МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное
образовательное учреждение
высшего профессионального образования
«КАЗАНСКИЙ ГОСУДАРСТВЕННЫЙ
ЭНЕРГЕТИЧЕСКИЙ УНИВЕРСИТЕТ»

С.Г. НИКОЛАЕВА

НЕЙРОННЫЕ СЕТИ. РЕАЛИЗАЦИЯ В MATLAB

Учебное пособие по дисциплине
«Интеллектуальные системы»

Казань 2015

УДК 681.3
ББК 22.18
Н62

Рецензенты:
кандидат технических наук, доцент Казанского национального исследовательского технологического университета *С.А. Понкратова;*
кандидат физико-математических наук, доцент Казанского государственного энергетического университета *Н.К. Петрова*

Николаева С.Г.

Н62 Нейронные сети. Реализация в Matlab: учебное пособие / С.Г. Николаева. – Казань: Казан. гос. энерг. ун-т, 2015. – 92 с.

В пособии изложены основы теории искусственных нейронных сетей, описаны некоторые типы нейросетей, а также рассмотрены достаточно распространенные методы обучения нейронных сетей.

Пособие содержит обзор функциональных возможностей стандартного расширения пакета Matlab Neural Network Toolbox с примерами реализации нейросетевых парадигм.

Предназначено для студентов направлений подготовки «Прикладная математика» и «Информатика и вычислительная техника».

УДК 681.3
ББК 22.18

Введение

Интерес к искусственным нейронным сетям быстро вырос за последнее десятилетие. Специалисты из самых разных областей науки и техники овладевают возможностями, предоставляемыми этой технологией, и ищут приложения им внутри своих дисциплин.

Искусственные нейронные сети могут менять свое поведение в зависимости от внешней среды. Этот фактор в большей степени, чем любой другой, ответствен за тот интерес, который они вызывают.

Материал пособия структурирован следующим образом: в первой главе описываются общие понятия теории нейронных сетей, важные характеристики нейронов, основные разновидности нейронных сетей; логическим продолжением является вторая глава, где для выбранных типов нейросетей приведены некоторые методы и алгоритмы их обучения; третья глава включает в себя обзор возможностей расширения Neural Network Toolbox (NNT), необходимых функций для расчета выбранной нейросети, средств визуализации и анализа полученных результатов. В последней главе описаны некоторые примеры решения конкретных задач с помощью реализации нейронных сетей в среде Neural Network Toolbox пакета MATLAB.

Пособие рассчитано на начинающего пользователя искусственных нейронных сетей, в связи с чем приводится достаточно информации для понимания основных идей. Тот, кто изучит пакет для работы с нейросетями, сможет реализовать любую из этих сетей.

Возможно начать освоение материала с просмотра и отработки демонстрационных примеров и проектирования сетей с помощью GUI-интерфейса пакета NNT, в состав которого входит инструментальное средство NNTool (прил. 2). Этот графический интерфейс позволяет, не обращаясь к командному окну системы Matlab, выполнять создание, обучение, моделирование, импорт и экспорт нейронных сетей и данных.

Интерфейс NNTool эффективен лишь на начальной стадии работы с нейросетями, так как допускает работу только с простейшими однослойными и двухслойными нейронными сетями.

Более опытному разработчику сетевых архитектур рекомендуется работать напрямую из командного окна, так как это позволяет воспользоваться всеми возможностями ППП NNT. Следует также обратить внимание на интерфейс NNT с приложением Simulink пакета Matlab, который дает возможность наглядно отображать архитектуру сети и выполнять моделирование как статических, так и динамических нейронных сетей.

Учебное пособие может оказать помощь в формировании у студентов общекультурных и профессиональных компетенций по дисциплине «Интеллектуальные системы» и созвучным ей дисциплинам, а также будет способствовать более полному раскрытию содержательной компоненты конкретных знаний, умений, владений. Поэтому прямо или косвенно материал пособия имеет отношение к следующим компетенциям:

– знание ведущих направлений развития искусственного интеллекта, типы задач, решаемых с помощью интеллектуальных систем (ИС), модели представления знаний;

– знание классификации ИС, методы создания и обучения ИС;

– знание принципов подготовки знаний для базы правил, принципов обработки, размещения информации в базе знаний;

– знание структуры представления фактов, правил предметной области для компьютерной обработки информации;

– знание технологий интеллектуального анализа данных, искусственных нейросистем и возможностей их применения в профессиональных задачах;

– знание технологий для разработки компонентов экспертных систем, баз знаний, программного обеспечения ИС;

– умение структурировать информацию в соответствии с выбранной моделью представления знаний;

– способность разрабатывать ИС (ЭС) для выбранной области знаний на уровне демонстрационного прототипа с применением современных технологий;

– готовность реализовывать такие модели представления знаний, как фреймы, системы продукций, семантические сети, а также генетические алгоритмы и нейронные сети;

– умение использовать компьютер как средство обработки и управления информацией;

– способность осваивать специализированные программные средства для реализации профессиональных задач;

– готовность представлять обоснованный выбор методов решения профессиональных задач с помощью ИС, тип и основные элементы ИС на стадии технического проектирования;

– владение навыками работы с инструментальными средствами создания ИС (CLIPS, нейропакеты);

– владение навыками работы с программным обеспечением для создания ИС и для обучения нейросетей;

– владение эффективными методиками теоретического и практического использования специальных программных средств;

– владение навыками разработки и программной реализации рабочих компонент ИС с помощью нейросетевых технологий.

ГЛАВА 1. НЕЙРОННЫЕ СЕТИ – ЧТО ЭТО?

1.1. Предпосылки создания и развития

Понимание функционирования нейрона и картины его связей позволило исследователям создать математические модели для проверки своих теорий. Эксперименты теперь могут проводиться на цифровых компьютерах без привлечения человека или животных, что решает многие практические и морально – этические проблемы. В первых же работах выяснилось, что эти модели не только повторяют функции мозга, но и способны выполнять функции, имеющие свою собственную ценность. Поэтому возникли и остаются в настоящее время две взаимно обогащающие друг – друга цели нейронного моделирования: первая – понять функционирование нервной системы человека на уровне физиологии и психологии и вторая – создать вычислительные системы (искусственные нейронные сети), выполняющие функции, сходные с функциями мозга.

Параллельно с прогрессом в нейроанатомии и нейрофизиологии психологами были созданы модели человеческого обучения. Одной из таких моделей, оказавшейся наиболее плодотворной, была модель Д. Хэбба, который в 1949 г. предложил закон обучения, явившийся стартовой точкой для алгоритмов обучения искусственных нейронных сетей. Дополненный сегодня множеством других методов, он продемонстрировал ученым того времени, как сеть нейронов может обучаться.

В пятидесятые и шестидесятые годы группа исследователей, объединив эти биологические и физиологические подходы, создала первые искусственные нейронные сети. Выполненные первоначально как электронные сети, они были позднее перенесены в более гибкую среду компьютерного моделирования, сохранившуюся и в настоящее время. Первые успехи вызвали взрыв активности и оптимизма. Минский, Розенблатт, Уидроу и другие разработали сети, состоящие из одного слоя искусственных нейронов. Часто называемые *персептронами*, они были использованы для такого широкого класса задач, как предсказание погоды, анализ электрокардиограмм и искусственное зрение. В течение некоторого времени казалось, что ключ к интеллекту найден и воспроизведение человеческого мозга является лишь вопросом конструирования достаточно большой сети.

Но эта иллюзия скоро рассеялась. Сети не могли решать задачи, внешне весьма сходные с теми, которые они успешно решали. С этих необъяснимых неудач начался период интенсивного анализа. Минский вместе с Пайпертом, используя точные математические методы, доказал, что

однослойные сети теоретически неспособны решить многие простые задачи, в том числе реализовать функцию «Исключающее ИЛИ».

Хотя на несколько десятилетий исследования в этой области фактически были прекращены, несколько наиболее настойчивых ученых, таких как Кохонен, Гроссберг, Андерсон продолжили научные изыскания. Постепенно сформировался теоретический фундамент, на основе которого сегодня конструируются наиболее мощные многослойные сети. Оценка Минского оказалась излишне пессимистичной, многие из поставленных в его книге задач решаются сейчас сетями с помощью стандартных процедур.

За последние несколько лет теория стала применяться в прикладных областях и появились организации, занимающиеся коммерческим использованием этой технологии.

1.2. Биологические основы искусственных нейронных сетей

Искусственные нейронные сети чрезвычайно разнообразны по своим конфигурациям. Несмотря на такое разнообразие, сетевые парадигмы имеют много общего.

Развитие искусственных нейронных сетей вдохновляется биологией. То есть рассматривая сетевые конфигурации и алгоритмы, исследователи мыслят их в терминах организации мозговой деятельности. Но на этом аналогия может и закончиться. Наши знания о работе мозга достаточно ограничены, поэтому разработчикам сетей приходится выходить за пределы современных биологических знаний в поисках структур, способных выполнять полезные функции. Основатели нейросетевых и нейрокомпьютерных технологий ставят целью создание электронных устройств, структурно и функционально адекватных мозгу человека.

Нервная система человека, построенная из элементов, называемых *нейронами*, невероятно сложна. Около 10^{11} нейронов участвуют в примерно 10^{15} передающих связях, имеющих длину метр и более. Уникальной способностью каждого нейрона является прием, обработка и передача электрохимических сигналов по нервным путям, которые образуют коммуникационную систему мозга. Нейрон состоит из трех частей: *тела клетки, дендритов и аксона.*

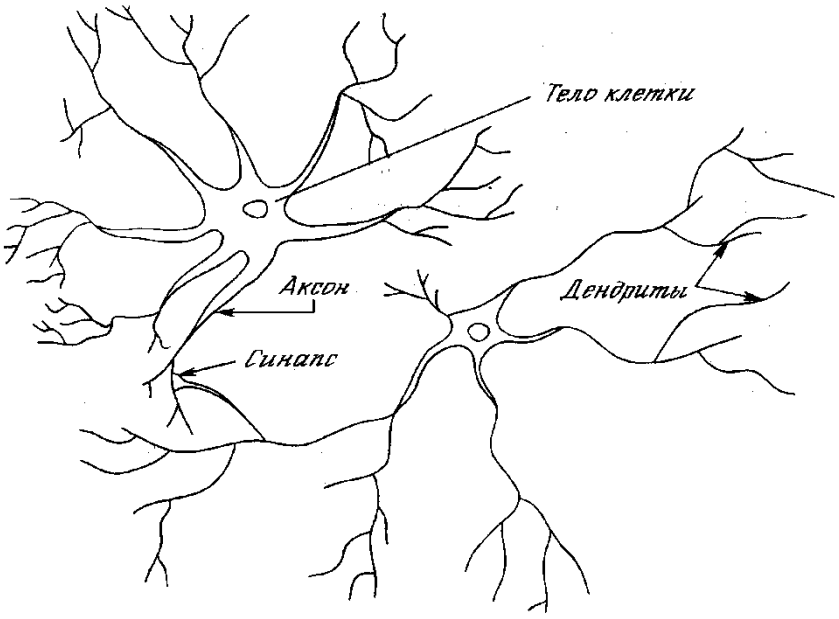

Рис. 1.1. Нейроны человеческого мозга

На рис. 1.1 схематично представлены два типичных биологических нейрона. *Дендриты* идут от тела нервной клетки к другим нейронам, где они принимают сигналы в точках соединения, называемых *синапсами*. Принятые синапсом входные сигналы подводятся к телу нейрона. Здесь они суммируются, причем одни входы стремятся возбудить нейрон, другие – воспрепятствовать его возбуждению. Когда суммарное возбуждение в теле нейрона превышает некоторый порог, нейрон возбуждается и посылает по *аксону* сигнал другим нейронам. Каждый нейрон может существовать в двух состояниях – возбужденном и невозбужденном. У этой основной функциональной схемы много усложнений и исключений, тем не менее большинство искусственных нейронных сетей моделируют лишь эти простые свойства.

1.3. Искусственный нейрон

Искусственный нейрон имитирует в первом приближении свойства биологического нейрона. На вход искусственного нейрона поступает некоторое множество сигналов, каждый из которых является выходом другого нейрона. Каждый вход умножается на соответствующий вес, аналогичный синаптической силе, и все произведения суммируются, определяя уровень активации нейрона.

На рис. 1.2 представлена модель, реализующая эту идею. Хотя сетевые парадигмы весьма разнообразны, в основе почти всех их лежит эта конфигурация. Здесь множество входных сигналов, обозначенных x_1, x_2,…,

x_n, поступает на искусственный нейрон. Эти входные сигналы, в совокупности обозначаемые вектором X, соответствуют сигналам, которые поступают в синапсы биологического нейрона.

Каждый сигнал умножается на соответствующий «вес» w_1, w_2,…, w_n, и поступает на суммирующий блок, обозначенный Σ. Каждый вес соответствует «силе» одной биологической синаптической связи. (Множество весов в совокупности обозначается вектором W.) Суммирующий блок, соответствующий телу биологического элемента, алгебраически складывает взвешенные входы, создавая выход, который обозначим как *S*.

В векторных обозначениях это можно записать следующим образом:

$$S = XW$$

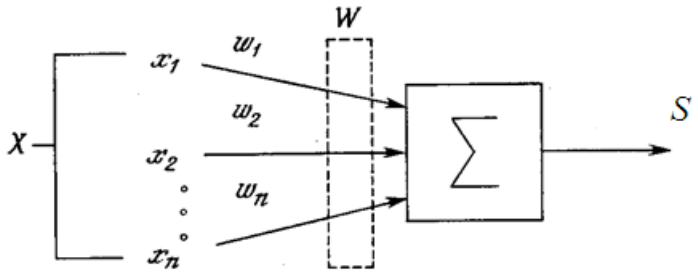

Рис. 1.2. Искусственный нейрон

Полученный сигнал *S* преобразуется *активационной* функцией нейрона, которая формирует выходной сигнал *Y*. От вида активационной функции зависит набор функциональных возможностей нейросети, а также способы ее обучения.

Математический нейрон, как и его биологический прототип, существует в двух состояниях. Если взвешенная сумма входных сигналов *S* не достигает некоторой пороговой величины, то математический нейрон не возбужден и его выходной сигнал равен нулю. Если же входные сигналы достаточно интенсивны и их сумма достигает порога чувствительности, то нейрон переходит в возбужденное состояние и на его выходе образуется сигнал $Y = 1$.

Активационная функция может быть обычной линейной функцией

$$Y = k(S),$$

где *k* – постоянная (рис. 1.3) или пороговой функцией вида

$$Y = 1, \text{ если } S \geq P,$$
$$Y = 0 \text{ в остальных случаях,}$$

где *P* – некоторая постоянная пороговая величина (рис. 1.4).

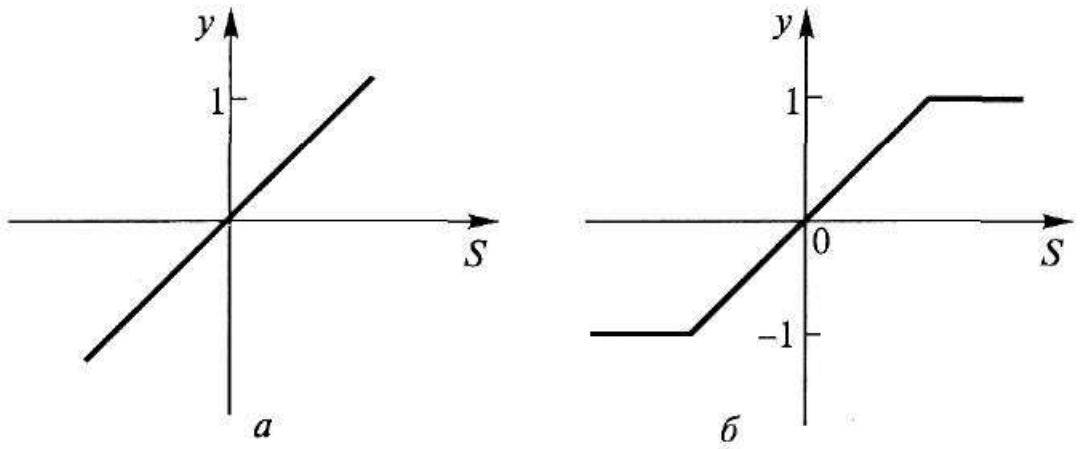

Рис. 1.3. Линейные функции активации:
а) – с неограниченной; б) – с ограниченной областью изменения

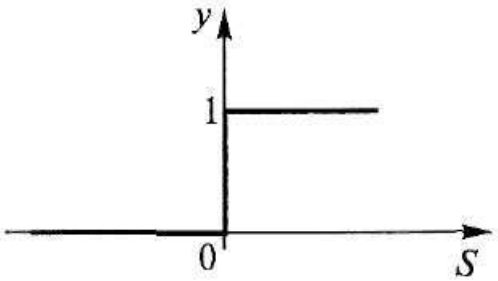

Рис. 1.4. Пороговая функция активации (функция Хевисайда)

Для того чтобы более точно смоделировать нелинейную передаточную характеристику биологического нейрона, в качестве активационной функции нейрона часто используется логистическая или «сигмоидальная» (*S*-образная) функция, показанная на рис. 1.5. Такая функция представляет нейронной сети гораздо большие возможности.

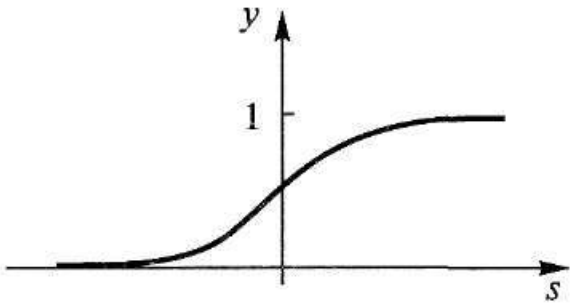

Рис. 1.5. Сигмоидальная логистическая функция

Эта функция была предложена к применению Уидроу и Хоффом, математически она выражается как $F(x) = 1/(1 + \exp(-x))$. Тогда

$$Y = \frac{1}{1 + \exp(-S)}.$$

С введением такой непрерывной нелинейной функции активации круг решаемых задач значительно расширился.

По аналогии с электронными системами активационную функцию можно считать нелинейной усилительной характеристикой искусственного нейрона. Коэффициент усиления представляется отношением приращения величины Y к вызвавшему его небольшому приращению величины S. Он выражается наклоном кривой при определенном уровне возбуждения и изменяется от малых значений при больших отрицательных возбуждениях (кривая почти горизонтальна) до максимального значения при нулевом возбуждении и снова уменьшается, когда возбуждение становится большим положительным.

Другой широко используемой активационной функцией является гиперболический тангенс. По форме она сходна с логистической функцией и часто используется биологами в качестве математической модели активации нервной клетки. В качестве активационной функции искусственной нейронной сети она записывается следующим образом:

$$Y = \text{th}(S).$$

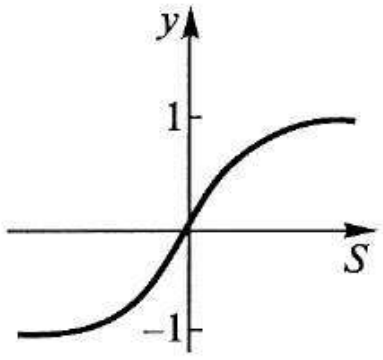

Рис. 1.6. Функция гиперболического тангенса

Так же, как и логистическая функция, гиперболический тангенс является S-образной функцией, но он симметричен относительно начала координат, и в точке $S = 0$ значение выходного сигнала Y равно нулю. В отличие от логистической функции гиперболический тангенс принимает значения различных знаков, что удобно для моделирования некоторых типов сетей.

Наряду с рассмотренными функциями в настоящее время в качестве активационной применяется функция Гаусса, которая имеет вид

$$Y = \exp\left(- \frac{S^2}{2\sigma^2} \right).$$

Здесь σ – параметр кривой Гаусса («ширина окна»), а S – евклидово расстояние между входным вектором X и центром активационной функции C: $S = \|X - C\|$.

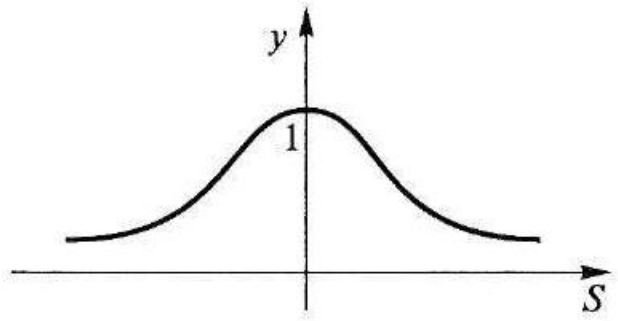

Рис. 1.7. Радиально-базисная активационная функция

Функция Гаусса используется нейронами радиально-базисных нейронных сетей (РБФ – сети).

Рассмотренная модель искусственного нейрона не принимает во внимание задержки во времени, которые влияют на динамику системы. Входные сигналы сразу же порождают выходной сигнал. Она также не учитывает воздействий функции частотной модуляции или синхронизирующей функции биологического нейрона, которые ряд исследователей считают основными. Несмотря на это, сети, построенные из таких нейронов, обнаруживают свойства, напоминающие биологическую систему.

1.4. Основные типы нейронных сетей

Различные способы объединения нейронов между собой и организации их взаимодействия привели к созданию сетей разных типов. Каждый тип сети, в свою очередь, тесно связан с соответствующим методом подбора весов межнейронных связей (т.е. обучения).

Среди множества существующих видов сетей в качестве важнейших можно выделить однослойный и многослойный персептрон, радиальные сети, сети с самоорганизацией в результате конкуренции нейронов, сети с самоорганизацией корреляционного типа, а также рекуррентные сети, в

которых имеются сигналы обратной связи. Отдельно от всех названных типов отстоят нечеткие нейронные сети, функционирование которых основано на принципах нечеткой логики.

Ниже приведена схема, объединяющая основные типы искусственных нейросетей.

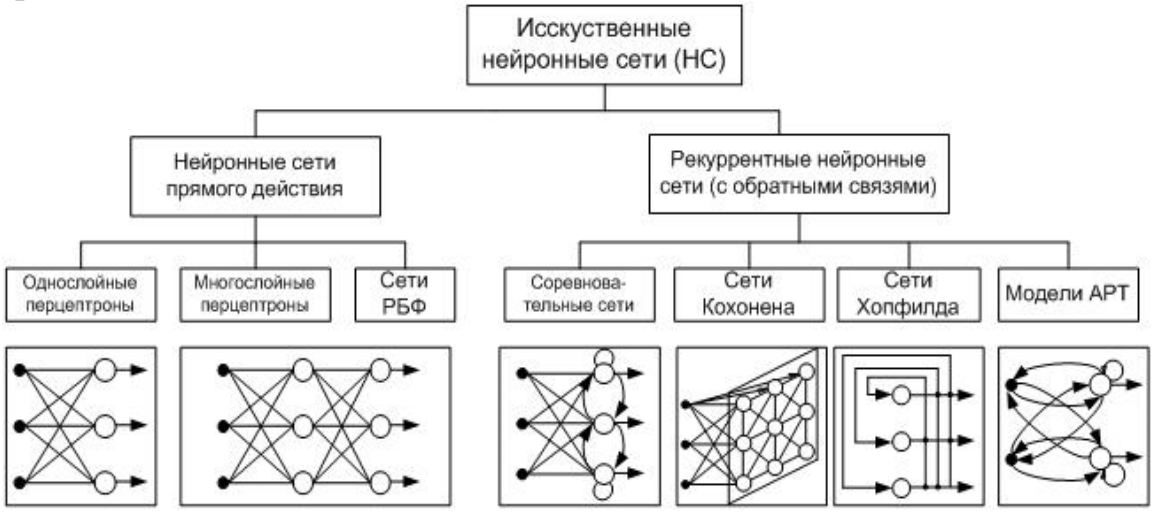

Рис. 1.8. Обобщенная классификация нейронных сетей

Интересным представляется объединение различных видов нейронных сетей между собой, особенно сетей с самоорганизацией и обучаемых с учителем. Такие комбинации получили название «гибридные сети». Первый компонент – это сеть с самоорганизацией на основе конкуренции, функционирующая на множестве входных сигналов и группирующая их в кластеры по признакам совпадения свойств. Она играет роль препроцессора данных. Второй компонент – в виде сети, обучаемой с учителем (например, персептронной), сопоставляет входным сигналам, отнесенным к конкретным кластерам, соответствующие им заданные значения (*постпроцессинг*). Подобная сетевая структура позволяет разделить фазу обучения на две части: вначале тренируется компонент с самоорганизацией, а потом – сеть с учителем. Дополнительное достоинство такого подхода заключается в снижении вычислительной сложности процесса обучения, а также в лучшей интерпретации получаемых результатов.

Однослойная нейронная сеть (однослойный персептрон)

Перцептрон, или персептрон (от лат. perceptio – восприятие) – кибернетическая модель биологического нейрона, предложенная в 1957 году Ф. Розенблаттом и реализованная в виде электронной машины Марк-1 в 1960 году. Персептрон, таким образом, стал одной из первых моделей нейросети, а «Марк-1» – первым в мире нейрокомпьютером. Несмотря на свою простоту,

перцептрон имеет способность к самообучению и может решать довольно сложные задачи. Основная математическая задача, с которой он справляется, – это линейное разделение любых нелинейных множеств.

