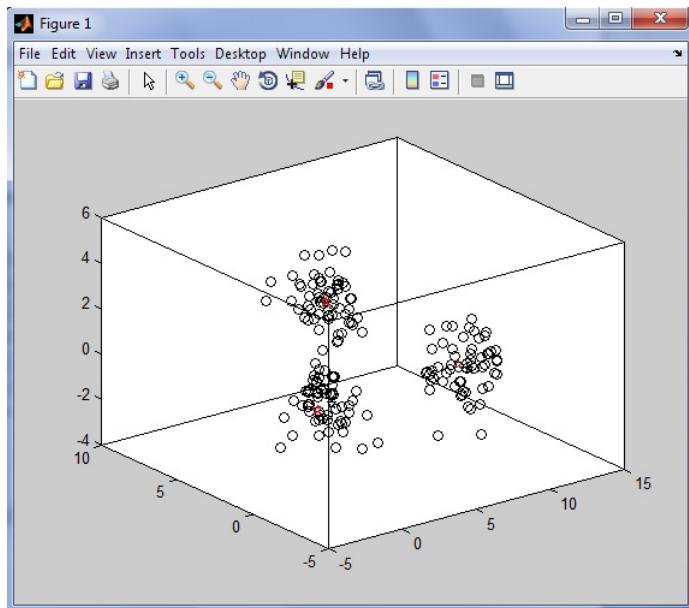


Рис. 7.10. Определение трех кластеров в пространстве

```

>> d2 = randn(3,20) + 3;    % кластер с центром (3, 3, 3)
>> d3 = randn(3,20); d3(1,:) = d3(1,:) + 9; % кластер с центром
(9, 0, 0)
>> d = [d1 d2 d3];          % массив для кластеризации
>> plot3(d(1,:), d(2,:), d(3,:), 'ko'), hold on, box on
>> net = newc(minmax(d), 3);
>> net.trainParam.epochs = 100;
>> net = train(net, d);
>> gcf,
>> w = net.IW{1};
>> plot3(w(:,1),w(:,2),w(:,3),'rp');

```

Результат кластеризации приведен на рис. 7.10 (центрам кластеров соответствуют звездочки).

7.4. Самоорганизующиеся карты Кохонена

Самоорганизующиеся карты Кохонена применяются для визуализации многомерных данных. Конечно, спроектировать многомерную выборку на плоскость без искажений невозможно, и SOM отражают лишь общую картину. Однако использование SOM позволяет анализировать особенности кластерной структуры многомерной выборки. Это бывает весьма важно для качественного анализа информации.

Идея заключается в том, чтобы спроектировать все объекты выборки на плоскую карту, точнее, на множество узлов прямоугольной сетки заранее заданного размера $M \times N$, которые могут иметь порядок десятков или сотен. Для того чтобы карта отражала кластерную структуру выборки, близкие объекты должны попадать в близкие узлы сетки. Нейроны SOM будут упорядочиваться так, чтобы иметь равномерное распределение при равномерном распределении входных векторов. Если же входные векторы распределены неравномерно, то и нейроны будут стремится попасть в центр кластеров.

Таким образом, SOM решает две задачи:

- понижение *размерности* данных с минимальной потерей информации (*анализ главных компонент* данных, выделение наборов независимых признаков);
- уменьшение *разнообразия* данных путем выделения конечно-го набора прототипов и отнесения данных к одному из таких типов (т. е. кластеризация).

Самоорганизующаяся карта Кохонена создается с помощью команды

```
net = newsom(PR,[d1,d2,...], tFcn, dFcn, olr, osteps, tlr, tnd),
```

где PR – $(R \times 2)$ -матрица минимальных и максимальных значений R входных элементов; d_i – размерность слоя (по умолчанию [5 8]); $tFcn$ – выбор топологии (по умолчанию 'hextop'); $dFcn$ – функция дистанции (по умолчанию 'linkdist'); olr – параметр скорости обучения на этапе размещения (по умолчанию 0,9); $osteps$ – число циклов обучения на этапе подстройки (по умолчанию 1000); tlr – параметр скорости на этапе подстройки (по умолчанию 0,02); tnd – размер окрестности на этапе подстройки (по умолчанию 1).

Функция $hextop$ соответствует гексагональной, $gridtop$ – прямоугольной и $randtop$ – случайной топологии.

Функция дистанции $dist$ соответствует евклидову расстоянию, $boxdist$ – максимальное координатное смещение, $mandist$ – расстояние суммарного координатного смещения, $linkdist$ – расстояние связи.

Настройка SOM производится по каждому входному вектору. Прежде всего определяется нейрон-победитель и корректируются его вектор весов и векторы соседних нейронов согласно соотношению

$$dW = lr \cdot A2 \cdot (P' - W),$$

где lr – параметр скорости обучения, равный olr для этапа упорядочения нейронов и tlr – для этапа подстройки; $A2$ – массив соседства для нейронов, расположенных в окрестности нейрона-победителя с номером i :

$$A2(i,q) = \begin{cases} 1, & a(i,q)=1, \\ 0,5, & a(i,q)=1, D(i,j) \leq nd, \\ 0 & \text{в остальных случаях.} \end{cases}$$

Здесь $a(i,q)$ – элемент выхода нейронной сети; $D(i, j)$ – расстояние между нейронами i и j ; nd – размер окрестности нейрона-победителя.

Таким образом, вес нейрона-победителя изменяется пропорционально половинному параметру скорости обучения, а веса соседних нейронов – пропорционально половинному значению этого параметра.

Весь процесс обучения карты Кохонена делится на этапы:

- упорядочения векторов весовых коэффициентов в пространстве признаков;

- подстройки весов нейронов по отношению к набору векторов входа.

На этапе упорядочения используется фиксированное число шагов. Начальный размер окрестности назначается равным максимальному расстоянию между нейронами для выбранной топологии и затем уменьшается до величины, используемой на следующем этапе. Он вычисляется по формуле

$$nd = 1,00001 + (\max(d) - 1)(1 - s/S),$$

где $\max(d)$ – максимальное расстояние между нейронами; s – номер текущего шага; S – число циклов на этапе упорядочения. Параметр скорости обучения изменяется по правилу

$$lr = tlr + (olr - tlr)(1 - s/S).$$

На этапе подстройки, который продолжается в течение оставшейся части процедуры обучения, размер окрестности остается постоянным и равным

$$nd = tnd + 0,00001,$$

а параметр скорости обучения изменяется по следующему правилу:

$$lr = tlrS/s.$$

Параметр скорости обучения продолжает уменьшаться, но очень медленно. Малое значение окрестности и медленное уменьшение параметра скорости обучения хорошо настраивают сеть при сохранении размещения, найденного на предыдущем этапе. Число шагов на этапе подстройки должно значительно превышать число шагов на этапе размещения. На этом этапе происходит тонкая настройка весов нейронов по отношению к набору векторов входов.

Нейроны карты Кохонена упорядочиваются так, чтобы при равномерной плотности векторов входа данные нейроны также были распределены равномерно. Если векторы входа распределены неравномерно, то и нейроны приобретают тенденцию распределяться в соответствии с плотностью размещения векторов входа.

Пример 7.6. Аппроксимация параболы одномерным слоем из десяти нейронов:

```
>> x=-1:0.05:1;
>> y=x.*x;
>> P=[x; y];
>> net=newsom([0 1;0 1],[10]);
>> net.trainParam.epochs =1000;
>> net1=train(net,P);
```

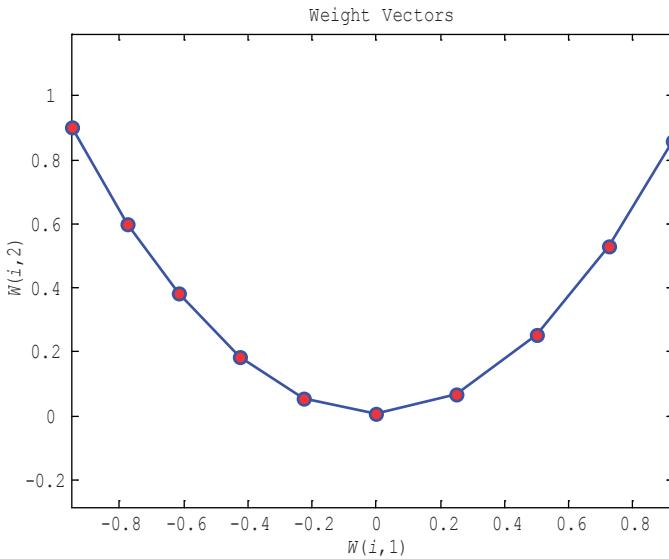


Рис. 7.11. Использование одномерной SOM

```
>> plotsom(net1.iw{1,1},net1.layers{1}.distances)
```

Результат обучения SOM приведен на рис. 7.11.

При подаче на вход сети конкретного вектора можно узнать номер сработавшего нейрона:

```
>> a=sim(net,[-0.6;0.4])
a =
(8,1)    1
```

Пример 7.7. Кластеризация случайно заданного массива из 40 точек на плоскости:

```
>> P = [rand(1,40)*2; rand(1,40)];
>> net = newsom([0 2; 0 1],[3 5]);
>> net = train(net,P);
>> plot(P(1,:),P(2,:),'g','markersize',20)
>> hold on
>> plotsom(net.iw{1,1},net.layers{1}.distances)
```

Результат кластеризации приведен на рис. 7.12.

В MatLab имеются дополнительные средства анализа SOM.

Функция

```
>> plotsompos(net,P);
```

порождает график, аналогичный показанному на рис. 7.12. Функция
`>> plotsomnd(net);`
позволяет визуализировать дистанцию между весами нейронов.
Пример приведен на рис. 7.13.

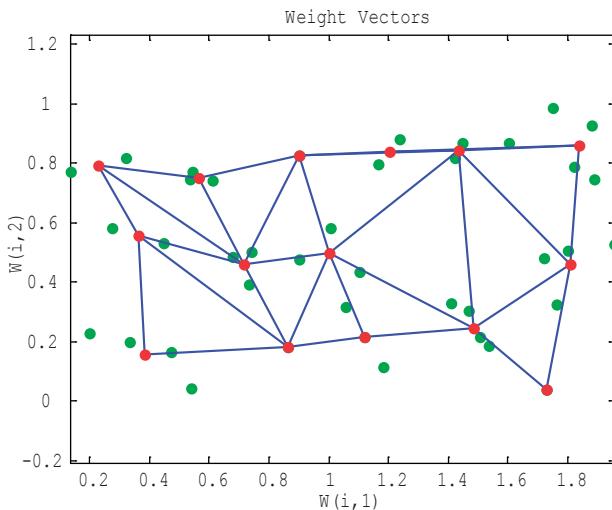


Рис. 7.12. Использование двумерной SOM

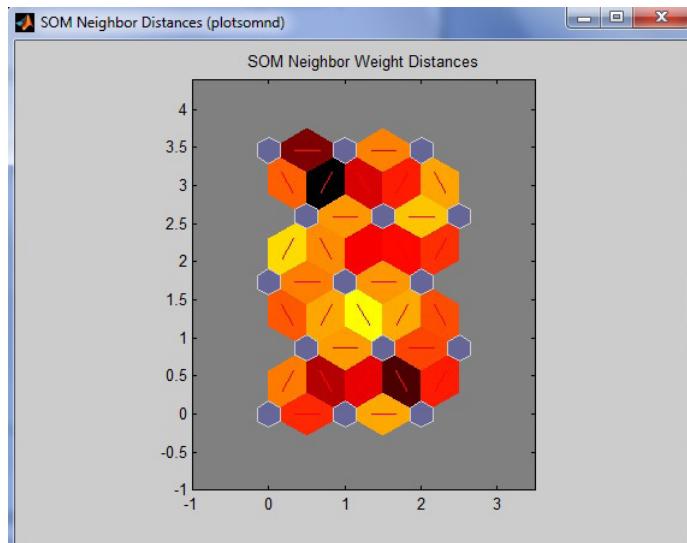


Рис. 7.13. Визуализация дистанции между весами SOM

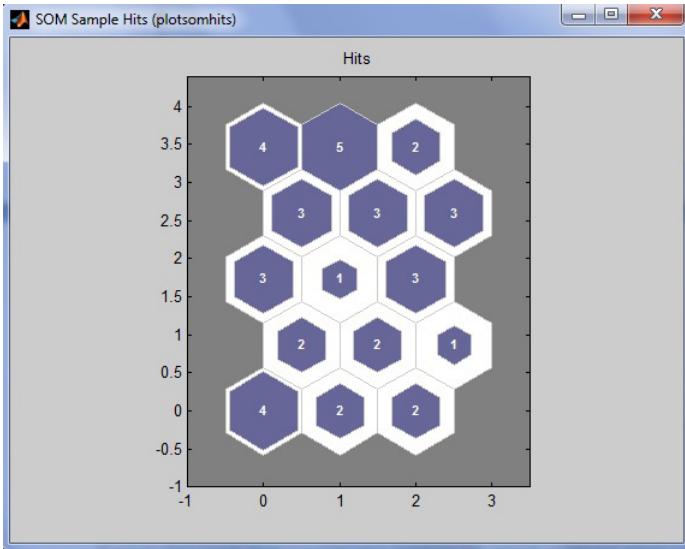


Рис. 7.14. Визуализация числа векторов, ассоциированных с нейронами двумерной SOM

Еще одна функция позволяет узнать число входных векторов, на которые реагирует каждый нейрон:

```
>> plotsomhits(net2,P);
```

Пример приведен на рис. 7.14.

Удобным инструментом визуализации данных является раскраска топографических карт аналогично тому, как это делают на обычных географических картах. Каждый признак данных порождает свою раскраску ячеек карты – по среднему значению этого признака у данных, попавших в данную ячейку.

7.5. Нейронные сети классификации

Нейронные сети для классификации входных векторов, или LVQ-сети (Learning Vector Quantization), выполняют и кластеризацию, и классификацию входных векторов. Это оказывается возможным благодаря добавления к слою Кохонена еще одного слоя – линейного (рис. 7.15).

Слой Кохонена (конкурирующий слой) способен поддерживать до S^1 кластеров (т. е. один нейрон на один кластер).

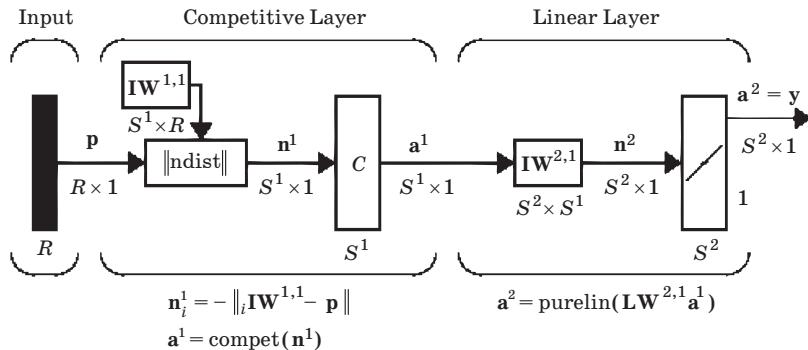


Рис. 7.15. Структура LVQ-сети

Линейный слой соответствует S^2 классам (где $S^2 \leq S^1$). При этом к одному классу могут принадлежать несколько кластеров.

Создание LVQ-сети происходит с помощью команды

`net = newlvq(PR, S1, PC, LR, LF),`

где PR – массив $R \times 2$ минимальных и максимальных значений для R элементов вектора входа; S^1 – число нейронов конкурирующего слоя; PC – вектор с S^2 элементами, определяющими процентную долю принадлежности входных векторов к известному классу; LR – параметр скорости настройки; LF – имя функции настройки.

Поскольку заранее известно, как кластеры 1-го слоя соотносятся с целевыми классами 2-го слоя, это позволяет заранее задать элементы матрицы весов последнего. Однако, чтобы найти правильный кластер для каждого вектора обучающего множества, необходимо выполнить процедуру обучения сети.

Обучение LVQ-сети происходит на основе обучающей последовательности, в которой каждый целевой вектор имеет единственный элемент, равный единице, а остальные элементы равны нулю:

$$\{p_1, t_1\}, \{p_2, t_2\}, \dots, \{p_N, t_N\}.$$

Пример 7.8. Даны десять векторов (точек на плоскости). Требуется разделить их на два класса в соответствии с целевым вектором T :

```
>> P = [-3 -2 -2 0 0 0 0 2 2 3; 0 1 -1 2 1 -1 -2 1 -1 0];
>> Tc = [1 1 1 2 2 2 2 1 1 1]; % индексы классов;
>> I1 = find(Tc==1); % вектор индексов первого класса
>> I2 = find(Tc==2); % вектор индексов второго класса
>> figure(1), clf, axis([-4,4,-3,3]), hold on
>> plot(P(1,I1),P(2,I1),'r')
```

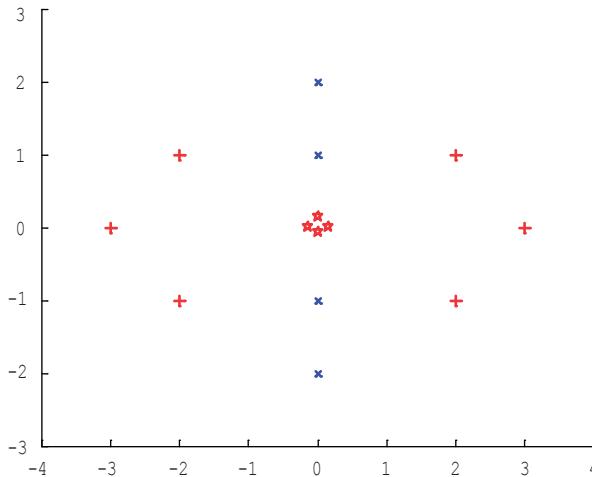


Рис. 7.16. Результат обучения LVQ-сети

```
>> plot(P(1,I2),P(2,I2),'xb')
>> T = ind2vec(Tc); % разреженная целевая матрица;
>> T = full(T); % полная целевая матрица;
>> net = newlvq(minmax(P), 4, [0.6 0.4]);
>> net.trainParam.epochs = 200;
>> net.trainParam.lr = 0.05;
>> net.trainParam.goal = 1e-5;
>> net = train(net,P,T);
>> w = net.IW{1};
>> plot(w(:,1),w(:,2),'rp');
```

Результат обучения приведен на рис. 7.16, где звездочки соответствуют центрам кластеров. Проверить работу сети можно с помощью команд

```
>> Y = sim(net,P);
>> Yc = vec2ind(Y)
Yc =
    1    1    1    2    2    2    2    1    1    1
```

Пример 7.9. Пусть на плоскости задано несколько групп точек, которые требуется разбить на два класса, обозначенных крестиками и кружками (рис. 7.17):

```
>> A = [rand(1,20) + 0.2; rand(1,20) + 0.2];
>> B = [rand(1,20) + 1.2; rand(1,20) + 1.2];
>> C = [rand(1,20) + 2.2; rand(1,20) + 2.2];
```

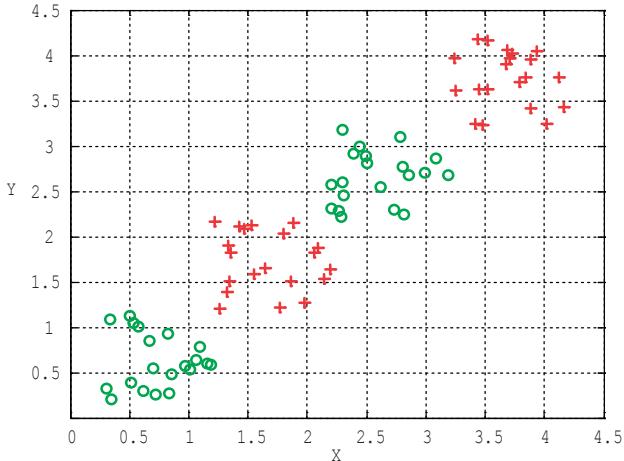


Рис. 7.17. Исходные данные для классификации

```
>> D = [rand(1,20) + 3.2; rand(1,20) + 3.2];
>> plot(A(1,:),A(2,:),'go')
>> hold on
>> plot(B(1,:),B(2,:),'r+')
>> plot(C(1,:),C(2,:),'go')
>> plot(D(1,:),D(2,:),'r+')
>> grid on
```

Сформируем данные для обучения:

```
>> t1=ones([1 20]); t2=t1+t1;
>> Tc=[t1 t2 t1 t2];
>> T = ind2vec(Tc);
>> P = [A B C D];
```

Определим LVQ-сеть и запустим процесс обучения:

```
>> net = newlvq(minmax(P), 4, [0.5 0.5]);
>> net.trainParam.epochs = 500;
>> net = train(net,P,T);
```

Проверку можно выполнить командами (рис. 7.18)

```
>> Y = sim(net,P);
>> Yc = vec2ind(Y);
>> figure; hold;
>> for i=1:length(P)
>> plot(P(1,i), P(2,i), ['.' colors(Yc(i))]);
```

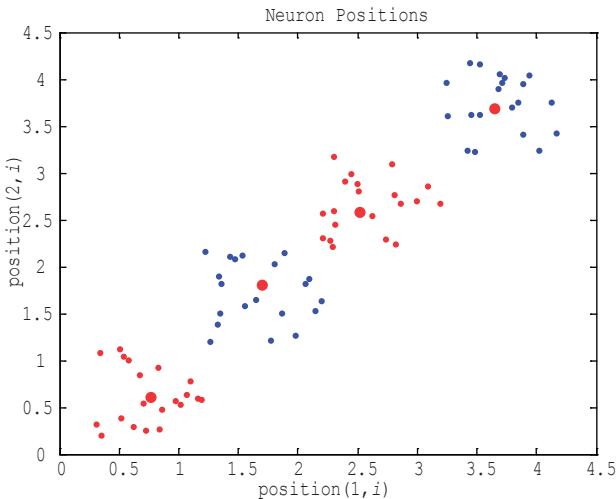


Рис. 7.18. LVQ-сеть в режиме классификации

```
>> end
>> plotsom(net.iw{1,1});
```

Вопросы для самопроверки

1. Какие задачи решают нейронные сети Кохонена?
2. Сколько слоев имеет нейронная сеть Кохонена?
3. В чем сходство работы сетей Кохонена и функционирования мозга?
4. Какие существуют варианты сетей Кохонена?
5. Как можно описать функцию близости нейронов Кохонена?
6. Что такое топология карты Кохонена? Какие бывают варианты топологий?
7. Как вычисляется близость векторов в сетях Кохонена?
8. Как описываются соревновательные связи между нейронами Кохонена?
9. Каков механизм реализации обучения без учителя в сетях Кохонена?
10. Какие проблемы могут возникать при обучении сетей Кохонена?
11. Что такое метод выпуклой комбинации?

12. Какие параметры задаются в MatLab при описании слоя Кохонена?
13. В чем заключается задача кластеризации точек на плоскости?
14. Как оценить качество решения задачи кластеризации?
15. Как используются самоорганизующиеся карты Кохонена?
16. Какие задачи выполняет SOM?
17. Какие параметры задаются в MatLab при описании SOM?
18. Какие этапы выполняются при обучении SOM?
19. Какие функции визуализации могут быть использованы при анализе SOM?
20. Для чего используются LVQ-сети?
21. Сколько слоев имеет LVQ-сеть?
22. Какие параметры задаются в MatLab при описании LVQ-сети?
23. В каких случаях кластеризация сеть Кохонена не может, а LVQ-сеть может обеспечить решение?

8. СТОХАСТИЧЕСКИЕ МЕТОДЫ ОБУЧЕНИЯ НЕЙРОННЫХ СЕТЕЙ

8.1. Задача коррекции динамической системы

Рассмотрим традиционную постановку задачи коррекции динамической системы.

Пусть задана исходная (располагаемая) динамическая система, описываемая передаточной функцией (ПФ) $W(s)$ (рис. 8.1, а). Если эта система неустойчива или не удовлетворяет заданным показателям качества, то ее поведение можно улучшить включением последовательного корректирующего устройства с ПФ $K(s)$ (рис. 8.1, б).

Для линейных систем задача синтеза корректирующего звена хорошо изучена. Для ее решения можно использовать, например, частотный синтез.

Если же рассматривается нелинейный объект управления, то универсальный подход к синтезу корректирующего звена обеспечивается использованием схемы, приведенной на рис. 8.2, где эталонная модель описывает поведение желаемой замкнутой системы.

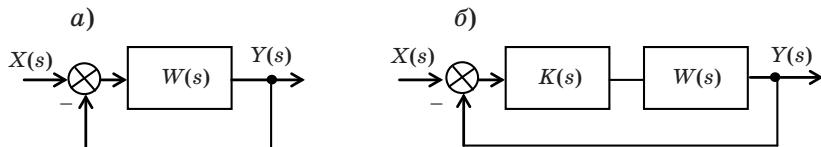


Рис. 8.1. Последовательная коррекция динамической системы

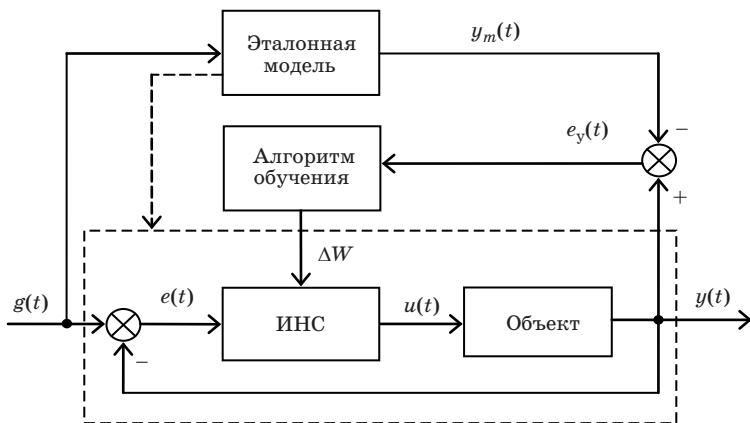


Рис. 8.2. Обучение последовательного нейросетевого регулятора

Задача обучения ИНС представляет собой задачу глобальной оптимизации некоторой целевой функции, описывающей качество функционирования ИНС.

8.2. Методы глобальной оптимизации

Задача глобальной минимизации описывается следующим образом:

$$F(X) \rightarrow \min_{x \in D}, \quad D \subset R^n,$$

где D – допустимое множество решений задачи; R^n – n -мерное евклидово пространство.

Можно рассматривать множество оптимальных решений задачи

$$X^{\text{opt}} = \arg \min_{x \in D} F(X)$$

и множество локально-оптимальных решений

$$X^{\text{lopt}} = \{X \in D : \exists \varepsilon > 0 : F(X) \leq F(X') \quad \forall X' \in B_\varepsilon(X) \cap D\},$$

где

$$B_\varepsilon(X) = \{X' \in R^n : |X' - X| < \varepsilon\}.$$

Таким образом, задача глобальной оптимизации заключается в поиске точки

$$X^{\text{opt}} \neq X^{\text{lopt}}.$$

Следует подчеркнуть, что в общем случае нельзя гарантировать точное решение задачи глобальной оптимизации за конечное число шагов. Для доказательства того, что найденное решение является глобальным оптимумом, необходимо выполнить полный перебор всех возможных значений вектора параметров. В большинстве случаев это невозможно, поэтому при глобальной оптимизации речь идет обычно о поиске не оптимального, а *субоптимального* решения:

$$X \in X^{\text{opt}}(\varepsilon) = \{X \in D : F(X) \leq F(X^{\text{opt}}) + \varepsilon\},$$

где ε – малая величина (с учетом специфики решаемой задачи).

Все известные методы глобальной оптимизации можно разделить на две категории: детерминированные и стохастические (рис. 8.3).

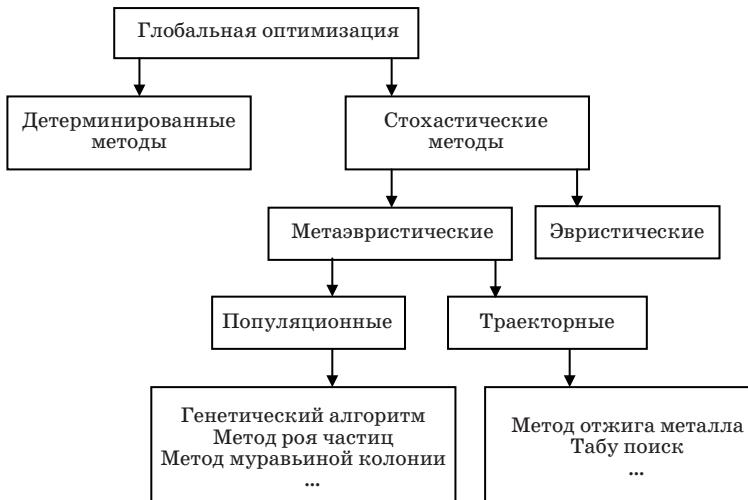


Рис. 8.3. Методы глобальной оптимизации

Детерминированные методы мало пригодны для решения задач глобальной оптимизации в больших поисковых пространствах, когда оптимизируемая функция имеет множество экстремумов, поэтому на практике широко используются стохастические методы глобальной оптимизации.

Стохастические эвристические методы работают по методу проб и ошибок. Смысл же определения «метаэвристические» заключается в том, что эти методы стохастической оптимизации сочетают в себе локальные поисковые процедуры нижнего уровня с глобальными стратегиями верхнего уровня.

Считается, что термин «метаэвристика» был впервые введен в работе [55]. Метаэвристики призваны обеспечивать поиск субоптимального решения сложной технической проблемы за приемлемое для разработчика время.

Метаэвристические методы делятся на два класса: популяционные и траекторные (рис. 8.3).

Траекторные методы предполагают, что в каждый момент времени рассматривается только одна точка поискового пространства, траектория движения которой должна постепенно приближаться к оптимуму. Траекторным методом является, в частности, метод отжига металла [11]. Траекторные методы не могут конкурировать с популяционными методами в больших поисковых пространствах,

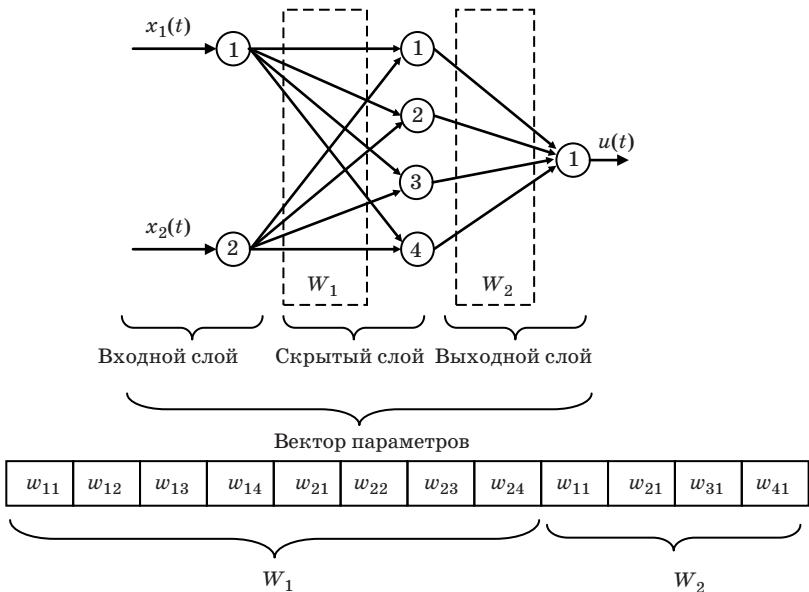


Рис. 8.4. Кодирование параметров ИНС

к тому же популяционные методы естественным образом допускают использование множества вычислителей.

Популяционные метаэвристические методы основаны на одновременном использовании множества точек поиска (популяции). Популяционные метаэвристические методы относятся к алгоритмам, основанным на биологическом подобии (*biologically inspired algorithms*). Для обозначения этого направления часто привлекается также термин *natural computing* (природные вычисления) [56, 57]. Самой известной метаэвристикой следует считать *генетический алгоритм* ([12, 51, 58] и др.).

Следует подчеркнуть, что метаэвристические методы являются предметно-независимыми, т. е. решаемая задача находит здесь отражение только в структуре оптимизируемого вектора, кодирующего решение.

Пример кодирования вектора параметров ИНС приведен на рис. 8.4.

При использовании метаэвристических методов кодируются, как правило, веса ИНС, но, в принципе, можно кодировать и структуру ИНС или параметры активационных функций. Важно лишь, чтобы все векторы популяции имели одинаковую длину.

8.3. Метод имитации отжига

Основная идея метода моделирования отжига (*simulated annealing*) [11, 59] исходит из физики процесса замерзания жидкостей или рекристаллизации металлов в процессе отжига. Целевая функция здесь является аналогом равновесия термодинамической системы и видоизменяется путем добавления случайных величин (условий температурного режима). Процесс повторяется достаточное число раз для каждой температуры, после чего она понижается и весь процесс происходит снова до состояния полной заморозки. Избегание попадания в незначительные локальные минимумы (замерзание) зависит от «схемы отжига», выбора начальной температуры, числа итераций для каждой температуры и насколько уменьшается температура на каждом шаге процесса «охлаждения».

В расплавленном металле атомы находятся в сильном беспорядочном движении, а по мере охлаждения все большее их число переходит в состояние с меньшими энергиями, пока в конце концов не будет достигнут глобальный минимум энергии. Распределение энергетических уровней при этом описывается формулой

$$P(E) = \exp\left(\frac{E}{kT}\right),$$

где $P(E)$ – вероятность нахождения системы в состоянии с уровнем энергии E ; k – постоянная Больцмана; T – температура по Кельвину.

При высоких температурах вероятность любого энергетического состояния близка к единице, а при уменьшении T вероятность высоких энергий уменьшается.

Алгоритм глобальной оптимизации с помощью имитации отжига представляет собой аналог физического процесса. Классический алгоритм имитации отжига можно описать следующим образом:

1. Искусственная температура получает максимальное значение: $T = T_{\max}$. Модельное время обнуляется: $t = 0$.
2. Выбирается начальная точка (аргумент функции) $x = x_0$.
3. Рассчитывается целевая функция $F(x)$.
4. Аргумент получает приращение $x' = x + \Delta x$.
5. Рассчитывается целевая функция $F(x')$.
6. Рассчитывается изменение целевой функции $\Delta F = F(x') - F(x)$.
7. Если $\Delta F < 0$, то $x = x'$ (значение аргумента сохраняется).
8. Если $\Delta F > 0$, то вероятность сохранения ΔF (и соответственно x') вычисляется по формуле

$$P(\Delta F) = \exp\left(-\frac{\Delta F}{kT}\right),$$

где k – константа, выбираемая из эвристических соображений.

Затем $P(\Delta F)$ сравнивается со случайным числом $n \in [0,1]$.

Если $P(\Delta F) > n$, то $x = x'$, при других условиях x не изменяется.

9. Искусственная температура уменьшается, модельное время увеличивается: $t = t + \Delta t$, происходит возврат к п. 4, и так до тех пор, пока T не уменьшится до заданного порогового значения.

Таким образом, пока искусственная температура T большая, алгоритм отжига может делать большие шаги даже в направлениях, увеличивающих значение минимизируемой функции. В результате этой особенности оказывается возможным попасть в окрестность глобального минимума. При этом, конечно, важно правильно определить начальное значение T_{\max} и задать закон изменения температуры. Например, эта величина должна быть обратно пропорциональна логарифму t :

$$T(t) = \frac{T_{\max}}{\log(1+t)}.$$

Данная формула означает, что искусственная температура должна меняться медленно.

Пример 8.1. Использование алгоритма отжига металла для поиска минимума функции одной переменной:

```
>> tm=100;
>> K=1;
>> x1=-10;
>> Tm=500;
>> F1=test(x1);
>> for i=1:200
    z=rand(1);
    if z > 0.5 z = 1; else z = -1; % направление шага – вперед
                                   или назад
    end;
    dx(i) = z*rand(1)*(2*Tm/500); % максимальный шаг
                                   равен 5
    x2(i)=x1+dx(i); % приращение аргумента
                      функции
    if x2(i) > 10 x2(i)=10; % проверка
    elseif x2(i) < -10 x2(i)=-10; % границы зоны поиска
    end;
```

```

F2(i) = test(x2(i));           % новое значение целевой функции
Fd(i) = F2(i) - F1;          % приращение целевой функции

P(i) = 0;
if Fd(i) < 0 x1 = x2(i); F1 = F2(i); elseif Fd(i) >= 0 P(i) = 1/exp(Fd(i)/K*Tm);
end;
if P(i) > rand(1) x1=x2(i); % обновление точки поиска
F1 = F2(i);                 % обновление целевой функции
end;
Tm=Tm-2.5; % уменьшение искусственной температуры
end;
>> figure; plot(x2); grid;      % график текущей точки поиска
>> figure; plot(F2); grid;      % график текущего значения целевой функции
>> figure; plot(Fd); grid;      % график приращения целевой функции

```

В примере использована следующая тестовая функция:

```

function s=test1(x)
s=20+x.^2+10.*cos(2*pi.*x))
end

```

Границы поиска определяются заранее: $x \in [-10, 10]$.

8.4. Генетический алгоритм

Генетический алгоритм (ГА) использует базовые представления об эволюции как о циклическом процессе смены популяций, в котором наибольшее потомство дают наиболее приспособленные особи, а новые свойства индивидуумов возникают при мутациях. Популяция состоит из хромосом, кодирующих решение некоторой задачи оптимизации.

Достаточно подробное изложение принципов использования ГА в MatLab можно найти в [51]. MatLab имеет в своем составе пакеты расширения (toolbox) Simulink и Control System toolbox. Пакет Simulink позволяет моделировать сложные нелинейные объекты управления и строить ИМ, учитывающие все значимые особенности объекта. Однако стандартные процедуры синтеза в Control System toolbox, такие, как частотный или модальный синтез, ориентированы на линейные системы. Синтез нелинейных систем управления (нейроконтроллеров) в общем случае связан с необходимостью оптимизации параметров регулятора.

Начиная с версии 7.0.1 в составе MatLab появился пакет GADS (Genetic Algorithm and Direct Search) toolbox, в котором реализован ГА. В версии 7.2.0 стало возможным вводить линейные и нелинейные ограничения на переменные, по которым ведется оптимизация, что дополнительно увеличило возможности пакета.

Использовать функции GADS можно с помощью обычных для MatLab приемов: из командной строки либо из файл-функций или файл-сценария. Все функции GADS являются М-файлами. Весьма удобно использовать графическое окно пользователя, которое вызывается командой gatool.

Рассмотрим принципы использования GADS toolbox в задачах синтеза нейросетевых регуляторов [60].

Генетический алгоритм использует стандартные представления об эволюции как процессе расширенного воспроизведения жизнеспособных особей и гибели неприспособленных. Эволюционные изменения особей происходят в результате работы генетических механизмов отбора, скрещивания и мутации. При синтезе системы управления в качестве «особи» выступает регулятор с конкретным набором параметров, кодируемым хромосомой. Его «жизнеспособность» соответствует качеству переходного процесса в системе. Таким образом, многократные эксперименты с имитационной моделью (ИМ) являются необходимостью при генетическом синтезе. Мощность современных компьютеров обеспечивает такую возможность.

Алгоритм эволюционного синтеза системы управления иллюстрирует рис. 8.5.

Рассмотрим алгоритм по шагам:

1. *Кодирование хромосом.* Хромосома состоит из генов, каждый из которых является параметром ИНС. Для представления генов могут использоваться двоичные или действительные числа. Помимо параметров хромосома может кодировать и (или) структуру регулятора.

2. *Выбор генетических операторов.* Работой ГА управляют три генетических оператора: селекция, скрещивание и мутация. Существуют различные модификации этих операторов, так что выбор конкретного варианта оказывается на скорости и качестве получения решения.

3. *Инициализация популяции.* Популяция состоит из множества хромосом – альтернативных вариантов параметров регулятора. При инициализации каждая хромосома получает случайные значения генов.