Персептрон состоит из трёх типов элементов: поступающие от сенсоров сигналы передаются *ассоциативным* элементам, а затем *реагирующим* элементам. Так, персептроны позволяют создать набор «ассоциаций» или связей между входными сигналами и ожидаемой реакцией на выходе. Согласно современной классификации, персептроны могут быть отнесены к нейронным сетям *с одним скрытым слоем; с пороговой передаточной функцией; с прямым распространением сигнала.*

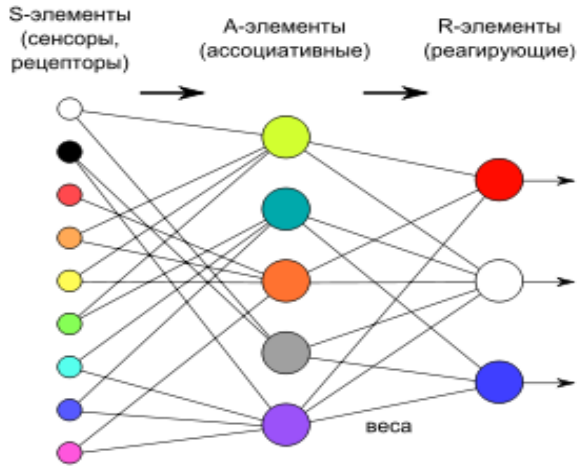

Рис. 1.9. Структура персептрона с тремя выходами

Хотя один нейрон и способен выполнять простейшие процедуры распознавания, сила нейронных вычислений проистекает от соединений нейронов в сетях. Простейшая сеть состоит из группы нейронов, образующих слой, как показано в правой части рис. 1.10.

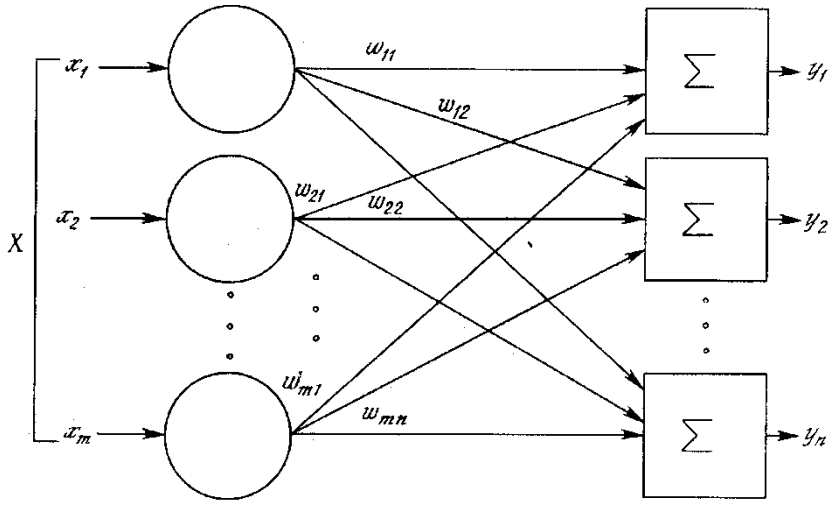

Рис. 1.10. Однослойная нейронная сеть

Отметим, что вершины – круги слева служат лишь для распределения входных сигналов. Они не выполняют каких- либо вычислений, и поэтому не будут считаться слоем. По этой причине они обозначены кругами, чтобы отличать их от вычисляющих нейронов, обозначенных квадратами.

Каждый элемент из множества входов X со своим весом соединен с каждым искусственным нейроном. А каждый нейрон выдает взвешенную сумму входов в сеть. В искусственных и биологических сетях многие соединения могут отсутствовать, все соединения показаны в целях общности. Могут иметь место также соединения между выходами и входами элементов в слое.

Удобно считать веса элементами матрицы W. Матрица имеет *m* строк и *n* столбцов, где m – число входов, а *n* – число нейронов. Например, $w_{2,3}$ – это вес, связывающий третий вход со вторым нейроном. Таким образом, вычисление выходного вектора N, компонентами которого являются выходы *Y* нейронов, сводится к матричному умножению N = XW, где N и X – векторы-строки.

Многослойные нейронные сети

Более крупные и сложные нейронные сети обладают, как правило, и большими вычислительными возможностями. Хотя созданы сети всех конфигураций, какие только можно себе представить, послойная организация нейронов копирует слоистые структуры определенных отделов мозга. Такие многослойные сети обладают большими возможностями, чем однослойные, и в последние годы были разработаны алгоритмы для их обучения.

Многослойные сети могут образовываться каскадами слоев; выход одного слоя является входом для последующего слоя (рис.1.11).

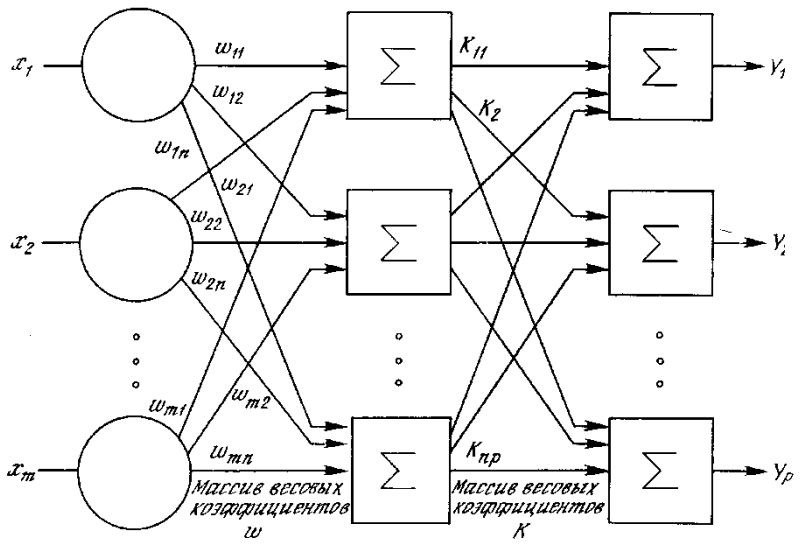

Рис. 1.11. Двухслойная нейронная сеть

До сего дня нет общепринятого способа подсчета числа слоев в сети. Многослойная сеть состоит, как показано на рис. 1.11, из чередующихся множеств нейронов и весов. Согласно рис. 1.10, входной слой не выполняет суммирования. Эти нейроны служат лишь в качестве разветвлений для первого множества весов и не влияют на вычислительные возможности сети. По этой причине первый слой не принимается во внимание при подсчете слоев, и сеть, подобная изображенной на рис. 1.11, считается двухслойной, так как только два слоя выполняют вычисления. Далее, веса слоя считаются связанными со следующими за ними нейронами. Следовательно, слой состоит из множества весов со следующими за ними нейронами, суммирующими взвешенные сигналы.

Многослойные сети не могут привести к увеличению вычислительной мощности по сравнению с однослойной сетью лишь в том случае, если активационная функция между слоями будет нелинейной. Вычисление выхода слоя заключается в умножении входного вектора на первую весовую матрицу с последующим умножением (если отсутствует нелинейная активационная функция) результирующего вектора на вторую весовую матрицу ($XW_1)W_2$. Так как умножение матриц ассоциативно, то $X(W_1W_2)$.

Это показывает, что двухслойная линейная сеть эквивалентна одному слою с весовой матрицей, равной произведению двух весовых матриц. Следовательно, любая многослойная линейная сеть может быть заменена эквивалентной однослойной сетью. Однослойные сети весьма ограничены по своим вычислительным возможностям. Таким образом, для расширения возможностей сетей по сравнению с однослойной сетью необходима нелинейная активационная функция.

Сети РБФ (сети с радиальными базисными функциями)

Нейронная сеть радиальных базисных функций содержит в наиболее простой форме три слоя: обычный входной слой, выполняющий распределение данных образца для первого слоя весов; слой скрытых нейронов с радиально симметричной активационной функцией, выходной слой (рис. 1.12).

Для построения сети РБФ необходимо выполнение следующих условий:

1) наличие эталонов, представленных в виде весовых векторов нейронов скрытого слоя;

2) наличие способа измерения расстояния входного вектора от эталона. Обычно это стандартное евклидово расстояние;

3) специальная функция активации нейронов скрытого слоя, задающая выбранный способ измерения расстояния. Обычно используется функция

Гаусса, которая существенно усиливает малую разницу между входным и эталонным векторами.

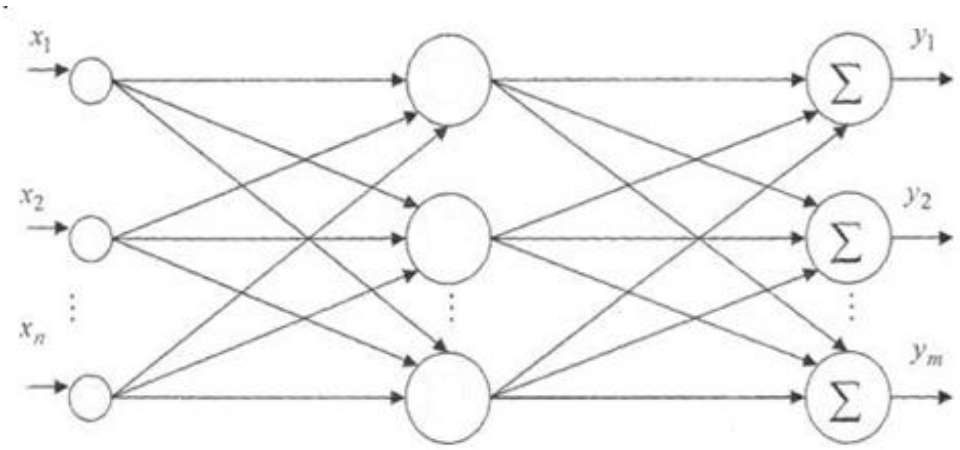

Рис. 1.12. Нейронная сеть с радиальными базисными функциями

Нейронные сети, построенные на РБФ, имеют ряд преимуществ перед рассмотренными многослойными сетями прямого распространения. Во-первых, они моделируют произвольную нелинейную функцию с помощью всего одного промежуточного слоя, тем самым избавляя разработчика от необходимости решать вопрос о числе слоев. Во-вторых, параметры линейной комбинации в выходном слое можно полностью оптимизировать с помощью известных методов линейной оптимизации, которые работают быстро и не испытывают трудностей с локальными минимумами, мешающими при обучении с использованием алгоритма обратного распространения ошибки. Поэтому сеть РБФ обучается очень быстро – на порядок быстрее, чем с использованием алгоритма ОР (обратного распространения).

Недостатки сетей РБФ: данные сети обладают плохими экстраполирующими свойствами и получаются весьма громоздкими при большой размерности вектора входов.

Сети с обратными связями

У сетей, рассмотренных до сих пор, не было обратных связей, т.е. соединений, идущих от выходов некоторого слоя к входам этого же слоя или предшествующих слоев. Этот специальный класс сетей, называемых сетями без обратных связей или сетями прямого распространения, представляет практический интерес и широко используется. Сети более общего вида, имеющие соединения от выходов к входам, называются *сетями с обратными связями*.

У сетей без обратных связей нет памяти, их выход полностью определяется текущими входами и значениями весов. В некоторых конфигурациях сетей с обратными связями предыдущие значения выходов возвращаются на входы; выход, следовательно, определяется как текущим входом, так и предыдущими выходами (рис. 1.13).

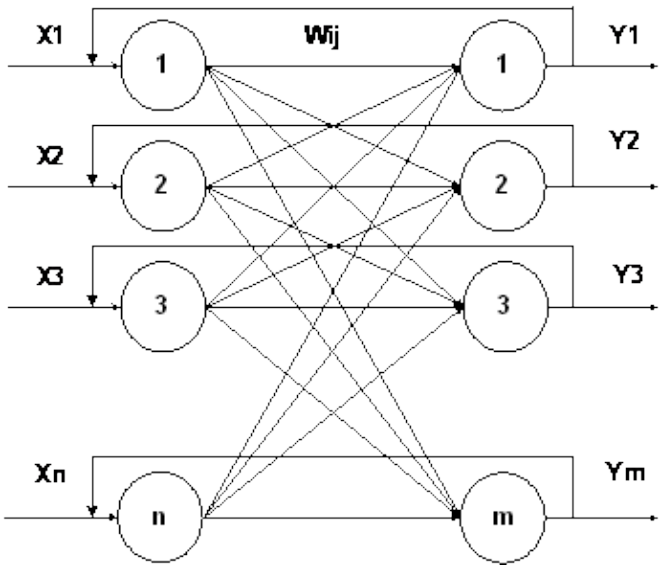

Рис. 1.13. Топология сети с обратными связями

По этой причине сети с обратными связями могут обладать свойствами, сходными с кратковременной человеческой памятью.

Сеть Кохонена

Сеть Кохонена еще называют «самоорганизующейся картой признаков». Сеть такого типа рассчитана на самостоятельное обучение, во время обучения сообщать ей правильные ответы необязательно. В процессе обучения на вход сети подаются различные образцы. Сеть улавливает особенности их структуры и разделяет образцы на кластеры, а уже обученная сеть относит каждый вновь поступающий пример к одному из кластеров, руководствуясь некоторым критерием «близости».

Сеть состоит из одного входного и одного выходного слоя, как это видно из рис. 1.14.

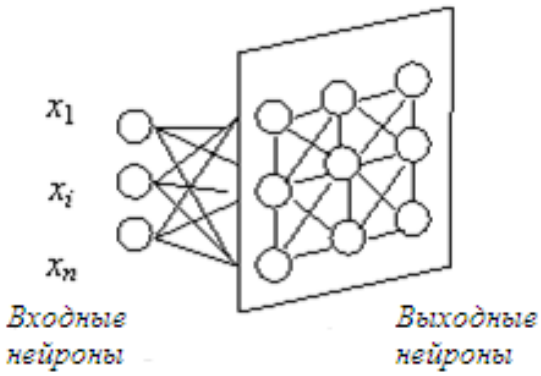

Рис. 1.14. Сеть Кохонена

Количество элементов в выходном слое определяет, сколько различных кластеров сможет распознать сеть. Каждый из выходных элементов получает на вход весь входной вектор. Как и во всякой нейронной сети, каждой связи соответствует некоторый синаптический вес. В большинстве случаев каждый выходной элемент соединен также со своими соседями. Эти внутрислойные связи играют важную роль в процессе обучения, так как корректировка весов происходит только в окрестности того элемента, который наилучшим образом откликается на очередной вход. Выходные элементы *соревнуются* между собой за право вступить в действие и «получить урок». Выигрывает тот из них, чей вектор весов окажется ближе всех к входному вектору.

Сеть Хопфилда

Сеть Хопфилда использует три слоя: входной, *слой Хопфилда* и выходной слой. Каждый слой имеет одинаковое количество нейронов. Входы слоя Хопфилда подсоединены к выходам соответствующих нейронов входного слоя через изменяющиеся веса соединений. Выходы слоя Хопфилда подсоединяются ко входам всех нейронов слоя Хопфилда, за исключением самого себя, а также к соответствующим элементам в выходном слое.

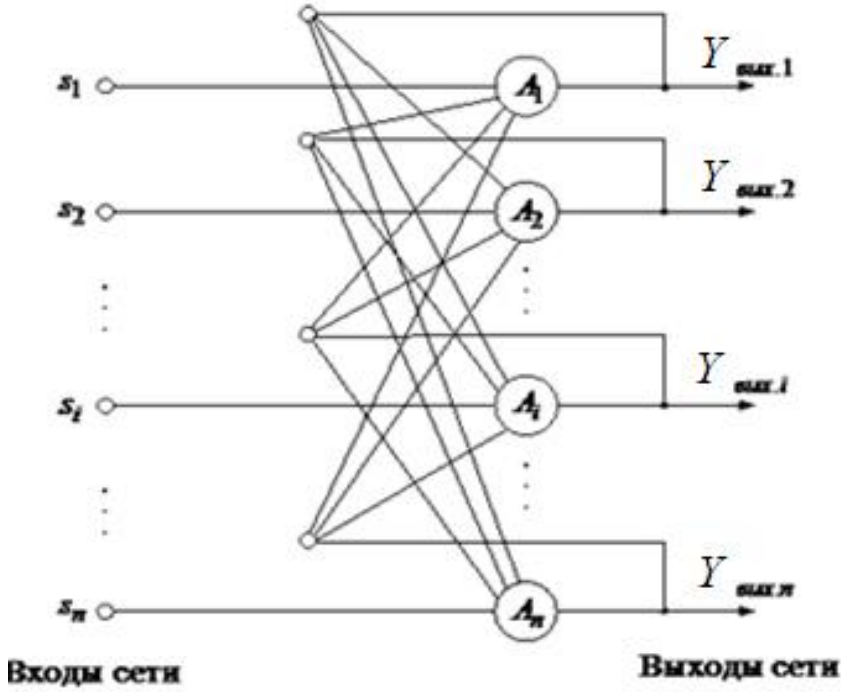

Рис. 1.15. Нейронная сеть Хопфилда

В режиме функционирования сеть направляет данные из входного слоя через фиксированные веса соединений к слою Хопфилда. Структурная схема сети Хопфилда представлена на рис. 1.15.

Сеть Хопфилда состоит из N искусственных нейронов. Граница емкости памяти для сети (то есть количество образов, которое она может запомнить) составляет приблизительно 15 % от числа нейронов в слое Хопфилда (N*0,15).

Размерности входных и выходных сигналов в сети ограничены при программной реализации только возможностями вычислительной системы, на которой моделируется нейронная сеть, при аппаратной реализации – технологическими возможностями. Размерности входных и выходных сигналов совпадают. Каждый нейрон системы может принимать одно из двух состояний (что аналогично выходу нейрона с пороговой функцией активации).

Нейронные сети APT (adaptive resonance theory (ART))

В большинстве нейронных сетей, обучаемых методом обратного распространения, генетическими алгоритмами, в двунаправленной ассоциативной памяти, сетях Хопфилда и т.д. очень часто обучение новому образу, ситуации или ассоциации заметно искажает или даже уничтожает плоды предшествующего обучения, требуя изменения значительной части весов связей или полного переобучения сети. В этом отношении указанные

нейронные сети резко отличаются от мозга человека, который, непрерывно обрабатывая потоки информации из внешней среды, может как модифицировать и уточнять хранящиеся в памяти образы, так и создавать новые, не уничтожая то, что уже хранится. Таким образом, мозг человека обладает высокой пластичностью к поступающей информации, позволяющей ему воспринимать новые образы и уточнять хранящуюся информацию по уже известным, и в то же время он имеет и высокую стабильность, сохраняя ранее полученные знания.

Невозможность с помощью уже известных нейронных сетей решить проблему стабильности – пластичности явилась одной из основных причин разработки принципиально новых конфигураций нейросетей. Примером таких сетей являются нейросети, полученные на основе адаптивной резонансной теории, разработанной Гроссбергом и Карпентером. Эти сети в известной мере позволяют решать противоречивые задачи чувствительности к новым данным и сохранения полученных знаний.

Нейронная сеть адаптивной резонансной теории (АРТ) относит входное изображение к одному из известных классов, если оно в достаточной степени подобно или резонирует с прототипом этого класса. Если найденный прототип с определенной точностью, задаваемой специальным параметром сходства, соответствует входному изображению, то он модифицируется, чтобы стать более похожим на предъявленное изображение. Когда входное изображение недостаточно подобно ни одному из имеющихся прототипов, то на его основе создается новый класс. Это возможно благодаря тому, что сеть имеет большое число избыточных или нераспределенных элементов, которые не используются до тех пор, пока в этом нет необходимости (если нет нераспределенных нейронов, то входное изображение не вызывает реакции сети). Таким образом, новые образы могут создавать новые классы, но не могут исказить существующую память.

Разработано несколько видов нейросетей на основе адаптивной резонансной теории, в частности, сети АРТ-1 и АРТ-2. АРТ-1 предназначена для работы с двоичными входными изображениями или векторами, а АРТ-2 – для классификации как двоичных, так и непрерывнозначных векторов. Хотя детали архитектуры и алгоритмов работы для АРТ-1 и АРТ-2 различны, однако они имеют общую базовую архитектуру.

Как видно из рис. 1.13, сеть типа АРТ-1 состоит из пяти функциональных модулей: двух слоев нейронов – *слоя сравнения* и *слоя распознавания*, и трех управляющих специализированных нейронов – *сброса*, *управления 1* и *управления 2*.

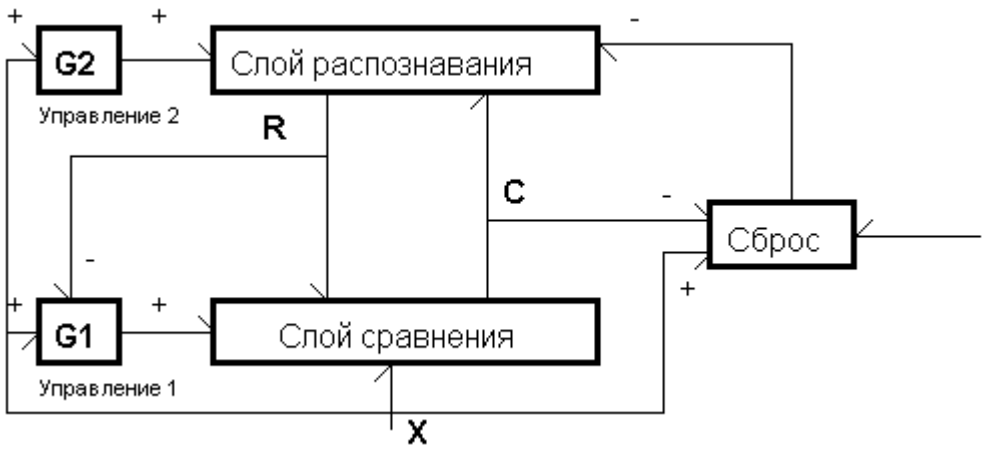

Рис. 1.16. Функциональная схема сети АРТ-1

Слой распознавания служит для отнесения входного вектора X с двоичными компонентами к одному из классов. Каждому классу соответствует один нейрон слоя распознавания. В результате распознавания может быть активирован всего один нейрон. После определения наиболее подходящего класса с помощью слоя сравнения определяется остаточный сигнал, который является разницей между входным образом и образом, соответствующим подобранному классу. Если разница существенна, нейронная сеть фиксирует входной образ в качестве нового класса, то есть в набор нейронов слоя распознавания поступает новый.

Контрольные вопросы

1. Опишите функционирование биологического нейрона.
2. Какие активационные функции подходят для решения нелинейных задач?
3. Как вычисляется значение выходного сигнала нейрона?
4. Можно ли заменить многослойную линейную сеть эквивалентной однослойной сетью?
5. Объясните схему работы двухслойной нейросети.
6. Что представляет собой гибридная сеть? Соревновательная сеть?
7. Дайте описание нейросети Хопфилда.
8. Какие типы элементов содержит персептрон? Каковы функции этих элементов?
9. Опишите особенности нейросетей с обратными связями.
10. Как изменится вычислительная мощность многослойной сети по сравнению с однослойной сетью при использовании нелинейной активационной функции?

ГЛАВА 2. МЕТОДЫ И АЛГОРИТМЫ ОБУЧЕНИЯ НЕЙРОСЕТЕЙ

2.1. Цель обучения

После того как сделан выбор типа сети, следует этап ее обучения. Далее обученную по какому-либо алгоритму нейросеть можно применять для решения прикладных задач. Ниже приведена укрупненная схема создания и использования нейросети (рис. 2.1).

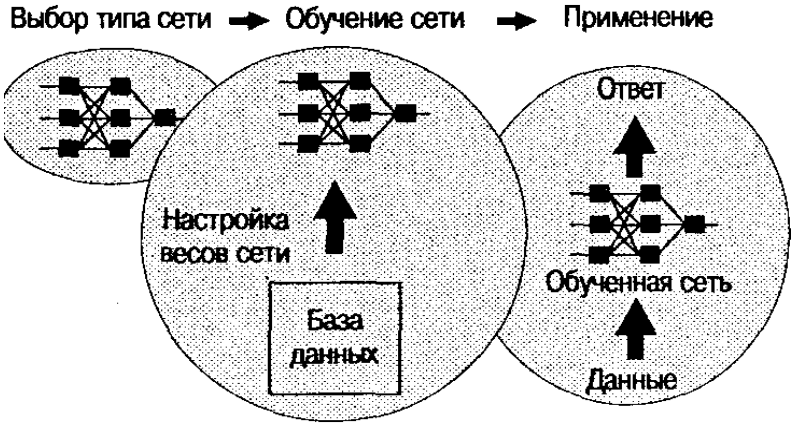

Рис. 2.1. Основные этапы нейросетевого проекта

Среди всех интересных свойств искусственных нейронных сетей ни одно не захватывает так воображения, как их способность к обучению. Их обучение до такой степени напоминает процесс интеллектуального развития человеческой личности, что может показаться, что достигнуто глубокое понимание этого процесса. Тем не менее, возможности обучения искусственных нейронных сетей ограничены, и нужно решить много сложных задач, чтобы достичь цели обучения. На рис. 2.2 обозначен процесс обучения нейронной сети.

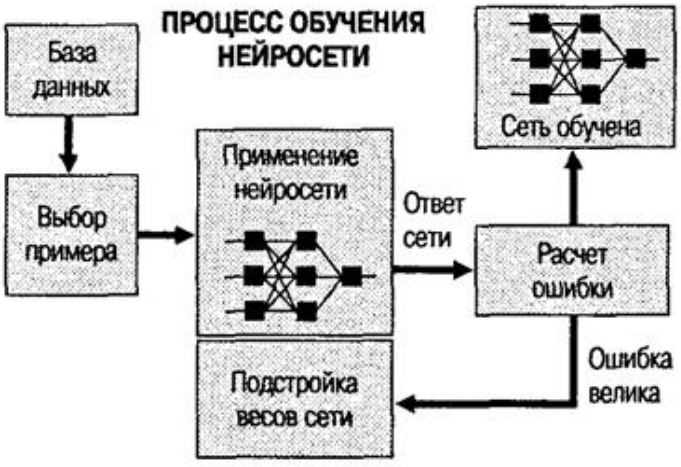

Рис. 2.1. Общий алгоритм обучения НС

Сеть обучается, чтобы для некоторого множества входов давать желаемое (или, по крайней мере, сообразное с ним) множество выходов. Каждое такое входное (или выходное) множество рассматривается как вектор. Обучение осуществляется путем последовательного предъявления входных векторов с одновременной подстройкой весов в соответствии с определенной процедурой. В процессе обучения веса сети постепенно становятся такими, чтобы *каждый входной вектор вырабатывал выходной вектор.*

2.2. Обучение с учителем и без учителя

На сегодня хорошо разработаны алгоритмы обучения нейронных сетей *с учителем и без учителя*. Обучение с учителем предполагает, что для каждого входного вектора существует целевой вектор, представляющий собой требуемый выход. Вместе они называются *обучающей парой*. Обычно сеть обучается на некотором числе таких обучающих пар. Предъявляется входной вектор, вычисляется выход сети и сравнивается с соответствующим целевым вектором, разность (ошибка) с помощью обратной связи подается в сеть и веса изменяются в соответствии с алгоритмом, стремящимся минимизировать ошибку.