4. *Тестирование хромосом.* На этом шаге каждая хромосома должна получить индивидуальную оценку пригодности в соответ-

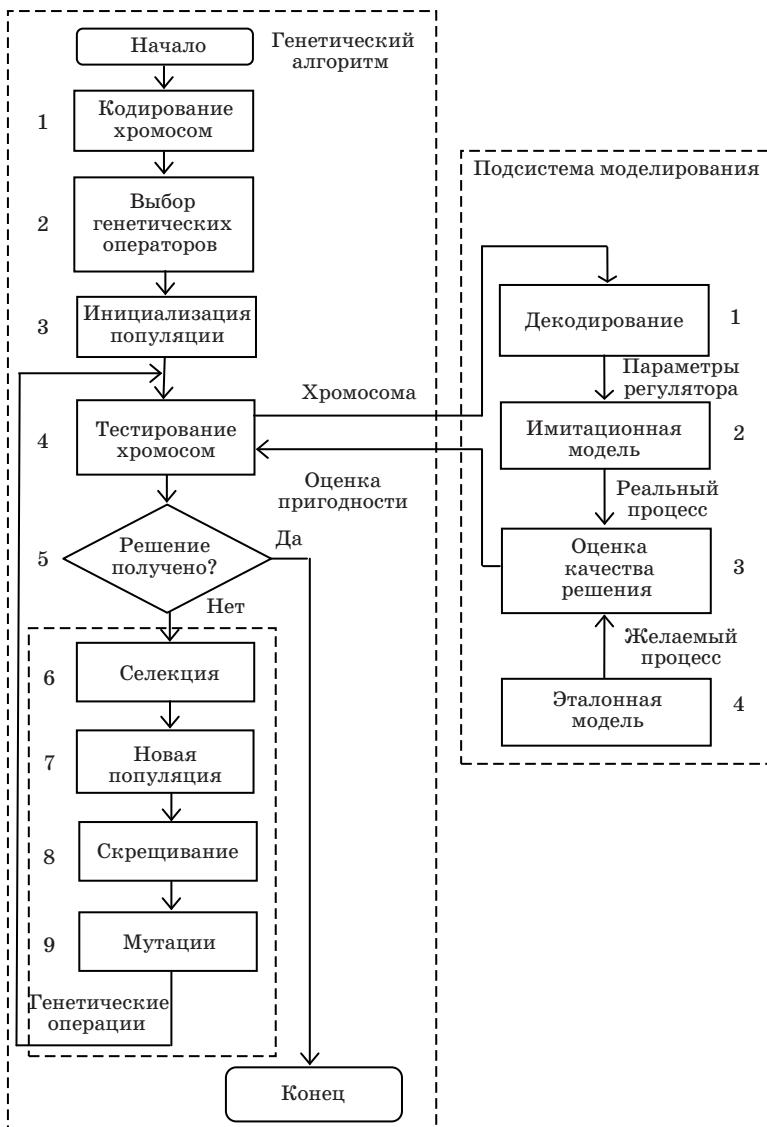


Рис. 8.5. Генетическое обучение регулятора

ствии с качеством решения задачи, которое она обеспечивает. Здесь необходимо использовать подсистему моделирования.

5. Проверка условий завершения работы алгоритма. Основным условием на этом шаге является достижение приемлемого качества

решения, при котором пригодность одной из хромосом превысит заданный порог. Дополнительными условиями выступают отсутствие улучшений на длительном интервале времени или просто истечение времени, отведенного на поиск решения.

Подсистема моделирования содержит две модели:

- *эталонную модель* (ЭМ), которая описывает желаемую реакцию объекта на тестовый сигнал. Такая модель может быть намного проще реального объекта, например, это может быть динамическое звено невысокого порядка или просто набор выходных координат в заданные моменты времени;
- *имитационную модель* системы управления, включающую в себя описание регулятора и объекта и реализованную в виде компьютерной программы или блок-схемы Simulink.

Эталонная и имитационная модели запускаются одновременно, порождая два переходных процесса: желаемый и реальный. Соответственно при моделировании с определенным шагом накапливаются два массива (вектора) точек. Пригодность хромосом обратно пропорциональна расстоянию между этими векторами.

Таким образом, блок оценки качества решения возвращает значение пригодности каждой хромосомы (фитнес-функцию), что необходимо для выполнения генетических операций.

Время работы ГА измеряется поколениями (генерациями). Каждое поколение заново подвергается тестированию и генетическим операциям.

Рассмотрим два примера использования ГА при синтезе нейронетевых регуляторов.

Пример 8.2. Генетический синтез нейроконтроллера двигателя постоянного тока.

Рассмотрим задачу управления скоростью вращения вала двигателя постоянного тока (ДПТ) с возбуждением от постоянных магнитов.

Математическая модель рассматриваемого ДПТ может быть выражена системой уравнений [61]:

$$\begin{cases} v_a = R_a i_a + L_a \frac{di_a}{dt} + e_a, \\ T_B = J \frac{d\omega_m}{dt} + T_L \operatorname{sgn}(\omega_m) + B_m \omega_m, \\ e_a = K \omega_m, \\ T_L = K i_a, \end{cases}$$

где v_a – напряжение питания обмотки якоря; R_a – активное сопротивление обмотки якоря; i_a – ток, индуцируемый в проводниках

обмотки якоря при вращении; L_a – индуктивное сопротивление обмотки якоря; e_a – ЭДС, индуцируемая в проводниках обмотки якоря при вращении; T_b – электромагнитный момент на валу ДПТ; J – момент инерции вращающихся частей якоря ДПТ; ω_m – угловая скорость вращения вала двигателя; B_m – коэффициент вязкого трения (коэффициент демпфирования); T_L – модуль момента сухого трения.

Электродвижущую силу самоиндукции обмотки якоря, обусловленную изменением тока, вычисляют по формуле

$$e = L_a \frac{di_a}{dt}.$$

Электромагнитный коэффициент

$$K = C\Phi$$

где C – конструктивная постоянная двигателя; Φ – магнитный поток.

Динамический момент (обусловленный момент инерции вращающихся частей двигателя)

$$T_{\text{дин}} = J \frac{d\omega_m}{dt}.$$

Статический момент, обусловленный силами трения, равен сумме моментов холостого хода $T_{x.x}$ и нагрузки T_h :

$$T_c = T_h + T_{x.x}.$$

Поскольку момент холостого хода мал по сравнению с моментом нагрузки,

$$T_{x.x} = 0 \text{ и } T_c = T_h.$$

Момент нагрузки

$$T_h = T_{\text{ст}} + T_{\text{вт}},$$

где $T_{\text{ст}} = T_L \operatorname{sgn}(\omega_m)$ – реактивный момент (момент сухого трения – момент от сжатия аморфных тел, трения, резания и т. п., препятствующий движению и изменяющий свой знак при изменении направления движения привода);

$T_{\text{вт}} = B_m \omega_m$ – демпферный момент (момент вязкого трения – момент от перемещения тела в жидкой среде).

При моделировании были использованы следующие параметры ДПТ:

$$R_a = 2 \text{ Ом}, L_a = 0,0052 \text{ Гн}, J = 1,5 \cdot 10^{-4} \text{ кг} \cdot \text{м}^2, B_m = 1 \cdot 10^{-3}, K = 0,1 \text{ Н} \cdot \text{м}/\text{А}.$$

Двигатель способен вращаться в обоих направлениях, реактивный момент нагрузки $T_{\text{ст}}$ в модели является знакопеременным и зависит от знака скорости ω_m (направления вращения вала двигателя). В связи с этим вводится блок определения знака (sgn) для скорости вращения вала ДПТ.

Присутствие сигнум-функции (sgn) в выражении для реактивного момента нагрузки $T_{\text{ст}}$ требуется, чтобы гарантировать пассивность нагрузки независимо от того, положительна скорость вращения вала двигателя или отрицательна. Наличие блока определения знака для реактивного момента нагрузки $T_{\text{ст}}$ в модели ДПТ делает эту модель нелинейной.

У ДПТ с возбуждением от постоянных магнитов возможно только якорное управление. Его характеристики аналогичны характеристикам управляемых двигателей с электромагнитным возбуждением. Якорное управление – изменение подводимого к якорю напряжения v_a при постоянном магнитном потоке [11].

В модели ДПТ используется схема управления, известная как подчиненное регулирование: два контура управления – по току и скорости. Главной управляющей величиной является напряжение питания обмотки якоря v_a . Контур по току включает в себя блок релейной характеристики (Relay), контур по скорости содержит регулятор.

Блок Relay – блок задержки сигнала, он реализует релейную характеристику с гистерезисом – имитирует работу широтно-импульсного модулятора, который входит в контур управления по току. Этот блок вводит в модель системы управления ДПТ дополнительную нелинейность.

Таким образом, модель ДПТ оказывается существенно нелинейной, что обосновывает возможность использования нейроконтроллера в контуре системы управления.

Сложность управляющей НС должна соответствовать сложности объекта управления, поэтому в рассматриваемой задаче можно применить достаточно простую двухслойную НС прямого распространения, представленную на рис 8.6.

Регулятор получает на входе ошибку управления по скорости $e(t)$, которая распространяется через две линии задержки, так что на входы НС подаются текущее значение ошибки и два ее предыдущих значения. Таким образом, статическая НС превращается в динамическое корректирующее звено. Каждый из четырех нейронов НС использует линейную с насыщением активационную функцию.

Использование нейроконтроллера (НК) предполагает реализацию процедуры обучения, в процессе которой происходит измене-

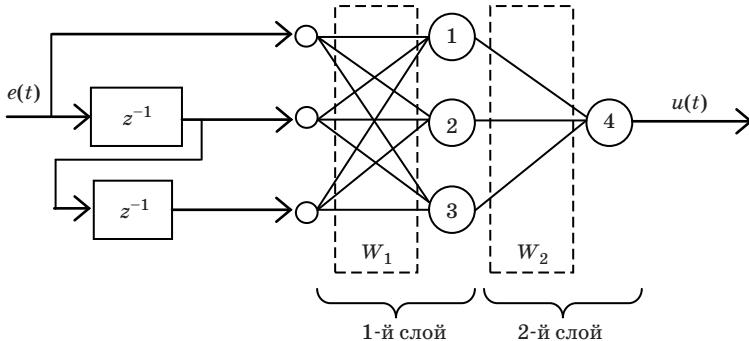


Рис. 8.6. Управляющая нейронная сеть

нение параметров НС с целью уменьшения ошибки управления. Параметрами НС являются веса межнейронных связей W_1 и W_2 , кодируемые действительными числами.

Хромосома в соответствии с рис. 8.6 имеет длину 12 генов (число весов НС). Популяция хромосом эволюционирует под воздействием генетических операторов отбора, скрещивания и мутации.

Оптимизируемая функция представляется в виде *m*-файла:

```
function z=set2(X);
global k1; global k2; global k3; global k4; global k5; global k6;
global k7; global k8; global k9; global k10; global k11; global k12;
k1=X(1); k2=X(2); k3=X(3); k4=X(4); k5=X(5); k6=X(6);
k7=X(7); k8=X(8); k9=X(9); k10=X(10); k11=X(11); k12=X(12);
sim('dpt_NC');
z=sum(abs(simout-simout1));
end
```

В тексте *m*-файла присутствуют три части:

- описание переменных, по которым осуществляется поиск;
- обращение к блоку моделирования (*simulink model 'dpt_NC'*);
- вычисление фитнес-функции (функции относительной пригодности), которая оценивает качество управления для конкретной хромосомы.

Для реализации последней задачи используются блоки *simout*, накапливающие массив точек переходного процесса по управляемой переменной, сравниваемый с заданным процессом (уставкой).

На рис. 8.7 показана блок-схема модели ДПТ, собранная в MatLab Simulink.

Запуск процесса оптимизации происходит из командной строки или с помощью интерфейса *gatool* [51].

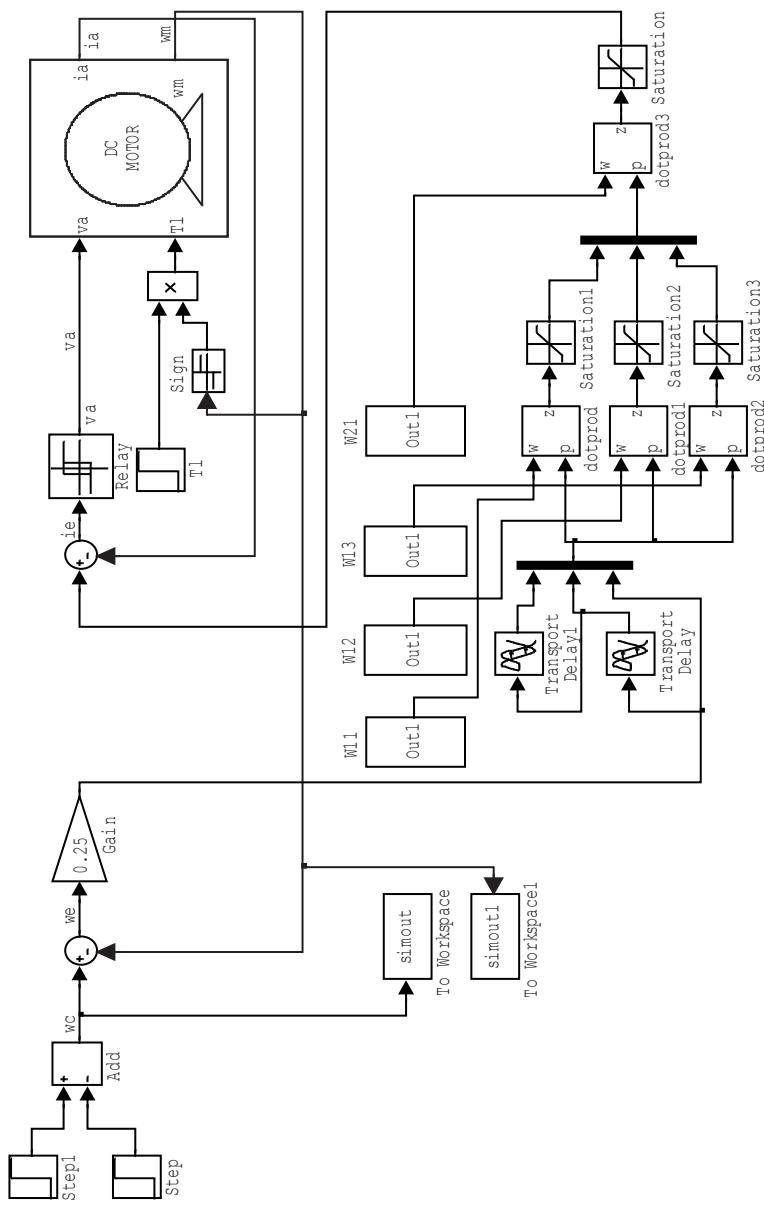


Рис. 8.7. Модель системы управления ДПТ в MatLab Simulink

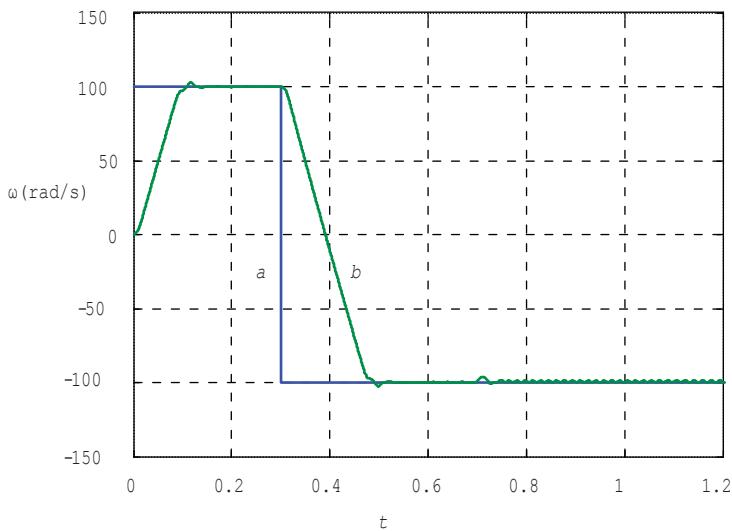


Рис. 8.8. Управление ДПТ: а – уставка по скорости; б – переходный процесс под управлением НК

Двигатель первоначально находится в бездействии и без нагрузки, потом подается команда скачка в скорость, затем, когда уставновившееся состояние достигнуто (скорость $\omega_m = 100$), срабатывает команда реверса скорости, сопровождаемая применением скачка нагрузки (на 0,07 с).

Процесс генетического обучения НК требует примерно 50–70 итераций при стандартном размере популяции 20 хромосом. После обучения были получены следующие значения весов НС:

$$W_1 = \begin{bmatrix} 3,6 & 2,4 & 8,6 \\ -2,3 & 1,7 & 5,3 \\ -0,47 & 3,5 & -2,3 \end{bmatrix}, \quad W_2 = [3,8 \quad 4,5 \quad -0,7].$$

На рис. 8.8 приведены переходные процессы в системе под управлением НК.

Как следует из рис. 8.6, использование НК обеспечивает достаточно высокое качество управления ДПТ. Полученный результат показывает целесообразность использования нейроконтроллеров в задачах управления электромеханическими объектами, математические модели которых содержат нелинейности.

Пример 8.3 (нейронный супервизор для управления нелинейным объектом).

Наиболее распространенным типом промышленных регуляторов в настоящее время являются ПИД-регуляторы. Около 90% регуляторов в промышленности используют ПИД-алгоритм. Причиной столь высокой популярности является простота построения и использования, ясность функционирования, пригодность для решения большинства практических задач и низкая стоимость [62].

Однако существующие методы расчета параметров ПИД-регуляторов ориентированы на линейные системы, поскольку сам регулятор является линейным динамическим звеном. Если же объект управления является существенно нелинейным, то трудно добиться высокого качества управления. В этой связи большой интерес представляют двухуровневые схемы, в которых на нижнем уровне располагается базовый ПИД-регулятор, а на верхнем – супервайзор, координирующий работу базового регулятора.

Поведение ПИД-регулятора описывается формулой

$$u(t) = K_p e(t) + K_d \frac{de(t)}{dt} + K_i \int e(t) dt,$$

где K_p , K_d , K_i – настраиваемые коэффициенты регулятора; $e(t)$ и $u(t)$ – ошибка и сигнал управления.

Общая схема ПИД-регулятора с нейронным супервайзором приведена на рис. 8.9 ($g(t)$ и $y(t)$ – задающее воздействие и выходной сигнал объекта).

Рассмотрим объект управления, в котором выделены линейная динамическая и нелинейная статическая часть (рис. 8.10).

Настройка ПИД-регулятора для линейной части (колебательное звено) средствами оптимизации MatLab не представляет трудностей (кривая 1 на рис. 8.11). Однако при введении нелинейности качество оптимизации коэффициентов заметно ухудшается (кривая 2 на рис. 8.11).

Таким образом, при постоянных значениях коэффициентов ПИД-регулятора не удается добиться хорошего качества управле-

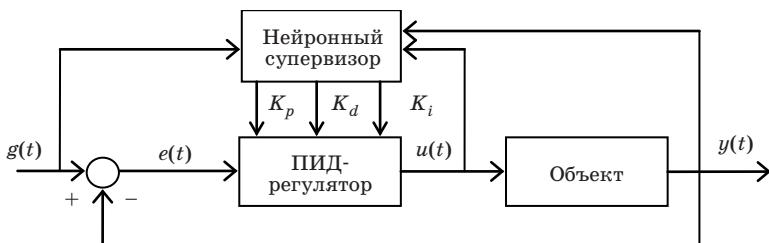


Рис. 8.9. Структура супервизорного управления ПИД-регулятором

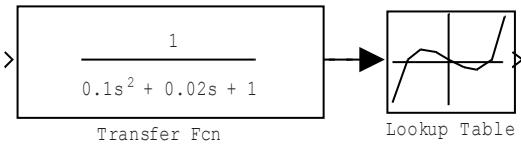


Рис. 8.10. Нелинейный объект управления в MatLab Simulink

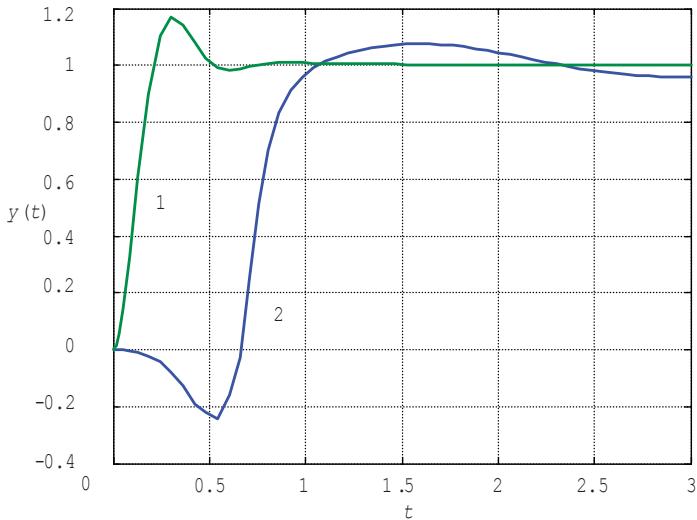


Рис. 8.11. Переходные процессы в системе управления

ния. Для оперативного изменения коэффициентов будем использовать двухслойную ИНС прямого распространения, содержащую по три нейрона в каждом слое (рис. 8.12).

На рис. 8.12 цифрами обозначены отдельные нейроны. Нейроны 1-го слоя имеют активационную функцию – гиперболический тангенс, а у нейронов 2-го слоя активационные функции линейные. На вход ИС поступают текущее и задержанное значения ошибки. Число линий задержки соответствует порядку динамической части объекта управления. Величина задержки зависит от скорости проекания переходного процесса (см. рис. 8.11), она может быть выбрана в пределах 0,01 – 0,1 с.

Задача конструирования нейросетевого супервизора подразумевает, таким образом, настройку двух весовых матриц W_1 и W_2 , содержащих по девять весов каждая. Использование для настройки алгоритма обратного распространения ошибки в данной ситуации оказывается затруднительным, поскольку невозможно сформировать обучающую выборку для ИС.

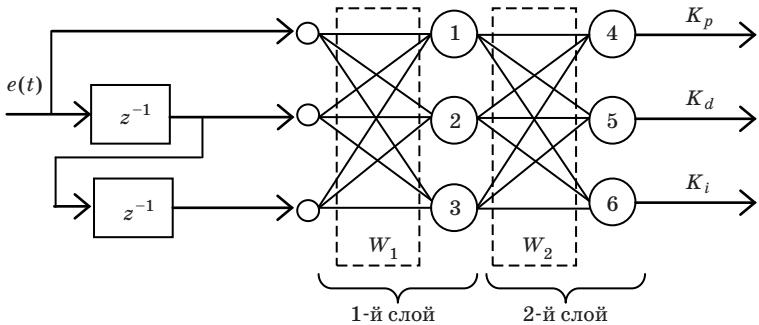


Рис. 8.12. Структура нейросетевого супервизора

Использование ГА предполагает кодирование параметров супервизора, т. е. весов НС, хромосомой длиной 18 действительных чисел (генов).

Хромосома представляет собой альтернативное решение задачи. Совокупность всех хромосом образует популяцию. Популяция эволюционирует под воздействием генетических операторов отбора, скрещивания и мутации. Для выполнения операции отбора необходимо иметь описание пригодности каждой хромосомы P_i . Эта оценка может быть найдена путем сравнения эталонного описания переходного процесса $y^*(t)$ и переходного процесса, полученного под управлением i -й хромосомы. Например,

$$P_i = \frac{1}{1 + E_i}, E_i = \sum_{k=1}^N |y_k^*(t) - y_k(t)|,$$

где N – рассматриваемое число точек переходного процесса.

На рис. 8.13 приведена схема моделирования работы нейросетевого супервизора в MatLab Simulink.

В нижней части рис. 8.13 представлена передаточная функция, служащая для описания эталонного процесса. Блоки simout и simout1 служат для накопления точек эталонного и реального переходного процесса.

Обучение супервизора выполнялось с помощью стандартного инструмента gatool в составе MatLab. Размер популяции 50 хромосом, для поиска решения потребовалось около 100 генераций. Были получены следующие весовые матрицы:

$$W_1 = \begin{bmatrix} -0,66 & 0,75 & 0,88 \\ 0,46 & 0,22 & 3,05 \\ 0,79 & 0,99 & -0,45 \end{bmatrix}, \quad W_2 = \begin{bmatrix} 0,41 & 1 & 0,97 \\ 0,85 & 1,89 & 1,52 \\ 0,84 & 0,09 & 0,02 \end{bmatrix}.$$

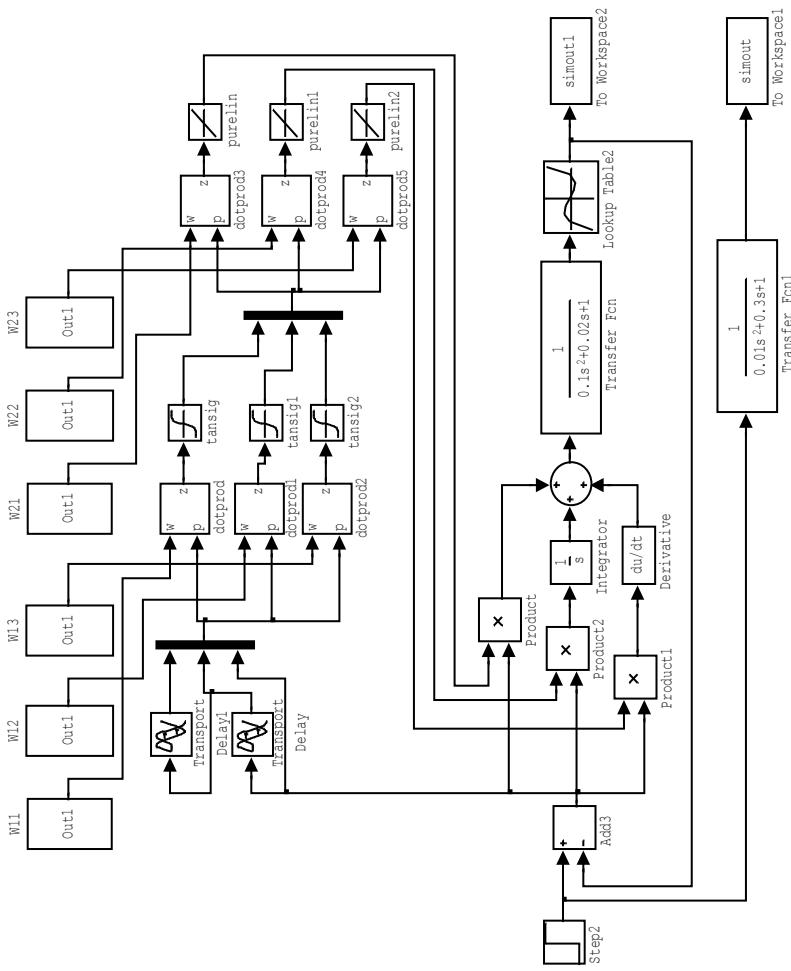
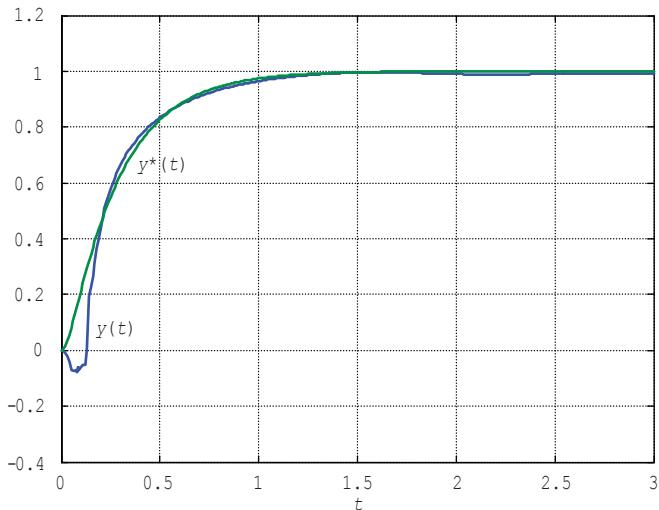
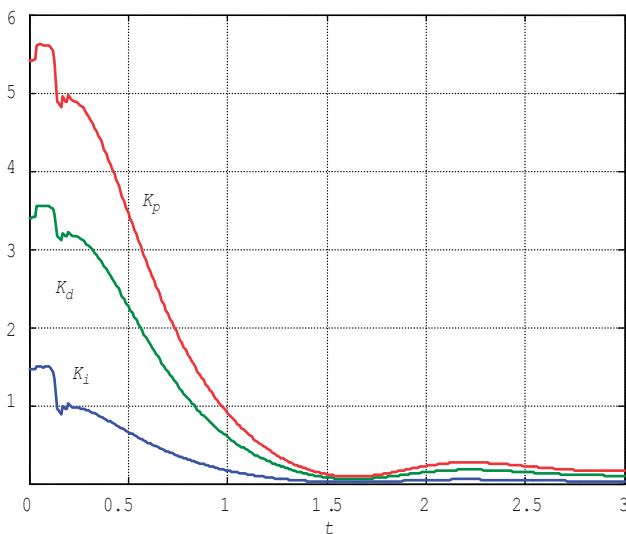


Рис. 8.13. Нейросетевой супервизор ПИД-регулятора в MatLab Simulink

На рис. 8.14 приведен переходный процесс в системе после обучения нейронного супервизора. На рис. 8.15 показаны графики изменения значений коэффициентов ПИД-регулятора во время переходного процесса.



*Рис. 8.14. Переходные процессы
в системе с нейронным супервизором*



*Рис. 8.15. Изменение коэффициентов ПИД-регулятора
во время переходного процесса*

Как следует из рис. 8.15, переходный процесс оказывается весьма близок к заданному эталонному процессу, и ПИД-регулятор с супервизором обеспечивает гораздо более высокое качество управления, чем ПИД-регулятор с постоянными коэффициентами (кризая 2 на рис. 8.11).

8.5. Метод роя частиц

Алгоритм Particle Swarm Optimization (PSO) был впервые предложен в работе [13]. Позднее вошел в употребление термин «swarm intelligence» [63], поскольку разработанная вычислительная технология оказалась пригодной для решения широкого круга сложных задач, относящихся к искусственному интеллекту. В отечественной технической литературе эти английские термины переводятся обычно как «стайный интеллект» или «ройный интеллект», а также «стайная оптимизация» или «оптимизация роем частиц».

Оптимизация роем частиц основывается на концепции социального взаимодействия при решении проблем. Такое взаимодействие наблюдается не только в человеческом обществе, но и у стаи птиц или рыб, роя насекомых, при функционировании колонии муравьев или пчел.

Пусть, например, стая птиц занята поиском корма. Каждая птица отыскивает его самостоятельно и наблюдает за другими птицами. Как только одна из птиц заметит признаки еды, она подает сигналы остальным, которые летают неподалеку. Уровень сигнала зависит от количества пищи. Получив сигналы от соседей, все птицы корректируют свои траектории. Так формируется локальное взаимодействие, распространяющееся на всю стаю. В конце концов стая собирается в месте с максимальным количеством корма, т. е. находит глобальный экстремум целевой функции.

Искусственные частицы, перемещающиеся в N -мерном поисковом пространстве, могут вести себя подобно стае птиц (или рою насекомых). Аналогом корма здесь является заданная целевая функция, минимум которой требуется найти. По аналогии с эволюционными вычислениями можно утверждать, что рой подобен популяции, а частицы (индивидуалы) соответствуют хромосомам. Частицы летают в поисковом пространстве, изменяя направление в соответствии с собственным опытом и опытом соседей.

В модели PSO каждая частица обладает вектором скорости и вектором позиции. При оптимизации N -мерной функции такие

векторы имеют размерность N . Как и в биологической жизни, предполагается, что каждая частица регулирует свои векторы позиции и скорости в соответствии с собственным опытом (когнитивная составляющая), а также по информации, полученной от других членов социума (социальный опыт). Когнитивный опыт частицы понимается как ее знание о лучшей позиции, в которой она сама находилась, а социальное знание частицы – как знание о лучшей позиции, через которую прошла одна из частиц в одной группе с ней. В задаче оптимизации лучшая позиция частицы понимается как позиция, в которой минимизируется значение функции.

Таким образом, состояние каждой частицы роя характеризуется:

1. Тремя N -мерными векторами:

- X – текущая позиция частицы в поисковом пространстве;
- G – лучшая позиция, найденная всем роем;
- v – скорость движения частицы.

2. Двумя скалярными значениями, описывающими качество решения задачи:

- P – фитнес-частицы (значение целевой функции в текущей позиции);
- G – фитнес-группы, в которую входит частица (значение целевой функции в лучшей позиции).

Изменение положения частицы задается простым добавлением v -вектора к X -вектору:

$$X_i = X_i + v_i. \quad (8.1)$$

Изначально значение вектора скорости генерируется случайным образом в пределах $[-v_{\max}, v_{\max}]$, где v_{\max} – максимально допустимое значение.

Скорость частицы модифицируется по формуле

$$v_i = v_i + c_1 r_1 (G - X_i) + c_2 r_2 (P_i - X_i), \quad (8.2)$$

где i – номер частицы; v_i – вектор скорости; X_i – вектор позиции частицы; P_i – вектор лучшей позиции, которой достигала частица; G – вектор лучшей позиции, которой достигала группа из I частиц; c_1, c_2 – константы скорости обучения, управляющие соответственно социальной и познавательной компонентой; r_1, r_2 – случайные числа в диапазоне $[0, 1]$, которые служат для поддержания разных траекторий частиц при поиске.

Константы скорости обучения контролируют степень важности индивидуального познания и социального знания. Частица одновременно обновляет свою позицию относительно лучшей позиции группы и собственной лучшей позиции, которая была в прошлом.

Если c_2 имеет преобладающее значение, то частица будет обследовать узкую область поискового пространства, а если преобладает c_1 , то происходит поиск в большом пространстве.

В формулу (8.2) может вводиться фактор инерции W :

$$v_i = Wv_i + c_1r_1(P_i - X_i) + c_2r_2(G - X_i). \quad (8.3)$$

Вначале фактор инерции $W = 1$, а потом он постепенно уменьшается во времени по формуле

$$W = W_{\max} - (W_{\max} - W_{\min})n/N,$$

где n – номер текущей итерации; N – заданное число итераций.

При инициализации частицы распределяются случайным образом в поисковом пространстве, их скорость получает случайные значения в допустимом диапазоне. Таким образом, работу алгоритма оптимизации PSO можно описать рис. 8.16.

Если ввести обозначения

$$K = c_1r_1(P_i - X_i),$$

$$S = c_2r_2(G - X_i),$$

то можно назвать величину K когнитивной компонентой скорости, или ностальгией частицы, поскольку она характеризует стремление частицы вернуться в то место, где ей было хорошо в прошлом. Величина S описывает социальные нормы, которым должен удовлетворять индивид, т. е. движение по направлению к позиции лучшей частицы группы.

Как и при работе ГА, решение считается полученным, если оно удовлетворяет поставленным критериям или истекло время, отведенное на поиск решения.

Большое значение для работы алгоритма имеет выбор топологии группы, в которой существует частица. Возможны две разные топологии: кольцевая и звездообразная. Соответственно различают два варианта: global best (gbest) и local best (lbest).

В варианте gbest используется звездообразная топология, и частица получает информацию от всех частиц группы. В варианте lbest применяется кольцевая топология, и частица получает информацию только от ближайших соседей. Однако области соседства частиц перекрывают друг друга.

Таким образом, gbest можно рассматривать как специальный случай lbest, в котором область соседства расширена на весь рой.

На рис. 8.17 приведен пример движения частицы в двумерном пространстве поиска. С течением времени лучшие позиции частицы и группы становятся одинаковыми.



Рис. 8.16. Алгоритм оптимизации роем частиц

Алгоритм PSO имеет очевидное сходство с генетическим алгоритмом:

- оба алгоритма являются стохастическими;
 - решение ищется на базе популяции индивидуумов;
 - начальная популяция генерируется случайным образом;
 - для работы алгоритма требуется расчет фитнеса каждого индивидуума;
 - области возможного использования алгоритма совпадают.
- Но существуют и отличия:
- число настраиваемых параметров алгоритма PSO меньше;
 - эволюционные операторы отсутствуют, частицы не «умирают».

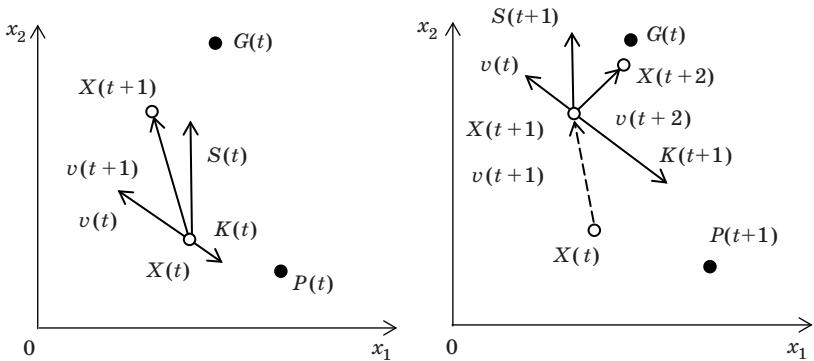


Рис. 8.17. Движение частицы в двумерном пространстве

Алгоритм PSO, описываемый формулами (8.1) и (8.2), обеспечивает более быструю сходимость, чем генетический алгоритм [64], однако при его использовании может возникать ряд особых проблем, к которым относятся:

- преждевременная сходимость, связанная с уменьшением скорости движения частиц;
- проблемная зависимость алгоритма, которая приводит к сильному влиянию значения констант скорости обучения на решение в конкретной задаче.

Для устранения этих проблем в классический алгоритм PSO вносятся модификации [65, 66], в частности, для управления процессом поиска в [66] предложено использовать нечеткий логический регулятор, работа которого основана на экспертных знаниях в виде лингвистических правил.

8.6. Другие метаэвристические алгоритмы

Стремление усовершенствовать ГА вызвало появление в 1990-е годы ряда работ, трансформирующих схему работы классического генетического алгоритма.

Так, в работах [67, 68] было предложено выполнять операцию отбора не над хромосомами популяции, а над генами хромосом. Увеличение или уменьшение числа генов в потомков определяется исходя из отношения средней пригодности хромосом, содержащих данный ген, к средней пригодности всей популяции. Операции скрещивания и мутации при этом сохраняются.

Идея отбора генов, а не хромосом лучше соответствует идеи симбиоза генетики и дарвинизма, который предполагает, что эволюционирует не конкретная особь, а популяция в целом. Эта идея получила развитие в алгоритме оценки распределения (Estimation of Distribution Algorithms – EDA), предложенном в работе [68].

Основная идея EDA заключается в отказе от использования операторов скрещивания и мутации классического ГА. Вместо этого в алгоритме EDA рассчитывается вероятностное распределение генов популяции, на основании которого формируется популяция потомков. Рассмотрим этот подход при бинарном представлении хромосом.

Пусть i -я популяция содержит шесть хромосом с оценками пригодности (табл. 8.1).

Допустим, что решается задача минимизации. Тогда при селекции отсечением в промежуточную популяцию попадут хромосомы с номерами 1, 4 и 5 (табл. 8.2).

Величина P_j из табл. 8.2 соответствует вероятности появления j -го гена хромосомы в популяции потомков. При формировании популяции потомков при выборе значения j -го гена хромосомы происходит сравнение P_j и случайного числа в диапазоне $[0, 1]$. Соответственно чем выше P_j , тем чаще будет появляться j -й ген в популяции потомков. Модификации EDA позволяют работать и при кодировании генов действительными числами.

Таблица 8.1

Исходная популяция

Номер хромосомы	Хромосома	Оценка пригодности
1	1001010	0,109
2	0100101	0,697
3	1101010	1,790
4	0110110	0,090
5	1001111	0,238
6	0001101	2,531

Таблица 8.2

Промежуточная популяция

Номер хромосомы	Хромосома при разных P_i						
	0,667	0,333	0,333	0,667	0,667	1	0,333
1	1	0	0	1	0	1	0
4	0	1	1	0	1	1	0
5	1	0	0	1	1	1	1

Метод дифференциальной эволюции (Differential Evolution – DE) был предложен в работах [69, 70].

Первоначально в поисковом пространстве случайным образом генерируется исходная популяция G из NP векторов длиной D :

$$G: \{X_{i,G}\}, \quad i = \overline{1, NP}.$$

Размер популяции не меняется в процессе оптимизации. Число популяций является одним из параметров алгоритма.

Очередная популяция формируется путем выполнения для каждого i -го вектора текущей популяции следующих действий:

1. Для вектора $X_{i,G}$ выбираются три вектора из остальной популяции со случайными индексами r_1, r_2, r_3 .

2. Формируется мутантный вектор (где $F \in [0, 2]$):

$$V_{i,G+1} = X_{r_1,G} + F(X_{r_2,G} - X_{r_3,G}).$$

3. Применяется оператор скрещивания, с помощью которого формируется пробный вектор (trial vector)

$$U_{i,G+1} = [u_{1i,G+1} \ u_{2i,G+1} \ \dots \ u_{Di,G+1}],$$

где

$$u_{ji,G+1} = \begin{cases} v_{ji,G+1}, & (\text{rand}(j) \leq CR) \text{or} (j = \text{rnbr}(i)), \\ x_{ji,G}, & (\text{rand}(j) > CR) \text{and} (j \neq \text{rnbr}(i)), \end{cases} \quad j = \overline{1, D};$$

$CR \in [0, 1]$ (это значение определяется пользователем; $\text{rnbr}(i)$ – случайный индекс ($\text{rnbr}(i) \in \{1, 2, \dots, D\}$)).

4. Пробный вектор сравнивается с исходным. Если пробный вектор $U_{i,G+1}$ имеет лучшее значение целевой функции, то он замещает исходный вектор, иначе $X_{i,G+1} = X_{i,G}$.

5. Шаги 1 – 4 выполняются для всех векторов популяции.

6. Проверяется условие окончания работы: если достигнуто максимальное число популяций, то в качестве решения выбирается вектор с наилучшим значением целевой функции.

Таким образом, метод DE отличается простотой, содержит мало управляющих параметров и легко программируется.

К числу новейших популяционных методов можно отнести метод поиска гармонии [71].

Метод поиска гармонии (Harmony Search – HS) – метаэвристический алгоритм оптимизации. Идея метода основана на подражании процедуре музыкальной импровизации.

Множество музыкантов составляют оркестр. Каждый музыкант генерирует свою ноту. Комбинация множества звуков может породить гармоничную мелодию, при которой отдельные звуки воспринимаются как единое целое. Музыканты импровизируют, пытаясь добиться гармонии.