Векторы обучающего множества предъявляются на обработку последовательно, вычисляются ошибки и веса подстраиваются для каждого вектора до тех пор, пока ошибка по всему обучающему массиву не достигнет допустимого низкого уровня.

Несмотря на значительные прикладные достижения, обучение с учителем критиковалось за некоторую биологическую неправдоподобность. Сложно представить обучающий механизм в мозге, который бы сравнивал желаемые и действительные значения выходов, выполняя коррекцию с помощью обратной связи. Если допустить подобный механизм в мозге, то откуда тогда возникают желаемые выходы? Обучение без учителя является намного более правдоподобной моделью обучения в биологической системе.

Развитая Кохоненом и многими другими, она не нуждается в целевом векторе для выходов и, следовательно, не требует сравнения с предопределенными идеальными ответами. Обучающее множество состоит лишь из входных векторов. Обучающий алгоритм подстраивает веса сети так, чтобы получались согласованные выходные векторы, т.е. чтобы предъявление достаточно близких входных векторов давало одинаковые выходы. Процедура обучения, следовательно, выделяет статистические свойства обучающего множества и группирует сходные векторы в классы.

Предъявление на вход вектора из данного класса даст определенный выходной вектор, но до обучения невозможно предсказать, какой выход будет производиться данным классом входных векторов. Следовательно, выходы подобной сети должны трансформироваться в некоторую понятную форму, обусловленную процессом обучения. Это не является серьезной проблемой. Обычно не сложно идентифицировать связь между входом и выходом, установленную сетью.

Большинство современных алгоритмов обучения выросло из концепций Хэбба. Им предложена такая модель обучения без учителя, в которой синаптическая сила (вес) возрастает, если активированы оба нейрона, источник и приемник. Таким образом, часто используемые пути в сети усиливаются, и феномен привычки и обучения через повторение получает объяснение.

В искусственной нейронной сети, использующей обучение по Хэббу, наращивание весов определяется произведением уровней возбуждения передающего и принимающего нейронов. Это можно записать как

$$w_{ij}(n+1) = w(n) + \alpha \mathrm{OUT}_i \, \mathrm{OUT}_j,$$

где $w_{ij}(n)$ – значение веса от нейрона i к нейрону j до подстройки, $w_{ij}(n+1)$ – значение веса от нейрона i к нейрону j после подстройки, α – коэффициент скорости обучения, OUT_i – выход нейрона i и вход нейрона j, OUT_j – выход нейрона j.

Сети, использующие обучение по Хэббу, конструктивно развивались, однако за последние 20 лет были развиты более эффективные алгоритмы обучения. В частности, были развиты алгоритмы обучения с учителем, приводящие к сетям с более широким диапазоном характеристик обучающих входных образов и большими скоростями обучения, чем использующие простое обучение по Хэббу.

В настоящее время используется огромное разнообразие обучающих алгоритмов. При решении с помощью нейронных сетей прикладных задач необходимо собрать достаточный и представительный объем данных для того, чтобы обучить нейронную сеть решению таких задач. Обучающий набор данных – это набор наблюдений, содержащих признаки изучаемого объекта. Первый вопрос, какие признаки использовать и сколько и какие наблюдения надо провести.

Выбор признаков, по крайней мере первоначальный, осуществляется эвристически на основе имеющегося опыта, который может подсказать, какие признаки являются наиболее важными. Сначала следует включить все признаки, которые, по мнению аналитиков или экспертов, являются существенными, на последующих этапах это множество будет сокращено.

Нейронные сети работают с числовыми данными, взятыми, как правило, из некоторого ограниченного диапазона. Это может создать проблемы, если значения наблюдений выходят за пределы этого диапазона или пропущены.

Вопрос о том, сколько нужно иметь наблюдений для обучения сети, часто оказывается непростым. Известен ряд эвристических правил, которые устанавливают связь между количеством необходимых наблюдений и размерами сети. Простейшее из них гласит, что количество наблюдений должно быть в 10 раз больше числа связей в сети. На самом деле это число зависит от сложности того отображения, которое должна воспроизводить нейронная сеть. С ростом числа используемых признаков количество наблюдений возрастает по нелинейному закону, так что уже при довольно небольшом числе признаков, скажем 50, может потребоваться огромное число наблюдений. Эта проблема носит название «проклятие размерности».

Для большинства реальных задач бывает достаточным нескольких сотен или тысяч наблюдений. Для сложных задач может потребоваться большее количество, однако очень редко встречаются задачи, где требуется менее 100 наблюдений. Если данных мало, то сеть не имеет достаточной информации для обучения, и лучшее, что можно в этом случае сделать, – это попробовать подогнать к данным некоторую линейную модель.

После того как определено количество слоев сети и число нейронов в каждом из них, нужно назначить значения весов и смещений, которые минимизируют ошибку решения. Это достигается с помощью *процедур обучения.*

Путем анализа имеющихся в распоряжении аналитика входных и выходных данных веса и смещения сети автоматически настраиваются так, чтобы минимизировать разность между желаемым сигналом и полученным на выходе в результате моделирования. Эта разность носит название *ошибки обучения*. Таким образом, процесс обучения – это процесс подгонки параметров той модели процесса или явления, которая реализуется нейронной сетью. Ошибка обучения для конкретной конфигурации нейронной сети определяется путем прогона через сеть всех имеющихся наблюдений и сравнения выходных значений с желаемыми, *целевыми значениями*. Эти разности позволяют сформировать так называемую *функцию ошибок* (*критерий качества обучения*). В качестве такой функции чаще всего берется сумма квадратов ошибок. При моделировании нейронных сетей с линейными функциями активации нейронов можно построить алгоритм, гарантирующий достижение абсолютного минимума ошибки обучения. Для нейронных сетей с нелинейными функциями активации в

общем случае нельзя гарантировать достижения глобального минимума функции ошибки.

При таком подходе к процедуре обучения может оказаться полезным геометрический анализ поверхности функции ошибок. Определим веса и смещения как свободные параметры модели и их общее число обозначим через N; каждому набору таких параметров поставим в соответствие одно измерение в виде ошибки сети. Тогда для всевозможных сочетаний весов и смещений соответствующую ошибку сети можно изобразить точкой в N+1-мерном пространстве, а все такие точки образуют некоторую поверхность, называемую *поверхностью функции ошибок*. При таком подходе цель обучения нейронной сети состоит в том, чтобы найти на этой многомерной поверхности глобальный минимум.

В случае линейной модели сети и функции ошибок в виде суммы квадратов такая поверхность будет представлять собой параболоид, который имеет единственный минимум, и это позволяет отыскать такой минимум достаточно просто.

В случае нелинейной модели поверхность ошибок имеет гораздо более сложное строение и обладает рядом неблагоприятных свойств, в частности может иметь локальные минимумы, плоские участки, седловые точки и длинные узкие овраги.

Определить глобальный минимум многомерной функции аналитически невозможно, и поэтому обучение нейронной сети, по сути дела, является процедурой изучения поверхности функции ошибок. Отталкиваясь от случайно выбранной точки на поверхности функции ошибок, алгоритм обучения постепенно отыскивает глобальный минимум. Как правило, для этого вычисляется градиент (наклон) функции ошибок в данной точке, а затем эта информация используется для продвижения вниз по склону. В конце концов алгоритм останавливается в некотором минимуме, который может оказаться лишь локальным минимумом, а если повезет, то и глобальным.

Таким образом, по существу алгоритмы обучения нейронных сетей аналогичны алгоритмам поиска глобального экстремума функции многих переменных. Среди последних можно выделить алгоритмы сопряженных градиентов и Левенберга – Марквардта.

Однако с учетом специфики нейронных сетей для них разработаны специальные алгоритмы обучения, среди которых следует выделить алгоритм обратного распространения ошибки.

При использовании алгоритма обратного распространения ошибки сеть рассчитывает возникающую в выходном слое ошибку и вычисляет вектор градиента как функцию весов и смещений. Этот вектор указывает направление кратчайшего спуска по поверхности для данной точки, поэтому если

продвинуться в этом направлении, то ошибка уменьшится. Последовательность таких шагов в конце концов приведет к минимуму того или иного типа. Определенную трудность здесь вызывает выбор величины шага.

При большой длине шага сходимость будет более быстрой, но имеется опасность перепрыгнуть через решение или уйти в неправильном направлении. Классическим примером такого явления при обучении нейронной сети является ситуация, когда алгоритм очень медленно продвигается по узкому оврагу с крутыми склонами, перепрыгивая с одного склона на другой. Напротив, при малом шаге, вероятно, будет выбрано верное направление, однако при этом потребуется очень много итераций. На практике величина шага выбирается пропорциональной крутизне склона (градиенту функции ошибок); такой коэффициент пропорциональности называется *параметром скорости настройки*. Правильный выбор параметра скорости настройки зависит от конкретной задачи и обычно осуществляется опытным путем; этот параметр может также зависеть от времени, уменьшаясь по мере выполнения алгоритма.

Алгоритм действует итеративно, и его шаги принято называть *эпохами* или *циклами*. На каждом цикле на вход сети последовательно подаются все обучающие наблюдения, выходные значения сравниваются с целевыми значениями и вычисляется функция ошибки. Значения функции ошибки, а также ее градиента используются для корректировки весов и смещений, после чего все действия повторяются. Начальные значения весов и смещений сети выбираются случайным образом, и процесс обучения прекращается либо когда реализовано определенное количество циклов, либо когда ошибка достигнет некоторого малого значения или перестанет уменьшаться.

2.3. Явление переобучения

Одна из наиболее серьезных трудностей при обучении сети заключается в том, что в ряде случаев минимизируется не та ошибка, которую на самом деле нужно минимизировать; требуется минимизировать ошибку, которая появляется в сети, когда на нее подаются совершенно новые наблюдения. Важно, чтобы нейронная сеть обладала способностью приспосабливаться к этим новым наблюдениям. В действительности сеть обучается минимизировать ошибку на некотором ограниченном обучающем множестве. Это не отвечает требованиям теории о наличии идеального и бесконечно большого обучающего множества. И это не соответствует той реальной ситуации, когда надо минимизировать конкретную функцию ошибок для заранее неизвестной модели.

Эти причины порождают проблему, которая известна как *явление переобучения*. Обратимся к задаче аппроксимации некоторой функции многочленом. Графики многочленов часто имеют весьма замысловатые формы, и чем выше степень многочлена, тем сложнее их форма. Если имеется некоторый набор данных, то можно поставить цель подобрать для него аппроксимирующий многочлен и таким образом получить подходящую математическую модель для этого набора данных. Поскольку исходные данные, как правило, заданы с погрешностями, то нельзя считать, что лучшая модель задается кривой, которая проходит точно через заданные точки. Многочлен низкого порядка может оказаться достаточно грубым для аппроксимации данных, в то время как многочлен высокого порядка может точно следовать данным, принимая при этом весьма замысловатую форму, не имеющую никакого отношения к форме истинной зависимости. Последняя ситуация и демонстрирует то, что называется *явлением переобучения*.

При работе с нейронными сетями пользователь сталкивается с той же проблемой. Сети с большим количеством весов позволяют воспроизводить очень сложные функции, и в этом смысле они склонны к переобучению. Сеть же с небольшим количеством весов может оказаться недостаточно гибкой, чтобы смоделировать имеющуюся зависимость. Например, однослойная линейная сеть способна воспроизводить только линейные функции. Если использовать многослойные линейные сети, то ошибка всегда будет меньше, но это может свидетельствовать не о хорошем качестве модели, а о том, что проявляется явление переобучения.

Для того чтобы выявить эффект переобучения, используется механизм *контрольной проверки*. Часть обучающих наблюдений резервируется как контрольные наблюдения и не используется при обучении сети. Вместо этого по мере работы алгоритма эти наблюдения применяются для независимого контроля результата. Вначале ошибка сети на обучающем и контрольном множествах будет одинаковой; если они существенно отличаются, то, вероятно, это означает, что разбиение наблюдений на 2 множества не обеспечило их однородность. По мере обучения сети ошибка убывает, и, пока обучение уменьшает функцию ошибок, ошибка на контрольном множестве также будет убывать. Если же контрольная ошибка перестала убывать или стала расти, это указывает на то, что сеть начала слишком близко следовать исходным данным и обучение следует остановить. В этом случае следует уменьшить количество нейронов или слоев, ибо сеть является слишком мощной для решения данной задачи. Если же, наоборот, сеть имеет недостаточную мощность, чтобы воспроизвести имеющуюся зависимость, то явление переобучения скорее всего наблюдаться не будет и обе ошибки – обучения и проверки – не достигнут достаточно малого уровня.

Возникающие при работе с нейронными сетями проблемы отыскания глобального минимума или выбора размера сети приводят к тому, что при практической работе приходится экспериментировать с большим числом сетей различных конфигураций, порой обучая каждую из них несколько раз и сравнивая полученные результаты. Главным критерием выбора в этих случаях является *контрольная погрешность*. При этом применяется правило, согласно которому из двух нейронных сетей с приблизительно равными контрольными погрешностями следует выбирать ту, которая проще.

Необходимость многократных экспериментов ведет к тому, что контрольное множество начинает играть ключевую роль в выборе модели нейронной сети, т.е. становится частью процесса *обучения*. Тем самым его роль как независимого критерия качества модели ослабляется, поскольку при большом числе экспериментов возникает риск *переобучения* нейронной сети на контрольном множестве. Для того чтобы гарантировать надежность выбираемой модели сети, резервируют еще одно – *тестовое* множество наблюдений. Итоговая модель тестируется на данных из этого множества, чтобы убедиться, что результаты, достигнутые на обучающем и контрольном множествах, реальны. Разумеется, для того чтобы хорошо играть свою роль, тестовое множество должно быть использовано только 1 раз: если его использовать повторно для корректировки процесса обучения, то оно фактически превратится в контрольное множество.

Итак, процедура построения нейронной сети состоит из следующих шагов:

– выбор начальной конфигурации сети; например, в виде одного слоя с числом нейронов, равным 1/2 общего количества входов и выходов;

– моделирование и обучение сети с оценкой контрольной ошибки и использованием дополнительных нейронов или промежуточных слоев;

– выявление эффекта переобучения и корректировки конфигурации сети.

2.4. Свойство обобщения

При описании процедуры обучения нейронных сетей неявно использовалось предположение, что обучающее, контрольное и тестовое множества являются *представительными* для решаемой задачи. Обычно в качестве обучающих берутся данные, испытанные на ряде примеров. Если обстоятельства изменились, то закономерности, имевшие место в прошлом, могут больше не действовать.

Кроме того, нейронная сеть может обучаться только на тех данных, которыми она располагает. Предположим, что известно обучающее множество для системы стабилизации самолета при полете в спокойной

атмосфере, а требуется спроектировать систему стабилизации на основе нейронной сети для условий полета при сильных возмущениях. Тогда едва ли можно ожидать от сети правильного решения в совершенно новой для нее ситуации.

Классическим примером непредставительной модели нейронной сети является следующая ситуация. При проектировании системы машинного зрения, предназначенной для автоматического распознавания целей, сеть обучалась на 100 картинках, содержащих изображения танков, и на 100 других картинках, где танков не было. После обучения сети был достигнут стопроцентно правильный результат. Но когда на вход сети были поданы новые данные, она безнадежно провалилась. В чем же была причина? Выяснилось, что фотографии с танками были сделаны в пасмурный, дождливый день, а фотографии без танков – в солнечный день. Сеть научилась улавливать разницу в общей освещенности. Чтобы сеть могла результативно работать, ее следовало обучать на данных, где бы присутствовали все погодные условия и типы освещения, при которых сеть предполагается использовать, и это не говоря еще о рельефе местности, угле и дистанции съемки и т.д.

Если сеть минимизирует общую погрешность, большое значение приобретают пропорции, в которых представлены данные различных типов. Сеть, обученная на 900 «хороших» и 100 «плохих» наблюдениях, будет искажать результат в пользу хороших наблюдений, поскольку это позволит алгоритму уменьшить общую погрешность. Если в реальной ситуации «хорошие» и «плохие» объекты представлены в другой пропорции, то результаты, выдаваемые сетью, могут оказаться неверными. Примером этого может быть задача выявления заболеваний. Пусть, например, при обычных обследованиях в среднем 90 % людей оказываются здоровыми и сеть, таким образом, обучается на данных, в которых пропорция «здоровые/больные» равна 90/10. Затем эта же сеть применяется для диагностики пациентов с определенными жалобами, среди которых соотношение «здоровые/больные» уже 50/50. В этом случае сеть будет ставить диагноз чересчур осторожно и не будет распознавать заболевание у некоторых больных. Если же, наоборот, сеть обучить на данных «с жалобами», а затем протестировать на «обычных» данных, то она будет выдавать повышенное число неправильных диагнозов о наличии заболевания. В таких ситуациях обучающие данные нужно скорректировать так, чтобы были учтены различия в распределении данных (например, можно повторить редкие наблюдения или удалить часто встречающиеся). Как правило, лучше всего постараться сделать так, чтобы наблюдения различных типов были представлены равномерно, и соответственно этому интерпретировать результаты, которые выдает сеть.

Способность сети, обученной на некотором множестве данных, выдавать правильные результаты для достаточно широкого класса новых данных, в том числе и не представленных при обучении, называется *свойством обобщения* нейронной сети.

Другой подход к процедуре обучения сети можно сформулировать, если рассматривать ее как процедуру, обратную моделированию. В этом случае требуется подобрать такие значения весов и смещений, которые обеспечивали бы нужное соответствие между входами и желаемыми значениями на выходе. Такая процедура обучения носит название *процедуры адаптации* и достаточно широко применяется для настройки параметров нейронных сетей.

Важно отметить, что вся информация, которую сеть имеет о задаче, содержится в наборе примеров. Поэтому качество обучения сети напрямую зависит от количества примеров в обучающей выборке, а также от того, насколько полно эти примеры описывают данную задачу. Так, например, бессмысленно использовать сеть для предсказания финансового кризиса, если в обучающей выборке кризисов не представлено. Считается, что для полноценной тренировки требуется хотя бы несколько десятков (а лучше сотен) примеров. Математически процесс обучения можно описать следующим образом.

В процессе функционирования нейронная сеть формирует выходной сигнал Y в соответствии с входным сигналом X, реализуя некоторую функцию $Y = G(X)$. Если архитектура сети задана, то вид функции G определяется значениями синаптических весов и смещений сети. Пусть решением некоторой задачи является функция $Y = F(X)$, заданная парами входных и выходных данных (X^1, Y^1), (X^2, Y^2),..., (X^N, Y^N), для которых $Y^k = F(X^k)$ $(k = 1, 2, ..., N)$.

Тогда обучение состоит в поиске (построении) функции G, близкой к F в смысле некоторой функции ошибки E.

Если выбрано множество обучающих примеров – пар $(X^к, Y^*)$ (где $k = 1, 2, ..., N$) и способ вычисления функции ошибки E, то обучение нейронной сети превращается в задачу многомерной оптимизации, имеющую очень большую размерность, при этом, поскольку функция E может иметь произвольный вид, обучение в общем случае является многоэкстремальной невыпуклой задачей оптимизации.

Для решения этой задачи могут быть использованы следующие алгоритмы:

– алгоритмы локальной оптимизации с вычислением частных производных первого порядка – градиентный алгоритм (метод наискорейшего спуска); методы с одномерной и двумерной оптимизацией целевой функции

в направлении антиградиента; метод сопряженных градиентов; методы, учитывающие направление антиградиента на нескольких шагах алгоритма;

– алгоритмы локальной оптимизации с вычислением частных производных первого и второго порядка – метод Ньютона, методы оптимизации с разреженными матрицами Гессе, квазиньютоновские методы, метод Гаусса –Ньютона, метод Левенберга – Марквардта и т.п.;

– стохастические алгоритмы оптимизации – поиск в случайном направлении, имитация отжига, метод Монте-Карло (численный метод статистических испытаний);

– алгоритмы глобальной оптимизации (задачи глобальной оптимизации решаются с помощью перебора значений переменных, от которых зависит целевая функция).

2.5. Алгоритм обратного распространения

На сегодняшний день имеется много серьезных демонстраций возможностей искусственных нейронных сетей: сеть научили превращать текст в фонетическое представление, которое затем с помощью уже иных методов превращалось в речь; другая сеть может распознавать рукописные буквы; сконструирована система сжатия изображений, основанная на нейронной сети. Все они используют алгоритм обратного распространения ошибки (back propagation) – один из наиболее распространенных алгоритмов обучения НС. Это итеративный градиентный алгоритм обучения, который используется с целью минимизации среднеквадратичного отклонения текущего выхода от желаемого выхода в многослойных нейронных сетях.

Алгоритм обратного распространения используется для обучения многослойных нейронных сетей с последовательными связями. Как отмечено выше, нейроны в таких сетях делятся на группы с общим входным сигналом – слои, при этом на каждый нейрон первого слоя подаются все элементы внешнего входного сигнала, а все выходы нейронов q-го слоя подаются на каждый нейрон слоя (q+1).

Нейроны выполняют взвешенное (с синаптическими весами) суммирование элементов входных сигналов; к данной сумме прибавляется смещение нейрона. Над полученным результатом затем выполняется нелинейное преобразование с помощью активационной функции. Значение функции активации есть выход нейрона.

В многослойных сетях оптимальные выходные значения нейронов всех слоев, кроме последнего, как правило, неизвестны, и персептрон с тремя и более слоями уже невозможно обучить, руководствуясь только величинами ошибок на выходах НС. Наиболее приемлемым вариантом обучения в таких

условиях оказался градиентный метод поиска минимума функции ошибки с рассмотрением сигналов ошибки от выходов НС к ее входам, то есть в направлении, обратном прямому распространению сигналов в обычном режиме работы. Этот алгоритм обучения НС получил название *процедуры обратного распространения.*

В данном алгоритме функция ошибки представляет собой сумму квадратов рассогласования (ошибки) желаемого выхода сети и реального. При вычислении элементов вектора градиента использован своеобразный вид производных функций активации сигмоидального типа. Алгоритм действует циклически (итеративно), и его циклы принято называть *эпохами.* На каждой эпохе на вход сети поочередно подаются все обучающие наблюдения, выходные значения сети сравниваются с целевыми значениями и вычисляется ошибка. Значения ошибки, а также градиента поверхности ошибок используются для корректировки весов, после чего все действия повторяются. Начальная конфигурация сети выбирается случайным образом, и процесс обучения прекращается либо когда пройдено определенное количество эпох, либо когда ошибка достигнет некоторого определенного уровня малости, либо когда ошибка перестанет уменьшаться (пользователь может сам выбрать нужное условие остановки).

Приведем словесное описание алгоритма.

Шаг 1. Весам сети присваиваются небольшие начальные значения.

Шаг 2. Выбирается очередная обучающая пара (*X, Y)* из обучающего множества; вектор X подается на вход сети.

Шаг 3. Вычисляется выход сети.

Шаг 4. Вычисляется разность между требуемым (целевым, *Y)* и реальным (вычисленным) выходом сети.

Шаг 5. Веса сети корректируются так, чтобы минимизировать ошибку (сначала веса выходного слоя, затем, с использованием правила дифференцирования сложной функции и отмеченного своеобразного вида производной сигмоидальной функции, – веса предыдущего слоя и т.п.).

Шаг 6. Шаги со 2-го по 5-й повторяются для каждой пары обучающего множества до тех пор, пока ошибка на всем множестве не достигнет приемлемой величины.

Шаги 2 и 3 подобны тем, которые выполняются в уже обученной сети.

Вычисления в сети выполняются послойно. На шаге 3 каждый из выходов сети вычитается из соответствующего компонента целевого вектора с целью получения ошибки. Эта ошибка используется на шаге 5 для коррекции весов сети.

Шаги 2 и 3 можно рассматривать как «проход вперед», так как сигнал распространяется по сети от входа к выходу. Шаги 4 и 5 составляют

«обратный проход», поскольку здесь вычисляемый сигнал ошибки распространяется обратно по сети и используется для подстройки весов.

Классический метод обратного распространения относится к алгоритмам с линейной сходимостью. Его известными недостатками являются: невысокая скорость сходимости (большое число итераций, требуемых для достижения минимума функции ошибки), возможность сходиться не к глобальному, а к локальным решениям (локальным минимумам отмеченной функции). Возможен также *паралич сети*, при котором большинство нейронов функционируют при очень больших значениях аргумента функции активации, то есть на ее пологом участке (поскольку ошибка пропорциональна производной, которая на данных участках мала, то процесс обучения практически замирает).

Для устранения этих недостатков были предложены многочисленные модификации алгоритма обратного распространения, которые связаны с использованием различных функций ошибки, различных процедур определения направления и величины шага и т.п.

Обратное распространение не свободно от проблем. Прежде всего нет гарантии, что сеть может быть обучена за конечное время. Много усилий, израсходованных на обучение, пропадает напрасно после затрат большого количества машинного времени. Когда это происходит, попытка обучения повторяется – без всякой уверенности, что результат окажется лучше. Нет также уверенности, что сеть обучится наилучшим возможным образом. Алгоритм обучения может попасть в ловушку так называемого локального минимума и будет получено худшее решение.