Можно провести параллель между процедурой достижения гармонии в музыке и поиском экстремума в процессе оптимизации. Так, число музыкантов оркестра соответствует размерности пространства поиска, нота, взятая музыкантом, – значение в одном из измерений поискового пространства, совокупность нот – точка пространства поиска. Импровизации – случайные поисковые движения.

Алгоритм HS можно представить в виде последовательности шагов:

1. Инициализация памяти гармоний. Генерируется hms (harmony memory size – размер памяти гармоний) случайных векторов, каждый из которых имеет длину n . С каждым вектором связывается значение целевой функции $F(X^i)$:

$$HM = \begin{bmatrix} x_1^1 & \dots & x_n^1 & F(X^1) \\ x_1^2 & \dots & x_n^2 & F(X^2) \\ \vdots & \vdots & \vdots & \vdots \\ x_1^{hms} & \dots & x_n^{hms} & F(X^{hms}) \end{bmatrix}.$$

2. Каждый вектор X преобразуется в X' , так что для каждой компоненты этого вектора:

- с вероятностью $hmcr$ (harmony memory considering rate) выбирается значение из памяти гармоний:

$$x'_i \leftarrow x_i^{\text{int}(\text{rand}(0.1)hms)+1};$$

- с вероятностью $1 - hmcr$ выбирается случайное значение из допустимого интервала.

3. Если компонента была выбрана из памяти гармоний, то с вероятностью par (pitch adjusting rate) выполняется дополнительная настройка:

- для дискретных переменных

$$x'_i \leftarrow x'_i \pm \delta;$$

- для непрерывных переменных

$$x'_i \leftarrow x'_i \pm w\text{rand}(0,1).$$

Если компонента была выбрана случайно, то она не меняется.

4. Если X' лучше, чем худший вектор X^{worst} в памяти, то X^{worst} заменяется на X' .

5. Повторять шаги со 2-го по 4-й до достижения заданного качества решения или до превышения времени на поиск решения.

Размер памяти hms выбирается обычно в пределах 50–100 векторов, вероятности выбираются в диапазонах $hmcr \in [0,7; 0,99]$, $par \in [0,1; 0,5]$. Величина δ – расстояние между двумя соседними значениями дискретного набора данных; w – шаг изменения непрерывных переменных.

Успех подхода (biologically inspired algorithms) стимулирует поиск новых природных аналогий решения задач глобальной оптимизации. Здесь помимо метода оптимизации роем частиц следует выделить *метод муравьиной колонии* (Ants Colony Optimization – ACO) [14]. Одним из новейших подходов является также *алгоритм пчелиного роя* (Bees Algorithm – BA) [72, 73], основанный на наблюдениях за процессом поиска пчелами наиболее медоносных участков. Можно также упомянуть «*обезьяний поиск*» (monkey search).

Рассмотрим подробнее *иммунологический алгоритм*, моделирующий работу искусственной иммунной системы ([74, 75]).

Иммунная система живого организма характеризуется способностью распознавать присутствие чужеродных белков (антител). Согласно клонально-селекционной теории [74] это распознавание обеспечивается большим разнообразием лимфоцитов – распознающих клеток, имеющих на своей поверхности антитела, способные обнаруживать антигены. Степень соответствия антител данному антигену носит название *аффинности*. Лимфоциты с антителами с наибольшей аффинностью подвергаются операции клонирования для обеспечения высокой концентрации антител и борьбы с инфекцией. В иммунной системе предусмотрен также механизм соматической гипермутации, позволяющий повысить уровень аффинности антител.

Применение искусственной иммунной системы (ИИС) позволяет успешно решать задачи оптимизации мультимодальных функций, распознавания образов, обеспечения компьютерной безопасности и другие задачи, связанные с глобальной оптимизацией в большом поисковом пространстве [75].

Иммунный алгоритм представлен на рис. 8.18.

Работу искусственной иммунной системы в задаче синтеза параметров ИИС можно описать следующим образом.

Пусть имеется n сгенерированных случайным образом антител, где каждое из них представляет собой вектор P длиной k , кодирующий параметры ИИС:

$$Ab_i = \{p_{i1}, p_{i2}, \dots, p_{ik}\}, \quad i = \overline{1, n}.$$

Эти векторы составляют начальную популяцию. Принцип работы ИИС в рассматриваемой задаче заключается в поиске антитела M , имеющего наиболее высокую аффинность по отношению к антигену

$$Ag = \{p_{opt1}, p_{opt2}, \dots, p_{optk}\},$$

где $p_{opt1}, p_{opt2}, \dots, p_{optk}$ – оптимальные параметры ИИС, обеспечивающие приемлемое значение критерия качества. Например, если рассматривается задача синтеза нейроконтроллера, то эти параметры могут быть заданы косвенно – эталонной моделью, порождаю-

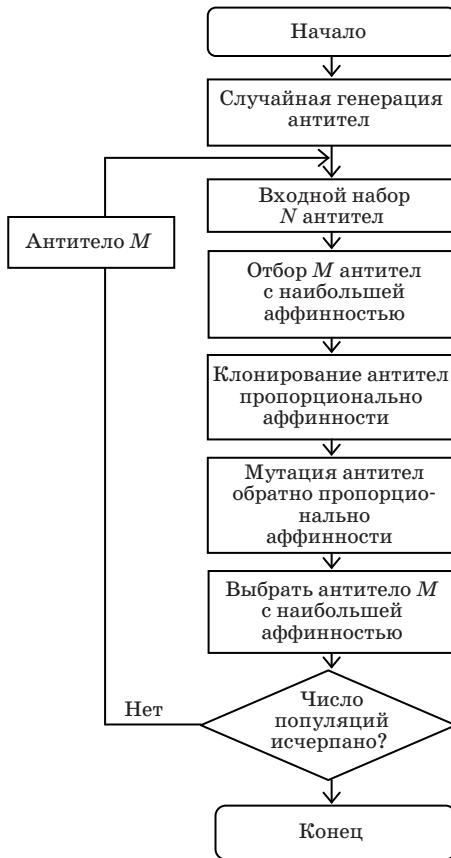


Рис. 8.18. Иммунный алгоритм оптимизации на принципах клонально-селекционной теории

щей выходной сигнал $Y^*(t)$. Тогда, зная реальный выходной сигнал объекта $Y_i(t)$ для i -го варианта таблицы правил, аффинность i -го антитела можно оценить по формуле

$$f_i(t) = \frac{1}{\sum_{k=1}^N |Y^*(t_k) - Y_i(t_k) + 1|}, \quad i = \overline{1, n},$$

где N – число точек, в которых сравниваются два переходных процесса.

Далее отбираются m антител с наиболее высокой аффинностью, которые подвергаются клонированию, при этом число клонов C_i антитела Ab_i будет пропорционально величине его аффинности:

$$C_i = \text{round}(k_c f_i), \quad i = \overline{1, n},$$

где k_c – масштабирующий коэффициент клонирования.

Клонированные антитела подвергаются процессу соматической гипермутации. Поскольку параметры представляются действительными числами, процесс мутации для антитела Ab_i будет заключаться в случайном изменении параметров на величину, кратную шагу мутации Δu .

Число мутаций α_i , которым будет подвергаться антитело Ab_i в процессе соматической гипермутации, будет равно

$$\alpha_i = \text{round}\left(k_{\text{mut}} \left(\frac{1}{f_i} - 1 \right)\right), \quad i = \overline{1, n},$$

где k_{mut} – масштабирующий коэффициент мутации.

Таким образом, чем больше аффинность антител, тем меньше они будут подвергаться процессу мутации.

После процесса соматической гипермутации отбирается антитело с наиболее высокой аффинностью. Оно запоминается как результат поиска M и помещается в следующую популяцию. После исчезновения каждой популяции значение M обновляется, если аффинность полученного антитела больше, чем предыдущий результат поиска M .

Работа алгоритма продолжается в течение N популяций антител, по истечении которых антитело M будет полученным результатом иммунного алгоритма оптимизации.

Можно предположить, что появление новых биологических аналогий в качестве метаэвристик не может фундаментально улучшить процесса глобальной оптимизации. В этой связи более перспектив-

ным выглядит направление, связанное с построением метаэвристик над метаэвристикой. Представителем этого подхода служит так называемый *культурологический алгоритм* (КА) [76, 77].

Культурологический алгоритм моделирует некоторые аспекты развития социальной системы.

Культуру можно описать как комплекс, включающий в себя знания, верования, искусство, мораль и обычаи, приобретаемые человеком как членом того или иного социума.

Культуру можно назвать также *системой коллективного программирования разума*, которая позволяет выделять отдельные нации, племена и сообщества.

В терминах эволюционных вычислений культура – это массив информации, определяющий поведение людей, входящих в некоторую популяцию.

Культурологический алгоритм состоит из социума (популяции) и пространства воззрений (верований, убеждений). Они взаимодействуют друг с другом в соответствии с протоколом взаимодействия. Внутри КА культура накапливает основные особенности поведения популяции.

Культурологический алгоритм представляет собой систему двойного наследования, содержащую два поисковых пространства:

- пространство популяции, подчиняющееся принципам дарвинской эволюции;
- пространство убеждений (*belief space*), в котором хранится культурная информация популяции.

Оба поисковых пространства функционируют параллельно, оказывая влияние друг на друга с помощью протокола взаимодействия. Этот протокол содержит два канала: один служит для адаптации убеждений под влиянием избранной группы индивидуумов, другой – для влияния из пространства убеждений на всех индивидуалов в пространстве популяции (рис. 8.19).

Главные особенности КА:

- поддержка иерархической структуры и двойное наследование (по двум уровням);
 - знания руководят эволюцией популяции;
 - область знаний отделена от индивидуумов;
 - происходит адаптация на обоих уровнях;
 - скорость эволюции на разных уровнях различна (культурная компонента развивается на порядок быстрее биологической компоненты);
 - гибридный подход к решению проблем.

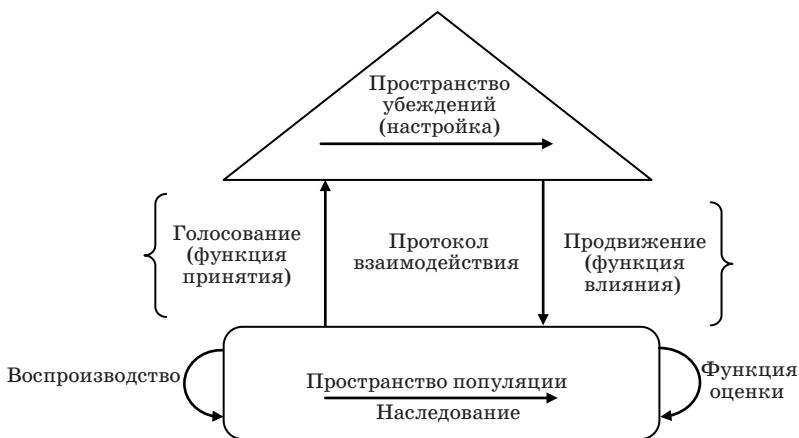


Рис. 8.19. Структура культурологического алгоритма

На каждой итерации (поколении) происходит оценка индивидуумов популяции.

Функция принятия определяет влияние лучших индивидуумов текущей популяции на пространство убеждений. Происходит настройка убеждений (эволюция культуры). Затем культура влияет на популяцию.

Поиск в пространстве популяции происходит под управлением ГА или PSO.

Пространство убеждений – коллективное хранилище знаний, его называют также *банком мемов*, где *мем* – единица информации, описывающая поведение индивидуума.

Мемы внутри пространства убеждений обобщают индивидуальный опыт из пространства популяции, накопленный в течение нескольких поколений. Это обобщение описывает представления об оптимальном поведении.

Можно отметить, что в пространстве популяции индивидуумы описывают отдельные точки поискового пространства, а в пространстве убеждений накапливается информация о перспективных направлениях поиска.

Пространство убеждений содержит, по крайней мере, две компоненты:

$$B(t) = (S(t), N(t)),$$

где $S(t)$ – ситуативная компонента, которая хранит лучшие решения, полученные в каждой генерации; а $N(t)$ – нормативная компонента, содержащая описание границ области поиска решения.

Конкретное представление знаний зависит от решаемой задачи.

Таким образом, можно утверждать, что стандартный эволюционный алгоритм не использует каких-либо дополнительных знаний для управления процессом поиска решения. Между тем такие знания могут ускорить поиск и направить его в желаемое русло.

Эволюционные вычисления основаны на принципах биологического наследования, которые занимают большие промежутки времени. Процессы культурных изменений протекают намного быстрее. Сочетание микроэволюции на уровне популяции и культурной макроэволюции призвано предупредить появление таких проблем глобального поиска, как преждевременная стагнация и низкая скорость поиска решения.

Завершая краткий обзор метаэвристик, можно подчеркнуть следующие ключевые моменты:

1. Арсенал биологических и иных природных аналогий для разработки популяционных метаэвристик далеко не исчерпан, новые методы регулярно предлагаются для обсуждения. Известные методы существуют также в виде множества вариантов и модификаций.

2. Проблема сравнения различных метаэвристик по их эффективности достаточно сложна, поскольку каждый алгоритм чувствителен к выбору параметров, и большое значение имеет первоначальное случайное распределение элементов популяции.

3. Наиболее перспективным представляется использование мета-метаэвристических алгоритмов (таких, как культурологический алгоритм), накапливающих знания о процессе поиска решения.

Вопросы для самопроверки

1. В каком случае можно гарантировать точное решение задачи глобальной оптимизации?

2. Чем субоптимальное решение задачи отличается от оптимального?

3. Как можно классифицировать методы глобальной оптимизации?

4. Чем метаэвристические методы глобальной оптимизации отличаются от эвристических?

5. Каковы два класса метаэвристических методов оптимизации?

6. В чем заключается предметная независимость методов глобальной оптимизации?

7. Какие параметры нейронной сети требуется оптимизировать с помощью процедур глобального поиска?

8. В чем заключается основная идея метода имитации отжига?
9. Как используется искусственная энергия в методе имитации отжига?
10. Как используется искусственная температура в методе имитации отжига?
11. Какова основная идея использования ГА для глобальной оптимизации?
12. Что представляет собой хромосома при генетическом синтезе нейронной сети?
13. Какие существуют генетические операторы?
14. Что такая пригодность хромосомы и как ее можно оценить?
15. В чем измеряется время работы генетического алгоритма?
16. Каковы принципы генетического синтеза нейроконтроллера для двигателя постоянного тока?
17. В чем заключаются принципы генетического синтеза нейронного супервизора для управления нелинейным объектом?
18. Какие идеи положены в основу оптимизации роем частиц?
19. Какими параметрами обладает частица при движении в пространстве?
20. Что такое социальный опыт частицы?
21. Что такое когнитивный опыт частицы?
22. Как регулируется соотношение социального и когнитивного опыта частицы в методе роя частиц?
23. Какие топологии могут быть использованы при работе метода роя частиц?
24. В чем заключается сходство ГА и метода роя частиц?
25. В чем состоят различия ГА и метода роя частиц?
26. Какие проблемы могут возникать при использовании метода роя частиц?
27. Чем алгоритм оценки распределения отличается от классического ГА?
28. Какие идеи положены в основу метода дифференциальной эволюции?
29. Какие идеи положены в основу метода поиска гармонии?
30. Как можно решать задачи глобальной оптимизации с помощью искусственной иммунной системы?
31. Что такое антиген?
32. Что такое антитело и его аффинность?
33. Как происходит клонирование антител?
34. Как происходит мутация антител?

35. Какова основная идея культурологического алгоритма?
36. Что такое культура с точки зрения глобальной оптимизации?
37. Какие изменения отличаются большей динамикой – культурные или эволюционные (биологические)?
38. Какие главные проблемы глобального поиска призвана решать двухуровневая структура культурологического алгоритма?

ЗАКЛЮЧЕНИЕ

Сфера использования ИНС постоянно расширяется, при этом существуют как приложения, в которых возможна работа в режиме offline, так и приложения, требующие работы в реальном времени.

В случаях, когда время реакции системы может измеряться минутами или даже часами, для реализации ИНС могут использоваться традиционные компьютеры, в которых нейронная сеть реализуется программно. Например, при анализе страховых рисков, прогнозировании продаж или анализе данных в ботанике сверхбыстрые реакции системы на входные данные не требуются.

Вместе с тем при работе в реальном времени необходима выработка управляющих решений в темпе поступления новой информации, так что вычислительные ресурсы традиционных компьютеров быстро оказываются исчерпанными, если реализуется достаточно сложная ИНС. Такая ситуация характерна для систем анализа и сжатия изображений, при управлении технологическими процессами и подвижными объектами и т. д. В этой связи большое практическое значение приобретают вопросы использования специализированных вычислительных средств, допускающих параллельную обработку информации. К таким средствам относятся:

- *специализированные параллельные компьютеры* – вычислительные системы, предназначенные для параллельной обработки данных. Часто используются для реализации сверхбольших нейронных сетей (десятки тысяч нейронов). Однако подобные компьютеры не компактны и отличаются высокой стоимостью;

- *аналоговые СБИС*. Нейроны в таких СБИС представляют собой пороговые усилители с сигмовидной передаточной функцией. Основные преимущества аналоговых СБИС заключаются в низком электропотреблении и высоком быстродействии, которое ограничивается только АЧХ усилителей и плотностью расположения элементов. К недостаткам следует отнести сложность получения высокой точности, обусловленную различиями компонентов из-за системы допусков при производстве, разные виды тепловых и электромагнитных помех, искажающих полезный сигнал. Проблемой является также сложность долгосрочного хранения весовых коэффициентов и организация операций аналогового умножения. Обучение в процессе работы затруднено. Указанные недостатки серьезно ограничивают распространение аналоговых СБИС в области моделирования нейронных сетей;

- цифровые СБИС также используются для реализации нейронных сетей. По сравнению с аналоговыми СБИС они обеспечивают большую точность и помехозащищенность, однако их создание является трудоемким процессом и экономически невыгодно в случае выпуска небольших серий микросхем;

- *программируемые логические интегральные схемы (ПЛИС).*

Данная технология позволяет не только создать сеть с распараллеливанием вычислений для каждого нейрона, но и сохранить гибкость реконфигурирования такой системы. Нейронные сети могут быть сформированы в специальной программной рабочей среде как в виде схем, так и в виде программы на языке описания аппаратуры (VHDL, Verilog). Тем самым существенно снижаются затраты на разработку топологии, ее реализацию и отладку.

В последние годы интерес к использованию ПЛИС в качестве элементной базы ИНС резко возрос. Этому способствовали появление на рынке высокочастотных ПЛИС и продолжающаяся тенденция к снижению их цены, а следовательно, и конечной стоимости разработок. Разнообразие устройств данной группы постоянно увеличивается: существуют как серии кристаллов с низким энергопотреблением для мобильных устройств, так и высокопроизводительные системы на кристалле, содержащие в себе не только блоки цифровой обработки сигналов (DSP), но и интерфейсы (PCI, Ethernet-MAC, LVDS) и даже полноценные процессоры (ARM, PowerPC). Современные ПЛИС изготавливаются такими производителями, как Altera, Atmel, Xilinx и другими.

Таким образом, уже существуют все возможности для эффективного использования ИНС при реализации широкого круга плохо формализуемых задач. Однако основные концепции ИНС настолько просты, что непреодолимой кажется пропасть между сложностью реального мозга и ограниченностью его нейросетевых моделей. В этой связи следует упомянуть концепции «сильного» и «слабого» искусственного интеллекта (ИИ):

- «слабый» ИИ – это набор отдельных технологий, позволяющих решать конкретные интеллектуальные задачи;

- «сильный» ИИ – это система, способная решать широкий комплекс задач, не уступающая по функциональности и эффективности человеческому интеллекту.

Очевидно, что ИНС уже достаточно давно являются важным инструментом при разработке систем «слабого» искусственного интеллекта. Между тем остается нерешенным вопрос, смогут ли ИНС по сложности и функциональным возможностям приблизиться к

природным нейронным сетям, составляющим основу мозга высших приматов и человека.

Если в конце минувшего века на этот счет и существовали разные мнения, то логика развития технологий XXI в. не оставляет места сомнениям. Классические компьютеры с их последовательной обработкой команд постепенно сменяются мощными много-процессорными системами, ориентированными на параллельную обработку информации.

Так, на конференции Supercomputing 2012 фирма IBM представила модель когнитивного компьютера, названную Compass. Модель была развернута на втором в мире по производительности суперкомпьютере Blue Gene/Q Sequoia, расположенному в лаборатории Лоуренса в США. С точки зрения структуры, Compass состоит из 2084 млрд нейросинаптических ядер, содержащих 530 млрд нейронов и 100 трлн синапсов, разделенных на 77 групп, в соответствии с разделами головного мозга.

Использование когнитивных компьютеров позволит в обозримом будущем выполнить информационное копирование мозга человека на внешний носитель. Такая полная или частичная «передача разума» открывает широкие возможности для автоматизации интеллектуального труда. При этом моделирование личностных качеств может быть перспективой более отдаленного будущего, важно лишь добиться «человеческого» уровня решения задач как в реальном, так и в виртуальном мире.

Библиографический список

1. Мак-Каллок У., Питтс У. Логические исчисления идей, относящихся к нервной активности. М.: ИЛ, 1956.
2. Hebb D. O. The organization of behavior: A Neuropsychological theory. New York: Wiley, 1949.
3. Розенблatt Ф. Принципы нейродинамики. Перцентрон и теория механизмов мозга. М.: Мир, 1965.
4. Widrow B., Hoff M. E. Adaptive switching circuits // 1960 IRE WESCON Convention Record. New York: IRE, 1960. Pt 4. P. 96–104.
5. Минский М., Пайперт С. Персептроны. М.: Мир, 1971.
6. Kohonen T. Correlation matrix memories // IEEE Trans. Comput. 1972. Vol.21. P. 353–359.
7. Grossberg S. Adaptive pattern classification and universal recoding. I. Parallel development and coding of neural feature detectors // Biol. Cybernet. 1976. Vol. 23. P. 121–134.
8. Hopfield J. J. Neural networks and phisical systems with emergent collective computational abilities // Proc. Nat. Acad. Sci. U.S.A. 1982. Vol. 79. P. 2554–2558.
9. Kohonen T. Self-organization and associative memory. Berlin: Springer, 1987.
10. Rumelhart D. E., Hinton G. E., Williams R. J. Learning internal representations by error propagation. I. Parallel distributed processing. 1986. Vol. 1. P. 318–362.
11. Aarts E. H. L., Laarhoven P. J. M. van. Simulated annealing: Theory and applications. London: Kluwer, 1987.
12. Goldberg D. E. Genetic algorithms in search, optimization and machine learning. New York: Addison-Wesley, Reading, MA. 1989.
13. Kennedy, J., Eberhart R. Particle swarm optimization // Proc. 1995 IEEE International Conference on Neural Networks IEEE Press. P. 1942–1948.
14. The ant system: Optimization by a colony of cooperating agents / Dorigo M. et al. // IEEE Trans. Systems Man Cybernetics. Pt B. Cybernetics. 1996. Vol. 26(1). P. 29–41.
15. Нейронные сети: История развития теории: Учеб. пособие для вузов / Под общ. ред. А. И. Галушкина, Я. З. Цыпкина. М.: ИПРЖР, 2001. Кн. 5.
16. Горбань А. Н. Обучение нейронных сетей. М.: ПараГраф, 1990.
17. Горбань А. Н., Россиев Д. А. Нейронные сети на персональном компьютере. Новосибирск: Наука, 1996.

18. Чертухин Ю. В. Нейропроцессоры. Таганрог: ТРТИ, 1994.
19. Галушкин А. И. Нейрокомпьютеры: Учеб. пособие для вузов / Общ. ред. А. И. Галушкина. М.: ИПРЖР, 2000. Кн. 3.
20. Терехов В. А., Ефимов Д. В., Тюкин И. Ю. Нейросетевые системы управления: Учеб. пособие для вузов / Общ. ред. А. И. Галушкина. М.: ИПРЖР, 2002. Кн. 8.
21. Нейрокомпьютеры в системах обработки сигналов / Под ред. Ю. В. Гуляева, А. И. Галушкина. М.: Радиотехника, 2003. Кн. 9.
22. Нейрокомпьютеры в космической технике / Под ред. В. В. Ефимова. М.: Радиотехника, 2004.
23. Васильев В. И., Ильясов Б. Г., Кусимов С. Т. Нейрокомпьютеры в авиации. М.: Радиотехника, 2004.
24. Городецкий А. Е., Тарасова И. Л. Управление и нейронные сети. СПб.: Изд-во Политех. ун-та, 2005.
25. Уоссермен Ф. Нейрокомпьютерная техника: Теория и практика. М.: Мир, 1992.
26. Омату С., Халид М., Юсоф Р. Нейроуправление и его приложения / Под ред. А. И. Галушкина, В. А. Птичкина. М.: ИПРЖР, 2000.
27. Осовский С. Нейронные сети для обработки информации / Пер. спольск. И. Д. Рудинского. М.: Финансы и статистика, 2002.
28. Хайкин С. Нейронные сети. М.: Изд. дом «Вильямс», 2006.
29. Будущее искусственного интеллекта. М.: Наука, 1991.
30. Сергеев Б. Ф. Тайны памяти. М.: Молодая гвардия, 1981.
31. Голдберг Э. Управляющий мозг: Лобные доли, лидерство и цивилизация. М.: Смысл, 2003.
32. Андерсон Дж. Когнитивная психология. СПб.: Питер, 2002.
33. Роуз С. Устройство памяти: От молекул к сознанию. М.: Мир, 1995.
34. Таунсенд К., Фохт Д. Проектирование и программная реализация экспериментных систем на персональных ЭВМ / Пер. с англ. М.: Финансы и статистика, 1990.
35. Дейкстра Э., Хоор К. Структурное программирование. М.: Мир, 1975.
36. Общая физиология нервной системы. Л.: Наука, 1979.
37. Дуда Р., Харт Р. Распознавание образов и анализ сцен. М.: Мир, 1976.
38. Бураков М. В. Нечеткие регуляторы: Учеб. пособие. СПб.: ГУАП. 2010.
39. Hornik K., Stinchcombe M., White H. Multilayer feedforward networks are universal approximators // Neural Network. 1989. Vol. 2. P. 359–366.

40. Генетические алгоритмы, искусственные нейронные сети и проблемы виртуальной реальности / Г. К. Вороновский, К. В. Махотило, С. Н. Петрашев, С. А. Сергеев. Харьков: Основа, 1997.
41. Бесекерский В. А., Попов Е. П. Теория систем автоматического регулирования. М.: Наука, 1972.
42. Hagan M. T., Demuth H. B. Neural networks for control // Proc. 1999 American Control Conference. San Diego: CA, 1999. P. 1642–1656.
43. Neural systems for control / O. Omidvar, D. L. Elliott eds. // New York: Academic Press. 1997. P. 272.
44. Галушкин А. И. Основы нейроуправления // Приложение к журналу «Информационные технологии». 2002. № 10. С. 2–16.
45. Чернодуб А. Н., Дзюба Д. А. Обзор методов нейроуправления // Проблемы программирования. 2011. № 2. С. 79–94.
46. Soloway D., Haley P. J. Neural generalized predictive control // Proc. 1996 IEEE International Symposium on Intelligent Control. 1996. P. 277–281.
47. Chen S., Billings S. A. Representation of nonlinear systems: The NARMA model // Int. J. Control. 1989. Vol. 49(3). P. 1013–1032.
48. Narendra K. S., Mukhopadhyay S. Adaptive control using neural networks and approximate models // IEEE Trans. Neural Networks. 1997. Vol. 8. P. 475–485.
49. Broomhead D. S., Lowe D. Multivariable functional interpolation and adaptive networks // Complex Systems. 1988. N 2. P. 321–355.
50. Yager R., Filev D. Essentials of fuzzy modeling and control. New York: John Wiley & Sons. 1984.
51. Бураков М. В. Генетический алгоритм: Теория и практика. СПб.: ГУАП. 2008.
52. Бураков М. В., Коновалов А. С., Шумилов Е. П. Интеллектуальные системы авиационной антисистемы. СПб.: Изд-во Политех. ун-та. 2005.
53. Elman J. L. Finding structure in time // Cognitive Sci. Ser. 1990. N 14. P. 179–211.
54. Медведев В. С., Потемкин В. Г. Нейронные сети. МАТЛАБ 6. М.: Диалог-МИФИ, 2002.
55. Glover F. Future paths for integer programming and links to artificial intelligence // Comput. Oper. Res. 1986. Vol. 13(5). P. 533–549.
56. Mohammadian M., Ruhul A., Xin Y. Computational intelligence in control. London: Idea Group inc. 2003.
57. Rudas I. F., Janos F., Kacprzyk J. Computational intelligence in engineering. Berlin; Heidelberg. Springer-Verlag: 2010.

58. Гладков Л. А., Курейчик В. В., Курейчик В. М. Генетические алгоритмы. М.: Физматлит, 2006.
59. Ali M., Storey C. Aspiration based simulated annealing algorithm // J. Global Optimization. 1996. N 11. P. 181–191.
60. Бураков М. В. Синтез нейронного регулятора // Изв. РАН. Теория и системы управления. 1999. № 3. С. 140–145.
61. Хрущев В. В. Электрические машины систем автоматики. Л.: Энергоатомиздат, 1985.
62. Astrom K. J., Hagglund T. Advanced PID control. Boston: ISA, 2006.
63. Kennedy J., Eberhart R. Swarm intelligence. San Francisco, CA: Morgan Kaufmann Publ. Inc., 2001.
64. Janson S. Middendorf M. A hierarchical particle swarm optimizer and its adaptive variant // IEEE Trans. Systems Man and Cybernetics. Pt B: Cybernetics. December 2005. Vol. 35. P. 1272–1282.
65. Angeline P. J. Using selection to improve particle swarm optimization // IEEE Int. Conference on Evolutionary Computation. May 1998. P. 84–89.
66. A fuzzy adaptive turbulent particle swarm optimisation / Liu et. al. // Int. J. Innovative Computing and Applications. 2007. Vol. 1. N 1. P. 39–47.
67. Burakov M. V., Konovalov A. S. Peculiarities of genetic algorithm usage when synthesizing neural and fuzzy regulators // Kluwer Int. Ser. Eng. and Comput. Sci. 2002. N 664. P. 39–48.
68. Muhlenbein H., Paab G. From recombination of genes to the estimation of distributions. I: Binary parameters // Lecture Notes in Comput. Sci. 1411: Parallel Problem Solving from Nature – PPSN IV. 1996. P. 178–187.
69. Storn R., Price K. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces // J. Global Optimization. 1997. Vol. 11. P. 341–359.
70. Price K., Storn R., Lampinen J. Differential evolution – a practical approach to global optimization. Heidelberg: Springer, 2005.
71. Geem Z. W. Music-inspired harmony search algorithm: Theory and applications. Berlin: Springer, 2009.
72. The bees algorithm / D. T. Pham, A. Ghanbarzadeh, E. Koc, S. Otri, S. Rahim, M. Zaidi. Technical Note, Manufacturing Engineering Centre, Cardiff University, UK, 2005.
73. Karaboga D. An idea based on honey bee swarm for numerical optimization: Technical report. Kayseri: Erciyes University, 2005.

74. *Castro L. N. de, von Zuben F. J.* Learning and optimization using the clonal selection principle // IEEE Trans. Evolutionary Computation. 2001. Special Issue on Artificial Immune Systems.

75. Искусственные иммунные системы и их применение / Под. ред. Д. Дасгупты / Пер. с англ.; Под ред. А. А. Романюхи. М.: 2006.

76. *Reynolds R. G.* Introduction to cultural algorithms / A. V. Sebald, J. Fogel // Proc. Third Annual Conference on Evolutionary Programming. Singapore: World Sci. Press, 1994. P. 131–139.

77. *Reynolds R. G.* An overview of cultural algorithms // Adv. Evolutionary Computation. New York: McGraw Hill Press, 1999.

1. Примеры проектирования нейрорегуляторов в MatLab

1.1. Нейрорегулятор с предсказанием

Рассмотрим пример использования нейрорегулятора с предсказанием, описанный в Neural Net toolbox, для объекта управления, показанного на рис. П.1.1.

Динамическая модель системы описывается уравнениями

$$\begin{cases} \frac{dh(t)}{dt} = w_1(t) + w_2(t) - 0,2\sqrt{h(t)}, \\ \frac{dC_b(t)}{dt} = (C_{b1} - C_b(t))\frac{w_1(t)}{h(t)} + (C_{b2} - C_b(t))\frac{w_2(t)}{h(t)} - \frac{k_1 C_b(t)}{(1 + k_2 C_b(t))^2}, \end{cases}$$

где $h(t)$ – уровень жидкости; $C_b(t)$ – концентрация продукта на выходе; w_1 и w_2 – скорость потоков компонентов смеси; C_{b1} и C_{b2} – концентрации компонентов смеси; $C_{b1} = 24,9$, $C_{b2} = 0,1$, k_1 и k_2 – константы процесса $k_1 = k_2 = 1$.

С целью упрощения в модели принято $w_1 = \text{const} = 0,1$, и сигналом управления является величина w_2 .

Пример вызывается командой predcstr (рис. П.1.2).

При щелчке по блоку NN Predictive Controller открывается окно, приведенное на рис. П.1.3.

Регулятор имеет следующие параметры:

- Cost Horizon (N_2) – верхний предел суммирования в показателе качества $N_2 = 7$, нижний предел фиксирован: $N_2 = 1$;

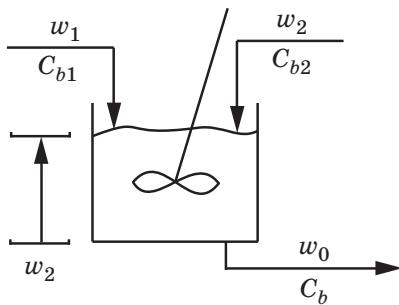


Рис. П.1.1. Каталитический реактор
с непрерывным перемешиванием

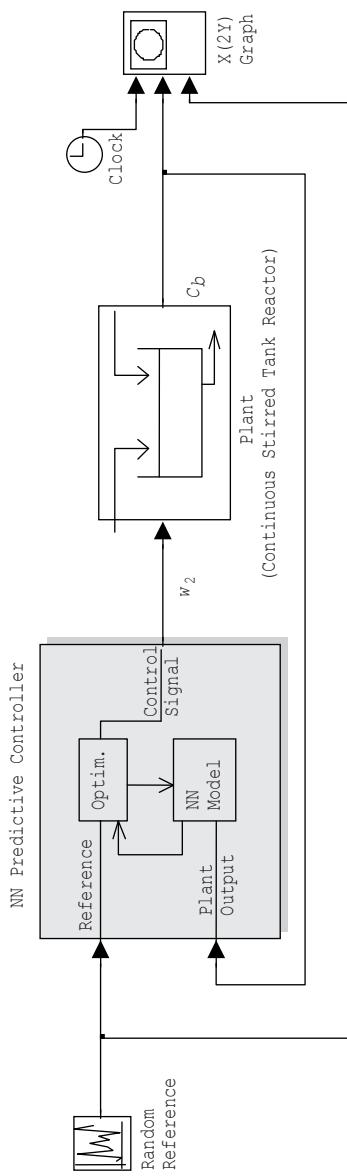


Рис. II.1.2. Модель системы управления смесительным баком

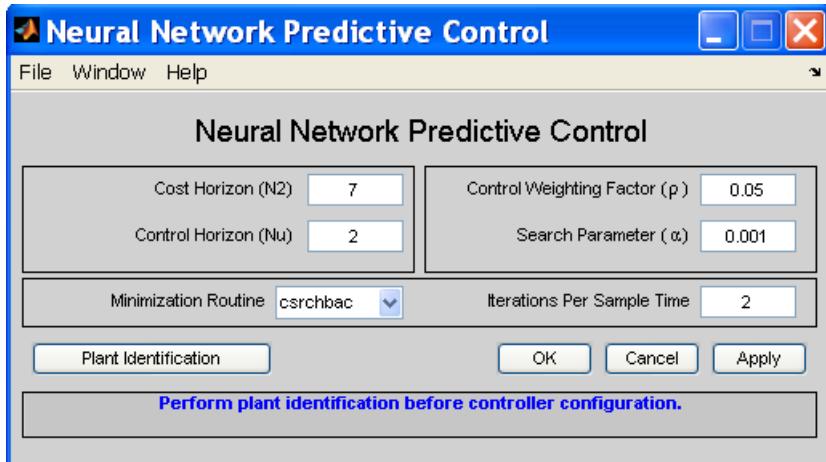


Рис. П.1.3. Интерфейс регулятора с предсказанием

- Control Horizon (Nu) – верхний предел суммирования при оценке мощности управления;
- Control Weighting Factor (ρ) – коэффициент веса для составляющей мощности управления $\rho = 0,05$;
- Search parameter (α) – параметр одномерного поиска, задающий порог уменьшения показателя качества $\alpha = 0,001$;
- Minimization Routine – выбор процедуры одномерного поиска;
- Iteration Per Sample Time – число итераций на один такт дискретности.

Надпись в нижней части окна предписывает выполнить идентификацию до установления параметров регулятора.

Панель идентификации приведена на рис. П.1.4.

Набор управляемых элементов для задания архитектурных параметров нейронной сети следующий:

- Size of the Hidden Layer – число нейронов в скрытом слое;
- No. Delayed Plant Inputs – число задержек для входного слоя;
- No. Delayed Plant Outputs – число задержек для выходного слоя;
- Sampling Interval – интервал квантования или шаг дискретности в секундах между двумя последовательными моментами отсчета данных;
- Normalize Training Data – переключатель нормирования для преобразования обучающих данных к диапазону [0 1].

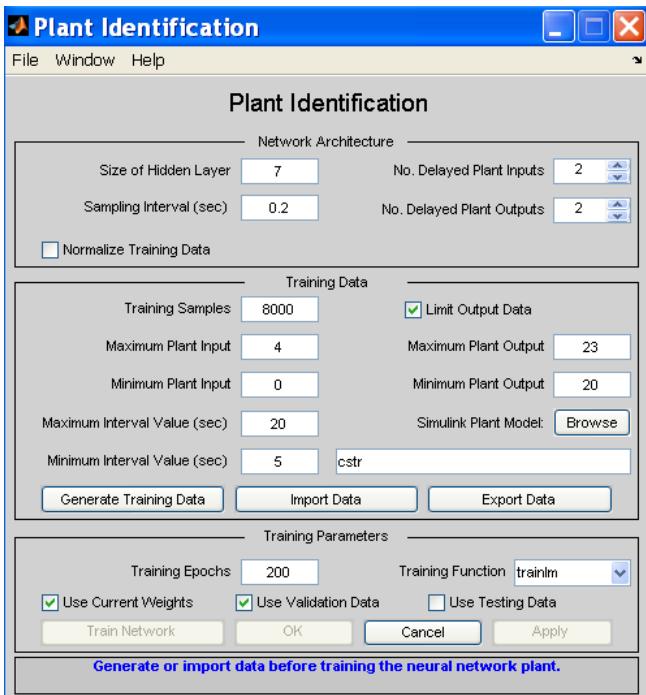


Рис. П.1.4. Интерфейс идентификации объекта

Набор управляемых элементов для задания характеристик обучающей последовательности включает в себя:

- Training Samples – число точек отсчета для получения обучающей последовательности в виде пар значений вход-выход для управляемого процесса, определяемого моделью Simulink;
- Maximum Plant Input – максимальное значение входного сигнала;
- Minimum Plant Input – минимальное значение входного сигнала;
- Maximum Interval Value (sec) – максимальный интервал идентификации (в секундах);
- Minimum Interval Value (sec) – минимальный интервал идентификации (в секундах);
- Limit Output Data – переключатель для ограничения значений выходного сигнала;
- Maximum Plant Output – максимальное значение выходного сигнала, задаваемое при включенном переключателе Limit Output Data;

- Minimum Plant Output – максимальное значение выходного сигнала, задаваемое при включенном переключателе Limit Output Data;
- Simulink Plant Model – окно задания модели управляемого процесса, реализованной с помощью блоков Simulink, имеющей порты входа и выхода и сохраненной в файле *.mdl; выбор модели производится с помощью кнопки Browse; имя модели отображается в специальном окне.

Параметры обучения задаются следующим образом:

- Training Epochs – число циклов обучения;
- Training Function – для задания обучающей функции;
- Use Current Weights – переключатель для использования текущих весов нейронной сети;
- Use Validation Data – переключатель для использования контрольного множества в объеме 25% от обучающего множества;
- Use Testing Data – переключатель для использования тестового множества в объеме 25% от обучающего множества.

Для идентификации управляемого процесса необходимо выполнить следующие действия:

- задать архитектуру ИНС, которая будет моделью управляемого процесса;
- задать параметры обучения;
- выбрать модель Simulink для управляемого процесса;
- сгенерировать обучающую последовательность заданного объема, запустив модель Simulink с помощью кнопки Generate Training Data.

Генерация обучающей последовательности производится с помощью воздействия ряда ступенчатых сигналов на модель управляемого процесса и снятия значений на входе и выходе модели через каждый шаг квантования. Графики входного и выходного сигналов отображаются в окне Plant Input-Output Data (рис. П.1.5).

По завершении генерации обучающей последовательности необходимо либо принять эти данные, нажав на кнопку Accept Data, и тогда они будут использованы для обучения нейронной сети, либо отвергнуть их, нажав кнопку Reject Data.