Разработано много других сетевых алгоритмов обучения, имеющих свои специфические преимущества. Следует подчеркнуть, что никакая из сегодняшних сетей не является панацеей, все они страдают от ограничений в своих возможностях обучаться и вспоминать.

Контрольные вопросы

1. Каковы цели обучения нейросети?
2. Назовите основные методы и алгоритмы обучения нейросетей.
3. В чем состоит свойство обобщения нейронной сети?
4. Какие этапы включает в себя процедура построения нейронной сети?
5. Что такое тестовое множество? Контрольное множество? Можно ли использовать тестовое множество более одного раза?
6. Охарактеризуйте обучение с учителем и обучение НС без учителя.
7. Дайте описание явления переобучения нейросети.

8. Как выполняется наращивание весов нейронной сети, обучаемой по Хеббу?

9. Какими достоинствами и недостатками обладает классический метод «back propagation»?

10. Какое количество элементов должен иметь входной вектор? Что такое «проклятие размерности»?

ГЛАВА 3. ИНСТРУМЕНТ NEURAL NETWORK TOOLBOX ПАКЕТА MATLAB

3.1. Возможности NNT

Пакет для работы с нейронными сетями Neural Network Toolbox представляет собой полноценную среду MATLAB для решения прикладных задач, обеспечивая поддержку проектирования, обучения и моделирования множества известных сетевых парадигм, от базовых моделей персептрона до самых современных ассоциативных и самоорганизующихся сетей. Пакет может быть использован для исследования и применения нейронных сетей к таким задачам, как обработка сигналов, нелинейное управление и финансовое моделирование.

Основные возможности пакета NNT:

– управляемые сетевые парадигмы: персептрон, линейные, обратного распространения, Левенберга, радиальный базис, Элмана, Хопфилда и самообучаемое квантование векторов;

– неуправляемые сети: Хэбб, Кохонен, конкурентные, карты признаков и самоорганизующиеся карты;

– конкурентные, предельные, линейные и сигмоидальные передаточные функции;

– неограниченное число элементов и взаимосвязей;

– настраиваемые на пользователя архитектуры и передаточные функции;

– *модульная организация.* Пакет использует согласованную, модульную реализацию, которая облегчает исследования и упрощает настройку на пользователя. Пакет не накладывает искусственные ограничения на размер сети или связность – т.е. не ограничено число нейронов в слое или в типе передаточной функции.

– *архитектуры и обучающие правила.* В пакет включены более 15 известных типов сетей и обучающих правил, позволяющих пользователю выбирать наиболее подходящую для конкретного приложения или исследовательской задачи парадигму. Для каждого типа архитектуры и

обучающих правил имеются функции инициализации, обучения, адаптации, создания и моделирования, демонстрации и пример приложения сети.

– *управляемые и неуправляемые сети.* Для управляемых сетей можно выбрать прямую или рекуррентную архитектуру, используя множество обучающих правил и методов проектирования, таких как персептрон, обратное распространение, обратное распространение Левенберга, сети с радиальным базисом и рекуррентные сети.

– *для неуправляемых сетей* можно выбрать ассоциативные или самоорганизующиеся сети, такие как конкурентные, карты свойств и самоорганизующиеся карты. Ассоциативные сети можно использовать как составляющие блоки для более сложных сетей с использованием обучающих правил Хебба, Кохонена, внутренних или внешних (instar or outstar) ассоциативных обучающих правил.

– *Neural Network как инженерная среда.* Пакет Neural Network предоставляет доступ к полному набору средств для исследования, проектирования и моделирования нейронных сетей. Средства анализа и моделирования MATLAB позволяют быстро оценивать поведение сети и ее качество в смысле окончательного результата проектирования.

3.2. Обзор функций Neural Networks Toolbox

В состав пакета Neural Networks Toolbox входят более 150 различных функций, которые разбиты на несколько групп, предоставляя пользователю широкие возможности по созданию, обучению и применению различных нейросетей.

Введя в окне «Command Window» команду

» help nnet

можно получить перечень входящих в пакет NNT функций. Для получения справки по интересующей функции используется команда

» help имя_функции

3.2.1. Функции активации (transfer functions) и связанные с ними функции

– compet(X) – «соревновательная» функция. В качестве аргумента используется матрица X, столбцы которой ассоциируются с векторами входов. Возвращает разреженную матрицу с единичными элементами, индексы которых соответствуют индексам наибольших элементов каждого столбца.

Пример 3.1:

» X = [0.9 -0.6; 0.1 0.4; 0.2 -0.5: 0 0.5];

```
» compete(X)
ans =
        (1,1)    1
        (4,2)    1
```

В виде запроса compet(code) возвращается служебная информация. Переменная code может принимать значения 'deriv' (имя производной функции), 'name' (полное имя), 'output' (диапазон выхода), 'active' (возможный диапазон входов).

Пример 3.2:

```
» compete ('deriv')
ans =
     ' '
» compet ('name')
ans =
Competitive

» compet ('output')
ans =
   0     1

» compet ('active')
ans =
   -Inf    Inf
```

Данная функция используется при создании НС со слоем «соревнующихся» нейронов (как, например, в сетях встречного распространения).

– hardlim(X) – пороговая функция активации с порогом $P = 0$; аргумент имеет тот же смысл, что и для предыдущей команды. Возвращает матрицу, размер которой равен размеру матрицы X, а элементы имеют значения 0 или 1 в зависимости от знака соответствующего элемента X.

Пример 3.3:

```
» X = [0.9  -0.6; 0.1  0.4; 0.2  -0.5; 0  0.5];
» hardlim(X)
ans =
1   0
1   1
1   0
1   1
```

В виде hardlim(code) функция возвратит информацию, аналогичную рассмотренной для функции compet.

– hardlims(X) – знаковая или сигнатурная функция активации; работает так же, как функция hardlim(X), но возвращает значения -1 или +1.

– logsig(X) – сигмоидальная логистическая функция. Результат: матрица, элементы которой являются значениями логистической функции от аргументов, которыми служат элементы матрицы X.

– poslin(X) – возвращает матрицу значений полулинейной функции.

– purelin(X) – возвращает матрицу значений линейной функции активации.

– radbas(X) – возвращает матрицу значений радиальной базисной функции.

– satlin(X) – возвращает матрицу значений полулинейной функции с насыщением.

– satlins(X) – возвращает матрицу значений линейной функции с насыщением.

– softmax(X) – возвращает матрицу, элементы которой вычисляются по формуле $\frac{e^{x_{ij}}}{\sum_{i=1`}^{N} e^{x_{ij}}}$, где N – число строк матрицы – аргумента X.

– tansig(X) – возвращает матрицу значений сигмоидальной функции (гиперболический тангенс).

– tribas(X) – возвращает матрицу значений треугольной функции принадлежности.

– dhardlim(X,Y) – производная пороговой функции активации. Аргументами являются матрица входов X и матрица выходов Y; матрицы имеют одинаковый размер. Результат: матрица того же размера с нулевыми элементами.

– dhardlms(X,Y) – производная знаковой функции активации. Возвращается матрица с нулевыми элементами.

– dlogsig(X.Y) – производная сигмоидальной логистической функции. Результат – матрица с элементами $y_{ij}(1 – y_{ij})$.

Пример 3.4:

```
» X = [0.1;  0.8;  -0.7];
» Y = logsig(X)
Y =
    0.5250
```

```
    0.6900
    0.3318

» dY_dX = dlogsig(X,Y)
dYdX =
          0.2494
          0.2139
          0.2217
```

– dposlin(X,Y) – производная полулинейной функции. Результатом служит матрица с элементами, равными 1 для соответствующих положительных элементов матрицы-аргумента Y и равными 0 в противоположном случае.

– dpurelin(X,Y) – производная линейной функции активации. Возвращается матрица с единичными элементами.

– dradbas(X,Y) – производная радиальной базисной функции. Будет возвращена матрица с элементами $-2x_{ij}\, y_{ij}$.

– dsatlin(X,Y) – возвращает матрицу значений производной полулинейной функции с насыщением. Элементы этой матрицы – единицы, если соответствующие элементы матрицы Y принадлежат интервалу (0, 1), и нули в противоположном случае.

– dsatlins(X,Y) – возвращает матрицу значений производной линейной функции с насыщением. Элементы матрицы – единицы, если соответствующие элементы матрицы Y принадлежат интервалу (-1, 1), иначе это нули.

– dtansig(X,Y) – возвращает матрицу значений производной сигмоидальной функции - гиперболического тангенса. Элементы матрицы вычисляются как $(1 - y_{ij}^2)$.

– dtribas(X,Y) – возвращает матрицу значений производной треугольной функции активации. Элементы матрицы определяются следующим образом: это 1, если $-1 < y_{ij} < 0$; -1, если $0 < y_{ij} < 1$; иначе 0.

3.2.2. Функции обучения нейронных сетей (training functions)

Данные функции позволяют задавать алгоритм и параметры обучения НС выбранной конфигурации. Перечислим входящие в группу функции.

[net,tr] = trainbfg(net,Pd,Tl,Ai,Q,TS,VV,TV) – функция обучения, реализующая разновидность квазиньютоновского алгоритма обратного распространения ошибки (BFGS). Аргументы функции подробнее:

– net – имя обучаемой НС;

– Pd – наименование массива задержанных входов обучающей выборки;

– Tl – массив целевых значений выходов;

– Ai – матрица начальных условий входных задержек;

– Q – количество обучающих пар в одном цикле обучения (размер «пачки»);

– TS – вектор временных интервалов;

– VV – пустой ([ ]) массив или массив проверочных данных;

– TV – пустой ([ ]) массив или массив тестовых данных.

Данная функция возвращает обученную нейронную сеть net и набор записей tr для каждого цикла обучения (tr. epoch – номер цикла, tr.perf – текущая ошибка обучения, tr.vperf – текущая ошибка для проверочной выборки, tr.tperf – текущая ошибка для тестовой выборки).

Обучение выполняется в соответствии со значениями следующих параметров (в скобках приведены значения по умолчанию):

net.trainParam.epochs (100) – заданное количество циклов обучения;

net.trainParam.show (25) – количество циклов для показа промежуточных результатов;

net.trainParam.goal (0) – целевая ошибка обучения;

net.trainParam.time (∞) – максимальное время обучения в секундах;

net.trainParam.min_grad (10^{-6}) – целевое значение градиента;

net.trainParam.max_fail (5) – максимально допустимая кратность превышения ошибки проверочной выборки по сравнению с достигнутым минимальным значением;

net.trainParam.searchFcn ('srchcha') – имя используемого одномерного алгоритма оптимизации.

Структуры и размеры массивов:

Pd – No ^{x} Ni ^{x} TS - массив ячеек, каждый элемент которого P{i, j, ts} есть матрица Dij ^{x} Q;

Tl – Nl ^{x} TS - массив ячеек, каждый элемент которого P{i, ts} есть матрица Vi ^{x} Q;

Ai – Nl ^{x} LD – массив ячеек, каждый элемент которого Ai{i, k} есть матрица Si ^{x} Q, где

Ni = net.numlnputs (количество входов сети);

N1 = net.numLayers (количество слоев сети);

LD = net.numLayerDelays (количество слоев задержки);

Ri = net.inputs{i}.size (размер i-го входа);

Si = net.layers{i}.size (размер i-го слоя);

Vi = net.targets{i}.size (размер целевого вектора);

Dij = Ri * length(net.inputWeights{i,j}.delays) (вспомогательная вычисляемая величина).

Если массив VV не пуст, то он должен иметь структуру, определяемую следующими компонентами:

VV.PD – задержанные значения входов проверочной выборки;

VV.Tl – целевые значения;

VV.Ai – начальные входные условия;

VV.Q – количество проверочных пар в одном цикле обучения;

VV.TS – временные интервалы проверочной выборки.

Эти параметры используются для задания останова процесса обучения, когда ошибка для проверочной выборки не уменьшается или возрастает.

Структуру, аналогичную структуре массива VV, имеет массив TV. Рассматриваемая функция, заданная в форме trainbfg (code), возвращает для значений аргумента 'pnames' и 'pdefauls', соответственно, имена параметров обучения и их значения по умолчанию.

Для использования функции в нейросети, задаваемой пользователем, необходимо:

Установить параметр net.trainFcn = 'trainbfg' (при этом параметры алгоритма будут заданы по умолчанию).

Установить требуемые значения параметров (net.trainParam). Обучение сети будет остановлено по выполнении любого из следующих условий:

– превышение заданного количества циклов обучения (net.trainParam.epochs);

– превышение заданного времени обучения (net.trainParam.time);

– ошибка обучения стала меньше заданной (net.trainParam.goal);

– градиент стал меньше заданного (net.trainParam.min_grad);

– возрастание ошибки проверочной выборки по сравнению с достигнутым минимальным превысило заданное значение (net. train Param.max_fail).

Пример 3.5:

```
» P = [0 1 2 3 4 5];                % Задание входного вектора
» T = [0 0 0 1 1 1];                % Задание целевого вектора
»   % Создание и тестирование сети
» net = newff([0  5],[2  1],{'tansig','logsig'},'traincgf');
» a = sim(net,P)
a =
0.0586    0.0772    0.0822    0.0870    0.1326    0.5901

»   % Обучение с новыми параметрами и повторное тестирование
» net.trainParam.searchFcn = 'srchcha';
```

```
» net.trainParam.epochs = 50;
» net.trainParam.show = 10;
» net.trainParam.goal = 0.1;
» net = train(net,P,T);
TRAINCGF – srchcha, Epoch 0/50, MSE 0.295008/0.1, Gradient 0.623241/le-006
TRAINCGF – srchcha, Epoch 1/50, MSE 0.00824365/0.1, Gradient 0.0173555/le-006
TRAINCGF, Performance goal met.
» a = sim(net,P)
a =
0.0706    0.1024    0.1474    0.9009    0.9647    0.9655
```

В этом примере созданная многослойная НС, обученная с установками по умолчанию, показывает вначале плохой результат отображения обучающей выборки, но после изменения параметров и повторного обучения сети результат становится вполне приемлемым.

[net,tr] = trainbr(net,Pd,Tl,Ai,Q,TS,VV) – функция, реализующая байесов-ский метод обучения, сущность которого заключается в подстройке весов и смещений сети на основе алгоритма Левенберга – Марквардта.

Алгоритм минимизирует комбинацию квадратов ошибок и весов с выбором оптимального варианта (для получения наилучших обобщающих свойств сети). Все аргументы, параметры, возвращаемые величины и их использование такие же, как у предыдущей функции. Это верно и для всех остальных функций данной группы.

[net,tr] = traincgb(net,Pd,Tl,Ai,Q,TS,VV) – функция обучения НС, реализующая разновидность алгоритма сопряженных градиентов.

[net,tr] = traincgf(net,Pd,Tl,Ai,Q,TS,VV) – функция обучения НС, реализующая алгоритм обратного распространения ошибки в сочетании с методом оптимизации Флетчера-Поуэлла.

[net,tr] = traincgp(net,Pd,Tl,Ai,Q,TS,VV) – то же, что в предыдущем случае, но с использованием метода Полака-Рибейры.

[net,tr] = traingd(net,Pd,Tl,Ai,Q,TS,VV) – функция, реализующая классический алгоритм обратного распространения ошибки.

[net,tr] = traingda(net,Pd,Tl,Ai,Q,TS,VV) – то же, что в предыдущем случае, но с адаптацией коэффициента скорости обучения.

[net,tr] = traingdm(net,Pd,Tl,Ai,Q,TS,VV) – функция, реализующая модифицированный алгоритм обратного распространения ошибки с введенной «инерционностью» коррекции весов и смещений.

[net,tr] = traingdx(net,Pd,Tl,Ai,Q,TS,VV) – функция, реализующая комбинированный алгоритм обучения, объединяющий особенности двух вышеприведенных.

[net,tr] = trainlm(net,Pd,Tl,Ai,Q,TS,VV) – данная функция возвращает веса и смещения НС, используя алгоритм оптимизации Левенберга-Марквардта.

[net,tr] = trainoss(net,Pd,Tl,Ai,Q,TS,VV) – функция, реализующая разновидность алгоритма обратного распространения ошибки с использованием метода секущих.

[net,tr] = trainrp(net,Pd,Tl,Ai,Q,TS,VV) – функция, реализующая разновидность алгоритма обратного распространения ошибки, так называемый *упругий* алгоритм обратного распространения (resilient backpropagation algorithm, RPROP).

[net,tr] = trainscg(net,Pd,Tl,Ai,Q,TS,VV) – данная функция возвращает веса и смещения НС, используя алгоритм масштабируемых сопряженных градиентов.

[net,tr] = trainwb(net,Pd,Tl,Ai,Q,TS,VV) – данная функция корректирует веса и смещения сети в соответствии с заданной функцией обучения нейронов.

[net,tr] = trainwb1(net,Pd,Tl,Ai,Q,TS,VV) – то же, что и предыдущая функция, но одновременно на вход сети предъявляется только один вектор входа.

[net,Ac,El] = adaptwb(net,Pd,Tl,Ai,Q,TS) – функция адаптации весов и смещений НС. Используется совместно с функциями newp и newlin (см. ниже). Возвращает массив выходов слоев Ac и массив ошибок слоев El.

3.2.3. Функции для настройки нейронных слоев

Функции этой группы являются вспомогательными при работе с некоторыми уже рассмотренными функциями обучения НС (например, trainwb, trainwb1, adaptwb), а также могут использоваться при настройках однослойных нейросетевых структур (персептронов, слоев Кохонена и т.п.).

[dB,LS] = learncon(B,P,Z,N,A,T,E,gW,gA,D,LP,LS) – функция настройки весов с введением «чувства справедливости». Аргументы подробнее:

– B - S ^{x} 1 – вектор смещений;

– P - 1 ^{x} Q – входной вектор;

– Z - S ^{x} Q – матрица взвешенных входов;

– N - S ^{x} Q – матрица входов;

– A - S ^{x} Q – матрица выходных векторов;

– T - S ^{x} Q – матрица целевых векторов слоя;

– E - S ^{x} Q – матрица ошибок;

– gW- S ^{x} R – градиент критерия эффективности по отношению к вектору весов;

– gA - S ^{x} Q – градиент критерия эффективности по отношению к вектору выхода;

– D - S ^{x} S – матрица расстояний между нейронами;

- LP – параметр обучения, LP = [ ];

– LS – состояние обучения, в начале – [ ].

Функция возвращает следующие величины:

– dB - S ^{x} 1 – вектор изменений весов (или смещений);

– LS – новое состояние обучения.

Функция в форме learncon(code) выдает следующую информацию:

– при аргументе 'pnames' – имена параметров обучения;

– при 'pdefaults' – значения параметров по умолчанию;

– при 'needg' – 1, если эта функция использует gW или gA. Алгоритм выполнения функции вначале вычисляет «чувство справедливости» нейрона по формуле c = (1 - lr)*c + lr*a, а затем корректирует вес в соответствии с формулой: b = exp(1 - log(c)) - b.

Пример 3.6:

```
» a = rand(3,1);
» b = rand(3,l);
» 1p.1r = 0.5;   % Задание параметра обучения
» dW = learncon(b,[],[],[],a,[],[],[],[],[],lp,[])
dW =
0.3449
0.7657
0.5405
```

learngd – функция коррекции весов и смещений, реализующая градиентный алгоритм оптимизации.

Синтаксис:

```
[dW,LS] = learngd(W,P,Z,N,A,T,E,gW,gA,D,LP,LS)
[db,LS] = learngd(b,ones(1,Q),Z,N.A,T,E,gW,gA,D,LP,LS)
info = learngd(code)
```

Аргументы функции: W – матрица весов или вектор смещения, остальные аргументы, а также возвращаемые параметры – как у функции learncon.

learngdm – функция практически аналогична предыдущей; используемый алгоритм оптимизации - градиентный метод с инерционной составляющей.

[dW,LS] = learnh(W,P,Z,N,A,T,E,gW,gA,D,LP,LS) – функция коррекции весов, использующая правило Хебба, в соответствии с которым веса корректируются по выражению dw = lr*a*p′.

[dW,LS] = learnhd(W,P,Z,N,A,T,E,gW,gA,D,LP,LS) – функция реализует модификацию правила Хебба, при котором корректировка весов выполняется по формуле dw = lr*a*p' - dr*w.

[dW,LS] = learnis(W,P,Z,N,A,T,E,gW,gA,D,LP,LS) – функция подстройки весов «входной звезды» (нейрона слоя Гроссберга), реализующая выражение dw = lr*a*(p' - w).

[dW,LS] = learnk(W,P,Z,N,A,T,E,gW,gA,D,LP,LS) – функция подстройки весов слоя Кохонена, реализующая выражение dw = lr*(p' - w), если a ≠ 0, и 0 в противном случае.

[dW,LS] = learnlv*(W,P,Z,N,A,T,E,gW,gA,D,LP,LS) – функции настройки сетей встречного распространения (вместо «*» может быть «1» или «2»).

[dW,LS] = learnos(W,P,Z,N,A,T,E,gW,gA,D,LP,LS) – функция настройки нейрона типа «выходная звезда», реализующая выражение dw = lr*(a - w)*p'.

[dW,LS] = learnp(W,P,Z,N,A,T,E,gW,gA,D,LP,LS) – функция, реализующая алгоритм обучения персептрона.

[dW,LS] = learnpn(W,P,Z,N,A,T,E,gW,gA,D,LP,LS) – то же, что и предыдущая функция, но с нормализацией входов. Работает эффективнее при больших изменениях входных сигналов.

[dW,LS] = learnsom(W,P,Z,N,A,T,E,gW,gA,D,LP,LS) – функция обучения самоорганизующихся карт.

[dW,LS] = learnwh(W,P,Z,N,A,T,E,gW,gA,D,LP,LS) – функция обучения, реализующая так называемый алгоритм Уидроу-Хоффа, основанный на соотношении dw = lr*e*pn' (дельта-правило или правило наименьших квадратов).

3.2.4. Функции, реализующие методы одномерной оптимизации

Данные функции можно считать вспомогательными для функций обучения нейронных сетей. Они реализуют различные алгоритмы одномерного поиска.

srchbac – функция реализует так называемый алгоритм перебора с возвратами (backtracking).

srchbre – функция реализует комбинированный метод оптимизации, объединяющий метод золотого сечения и квадратичную интерполяцию.

srchcha – функция реализует разновидность метода оптимизации с применением кубической интерполяции.

srchgol – функция реализует метод золотого сечения.

srchhyb – функция реализует комбинированный метод оптимизации, объединяющий метод дихотомии и кубическую интерполяцию.

3.2.5. Функции инициализации слоев и смещений

Некоторые нейросети требуют выполнить этап инициализации перед проведением обучения (смысл инициализации – задание обычно выбираемых случайным образом весов и смещений сети). Этой цели служат нижеприведенные функции.

initcon(s,pr) – функция, устанавливающая смещения нейронов в зависимости от среднего выхода нейрона. Аргументы: s – количество нейронов, pr = [Pmin Pmax] – матрица (с двумя столбцами) минимальных и максимальных значений входов, по умолчанию [1 1]. Возвращает вектор смещений, используется совместно с командой learncon.

Пример 3.7:

```
» b = initcon(3)
b =
8.1548
8.1548
8.1548
```

initzero – функция задания нулевых начальных значений весам или смещениям. Аргументы как и в предыдущей команде.

midpoint(S,PR) – функция инициализации, устанавливающая веса в соответствии со средними значениями входов.

randnc(S,R) – функция задания матрицы весов. Результат – матрица размером $S \times R$ со случайными элементами, нормализованная по столбцам (векторы-столбцы имеют единичную длину).

randnr(S,R) – то же, что предыдущая функция, с возвратом матрицы весов, нормализованной по строкам.

rands – функция инициализации весов/смещений заданием их случайных значений из диапазона [-1, 1].

Синтаксис:

W = rands(S,PR)

M = rands(S,R)

V = rands(S)

Аргументы – те же, что и для функции initcon; значение R по умолчанию - 1. Результат – матрица соответствующего размера.

3.2.6. Функции создания нейронных сетей

network – создает нейронной сети пользователя.

Синтаксис:

net = network

net =
network(numInputs,numlayers,biasConnect,inputConnect,layerConnect,
outputConnect,targetConnect)

Функция network возвращает созданную нейронную сеть с именем net и со следующими характеристиками (в скобках даны значения по умолчанию):

– numInputs – количество входов (0);

– numLayers – количество слоев (0);

– biasConnect – булевский вектор с числом элементов, равным количеству слоев (нули);

– inputConnect – булевская матрица с числом строк, равным количеству слоев, и числом столбцов, равным количеству входов (нули);

– layerConnect – булевская матрица с числом строк и столбцов, равным количеству слоев (нули);

– outputConnect – булевский вектор-строка с числом элементов, равным количеству слоев (нули);

– targetConnect – вектор-строка такая же, как предыдущая (нули).

net = newc(PR,S,KLR,CLR) – функция создания слоя Кохонена. Описание аргументов:

– PR - R ^{x} 2 – матрица минимальных и максимальных значений для R входных элементов;

– S – число нейронов;

– KLR – коэффициент обучения Кохонена (по умолчанию 0,01);

– CLR – коэффициент «справедливости» (по умолчанию 0,001).

Функция возвращает слой Кохонена с заданным именем.

net = newcf(PR,[S1 S2...SN1],{TF1 TF2...TFN1},BTF,BLF,PF) – функция создания разновидности многослойной НС с обратным распространением ошибки – каскадной нейросети. Такая сеть содержит N1 скрытых слоев, использует входные функции типа dotprod и netsum, инициализация сети осуществляется функцией initnw. Аргументы функции:

– PR - R ^{x} 2 – матрица минимальных и максимальных значений R входных элементов;

– Si – размер i-го скрытого слоя, для N1 слоев;

– TFi – функция активации нейронов r-го слоя, по умолчанию 'tansig';

– BTF – функция обучения сети, по умолчанию 'traingd';

– BLF – функция настройки весов и смещений, по умолчанию 'learngdm';

– PF – функция ошибки, по умолчанию 'mse'.