После получения обучающей последовательности необходимо установить требуемые параметры обучения и с помощью кнопки Train Network запустить процесс обучения нейронной сети.

После завершения обучения его результаты отображаются на графиках изменения ошибки сети для обучающей и тестирующей последовательностей, а также в окне выходных значений модели и сети при подаче на вход указанных последовательностей.

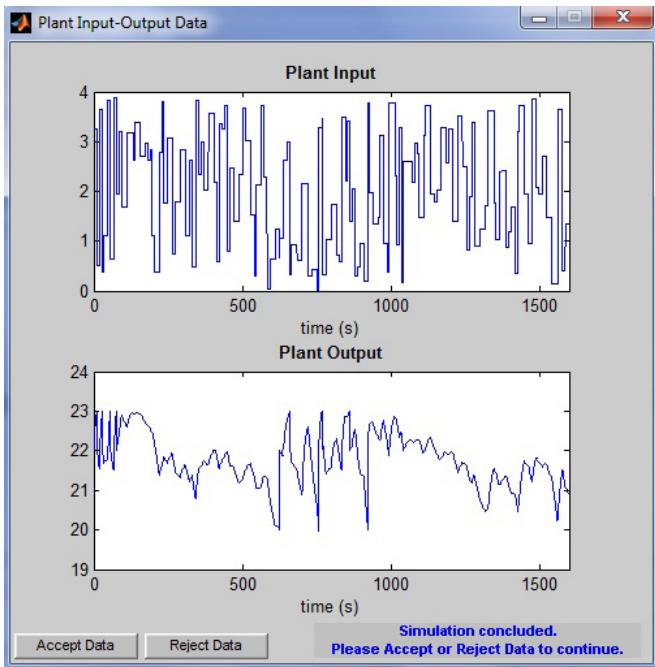


Рис. П.1.5. Данные для обучения нейронной сети

Если результаты обучения приемлемы, то необходимо сохранить параметры нейросетевой модели управляемого процесса и приступить к синтезу регулятора, нажав кнопки Apply и OK. В противном случае следует нажать кнопку Cancel и повторить процесс идентификации, сначала изменив архитектуру сети и параметры обучающей последовательности.

Обучающую последовательность можно импортировать из рабочей области или из файла, нажав на кнопку Import Data. Кнопка Export Data позволяет сохранить обучающую последовательность в рабочей области или в файле.

Кнопка Erase Generated Data удаляет сгенерированные данные.

Таким образом, диалоговая панель Plant Identification позволяет идентифицировать управляемый процесс, представленный в виде модели Simulink, построить двухслойную нейронную сеть прямого распространения с необходимым числом нейронов и линий задержки, и обучить эту сеть для получения нейронной модели управляемого процесса, а также оценить качество обучения и работу нейронной сети.

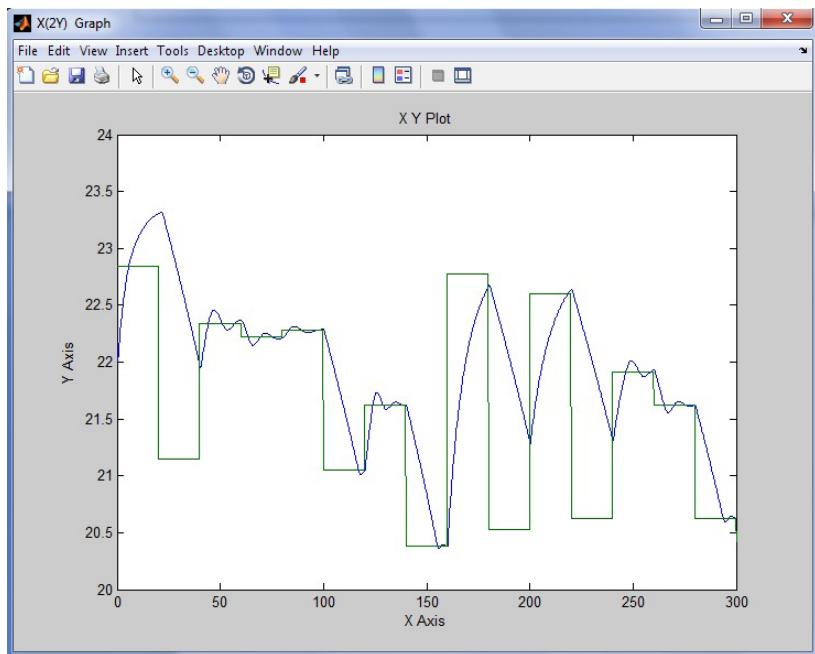


Рис. П.1.6. Пример функционирования регулятора с предсказанием

По завершении процесса идентификации необходимо вернуться к окну параметров регулятора, нажать кнопки **Apply** и **OK**, после чего можно приступить непосредственно к моделированию.

На рис. П.1.6 приведен пример отработки системой ступенчатых входных сигналов со случайным периодом и амплитудой.

1.2. Нейроуправление с эталонной моделью

Рассмотрим проектирование нейрорегулятора с эталонной моделью для механического звена робота, динамика которого описывается нелинейным дифференциальным уравнением второго порядка с постоянными коэффициентами:

$$\frac{d^2\varphi}{dt^2} = -2 \frac{d\varphi}{dt} - 10 \sin \varphi + u,$$

где φ – угол поворота звена; u – момент, развиваемый двигателем постоянного тока.

Задача нейрорегулятора состоит в том, чтобы при любом внешнем воздействии r система воспроизводила с заданной точностью реакцию эталонной модели, описываемой дифференциальным уравнением

$$\frac{d^2y_r}{dt^2} = -6 \frac{dy_r}{dt} - 9y_r + 9r,$$

где y_r – выход эталонной модели.

Сравним переходные процессы объекта и модели в MatLab (рис. П.1.7 и П.1.8).

Как следует из рис. П.1.8, эталонная модель обеспечивает устойчивый переходный процесс, в то время как объект управления без регулятора оказывается неустойчивым.

Рассмотрим демонстрационный пример, иллюстрирующий систему управления с эталонной моделью – звеном манипулятором робота. Пример вызывается командой mrefrobotarm (рис. П.1.9).

Для начала работы с демонстрационным примером необходимо активизировать блок Model Reference Controller двойным щелчком левой кнопки мыши. Появится окно, приведенное на рис. П.1.10.

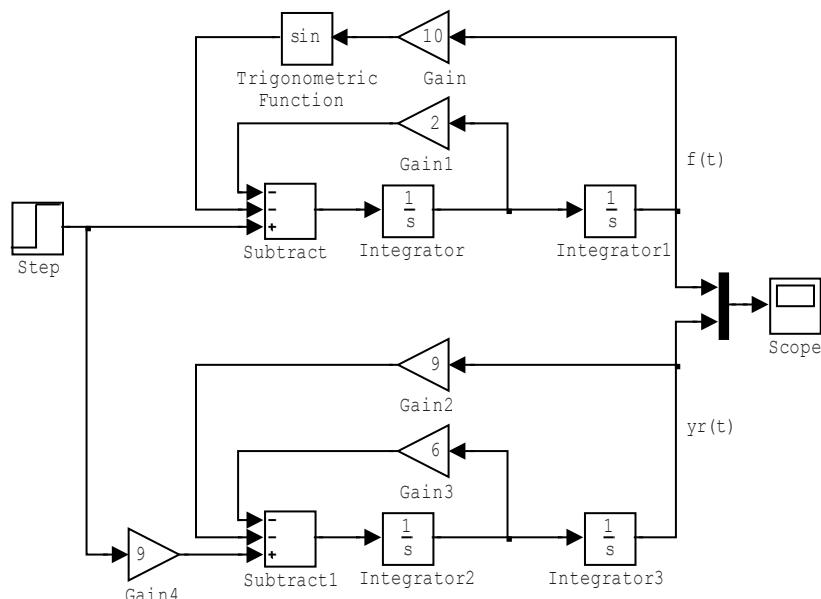


Рис. П.1.7. Сравнение динамики объекта и модели в MatLab

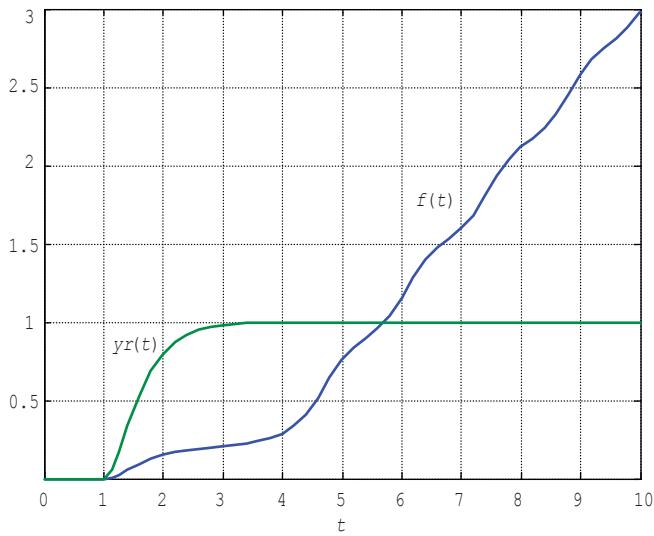


Рис. П.1.8. Переходные процессы объекта и модели

Настройка контроллера осуществляется в два этапа. На первом этапе проводится идентификация объекта управления (см. рис. П.1.4), на втором – обучение нейроконтроллера.

Процесс обучения контроллера в свою очередь можно разбить на три стадии.

Первая стадия – выбор конфигурации управляющей сети.

Здесь параметры Sampling Interval и Normalize Training Data выбираются такими же, как и при обучении идентификационной сети автоматически.

No. Delayed Reference Inputs – число элементов задержки на входе, на который подается тестовый сигнал.

No. Delayed Controller Outputs – число элементов задержки на выходе управляющей сети.

No. Delayed Plant Outputs – число элементов задержки на выходе идентификационной сети.

Вторая стадия – получение данных для обучения.

На этой стадии, как и при обучении идентификационной сети, возможно импортирование данных из mat-файла или непосредственно из рабочего пространства MatLab, а также данных на основе заранее подготовленной эталонной (reference) модели.

Reference model – это модель, которая описывает желаемое поведение замкнутой системы управления. Она задается так же, как

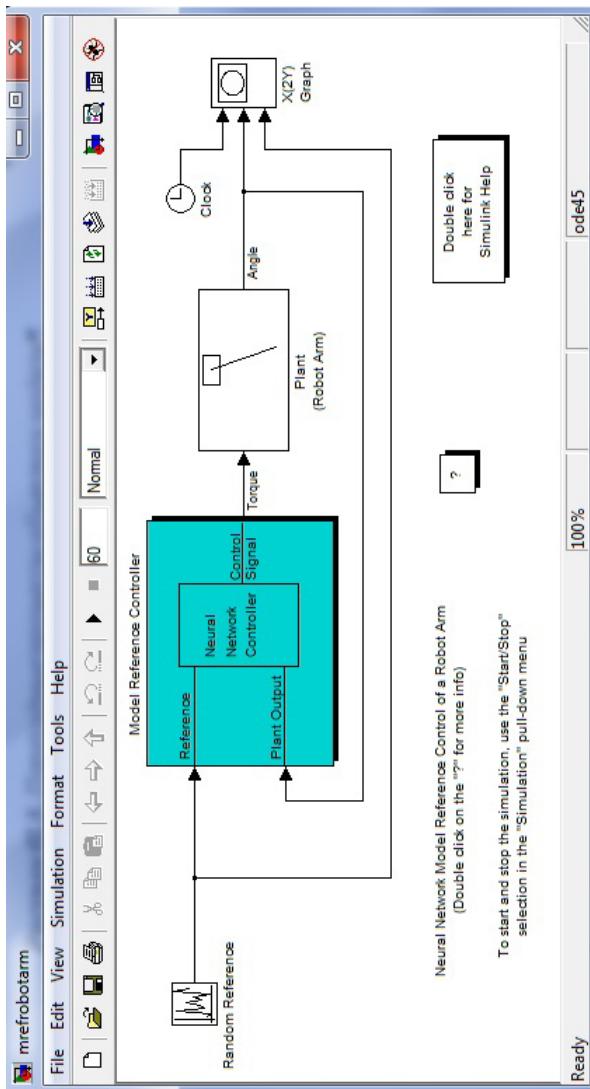


Рис. II.1.9. Нейросетевой регулятор с эталонной моделью

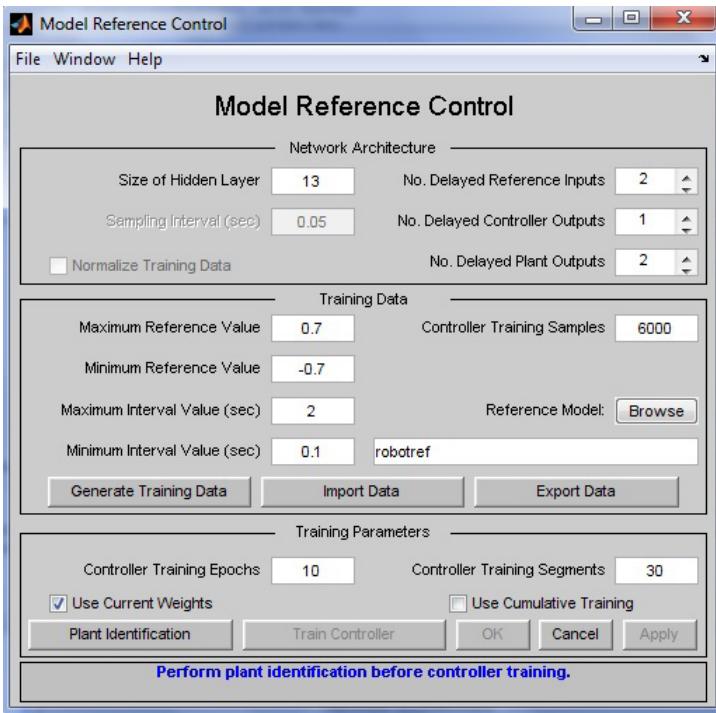


Рис. П.1.10. Окно настроек нейросетевой системы управления

и модель объекта управления на этапе идентификации, т. е. в виде модели Simulink.

Controller Training Samples – число дискретных отсчетов в тестовом сигнале. Оно должно быть, как и при идентификации объекта, достаточно большим.

Все параметры, как и сам процесс формирования тестовой последовательности, аналогичны используемым на этапе идентификации объекта управления.

Третья стадия – обучение контроллера (управляющей сети). Для обучения необходимо задать следующие параметры:

Controller Training Epochs – число циклов обучения. Значение этого параметра нельзя определить заранее, его необходимо подобрать в процессе обучения.

Controller Training Segments – число сегментов, на которые будут разделены тестовые данные. При этом для каждого сегмента буд-

дет проведено указанное число циклов. Желательно разбивать данные на сегменты таким образом, чтобы каждый сегмент содержал хотя бы один переходный процесс.

Use Current Weights – использовать текущие весовые коэффициенты. В противном случае данные коэффициенты будут выбраны случайным образом. Этот параметр следует применять при повторном обучении.

Use Cumulative Training – при выборе этого параметра первоначальное обучение будет выполнено с первым сегментом тестовых данных, а при последующем обучении остальные сегменты будут добавляться к данным, используемым на предыдущем этапе обучения. Таким образом, на финальной стадии обучения будет использован весь набор тестовых данных. Этот параметр следует применять осторожно, так как он увеличивает время обучения.

После задания всех параметров необходимо приступить к обучению управляющей сети, для чего необходимо нажать на кнопку Train Controller. При этом откроется окно, в котором будет отображаться процесс обучения (так же, как и при обучении идентификационной сети).

После завершения обучения открывается окно Plant Response for NN Model Reference Control, в котором отображаются результаты обучения.

После успешного завершения процесса обучения управляющей сети контроллер готов к использованию в составе системы управления. На рис. П.1.11 приведен пример отработки контроллером заданного входного ступенчатого сигнала.

Блок Model Reference Controller может быть использован для произвольного объекта управления. При этом могут возникать следующие проблемы:

1. Процесс обучения идентификационной сети идет медленно или прекращается самопроизвольно, что определяется по тому, что ошибка обучения не уменьшается с какого-либо урока. Это может происходить по следующим причинам:

- неверно выбрана конфигурация ИНС – необходимо задать большее число нейронов в скрытом слое;
- неверно задан набор тестовых данных, не содержащий достаточного для обучения количества информации;
- выбрано слишком малое число уроков обучения, и его необходимо увеличить.

2. Процесс обучения управляющего контроллера идет медленно, и ошибка обучения практически не уменьшается.

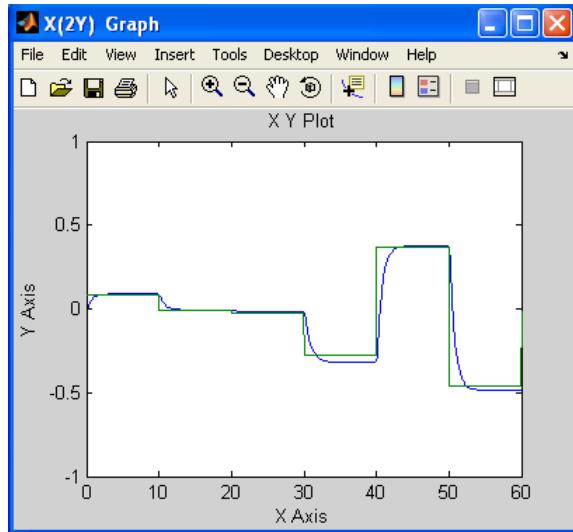


Рис. П.1.11. Отработка системой управления эталонного сигнала

Это может происходить в следующих случаях:

- неудачно заданы параметры тестовой последовательности при идентификации объекта, и необходимо повторить процесс обучения сети с другими параметрами в блоке Training Data; следует использовать близкие по форме и параметрам тестовые сигналы при обучении обеих ИНС;
- выбрано слишком малое число элементов в скрытом слое, и его необходимо увеличить.

3. Прямые показатели качества моделируемой системы существенно отличаются от аналогичных показателей для эталонной модели.

Это возможно в следующих случаях:

- идентификация объекта произведена неудачно;
- конфигурация управляющей сети не позволяет решать задачу управления, и необходимо повторить обучение с увеличенным числом элементов в скрытом слое.

1.3. Использование NARMA – L2-контроллера

Рассмотрим этапы проектирования нелинейного регулятора NARMA – L2 на демонстрационном примере управления магнитной подушкой, реализованного в пакете Neural Network toolbox.

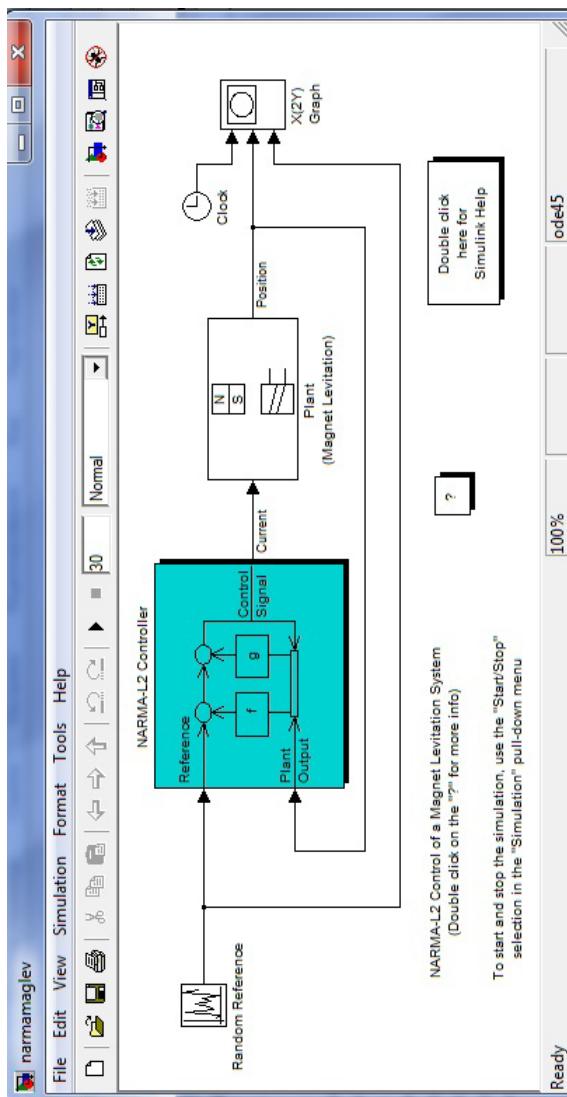


Рис. П.1.12. Пример использования NARMA – L2-контроллера

Пример вызывается вводом команды narmamaglev (рис. П.1.12).

Как следует из рис. П.1.12, в систему управления магнитной подушкой входят:

- блок управляемого процесса Plant;
- блок контроллера NARMA – L2 Controller;
- блок генерации эталонного ступенчатого сигнала со случайной амплитудой Random Reference;
- блок отсчета времени Clock;
- блок построения графиков Graph.

В демонстрационном примере управляемым объектом является постоянный магнит, который движется в магнитном поле только в вертикальном направлении, как это схематично показано на рис. П.1.13.

Уравнение движения постоянного магнита имеет вид

$$\frac{d^2y(t)}{dt^2} = -g + \frac{\alpha}{M} \frac{i^2}{y(t)} - \frac{\beta}{M} \frac{dy(t)}{dt},$$

где $y(t)$ – перемещение магнита, отсчитываемое от электромагнита; g – ускорение силы тяжести; α – постоянная, зависящая от числа витков обмотки и намагниченности стального сердечника; i – управляющий ток в обмотке электромагнита; M – масса постоянного магнита.

Для начала работы необходимо активизировать блок NARMA – L2 Controller двойным щелчком левой кнопки мыши в окне системы Simulink. Появится окно Plant Identification – NARMA – L2, параметры которого были рассмотрены ранее.

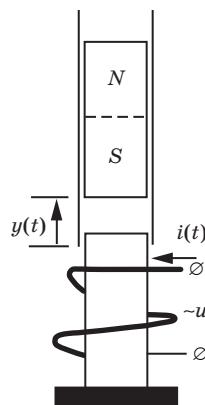


Рис. П.1.13. Структура объекта управления

После идентификации для обучения нейронной сети следует воспользоваться кнопкой Train Network. Начнется обучение нейросетевой модели. После завершения обучения его результаты отображаются на графиках.

Текущее состояние отмечено в окне Plant Identification сообщением «Обучение завершено. Можно сгенерировать или импортировать новые данные, продолжить обучение или сохранить полученные результаты, выбрав кнопки OK или Apply». В результате параметры нейросетевой модели управляемого процесса будут введены в блок NN NARMA – L2 Controller.

Выбрав опцию Start из меню Simulation, можно начать моделирование. В процессе моделирования выводятся графики входа и выхода управляемого процесса (см. рис. П.1.13).

ПРИЛОЖЕНИЕ 2

Работа с интерфейсом NNtool. Создание простейших нейронных сетей

Графический интерфейс пользователя NNtool позволяет выбирать структуры искусственных нейронных сетей из обширного перечня и предоставляет множество алгоритмов обучения для каждого типа сети.

NNtool позволяет решать следующие задачи:

- подготовка данных;
- создание нейронной сети;
- обучение сети;
- прогон сети;
- импорт и экспорт нейронных сетей и данных.

Интерфейс NNtool допускает работу только с простыми однослойными и двухслойными НС, поэтому он удобен для начального ознакомления с работой нейронных сетей.

Для запуска NNtool необходимо выполнить одноименную команду в командном окне MatLab:

```
>> nntool
```

после этого появится главное окно NNtool (рис. П.2.1).

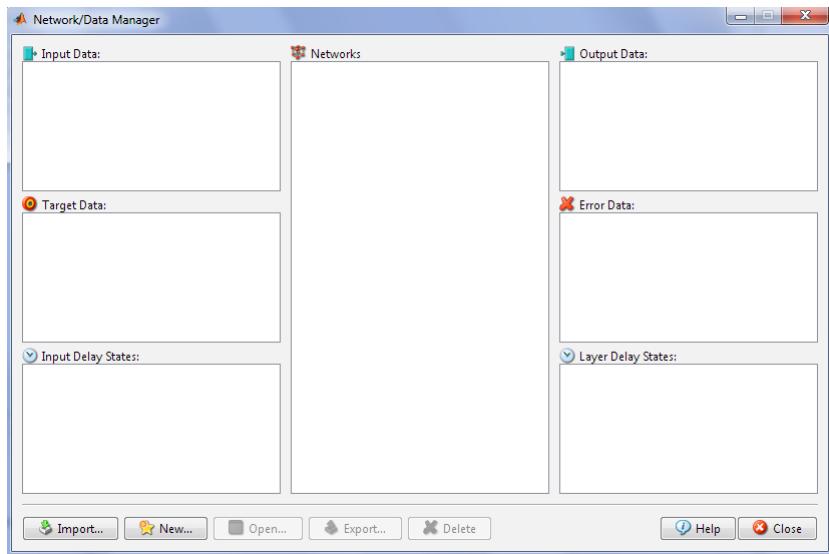


Рис. П.2.1. Главное окно NNtool

Работа с интерфейсом начинается нажатием кнопки New, после чего можно выбрать параметры ИНС для решаемой задачи. Допустим, мы хотим реализовать простейшую логическую функцию XOR. Для этой цели потребуется двухслойная сеть прямого распространения (рис. П.2.2).

Здесь же можно выбрать функцию обучения и адаптации, а также функцию оценки качества работы. Вкладка Data служит для ввода обучающих данных (рис. П.2.3).

После нажатия кнопки Create нейронная сеть готова к обучению (рис. П.2.4).

Двойной щелчок левой кнопкой мыши по названию нейронной сети открывает окно параметров обучения (рис. П.2.5).

Основной в этом окне является вкладка Train, которая требует указать данные для обучения и позволяет выбирать параметры ал-

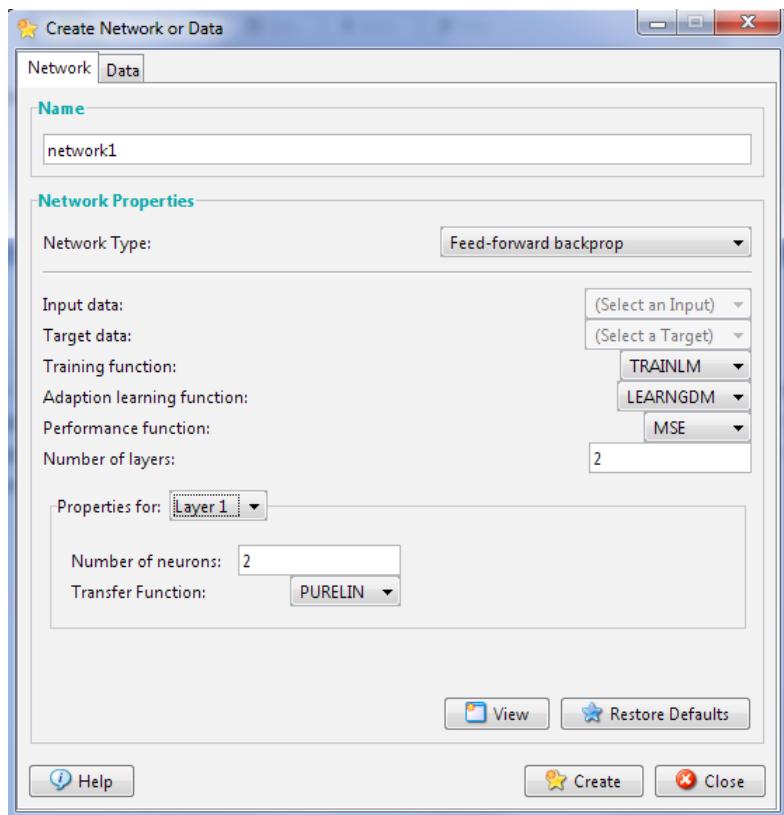


Рис. П.2.2. Выбор параметров нейронной сети

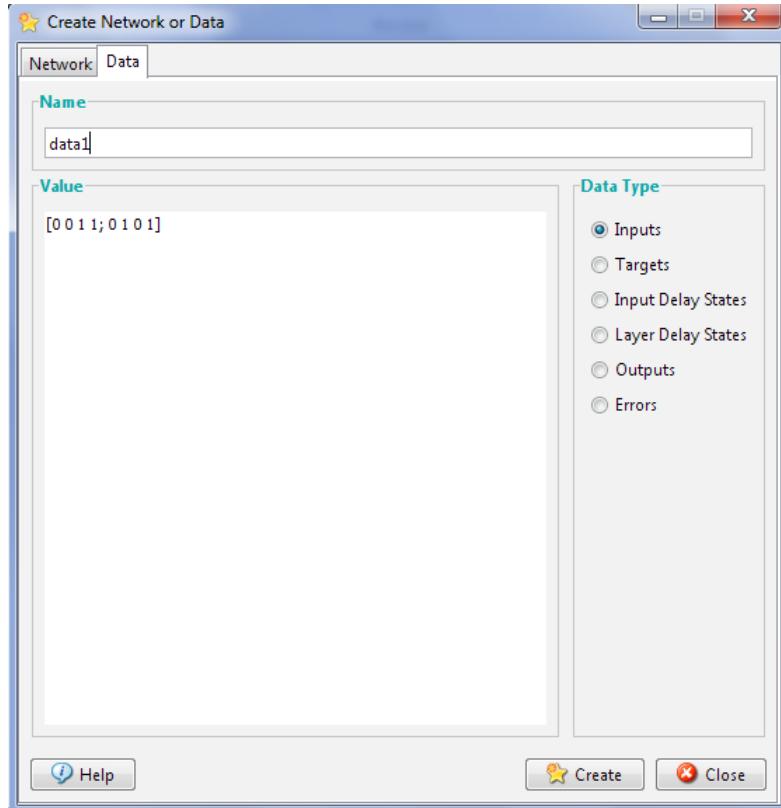


Рис. П.2.3. Ввод обучающих данных

горитма обучения. Остальные вкладки позволяют использовать возможности по моделированию работы ИНС, описать параметры адаптации сети, а также инициализировать и просматривать веса межнейронных связей.

Нажатием кнопки Train Network (рис. П.2.6) запускается процесс обучения.

На вкладке Training Parameters можно установить поля:

- число эпох (epochs) определяет число эпох, по прошествии которых обучение будет прекращено;
- достижение цели (goal) задает абсолютную величину функции ошибки, при которой цель будет считаться достигнутой;
- период обновления (show) – период обновления графика кривой обучения, выраженный числом эпох;

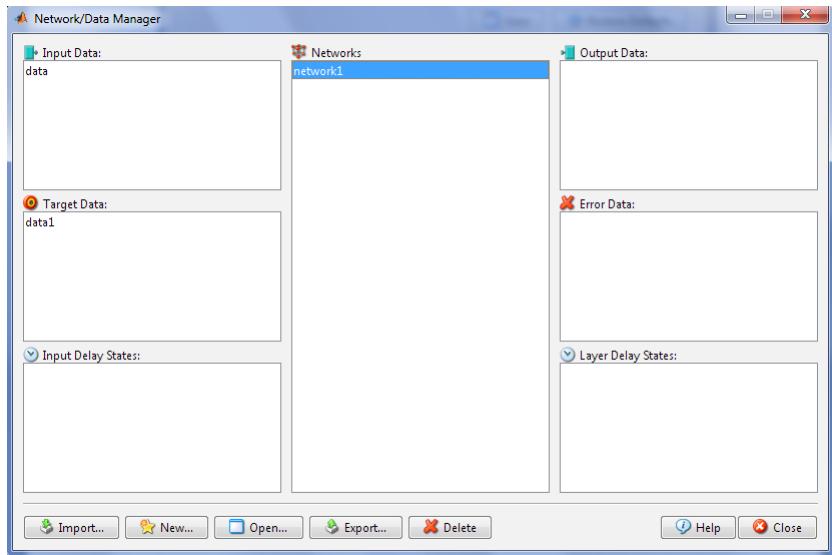


Рис. П.2.4. Главное окно NNtool после ввода исходной информации

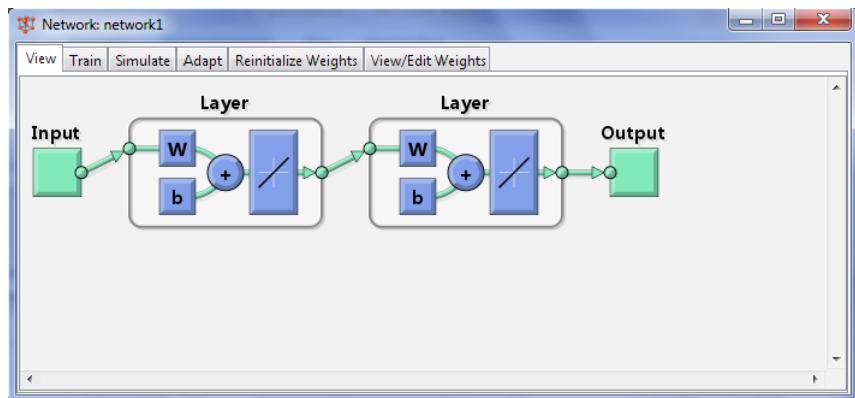


Рис. П.2.5. Окно обучения ИНС

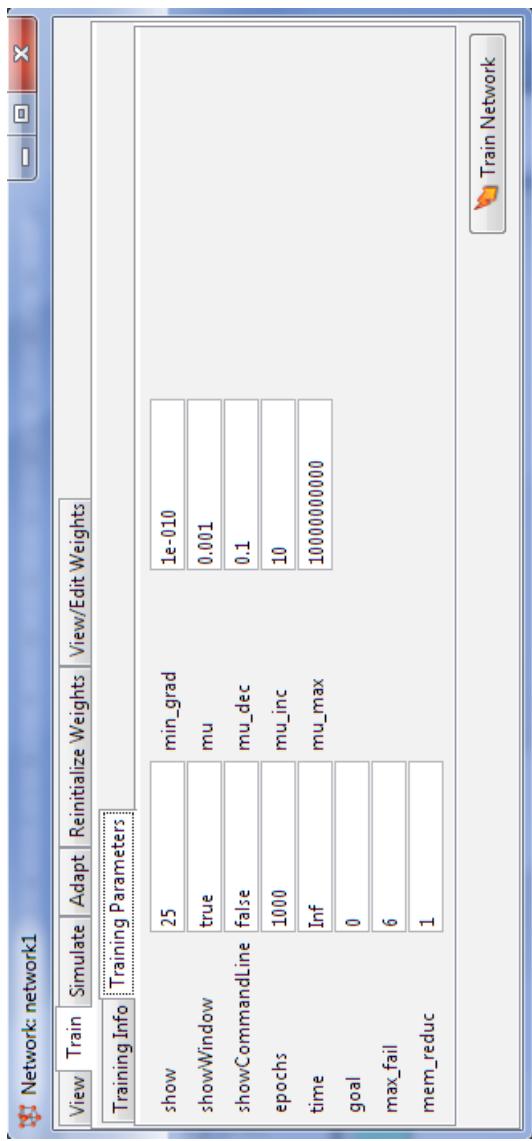


Рис. II.2.6. Окно ввода параметров обучения с кнопкой запуска обучения

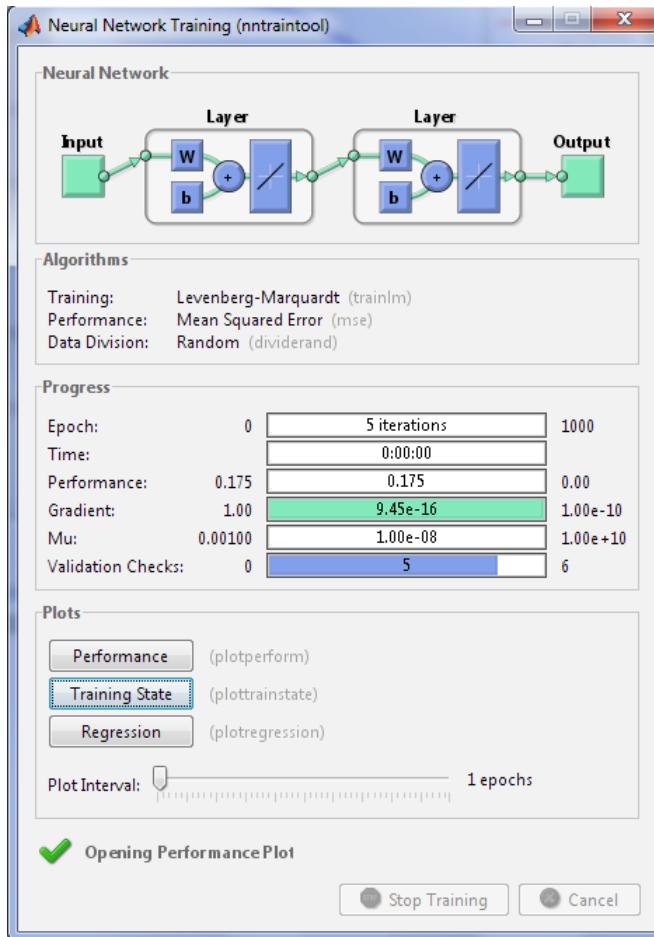


Рис. П.2.7. Окно контроля процесса обучения

- время обучения (time) – по истечении указанного временного интервала, выраженного в секундах, обучение прекращается.

После запуска обучения открывается следующее окно (рис. П.2.7), которое позволяет контролировать ход обучения, отображая время обучения, число эпох и текущее качество обучения.

СОДЕРЖАНИЕ

Введение.....	3
1. Нейронные сети. Базовые понятия	8
1.1. Некоторые сведения о мозге человека.....	8
1.2. Биологические представления о нейроне	13
1.3. Понятие нейрокомпьютера	16
1.4. Классификация нейронных сетей	20
1.5. Задача распознавания и линейная машина.....	22
1.6. Искусственный нейрон.....	26
1.7. Проблема линейной разделимости	27
1.8. Правило обучения Хебба	32
1.9. Концепция входной и выходной звезды	33
1.10. Парадигмы обучения.....	35
1.11. Предварительная обработка информации и оценка качества работы нейросети	36
Вопросы для самопроверки.....	38
2. Однослойные нейронные сети	41
2.1. Описание искусственного нейрона в MatLab	41
2.2. Персептрон.....	43
2.3. Линейная нейронная сеть	53
2.4. Рекуррентный метод наименьших квадратов	56
2.5. Линейная сеть с линией задержки	61
Вопросы для самопроверки.....	66
3. Нейронные сети прямого распространения.....	68
3.1. Топология и свойства	68
3.2. Алгоритм обратного распространения ошибки.....	70
3.3. Реализация логических функций	75
3.4. Аппроксимация функций	77
3.5. Распознавание символов.....	84
3.6. Моделирование статистических зависимостей.....	88
3.7. Масштабирование и восстановление данных	95
Вопросы для самопроверки.....	96
4. Нейроуправление	98
4.1. Идентификация динамических звеньев	98
4.2. Нейроэмитаторы и нейропредикторы.....	107
4.3. Концепция нейроуправления	108
4.4. Инверсное нейроуправление	115
4.5. Нейроконтроллеры в MatLab.....	120
Вопросы для самопроверки.....	126
5. Радиальные нейронные сети.....	128
5.1. Структура радиальной нейронной сети	128
5.2. Расчет параметров радиальной нейронной сети.....	131
5.3. Обучение радиальной нейронной сети.....	136

5.4. Радиальные нейронные сети в MatLab	137
5.5. Радиальные нейронные сети и нечеткие системы	146
Вопросы для самопроверки.....	148
6. Модели ассоциативной памяти	150
6.1. Нейронная сеть Элмана	150
6.2. Сети Хопфилда	155
6.3. Двунаправленная ассоциативная память	172
6.4. Нейронная сеть Хэмминга	175
6.5. Адаптивные резонансные нейронные сети.....	182
Вопросы для самопроверки.....	189
7. Нейронные сети Кохонена.....	191
7.1. Структура сети Кохонена.....	191
7.2. Обучение сети Кохонена	196
7.3. Слой Кохонена	199
7.4. Самоорганизующиеся карты Кохонена	204
7.5. Нейронные сети классификаций.....	209
Вопросы для самопроверки.....	213
8. Стохастические методы обучения нейронных сетей	215
8.1. Задача коррекции динамической системы	215
8.2. Методы глобальной оптимизации	216
8.3. Метод имитации отжига	219
8.4. Генетический алгоритм	221
8.5. Метод роя частиц.....	235
8.6. Другие метаэвристические алгоритмы.....	240
Вопросы для самопроверки.....	248
Заключение	251
Библиографический список	254
Приложение 1	259
Приложение 2	275

Учебное издание

Бураков Михаил Владимирович

НЕЙРОННЫЕ СЕТИ
И НЕЙРОКОНТРОЛЛЕРЫ

Учебное пособие

Редактор *А. А. Гранаткина*
Верстальщик *С. Б. Мацапура*

Сдано в набор 05.02.13. Подписано к печати 25.04.13.
Формат 60×84 1/16. Бумага офсетная. Усл. печ. л. 16,39.
Уч.-изд. л. 16,36. Тираж 100 экз. Заказ № 200.

Редакционно-издательский центр ГУАП
190000, Санкт-Петербург, Б. Морская ул., 67

Для заметок