Пример 3.8:

» P = [0 1 2 3 4 5 6 7 8 9 10];

» T= [0 123432123 4];

» net = newcf([0 10],[5 1], {'tansig' 'purelin'}); % Создание новой сети

» net.trainParam.epochs = 50; % Задание количества циклов обучения

» net = train(net,P,T); % Обучение НС

TRAINLM, Epoch 0/50, MSE 7.77493/0, Gradient 138.282/le-010

TRAINLM, Epoch 25/50, MSE 4.01014e-010/0, Gradient 0.00028557/le-010 TRAINLM, Epoch 50/50, MSE 1.13636e-011/0, Gradient 1.76513e-006/le-010 TRAINLM, Maximum epoch reached, performance goal was not met.

» Y = sim(net,P); % Использование НС

» plot(P,T,P,Y,'o') % Графическая иллюстрация работы сети

На рис. 3.1 элементы обучающей выборки отображены точками, сплошная линия - выход сети.

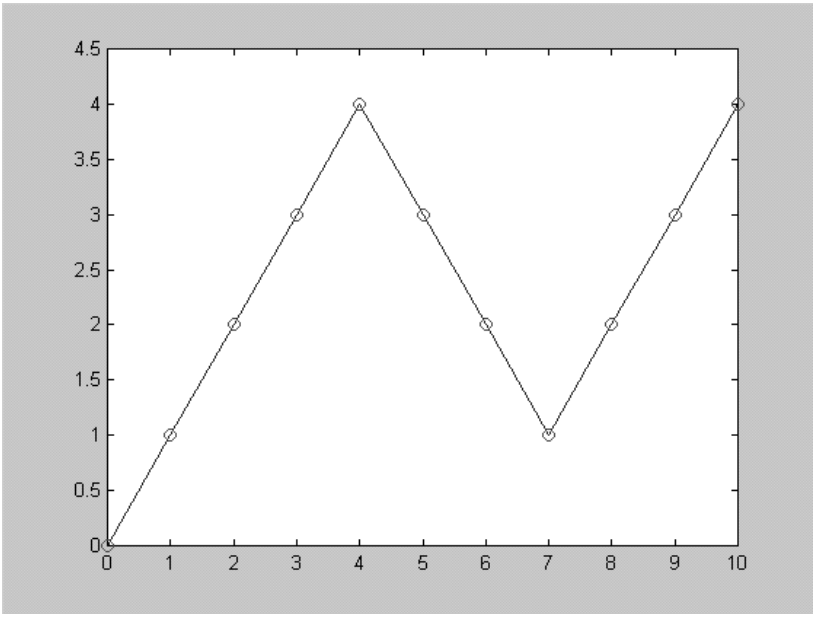

Рис. 3.1. Работа нейросети

net = newelm(PR,[S1 S2...SN1],{TF1 TF2...TFN1},BTF,BLF,PF) – функция создания сети Элмана. Аргументы – такие же, как и у предыдущей функции.

net = newff(PR,[S1 S2...SN1], {TF1 TF2…TFN1},BTF,BLF,PF) – функция создания «классической» многослойной НС с обучением по методу обратного распространения ошибки.

net = newfftd(PR,ID,[S1 S2..SN1],{TF1 TF2...TFN1},BTF,BLF,PF) – выполняет то же, что и предыдущая функция, но с наличием задержек по входам. Аргумент ID здесь – это вектор входных задержек.

net = newgrnn(P,T,spread) – функция создания обобщенно-регрессионной сети. Значения аргументов:

– P - R ^{x} Q – матрица Q входных векторов;

– T - S ^{x} Q – матрица Q целевых векторов;

spread – отклонение (по умолчанию 1,0).

net = newhop(T) – функция создания сети Хопфилда. Имеет только один аргумент:

– T - R ^{x} Q – матрица Q целевых векторов (значения элементов должны быть +1 или -1).

net = newlin(PR,S,ID,LR) – функция создания слоя линейных нейронов. Аргументы:

– PR – R ^{x} 2 – матрица минимальных и максимальных значений для R входных элементов;

– S – число элементов в выходном векторе;

– ID – вектор входной задержки (по умолчанию [0]);

– LR – коэффициент обучения (по умолчанию 0,01).

В качестве результата функция возвращает новый линейный слой. При записи в форме net = newlin(PR,S,0,P) используется аргумент P – матрица входных векторов, при этом возвращается линейный слой с максимально возможным коэффициентом обучения при заданной матрице P.

net = newlind(P,T) – функция проектирования нового линейного слоя. Данная функция по матрицам входных и выходных векторов методом наименьших квадратов определяет веса и смещения линейной НС.

net = newlvq(PR,S1,PC,LR,LF) – функция создания сети встречного распространения. Аргументы:

– PR - R ^{x} 2 – матрица минимальных и максимальных значений R входных элементов;

– S1 – число скрытых нейронов;

– PC – S2 элементов вектора, задающих доли принадлежности к различным классам;

– LR – коэффициент обучения, по умолчанию 0,01;

– LF – функция обучения, по умолчанию 'learnlv1'.

net = newp(PR,S,TF,LF) – функция создания персептрона. Аргументы:

– PR - R ^{x} 2 – матрица минимальных и максимальных значений R входных элементов,

– S – число нейронов;

– TF – функция активации, по умолчанию 'hardlim';

– LF – функция обучения, по умолчанию 'learnp'.

net = newpnn(P,T,spread) – функция создания вероятностной НС. Аргументы - как у функции newgrnn.

net = newrb(P,T,goal,spread) – функция создания сети с радиальными базисными элементами. Аргументы P, T, spread – такие же, как у функции newgrnn; аргумент goal – заданная среднеквадратичная ошибка.

net = newrbe(P,T,spread) – функция создания сети с радиальными базисными элементами с нулевой ошибкой на обучающей выборке.

net = newsom(PR,[D1,D2,...],TFCN,DFCN,OLR,OSTEPS,TLR,TND) – функция создания самообучающейся карты со следующими аргументами:

– PR - R ^{x} 2 – матрица минимальных и максимальных значений R входных элементов;

– I – размеры i-го слоя, по умолчанию [5 8];

– TFCN – топологическая функция, по умолчанию 'hextop';

– DFCN – функция расстояния, по умолчанию 'linkdist';

– OLR – коэффициент обучения фазы упорядочивания, по умолчанию 0,9;

– OSTEPS – число шагов фазы упорядочивания, по умолчанию 1000;

– TLR – коэффициент обучения фазы настройки, по умолчанию 0,02;

– TND – расстояние для фазы настройки, по умолчанию 1.

3.2.7. Функции, преобразующие входы нейросети

Функции данной группы преобразуют значения входов с использованием операций умножения или суммирования.

netprod(Z1,Z2,…) – возвращает матрицу, элементы которой определяются как произведения элементов входных векторов и смещений. Аргументы Zl, Z2,... – матрицы, чьи столбцы ассоциированы с входами или смещениями.

Пример 3.9:

```
» zl = [1 2 4;3 4 1];
» z2 = [-1 2 2; -5 -6 1];
» n = netprod(zl;z2)
n =
-1 4  8
- 15    -24        1

» b = [0; -1];
» % Функция concur(b,3) создает 3 копии вектора смещения
» n = netprod(zl,z2,concur(b,3))
n =
0       0       0
15      24      -1
```

netsum(Z1,Z2,...) – то же, что в предыдущем случае, но вместо умножения используется суммирование.

dnetprod(Z,N) – возвращает матрицу значений первой производной входов, преобразованных функцией N = netprod(Zl,Z2,..).

Пример 3.10:

» Z1 = [0; 1; -1];
» Z*2* = [1; 0.5; 1.2];
» N = netprod(Zl,Z2)
N =
0
0.5000
-1.2000
» dN_dZ2 = dnetprod(Z2,N)
dN_dZ2 =
0
1
-1

dnetsum(Z,N) – то же, что и в предыдущем случае, но по отношению к функции netsum(Zl,Z2,...).

3.2.8. Функции расстояний и весов

boxdist(pos) – функция определения box-расстояния между нейронами в слое. Имеет один аргумент pos – матрицу размером N x S**,** элементы которой определяют координаты нейронов, возвращает матрицу расстояний размером S x S.Расстояния (элементы возвращаемой матрицы) вычисляются по выражению D_{ij} = max(abs(P_i - P_j)), где P_i и P_j – векторы, содержащие координаты нейронов *i* и *j*.

dist – функция вычисления евклидова расстояния.

Синтаксис:

Z = dist(W,P) – возвращает матрицу, элементы которой являются евклидовыми расстояниями между строками (векторами) матрицы W и столбцами матрицы P (матрицы должны иметь соответствующие размеры).

D = dist(pos) – в такой форме функция аналогична функции boxdist(pos) за тем исключением, что возвращается матрица евклидовых расстояний.

negdist(W,P) – функция идентична предыдущей, но элементы возвращаемой матрицы являются евклидовыми расстояниями, взятыми со знаком «минус».

mandist(W,P) – функция аналогична предыдущей, но элементы возвращаемой матрицы являются расстояниями по Манхэттену, которое для векторов x и y определяется соотношением D = sum(abs(*x*-*y*)).

linkdist(pos) – функция определения линейного расстояния между нейронами в слое. Аналогична функции boxdist, отличаясь от последней алгоритмом определения расстояния:

– $D_{ij} = 0$, если $i = j$

– $D_{ij} = 1$, если евклидово расстояние между P_i и P_j меньше или равно 1;

– $D_{ij} = 2$, если существует k такое, что $D_{ik} = D_{kj} = 1$;

– $D_{ij} = 3$, если существуют k_1 и k_2 такие, что $D_{ik1} = D_{k1k2} = D_{k2j} = 1$;

– $D_{ij} = N$, если существуют $k_1, k_2, ..., k_N$ такие, что $D_{ik1} = D_{k1k2} = \ldots = D_{kNj} = 1$;

– $D_{ij} = S$, если не выполнено ни одно из предыдущих условий.

dotprod (W,P) – функция придания входам P некоторых весов W. Результат – матрица Z = W*P.

normprod (W,P) – функция аналогична предыдущий, но каждый элемент возвращаемой матрицы дополнительно делится на сумму элементов соответствующего столбца-сомножителя матрицы P.

3.2.9. Функции размещения нейронов (topology functions)

Функции данной группы используются для создания самоорганизующихся карт.

gridtop(diml,dim2,…,dimN) – функция размещения N слоев нейронов в узлах регулярной прямоугольной N-мерной решетки, diml, dim2, ..., dimN - число нейронов в слоях. Возвращает матрицу, содержащую *N* строк и (diml ^{x} dim2 ^{x} ... ^{x} dimN) столбцов с координатами нейронов.

Пример 3.11:

```
» pos = gridtop(2,3)
pos =
0    1    0    1    0    1
0    0    1    1    2    2
```

hextop(dim1,dim2,…,dimN) – функция аналогична предыдущей, но размещение нейронов производится в узлах гексагональной (шестиугольной) решетки (см. рис. 3.2).

Пример 3.12:

```
» pos = hextop(8,5);
» plotsom(pos)
```

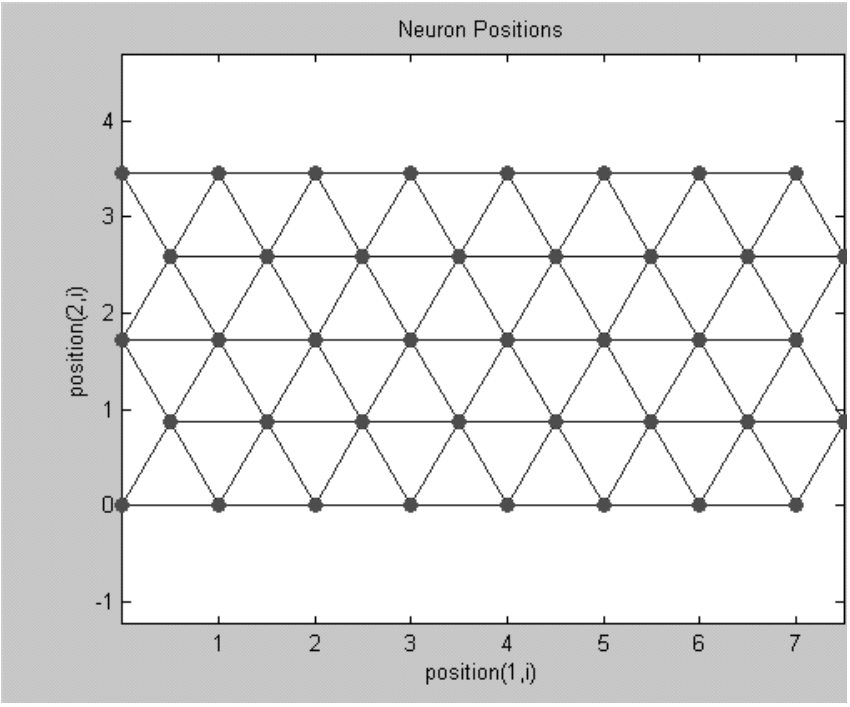

Рис. 3.2. Иллюстрация работы команды hextop(8,5)

randtop(dim1,dim2,…,dimN) – аналогична функции gridtop(diml,dim2**,…,** dimN), но координаты нейронов выбираются случайным образом.

Пример 3.13 (рис. 3.3):

» pos = randtop(16,12);

» plotsom(pos)

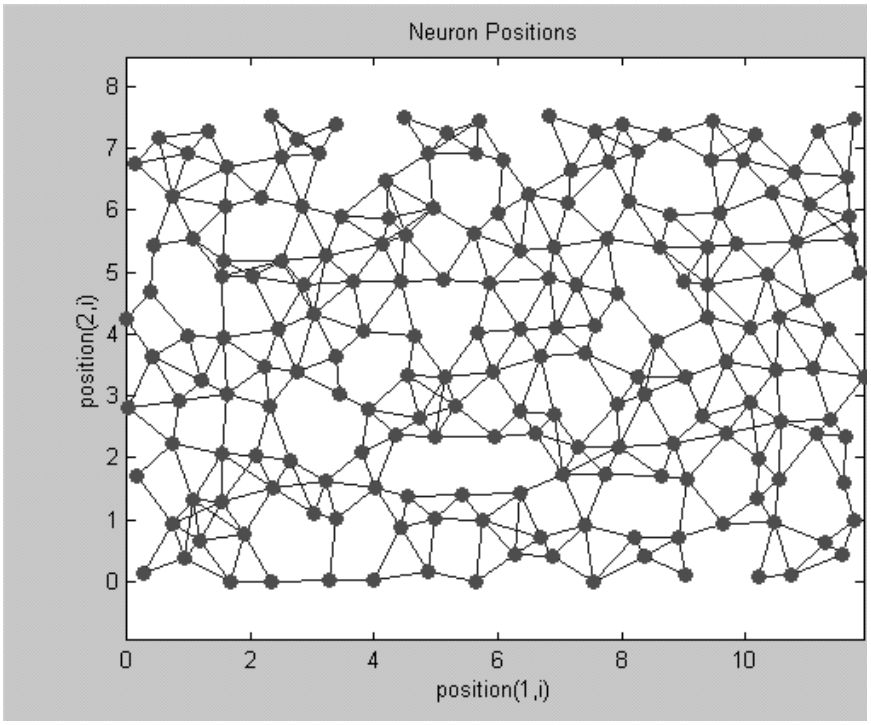

Рис. 3.3. Иллюстрация работы команды randtop(16,12)

Как видим, нейроны здесь расположены хаотично.

3.2.10. Функции использования нейронных сетей (using functions)

[Y,Pf,Af] = sim(net,P,Pi,Ai) – функция, симулирующая работу нейронной сети. Аргументы: net - имя сети, P – ее входы, Pi – массив начальных условий входных задержек (по умолчанию они нулевые), Ai – массив начальных условий задержек слоя нейронов (по умолчанию они нулевые).

Результат работы функции - значения выходов Y и массивы конечных условий задержек. Аргументы Pi, Ai, Pf, Af используются только в случаях, когда сеть имеет задержки по входам или по слоям нейронов.

Значение аргументов:

– P – массив размером Ni x TS, каждый элемент которого P{i,ts} является матрицей размером Ri x Q;

– Pi – массив размером Ni x ID, каждый элемент которого Pi{i,k} (i-й вход в момент ts = k - ID) является матрицей размером Ri xQ (по умолчанию ноль);

– Ai – массив размером Nl x LD, каждый элемент которого Ai{i,k} (выход i-го слоя в момент ts = k - LD) является матрицей размером Si x Q (по умолчанию ноль);

– Y – массив размером No x TS, каждый элемент которого Y{i,ts} является матрицей размером Ui x Q;

– Pf – массив размером Ni x ID, каждый элемент которого Pf{i,к} (i-й вход в момент ts = TS + k - ID) является матрицей размером Ri x Q;

– Af – массив размером N1 x LD, каждый элемент которого Af{i,k} (выход i-го слоя в момент ts = TS + k - LD) является матрицей размером Si x Q.

Здесь:

Ni = net.numlnputs – количество входов сети;

Nl = net.numLayers – количество ее слоев;

No = net.numOutputs – количество выходов сети;

ID = net.numlnputDelays – входные задержки;

LD = net.numLayerDelays – задержки слоя;

TS – число временных интервалов;

Q – размер набора подаваемых векторов;

Ri = net.inputs{ i }.size – размер i-го вектора входа;

Si = net.layers{ i }.size – размер i-го слоя;

Ui = net.outputs{ i }.size – размер i-го вектора выхода.

net = init(net) – функция инициализирует нейронную сеть с именем net и устанавливает веса и смещения сети в соответствии с установками net.initFcn и net.initParam.

[net,Y,E,Pf,Af] = adapt(net,P,T,Pi,Ai) – функция адаптации нейросети. Выполняет адаптацию сети в соответствии с установками net.adaptFcn и net.adaptParam. Здесь E – ошибки сети, T – целевые значения выходов (по умолчанию ноль); остальные аргументы – как у команды sim.

[net,tr] = train(net,P,T,Pi,Ai) - функция выполняет обучение нейросети в соответствии с установками net.trainFcn и net.trainParam. Здесь tr - информация о ходе процесса обучения (количество циклов и ошибка обучения).

disp(net) - функция возвращает развернутую информацию о структуре и свойствах НС.

Пример 3.14:

```
» net = newp([-1 1; 0 2],3);   % Создание НС типа персептрон
» disp(net)
Neural Network object:
architecture:
numInputs: 1
numLayers: 1
biasConnect: [1]
inputConnect: [1]
layerConnect: [0]
outputConnect: [1]
targetConnect: [1]

numOutputs: 1 (read-only)
numTargets: 1 (read-only)
numInputDelays: 0 (read-only)
numLayerDelays: 0 (read-only)

subobject structures:
inputs: {1x1 cell} of inputs
layers: {1x1 cell} of layers
outputs: {1x1 cell} containing 1output
targets: {1x1 cell} containing 1target
biases: {1x1 cell} containing 1 bias
inputWeights: {1x1 cell} containing 1 input weight
layerWeights: {1x1 cell} containing no layer weights
functions:
```

adaptFcn: 'adaptwb'
initFcn: 'initlay'
performFcn:'mae'
trainFcn: 'trainwb'
parameters:
adaptParam:.passes
initParam: (none)
performParam: (none)
trainParam: .epochs, ,goal, ,max_fail, ,show, ,time
weight and bias values:
IW: {lxl cell} containing 1 input weight matrix
LW: {lxl cell} containing no layer weight matrices
b: {lxl cell} containing 1 bias vector
other:
userdata: (user stuff)

display(net) – то же, что предыдущая команда, но здесь дополнительно возвращается имя нейронной сети.

3.2.11. Графические функции

hintonw(W,maxw,minw) – функция возвращает так называемый хинтоновский график матрицы весов, при котором каждый весовой коэффициент отображается квадратом площадью, пропорциональной величине данного коэффициента. Знак отображается цветом квадрата (см. рис. 3.4)

Аргументы: W – матрица весов, maxw и minw – минимальное и максимальное значения ее коэффициентов (могут не задаваться).

Пример 3.15:

» W = rands(2,3)
W =
-0.8842 0.6263 -0.7222
-0.2943 -0.9803 -0.5945
» hintonw(W)

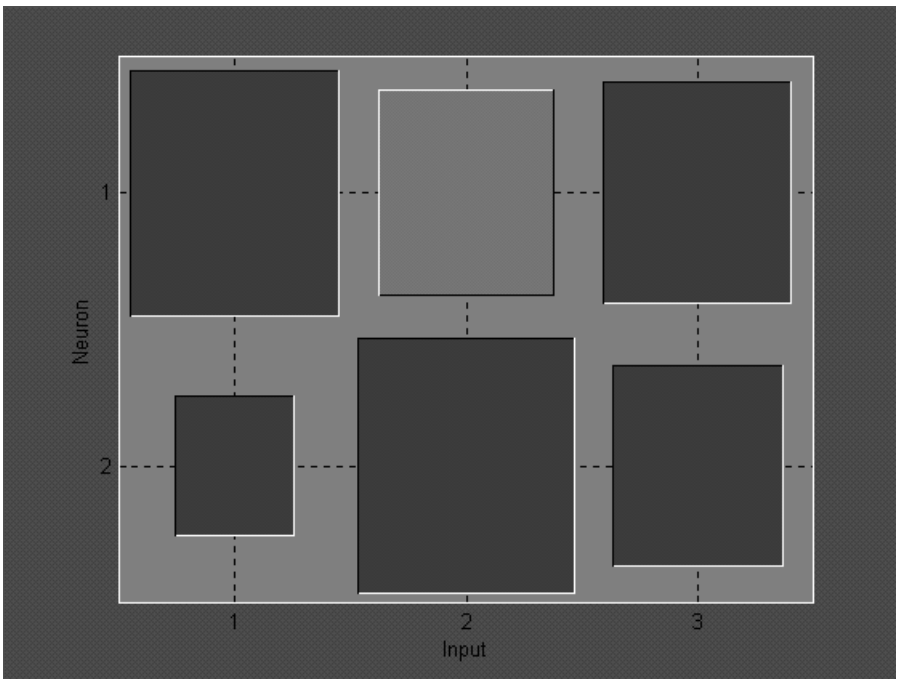

Рис. 3.4. Иллюстрация работы функции hintonw

hintonwbM(W,b,maxw,minw) – то же, что и предыдущая функция, но на графике отображаются не только веса, но и смещения.

plotbr(tr,name,epoch) – функция возвращает графики изменения критерия качества нейросети в процессе обучения при использовании байесовского метода. Аргументы: tr – запись процесса обучения, name – имя сети, epoch – количество циклов обучения (по умолчанию – длина записи обучения). Результат работы функции приведен на рис. 3.5.

Пример 3.16:

```
» p = [-1; .05; 1];
»  t = sin(2*pi*p)+0.1*randn(size(p));
»            % Создание новой сети
» net = newff([-l  1],[20,1],{'tansig','purelin'},'trainbr');
» [net,tr] = train(net,p,t);            % Обучение сети
TRAINBR, Epoch 0/100,  SSE 228.933/0,  SSW 21461.7, Grad 2.33e+002/1.00e-010, #Par 6.10e+001/61
TRAINBR, Epoch 25/100,  SSE 0.235423/0,  SSW 211.044, Grad 9.43e-002/1.00e-010,  #Par 1.35e+001/61
TRAINBR, Epoch 50/100,  SSE 0.240881/0,  SSW 121.647, Grad 1.87e-001/1.00e-010,  #Par 1.23e+001/61
TRAINBR, Epoch 75/100,  SSE 0.239867/0,  SSW 116.884,  Grad 1.62e-002/1.00e-010,  #Par 1.22e+001/61
TRAINBR, Epoch 100/100,  SSE 0.239762/0,  SSW 116.871, Grad 9.60e-003/1.00e-010,  #Par 1.22e+001/61
```

TRAINBR, Maximum epoch reached.

» plotbr(tr)

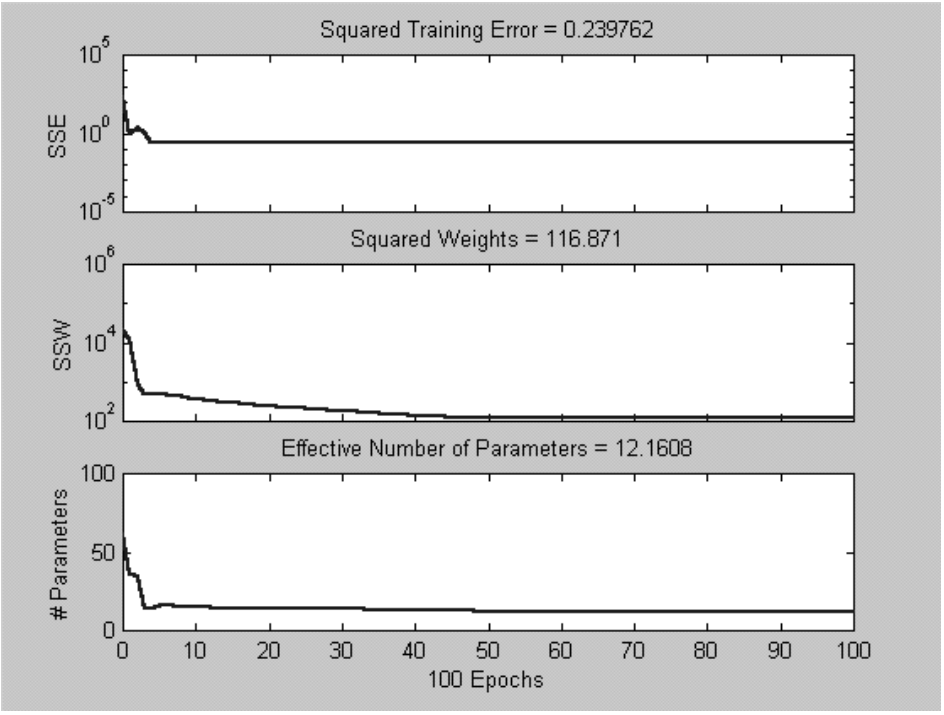

Рис. 3.5. Иллюстрация работы функции plotbr

plotep(w,b,e) – функция отображает позиции весов и смещений на поверхности ошибки НС. Аргументы: w, b, e – соответственно, матрицы весов, смещений и ошибок. Результат – вектор, используемый для продолжения графика, созданного функцией plotes.

plotes(wv,bv,e,v) – функция возвращает график поверхности ошибки одновходового нейрона. Аргументы: wv, bv – соответственно, наборы значений веса и смещения нейрона, e – матрица значений ошибки, v – опция вида изображения (по умолчанию [-37.5, 30]).

plotpc(W,b) – функция возвращает график линии решения для персептрона. Аргументы: W – матрица весов, b – вектор смещений. Применяется вместе с функцией plotpv (см. пример ниже).

plotperf(tr,goal,name,epoch) – возвращает график изменения критерия качества НС в процессе обучения. Аргументы: tr – запись процесса обучения, goal - целевое значение критерия, name - имя сети, epoch - количество циклов обучения.

plotpv(p,t) – функция возвращает графическое отображение входных p и целевых t векторов персептрона.

Пример 3.17:

» p = [0 0 1 1; 0 1 0 1];

» t = [0 0 0 1];
» plotpv(p,t)

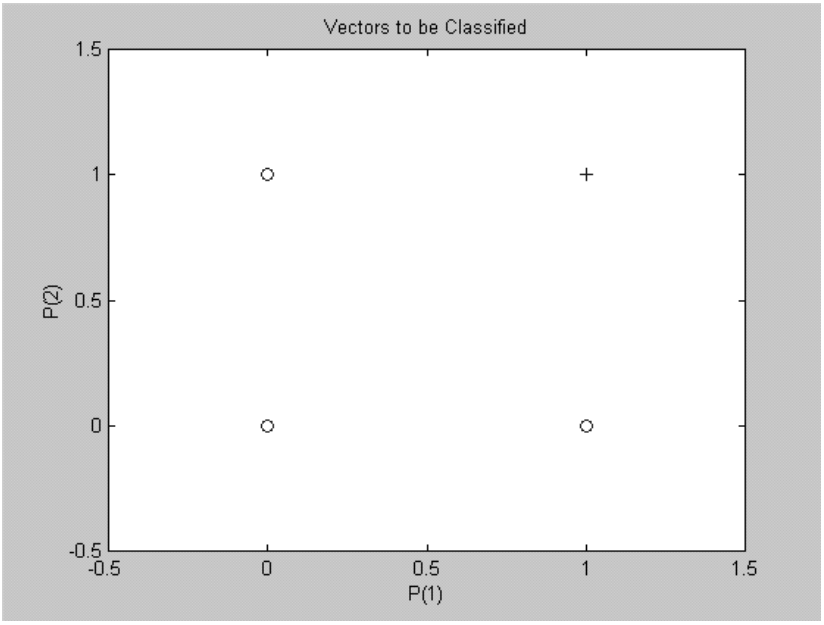

Рис. 3.6. Иллюстрация работы функции plotpv

plotsom(pos) – функция возвращает графическое представление расположения нейронов в самоорганизующихся картах.

plotv(M,t) – функция для графического изображения векторов. Аргументами служат: M – двухстрочная матрица, столбцы которой ассоциированы с векторами, t – параметр, определяющий тип линии. Результат выполнения данной функции приведен на рис. 3.7.

Пример 3.18:

» plotv([-.4 0.7 .2; -0.5 .1 0.5], '-')

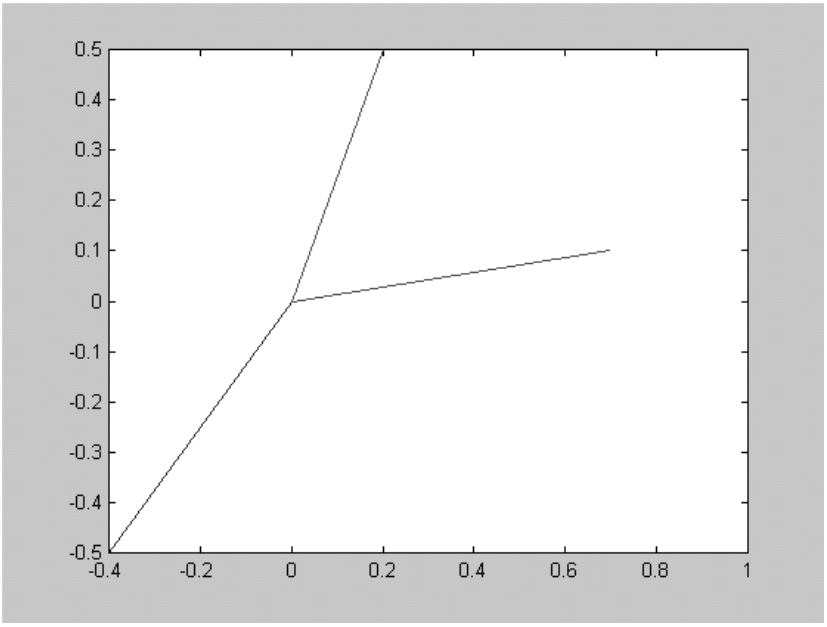

Рис. 3.7. Иллюстрация работы функции plotv

plotvec(M,C,m) – функция для цветного графического изображения векторов. Аргументы функции: M – двухстрочная матрица, столбцы которой ассоциированы с векторами, C – строка задания цветов, m – тип точек, указывающих окончания векторов (по умолчанию изображаются в виде '+').

3.2.12. Функции, не вошедшие в основные группы

errsurf(p,t,wv,bv,f) – функция, возвращающая матрицу значений поверхности ошибок нейрона с одним входом и одним выходом в зависимости от значений веса и смещения.

Аргументы:

– p – вектор значений входа;

– t – вектор значений выхода;

– wv – набор значений веса нейрона;

– bv – набор значений смещения;

– f – название реализуемой функции активации (строка).

Размер возвращаемой матрицы – (количество значений bv) x (количество значений wv).

Пример 3.19:

```
» p = [-6.0  -6.1  -4.1  -4.0  +4.0  +4.1  +6.0  +6.1];
» t =  [+0.0  +0.0  +.97  +.99  +.01  +.03  +1.0  +1.0];
» wv = -1: .1: 1;
» bv = -2.5: .25: 2.5;
» es = errsurf(p,t,wv,bv,'logsig');
```

» plotes(wv,bv,es,[60 30])

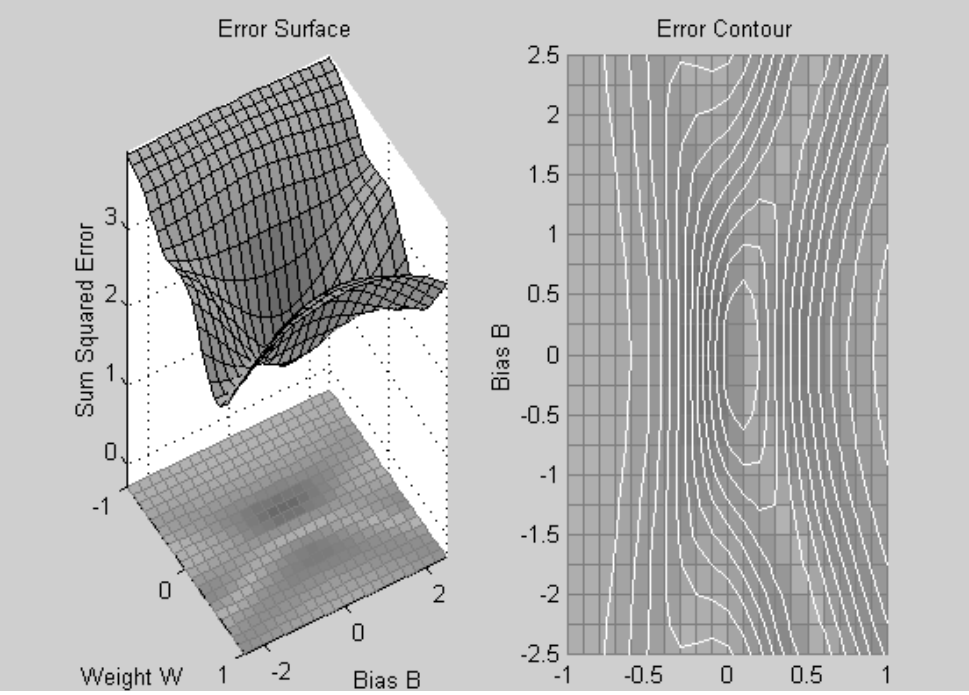

Рис. 3.8. Результаты работы функции errsurf

maxlinlr(P) – функция возвращает максимальную величину коэффициента обучения для линейного слоя нейронов, где P – матрица входов. В форме maxlinlr(P, 'bias') результатом работы функкции будет максимальная величина коэффициента обучения для линейного слоя нейронов со смещением.

gensim(net,st) –генерация нейросетевого блока Simulink для последующего моделирования средствами этого пакета. Здесь net – имя созданной НС, st – интервал дискретизации (если НС не имеет задержек, ассоциированных с ее входами или слоями, значение данного аргумента устанавливается равным -1).

Пример 3.20:

```
» net
» newff([0 1],[5 1]);       % Создание новой нейросети
» gensim(net)
```

initlay(net) – запрос инициализирует слои нейронной сети. Аргумент: используется имя (идентификатор) сети net. Возвращает нейронную сеть, слои нейронов в которой инициализированы в соответствии с функцией net.layers{i}.initFcn. В виде initlay(code), где строковая переменная code может принимать значения 'pnames' или 'pdefaults', функция возвращает

информацию о именах или о значениях по умолчанию параметров инициализации.

initnw(net,i) – функция инициализации i-го слоя. Возвращает сеть**,** веса и смещения в i-м слое которой обновлены в соответствии с алгоритмом инициализации Нгуен-Уидроу (таким образом, что зоны «влияния» каждого нейрона в слое распределены равномерно).

initwb(net,i) – практически то же, что в предыдущем случае, но веса и смещения i-го слоя инициализируются в соответствии с их собственными функциями инициализации.

ddotprod – операция возвращает производную от результата Z умножения матрицы весов W на матрицу входов P.

Синтаксис:

dZ_dP - ddotprod (' p ', W, P, Z)

dZ_dW - ddotprod('w', W, P, Z)

Пример 3.21:

» W = [0 -1 0.2; -1.1 1 0];

» P = [0.1; 0.6; -0.2];

» % Вычисление произведения Z=W*P

» Z = dotprod(W,P)

Z =

0.6400

0.4900

» dZ_dP = ddotprod('p',W, P, Z)

dZ_dP =

0 -1.0000 0.2000

-1.1000 1.0000 0

» dZ_dW = ddotprod('w', W, P, Z)

dZ_dW =

0.1000

0.6000

-0.2000

Контрольные вопросы

Какие группы функций пакета Neural Networks Toolbox вы знаете?

Назовите функции, с помощью которых можно создать сеть.

Как получить справочную информацию по интересующей вас функции?

Опишите «соревновательную» функцию NNT. Какие параметры можно задавать в соответствующем запросе?

Назовите различия использования функций hardlim(X) и hardlims(X). Что может быть задано в качестве параметра X? Привести примеры.

Охарактеризуйте аргументы net, Pd, Tl, Ai, Q, TS, VV, TV функции trainbfg().

Как можно задать максимальное количество циклов обучения сети и начальные входные условия?

Какую информацию возвращает функция learncon(code)?

Назовите функции, выполняющие одномерную оптимизацию НС.

ГЛАВА 4. РЕАЛИЗАЦИЯ НЕЙРОННЫХ СЕТЕЙ В NEURAL NETWORK TOOLBOX

4.1. Приближение функций

Аппроксимируем функцию $y = x^2$ с изменением аргумента на отрезке [-1, 1]. Для этого создадим обобщенно-регрессионную нейронную сеть под именем «a». Аппроксимация проводится на следующих данных:

x = [-1 -0.8 -0.5 -0.2 0 0.1 0.3 0.6 0.9 1],

y = [1 0.64 0.25 0.04 0 0.01 0.09 0.36 0.81 1]

Реализация (создание и использование данной сети). В командном окне пакета Matlab последовательно наберем:

```
» % Задать вектор входных значений
» P = [-1 -0.8 -0.5 -0.2 0 0.1 0.3 0.6 0.9 1];
» % Задать вектор выходных значений
» T = [1 0.64 0.25 0.04 0 0.01 0.09 0.36 0.81 1];
»  a = newgrnn(P,T,0.01);   % Создание НС с отклонением 0.01
» Y = sim(a,[-0.9 -0.7 -0.3 0.4 0.8])   % Опрос НС
Y =
0.8200     0.6400     0.0400     0.0900     0.8100
```

Как видим, точность аппроксимации в данном случае получилась недостаточно высокой.

Можно попробовать улучшить качество аппроксимации вариацией величины отклонения, но здесь неплохой результат легко достигается при использовании сети с радиальными базисными элементами:

```
» a = newrbe(P,T);
» Y = sim(a,[-0.9 -0.7 -0.3 0.4 0.8])    % Опрос НС
Y =
0.8100     0.4900     0.0900     0.1600     0.6400
```

Чтобы сохранить построенную сеть, в командной строке надо набрать оператор *save('a');* при этом будет создан файл *a.mat*, содержащий имя нейросети, с расширением *mat*. В последующих сеансах работы сохраненную сеть можно загрузить, используя функцию *load('a')*.

Рассмотрим использование *линейной* нейросети для решения аналогичной задачи. Пусть экспериментальные данные таковы:

x = [+1.0 +1.5 +3.0 -1.2], y = [+0.5 +1.1 +3.0 -1.0]

Процесс создания, обучения и использования линейной НС с именем b иллюстрируется приведенным ниже листингом и рисунком 4.1.

```
» P = [+1.0 +1.5 +3.0 -1.2];
» T = [+0.5 +1.1 +3.0 -1.0];
»   % Определение величины коэффициента обучения
» maxlr = maxlinlr(P,'bias');
» b = newlin([-2 2], 1, [0], maxlr);    % Создание линейной НС с именем b
» b.trainParam.epochs = 15; % Задание количества циклов обучения
» b = train(b,P,T);      % Обучение НС
TRAINWB, Epoch 0/15,  MSE 2.865/0.
TRAINWB, Epoch 15/15,  MSE 0.0730734/0.
TRAINWB, Maximum epoch reached.
» p = -1.2;
» y = sim(b, p)  % Опрос сети
y  =
-1.1803
```

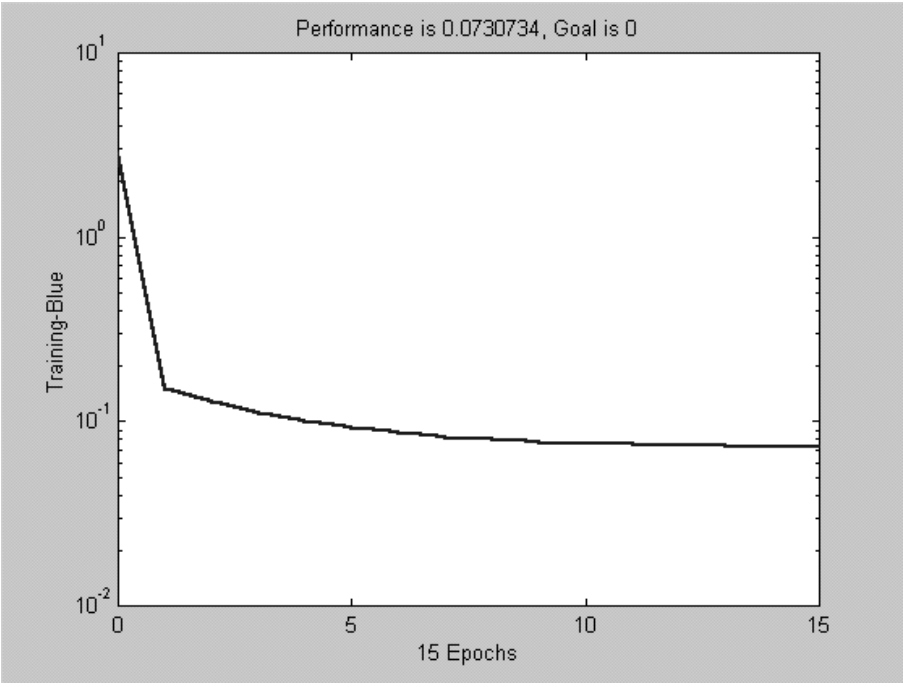

Рис. 4.1. Процесс обучения нейросети

Этот пример служит иллюстрацией того, как уменьшается ошибка в процессе обучения нейросети.

4.2. Задача прогнозирования

Пусть имеется функция времени, описываемая соотношением x(t) = sin(4πt), причем сигнал – дискретный с интервалом 0,025 с.

Построим линейную нейронную сеть, позволяющую прогнозировать будущее значение подобного сигнала по 5 предыдущим. Вариант решения данной задачи приведен ниже.

```
» t = 0:0.025:5;  % Задание диапазона времени от 0 до 5 секунд
»x = sin(t*4*pi); % Предсказываемый сигнал
» Q = length(x);
» % Создание входных векторов
» P = zeros(5,Q); % Создание нулевой матрицы P
» P(1,2:Q) = x(1,1:(Q-1));
» P(2,3:Q) = x(1,1:(Q-2)):
» P(3, 4:Q) =  x( l, l:(Q-3));
» P(4,5:Q) = x(l,l:(Q-4));
» P(5,6:Q) = x(l,1:(Q-5));
» s = newlind(P,x);   % Создание новой НС с именем s
» y = sim(s,P);          % Расчет прогнозируемых значений
```

```
»  % Вывод графиков исходного сигнала и прогноза
» plot(t,y,t,x,'*');
» xlabel('Время');
» ylabel( 'Прогноз - Сигнал +');
» title('Выход сети и действительные значения');
» % Расчет и создание графика ошибки прогноза
» e = x - y;
» plot(t,e);
» hold on;
» plot([min(t) max(t)], [0 0], ':r');
» hold off;
» xlabel('Время');
» ylabel('Ошибка');
» title('Сигнал ошибки');
```

В данном случае сеть была построена с помощью функции newlind, при использовании которой не требуется дополнительного обучения.

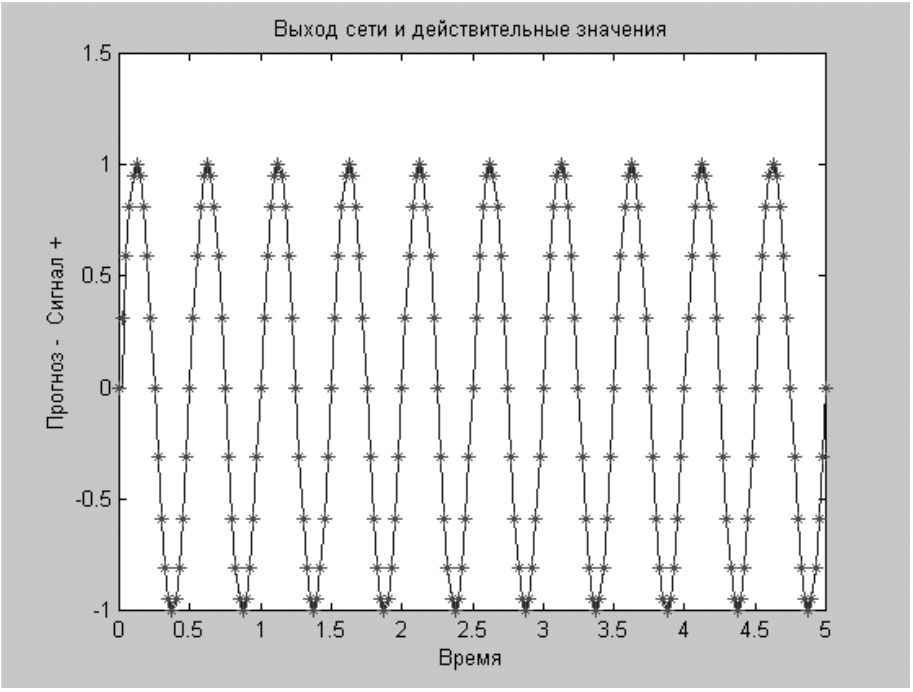

Рис. 4.2 Исходный сигнал и прогноз

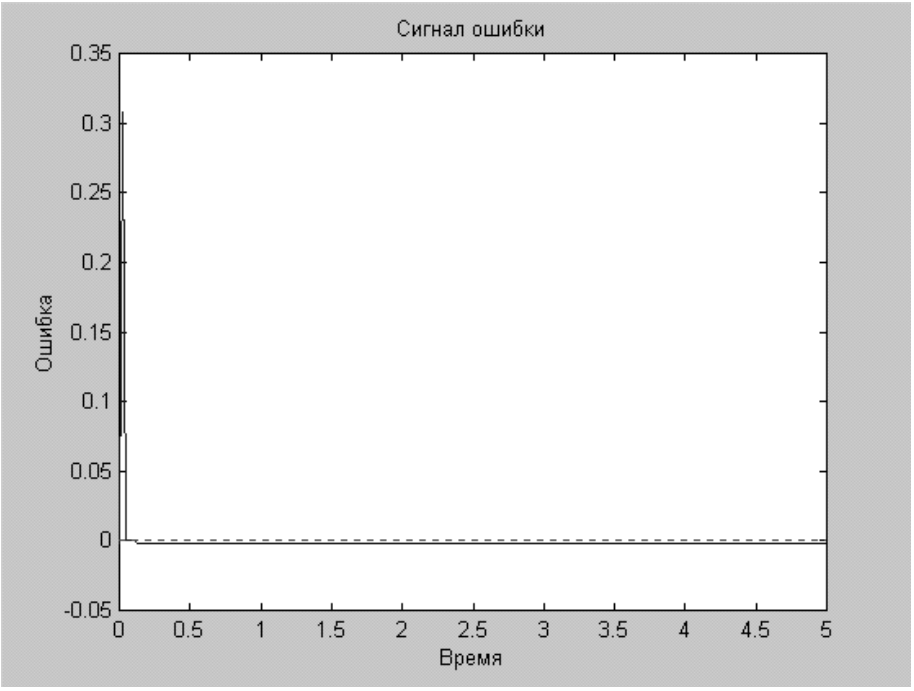

Рис. 4.3 Ошибка прогноза

Судя по графикам результатов, приведенным на рис. 4.2 и 4.3, точность прогноза с использованием линейной НС можно считать достаточно хорошей.

4.3. Нейронная сеть со слоем Кохонена

Приведем возможное решение задачи автоматического определения (в режиме обучения без учителя) центров кластеров входов для двумерного случая с использованием слоя Кохонена.

```
» % Задание диапазонов возможного положения центров кластеров
» X = [0 1: 0 1];
» % Задание параметров для моделирования исходных данных,
» % принадлежащих 8 классам (кластерам)
» clusters = 8;
» points =10;
» std_dev =0.05;
» P = nngenc(X, clusters, points,s td_dev);  % Моделирование входных данных
» h=newc([0 1; 0 1], 8, .1);      % Создание слоя Кохонена
» h.trainParam.epochs = 500;  % Задание количества циклов обучения
» h=init(h);       % Инициализация сети
» h=train(h,P);  % Обучение сети
```

```
» w=h.IW{1};
» % Вывод графика исходных данных и выявленных центров кластеров
» plot(P(l, :), P(2, :), '+r');
» hold on;
» plot(w(:, 1),w(:, 2), 'ob');
» xlabel('p(1)');
» ylabel('p(2)');
» p = [0; 0.2];          % Задание нового входного вектора
» y = sim(h, p);        % Опрос сети
y =
(3, 1)
```

Работу обученной сети иллюстрируют рис. 4.4 и результат ее опроса (матрица y в конце приведенного листинга, которая выдается в форме разреженной матрицы).

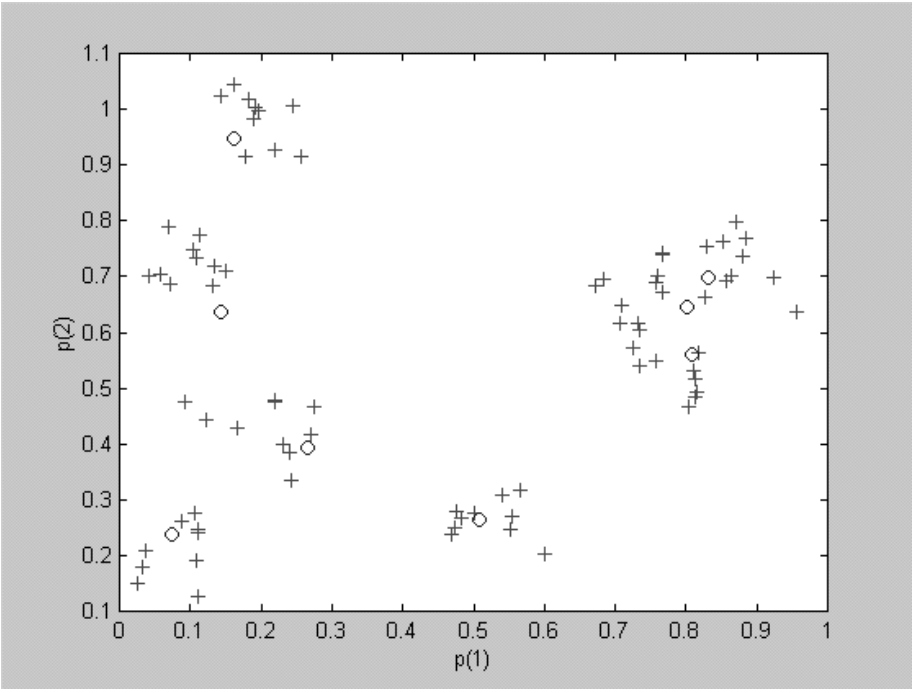

Рис. 4.4. Исходные данные и распознанные центры классов (кластеров)

Для заданных условий предъявленный вектор отнесен к третьему классу.

4.4. Сеть Хопфилда с двумя нейронами

Рассмотрим сеть Хопфилда, которая имеет два нейрона и два устойчивых состояния, отображаемых векторами [1 -1] и [-1 1] (см. рис. 4.5).

```
» T = [+1 -1; -1 +1];
» plot(T(l, :), T(2, :), 'r*');
» axis([-1.1  1.1  -1.1  1.1]);
» title('Пространство векторов сети Хопфилда');
» xlabel('a(1)');
» ylabel('a(2)');
```

Создадим сеть Хопфилда с именем H и проверим ее работу, подав на вход векторы, соответствующие устойчивым точкам. Если сеть работает правильно, она должна выдавать эти же векторы без каких–либо изменений.

```
» H=newhop(T); % Создание НС Хопфилда
» [Y,Pf,Af] = sim(H,2,[], T);
Y                    % Опрос сети Хопфилда
Y =
-1
-1     1
```

Как следует из результата опроса, нейронная сеть работает достоверно. Подадим теперь на ее вход произвольный вектор (см. рис. 4.6).

```
» a = {rands(2, l)}; % Задание случайного вектора
a =
[2x1 double]

» [y,Pf,Af] = sim(H, {l 50}, {}, a);
» plot(T(l, :), T(2, :), 'r*');
» axis([-1.1  1.1  -1.1  1.1]);
» record = [cell2mat(a) cell2mat(y)];
» start = cell2mat(a);
» hold on;
» plot (start (1,1), start (2,1),' bx', record(1, :), record(2, :));
» xlabel('a(l)');  ylabel('a(2)');
» title('Результат работы нейросети Хопфилда');
```

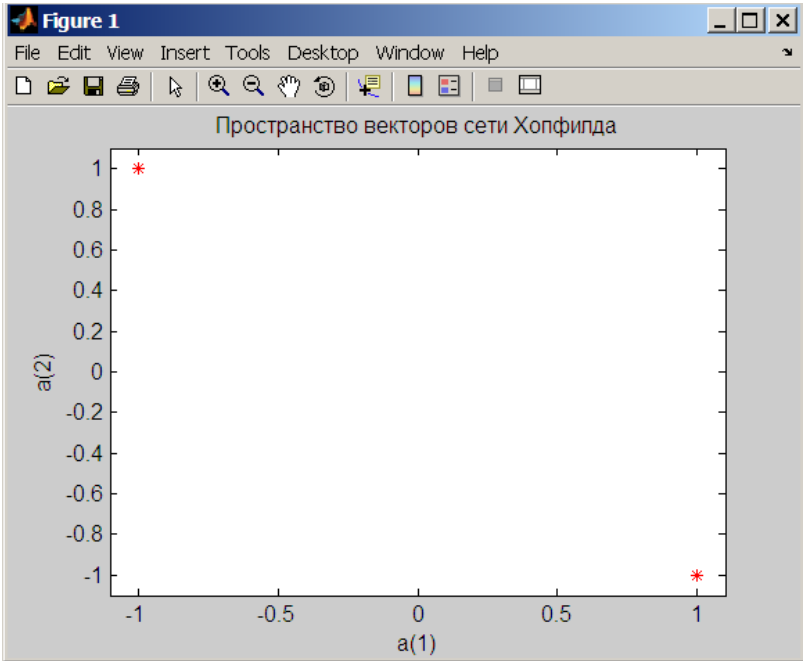

Рис. 4.5. Устойчивые точки сети Хопфилда

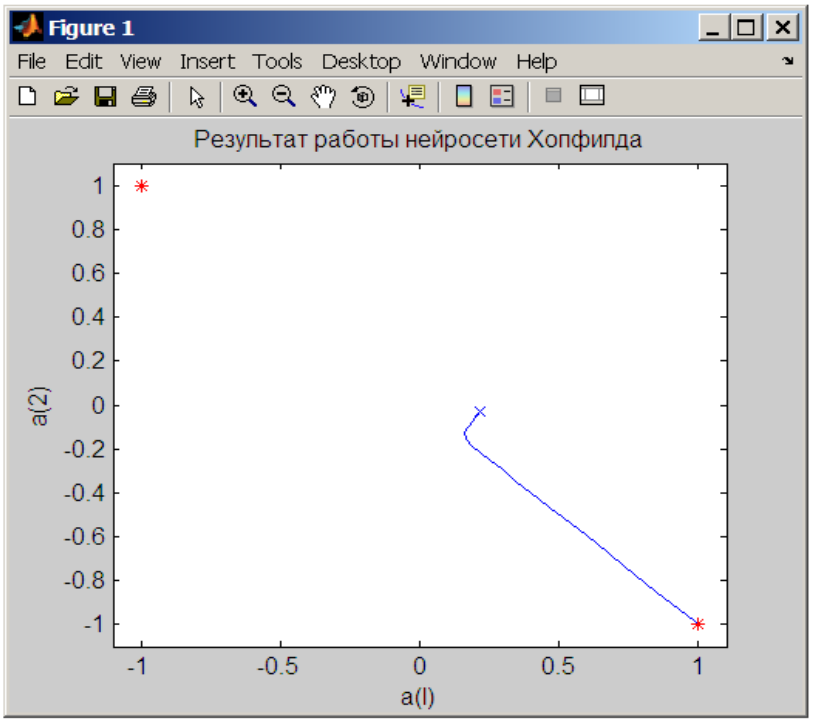

Рис. 4.6. Иллюстрация работы сети Хопфилда

На рис. 4.5 представлены векторы [1 -1] и [-1 1], отражающие устойчивые состояния НС, а на рис. 4.6 виден результат работы НС в случае подачи произвольного вектора.

4.5. Классификация с помощью персептрона

Следующий пример иллюстрирует решение задачи классификации с помощью персептрона. Исходные входные векторы (с указанием их принадлежности к одному из двух классов) и результат настройки персептрона (с именем My_net) представлены на рис. 4.7. и 4.8.

```
» % Задание входных векторов с указанием их принадлежности
» % одному из двух классов
» P = [-0.5 -0.5 +0.3 -0.1; -0.5 +0.5 -0.5 +1.0];
» T = [1 1 0 0];
» plotpv(P,T);
```

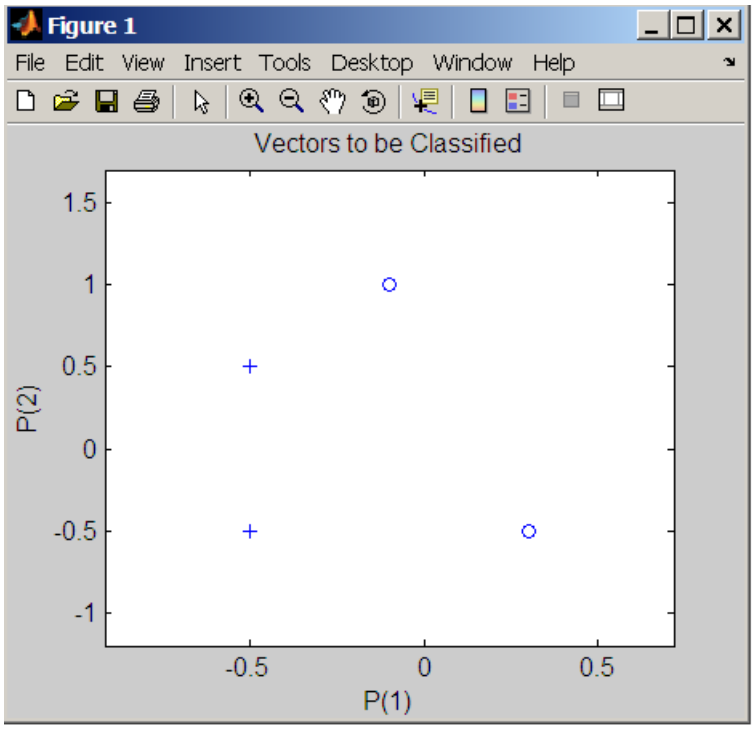

Рис. 4.7. Графическое представление исходных векторов

```
» % Создание персептрона с указанием границ изменений входов
» %  и одним нейроном
» My_net = newp([-1 1; -1 1], 1);
» E = 1;
» % Инициализация персептрона
»  My_net = init(My_net);
» % Организация цикла адаптивной настройки персептрона
» % с выводом  графика разделяющей линии
» while (sse(E))
```

```
[My_net, Y, E] = adapt(My_net, P, T);
linehandle = plotpc(My_net.IW{1},  My_net.b{l});
drawnow;
end;
```

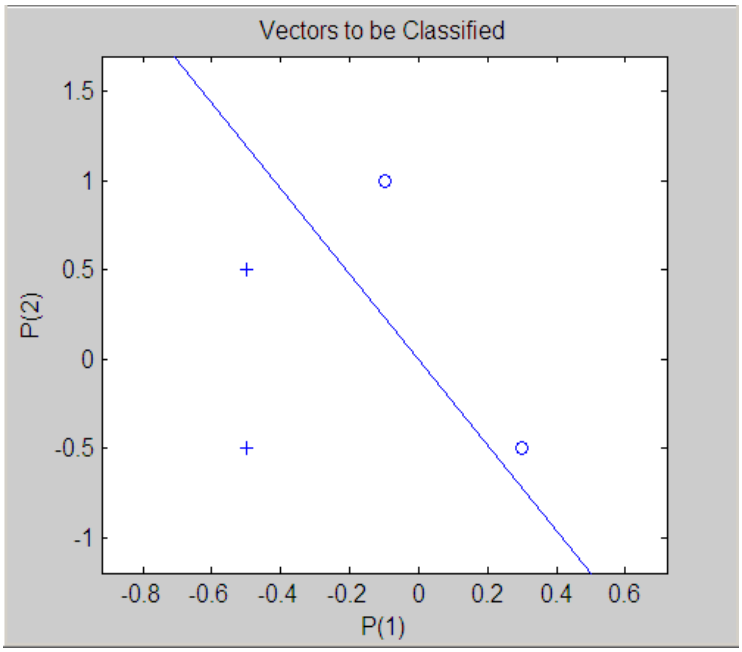

Рис. 4.8. Входные векторы и разделительная линия

4.6. Создание и использование самоорганизующейся карты

Уже отмечалось, что самоорганизующиеся карты можно рассматривать как усовершенствованную модификацию слоя конкурирующих нейронов (слоя Кохонена). От последнего данный вид НС отличается следующим:

нейроны распределяются некоторым пространственным образом (по одному из трех возможных вариантов: в узлах прямоугольной решетки, гексагональной решетки или случайно);

на этапе самообучения корректируются веса не только нейрона–«победителя», но и группы нейронов в его некоторой пространственной окрестности.

Назначение самоорганизующихся карт такое же, как у слоя Кохонена – выявление в режиме самообучения центров кластеров входных векторов.

Рассмотрим создание и использование самоорганизующейся карты на примере кластеризации двумерных векторов (исходные данные см. на рис. 4.9).

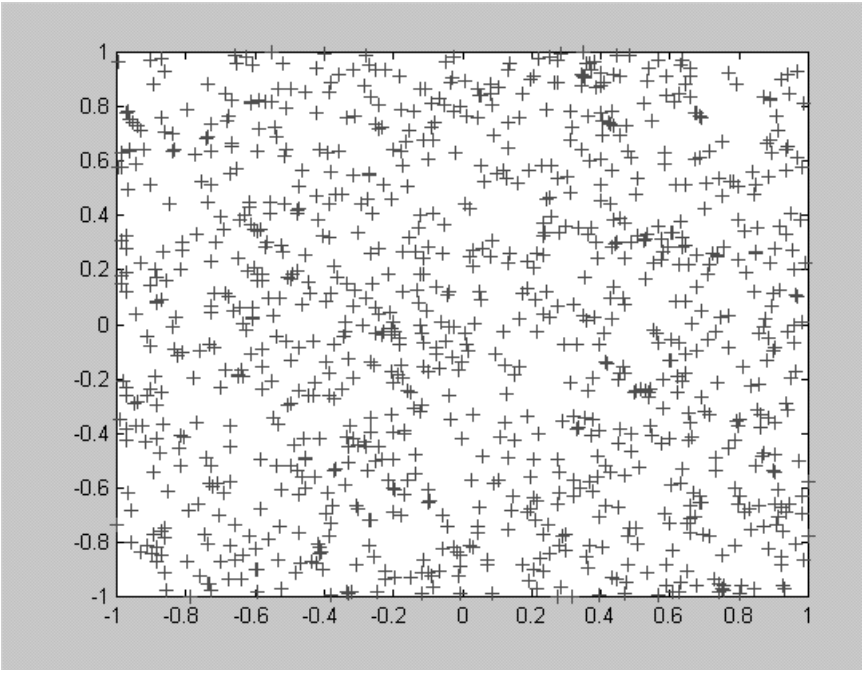

Рис. 4.9. Входные векторы

» P = rands(2,1000); % Задание случайных двумерных входных векторов
» plot(P(l, :), P(2,:), ' +r'); % Визуальное изображение входных векторов
» Создание НС с 5x6 = 30 нейронами: все установки – по умолчанию
» net = newsom([0 1: 0 1], [5 6]);
» net.trainParam.epochs = 1000; % Задание числа циклов настройки
» net.trainParam.show = 200; % Задание периодичности вывода информации
» net = train(net,P); % Настройка сети
TRAINWB1, Epoch 0/1000
TRAINWB1, Epoch 200/1000
TRAINWB1, Epoch 400/1000
TRAINWB1, Epoch 600/1000
TRAINWB1, Epoch 800/1000
TRAINWB1, Epoch 1000/1000
TRAINWB1, Maximum epoch reached.
» % Выявленные центры кластеров
» piotsom(net.iw{1,1}, net.layers{1}.distances);
» p = [0.5: 0.3];
» a = sim(net, p); % Опрос сети
a =
(11, 1)

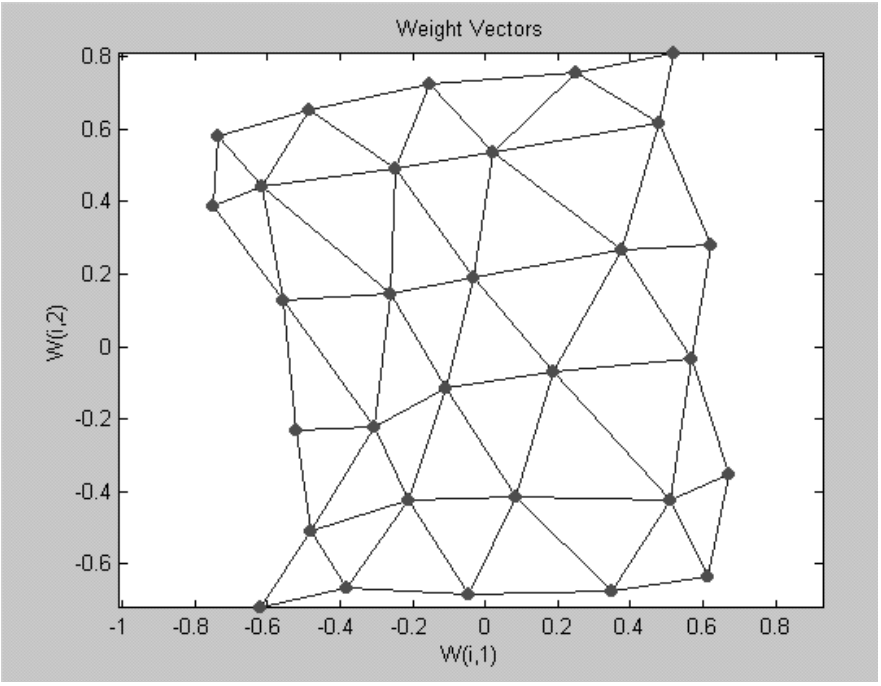

Рис. 4.10. Распознанные центры кластеров

Предъявленный на этапе тестирования вектор отнесен сетью к классу 11. Выявленные центры кластеров представлены на рис. 4.10.

4.7. Работа с приложением Simulink

Simulink – интерактивный инструмент для моделирования, имитации и анализа динамических систем. Он дает возможность строить графические блок-диаграммы, имитировать динамические системы, исследовать работоспособность систем и совершенствовать проекты. Simulink полностью интегрирован с MATLAB, предоставляя доступ к широкому спектру инструментов анализа и проектирования, в том числе нейросетей.

Пакет Neural Network Toolbox содержит библиотеку блоков, которые либо могут быть непосредственно использованы для построения нейронных сетей в среде Simulink, либо применяться вместе с функцией gensim.

Для вызова набора блоков, в командной строке необходимо набрать команду neural, после выполнения которой появляется окно (рис. 4.11).

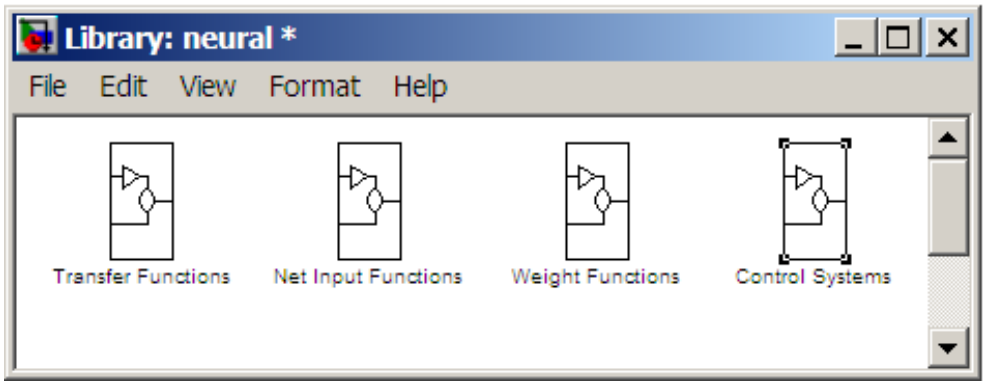

Рис. 4.11. Основные нейросетевые блоки Simulink

Каждый блок является библиотекой функциональных блоков. Выбор блока Transfer Functions приводит к появлению библиотеки функций активации (рис. 4.12).

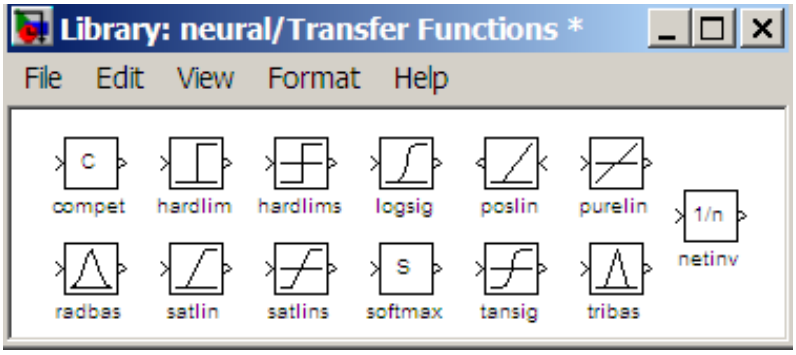

Рис. 4.12. Библиотека функций активации

Каждый из этих блоков данной библиотеки преобразует подаваемый на него вектор в соответствующий вектор той же размерности.

Блок Net Input Functions открывает следующую библиотеку блоков:

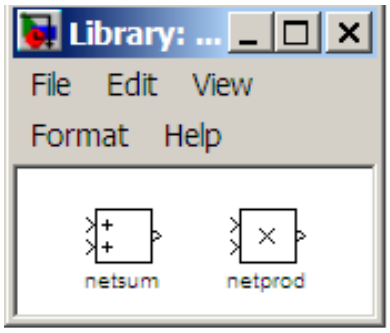

Рис. 4.13. Библиотека блоков преобразований сигналов

Блоки данной библиотеки реализуют рассмотренные выше функции преобразования входов сети. Выбор блока Weight Functions приводит к библиотеке блоков, реализующих некоторые функции весов и расстояний.

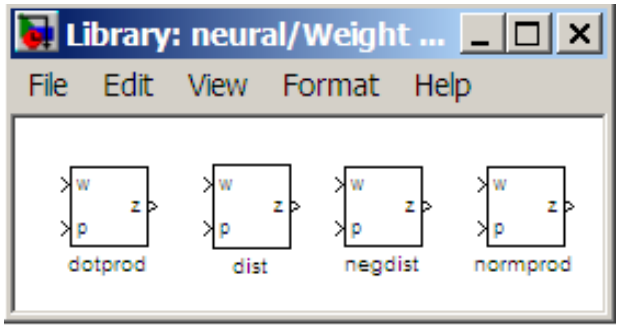

Рис. 4.14. Библиотека блоков весовых коэффициентов

Основной функцией для формирования нейросетевых моделей в среде Simulink является функция gensim, рассмотренная ранее.

Рассмотрим пример использования средств Simulink; входной и целевой векторы имеют вид:

p = [1 2 3 4 5];

t = [1 3 5 7 9].

Создадим линейную нейросеть и затем протестируем ее:

```
» p = [1 2 3 4 5];
» t = [2 4 6 8 10];
» net = newlind(p,t);
» Y = sim(net,p)
Y =
2.0000    4.0000    6.0000    8.0000    10.0000

»  %  Запуск Simulink
» gensim(net,-1)
```

Результат отражен на блок-диаграмме на рис. 4.15.

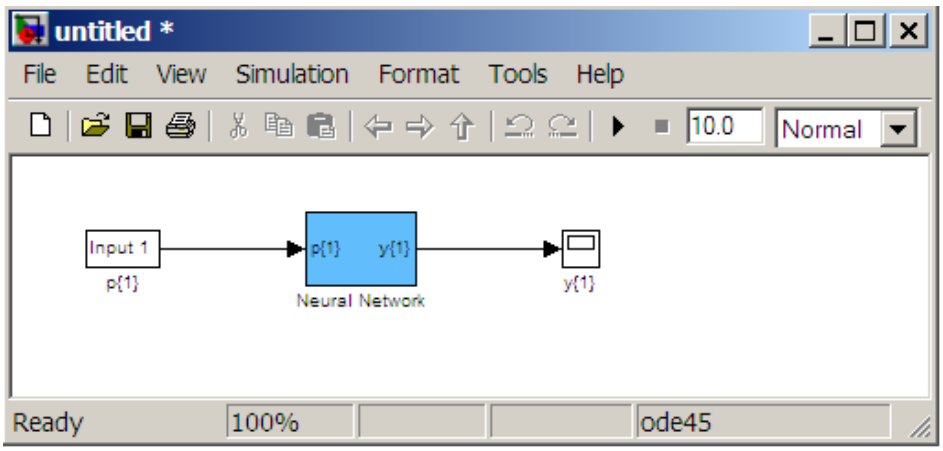

Рис. 4.15. Созданная нейросетевая модель Simulink

Для того чтобы протестировать модель, щелкнем мышью по левому блоку (Input 1), что приведет к открытию диалогового окна:

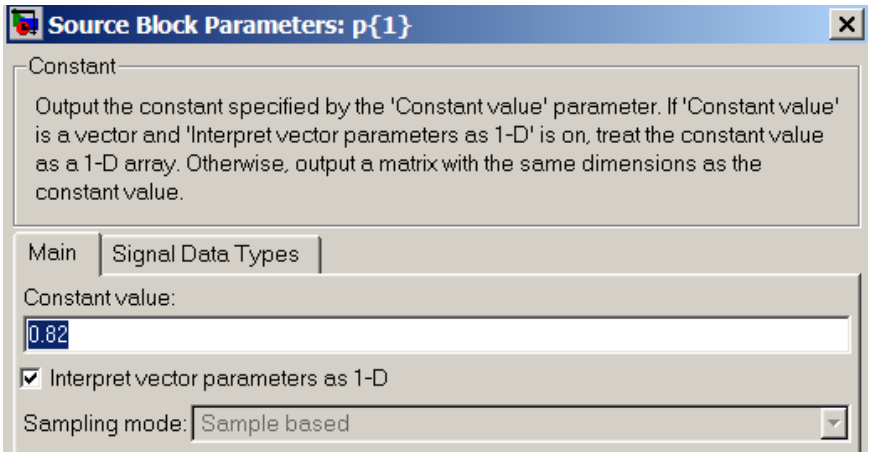

Рис. 4.16. Окно определения входа НС

Блок Input 1 (Вход 1) является стандартным блоком задания значения константы (Constant). Изменим значение по умолчанию, например, на 3 и нажмем кнопку OK. Затем выберем опцию Simulation (см. рис. 4.15) и нажмем кнопку Start в меню моделирования. Сеть рассчитает соответствующее значение. Для его вывода необходимо дважды щелкнуть мышью по блоку y(1). Результат вычислений приведен на рис. 4.17, – он равен 6, что и требуется.

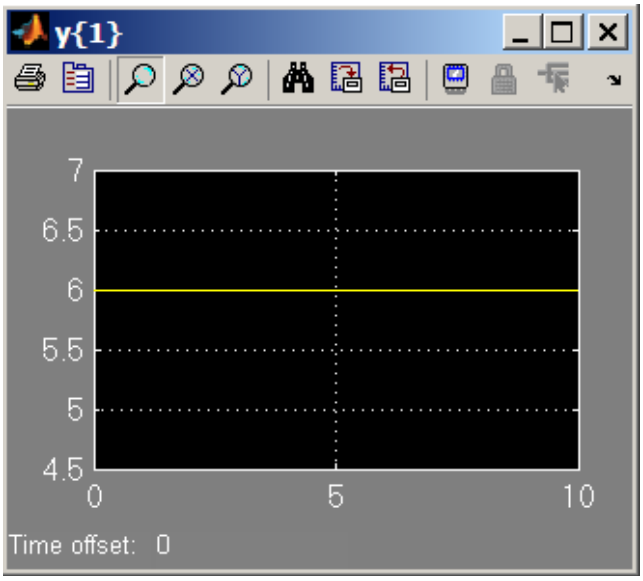

Рис. 4.17. Окно, отображающее выход НС

Заметим, что щелкнув левой кнопкой мыши по блоку Neural Network, затем – по блоку Layer 1, можно получить графическое изображение структуры нейросети (рис. 4.18).

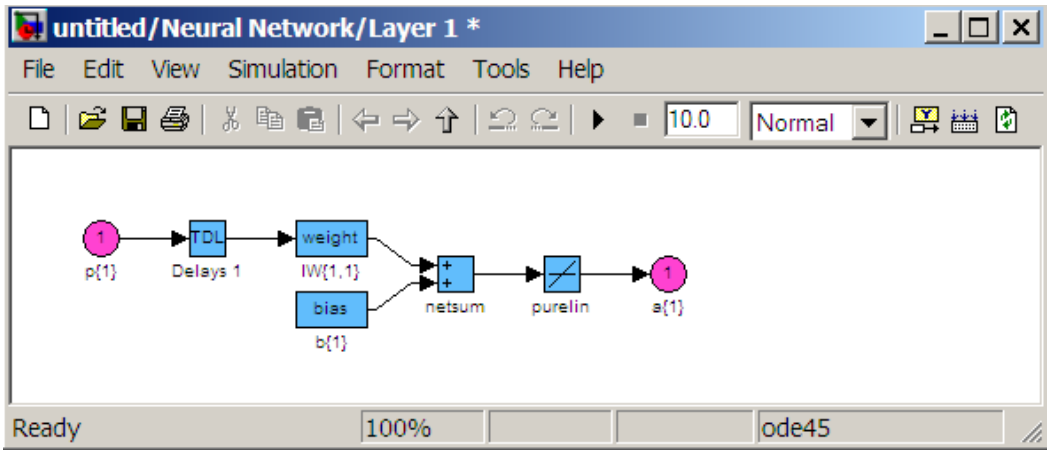

Рис. 4.18. Визуализация структуры созданной НС

С созданной сетью можно проводить различные эксперименты, возможные в среде Simulink. С помощью команды gensim осуществляется интеграция созданных нейросетей в блок-диаграммы этого пакета с использованием имеющихся при этом инструментов моделирования различных систем.

Контрольные вопросы

Какие функции используются для создания нейронной сети Хопфилда с двумя нейронами?

Объясните результаты работы обученной сети со слоем Кохонена, приведенные на рис. 4.4.

Как можно вывести графики работ с нейросетями? Какие параметры необходимо задавать для этого?

Приведите пример создания линейной нейронной сети с прогнозом сигнала по 4 известным значениям.

Как сохранить и вновь вызвать созданную сеть? Какое расширение будет иметь соответствующий файл?

Назовите основные моменты решения задачи распознавания (классификации) с помощью персептрона.

Как проводится опрос сети Хопфилда? Можно ли на основании результатов опроса НС сделать вывод о правильной работе сети?

Какие функции задействованы при создании самоорганизующихся карт? Каким образом выполняется настройка сети?

Сравните результаты работы слоя Кохонена для примеров 4.3 и 4.6.

Приведите примеры задач, которые возможно решить в пакете NNT.

Заключение

Искусственные нейронные сети предложены для задач, простирающихся от управления боем до присмотра за ребенком. Потенциальными приложениями являются те, где человеческий интеллект малоэффективен, а обычные вычисления трудоемки или неадекватны. Этот класс приложений во всяком случае не меньше класса, обслуживаемого обычными вычислениями, и можно предполагать, что искусственные нейронные сети займут свое место наряду с обычными вычислениями в качестве дополнения такого же объема и важности.

Это возрождение интереса было вызвано как теоретическими, так и прикладными достижениями. Неожиданно открылись возможности использования вычислений в сферах, до этого относящихся лишь к области человеческого интеллекта, возможности создания машин, способность которых учиться и запоминать удивительным образом напоминает мыслительные процессы человека.

Было разработано множество обучающих алгоритмов, каждый со своими сильными и слабыми сторонами. Все еще существуют проблемы относительно того, чему сеть может обучиться и как обучение должно проводиться.

Представленные в пособии типовые нейросетевые парадигмы дают представление об искусстве конструирования сетей в целом, а многие другие парадигмы при тщательном рассмотрении оказываются лишь их модификациями. Поэтому понимание описанных парадигм позволит следить за прогрессом в этой быстро развивающейся области.

Для желающих продолжать саморазвитие в области искусственных нейронных сетей в литературе приводятся ссылки на более полные и строгие источники, а навыки работы с пакетом Neural Network Toolbox системы Matlab позволят реализовать сети сложной архитектуры.

ПРИЛОЖЕНИЯ

Приложение 1

Типы нейросетей NNT

Тип сети	Название сети	Число слоев	Обучаемые параметры
1	2	3	4
Competitive	Конкурирующая сеть	1	IW{1, 1}, b{1}
Cascade-forward backprop	Каскадная сеть с прямым распространением сигнала и обратным распространением ошибки	2	IW{1, 1}, b{1}, LW{2, 1}, IW{2, 1}, b{2}
Elman backprop	Сеть Элмана с обратным распространением ошибки	2	IW{1, 1}, b{1}, LW{2, 1}, b{2}, LW{1, 1}
Feed-forward backprop	Сеть с прямым распространением сигнала и обратным распространением ошибки	2	IW{1, 1}, b{1}, LW{2, 1}, b{2}
Time delay backprop	Сеть с запаздыванием и обратным распространением ошибки	2	IW{1, 1}, b{1}, LW{2, 1}, b{2}
Generalized regression	Обобщенная регрессионная сеть	2	IW{1, 1}, b{1}, LW{2, 1}
Hopfield	Сеть Хопфилда	1	LW{1, 1}, b{1}
Linear layer (design)	Линейный слой (создание)	1	IW{1, 1}, b{1}
Linear layer (train)	Линейный слой (обучение)	1	IW{1, 1}, b{1}
LVQ	Сеть для классификации вход-ных векторов	2	IW{1, 1}, LW{2, 1}

1	2	3	4
Perceptron	Персептрон	1	IW{1, 1}, b{1}
Probabilistic	Вероятностная сеть	2	IW{1, 1}, b{1}, LW{2, 1}
Radial basis (exact fit)	Радиальная базисная сеть с нулевой ошибкой	2	IW{1, 1}, b{1}, LW{2, 1}
Radial basis (fewer neurons)	Радиальная базисная сеть с минимальным числом нейронов	2	IW{1, 1}, b{1}, LW{2, 1}, b{2}
Self-organizing map	Самоорганизующаяся карта Кохонена	1	IW{1, 1}

Демонстрационные примеры NNT

По команде help nndemos можно получить перечень демонстрационных примеров, включенных в расширение NNT.

Персептроны	
demop1	Классификация с использованием персептрона с двумя входами
demop4	Формирование входных векторов внешнего слоя
demop5	Обучение с использованием нормированной функции настройки
demop6	Пример линейно неразделимых векторов
Линейные сети	
demolin1	Пример функционирования линейной сети
demolin2	Обучение линейного нейрона
demolin4	Задача линейной аппроксимации
demolin5	Задача с неполными данными
demolin6	Задача с линейно зависимыми данными
demolin7	Оценка влияния параметра скорости настройки
demolin8	Адаптируемый линейный слой

Радиальные базисные сети	
demorb1	Радиальные базисные сети
demorb3	Пример неперекрывающихся функций активации
demorb4	Пример перекрывающихся функций активации
demogrn1	Сеть GRNN и аппроксимация функций
demopnn1	Сеть PNN и классификация векторов
Сети кластеризации и классификации данных	
Самоорганизующиеся сети	
democ1	Настройка слоя Кохонена
demosm1	Одномерная карта Кохонена
demosm2	Двумерная карта Кохонена
LVQ-сети	
demolvq1	Классификация векторов
Рекуррентные сети	
Сети Элмана	
appelm1	Сеть Элмана
Сети Хопфилда	
demohop1	Пример двумерной модифицированной сети Хопфилда
demohop2	Пример неустойчивой точки равновесия
demohop3	Пример трехмерной модифицированной сети Хопфилда
demohop4	Пример устойчивых паразитных точек равновесия
Применение нейронных сетей	
applin1	Предсказание стационарного сигнала
applin2	Предсказание нестационарного сигнала
appelm1	Детектирование амплитуды с помощью сети Элмана
appcr1	Распознавание символов
Нейронные сети и системы управления (среда Simulink)	
predcstr	Управление каталитическим реактором
narmamaglev	Управление магнитной подушкой
mrefrobotarm	Управление звеном робота

Приложение 2

Работа с GUI-интерфейсом

GUI-интерфейс NNTool можно вызвать командой nntool из командной строки Matlab. После вызова на экране терминала появляется окно вида

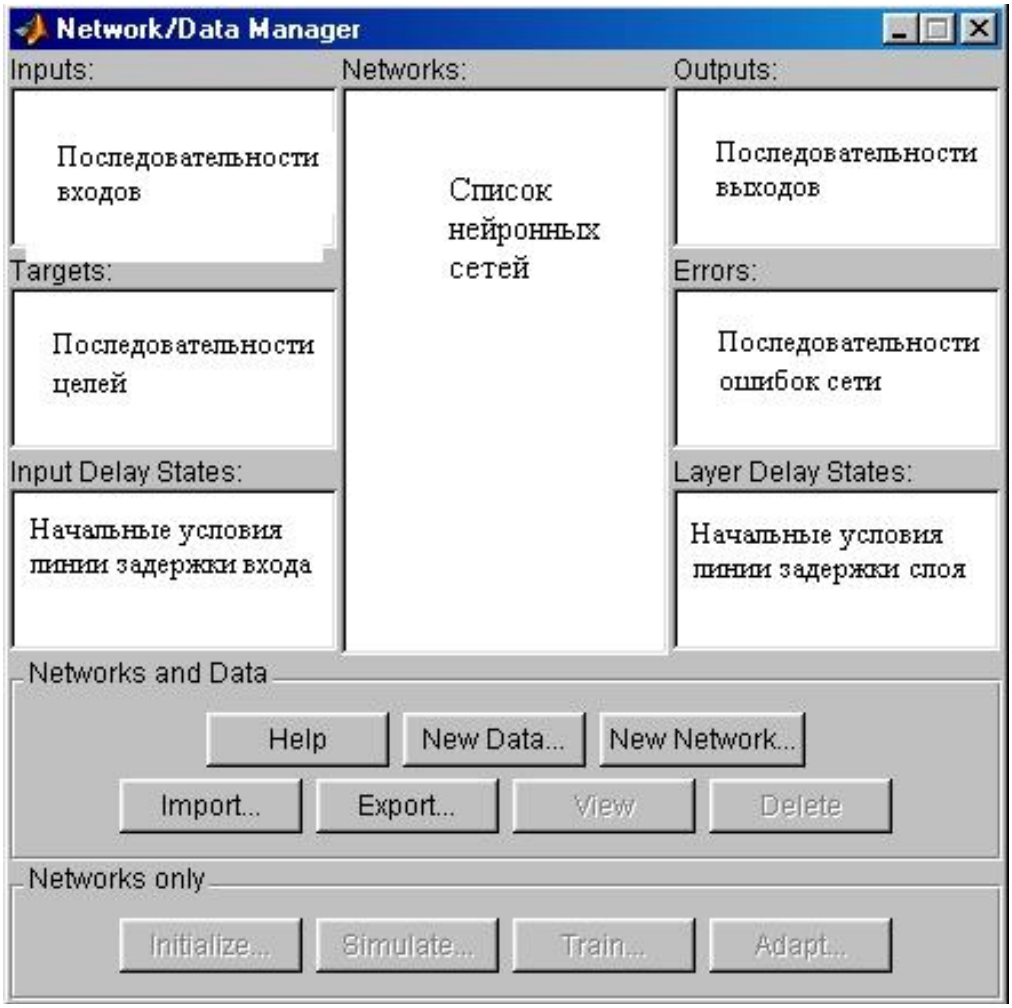

Рис. 1. Окно управления сетью/данными

Описание кнопок:

Help – кнопка вызова окна подсказки Network/Data Manager Help;

New Data… – кнопка вызова окна формирования данных Create New Data;

New Network… – кнопка вызова окна создания новой нейронной сети Create New Network;

Import…– кнопка вызова окна для импорта или загрузки данных Import or Load to Network/Data Manager;

Export… – кнопка вызова окна для экспорта или записи данных в файл Export or Save from Network/Data Manager.

Кнопки View, Delete будут активны только после создания и активизации данных, относящихся к последовательностям входа, цели, выхода или ошибок сети. Кнопка View позволяет просмотреть, а кнопка Delete удалить активизированные данные.

Кнопки View, Delete, Initialize…, Simulate…, Train…, Adapt… становятся активными после создания и активизации самой нейронной сети. Они позволяют просмотреть, удалить, инициализировать, смоделировать, обучить или адаптировать нейронную сеть.

Опишем рабочие окна интерфейса.

Окно **Network/Data Manager Help**. Это окно помощи показано на рис. 2 и описывает правила работы с диспетчером Network/Data Manager при создании нейронной сети.

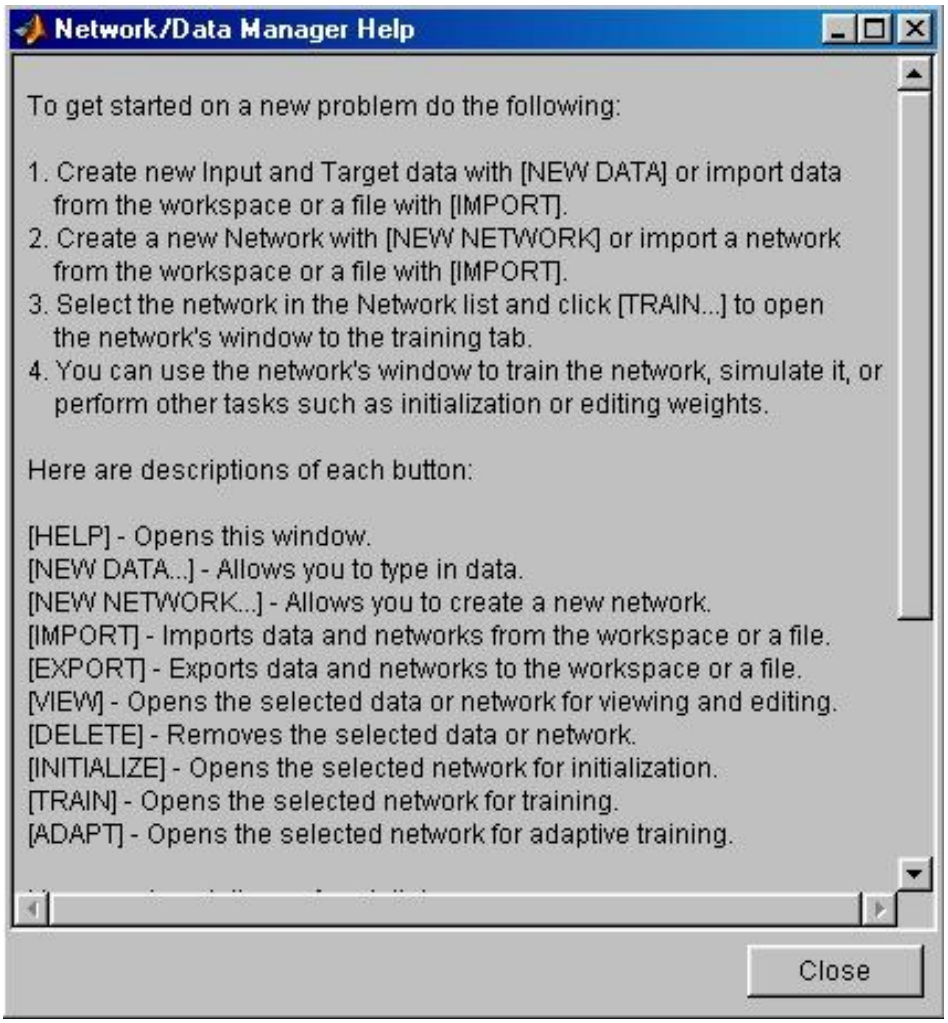

Рис. 2. Окно помощи

Чтобы создать нейронную сеть, необходимо выполнить следующие операции:

- Сформировать последовательности входов и целей (кнопка New Data), либо загрузить их из рабочей области системы MATLAB или из файла (кнопка Import).
- Создать новую нейронную сеть (кнопка New Network), либо загрузить ее из рабочей области системы MATLAB или из файла (кнопка Import).
- Выбрать тип нейронной сети и нажать кнопку Train…, чтобы открыть окно для задания параметров процедуры обучения.
- Открыть окно Network для просмотра, инициализации, моделирования, обучения и адаптации сети.

Окно **Create New Data**. Это окно включает 2 области редактирования текста для записи имени вводимых данных (область Name) и ввода самих данных (область Value), а также 6 кнопок для указания типа вводимых данных.

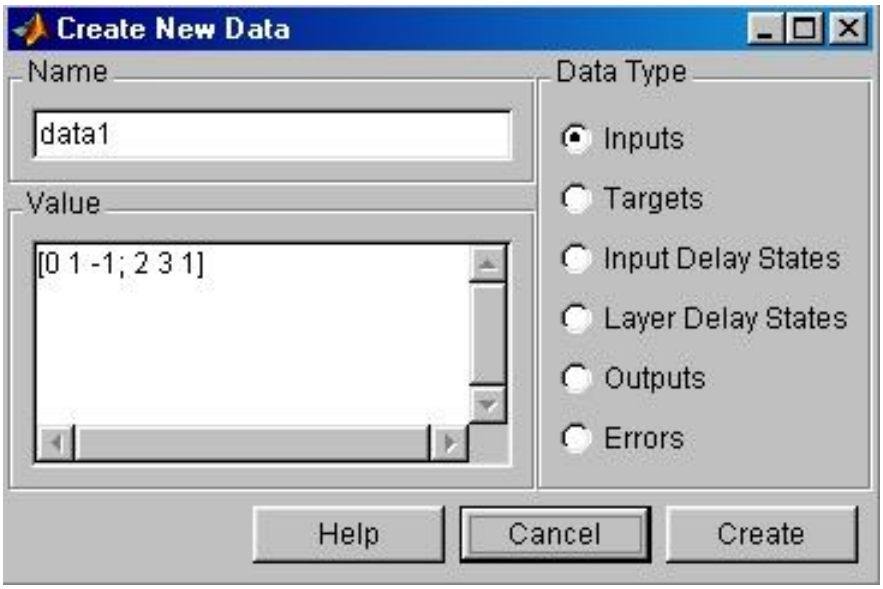

Рис. 3. Окно для создания новых данных

Здесь можно определить следующие типы данных:

Inputs (Входы) – последовательность значений входов;

Targets (Цели) – последовательность значений цели;

Input Delay States (Состояния ЛЗ входа) – начальные условия линии задержки на входе;

Layer Delay States (Состояния ЛЗ слоя) – начальные условия линии задержки в слое;

Outputs (Выходы) – последовательность значений выхода сети;

Errors (Ошибки) – разность значений целей и выходов.

Обычно пользователь задает только последовательности входа и цели, т. е. типы данных Inputs и Targets. Следует помнить, что при адаптации нейронной сети данные должны быть представлены в виде массива ячеек.

Окно **Create New Network**. Это окно показано на рис. 4 и включает поля для задания параметров создаваемой сети. В зависимости от типа сети количество полей и их названия изменяются.

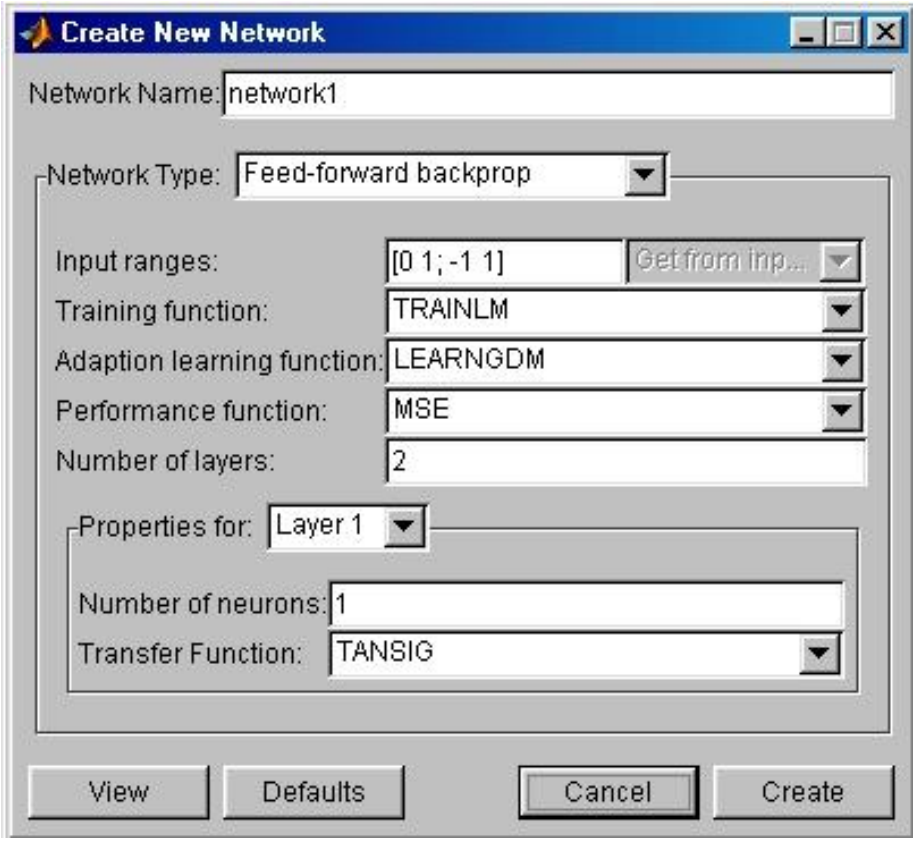

Рис. 4. Окно создания сети

Описание полей:

Network Name (Имя сети) – стандартное имя сети, присваиваемое GUI-интерфейсом NNTool; в процессе создания новых сетей порядковый номер будет изменяться автоматически.

Input ranges (Диапазоны входа) – допустимые границы входов, которые либо назначаются пользователем, либо определяются автоматически по имени входной последовательности, выбираемой из списка Get from Inp… .

Training function (Функция обучения) – список обучающих функций.

Adaption learning function (Функции настройки для режима адаптации) – список функций настроек.

Performance function (Функция качества обучения) – список функций оценки качества обучения.

Number of layers (Количество слоев) – количество слоев нейронной сети.

Properties for (Свойства) – список слоев: Layer 1 (Слой 1), Layer 2 (Слой 2).

Number of neurons (Количество нейронов) – количество нейронов в слое.

Transfer function (Функция активации) – функция активации слоя.

Окно **Import or Load to Network/Data Manager**. Это окно показано на рис. 5 и включает 3 поля.

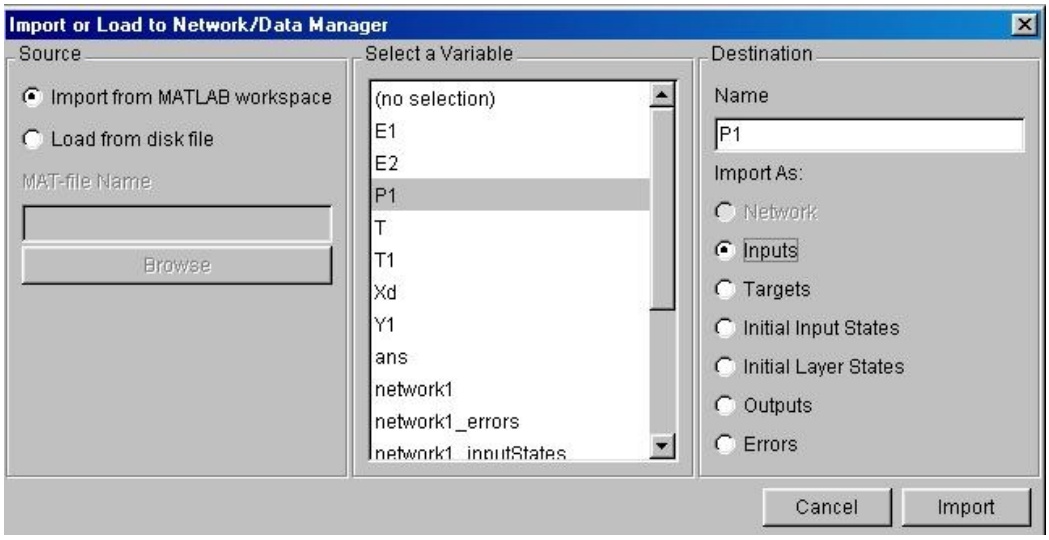

Рис. 5. Окно для импорта или загрузки данных

Source (Источник) – поле для выбора источника данных. Это либо рабочая область системы MATLAB (кнопка выбора Import from MATLAB Workspace), либо файл (кнопка выбора Load from disk file).

Если выбрана первая кнопка, то в поле Select a Variable можно видеть все переменные рабочей области и, выбрав одну из них, например P1, можно определить ее в поле Destination (Назначение) как последовательность входа Inputs (Входы).

Если выбирается кнопка Load from disk file, то активизируется поле MAT-file Name и кнопка Browse, что позволяет начать поиск и загрузку файла из файловой системы.

Окно **Export or Save from Network/Data Manager**. Окно показано на рис. 6 и позволяет передать данные из рабочей области GUI-интерфейса NNTool в рабочую область системы MATLAB или записать их в виде файла на диске.

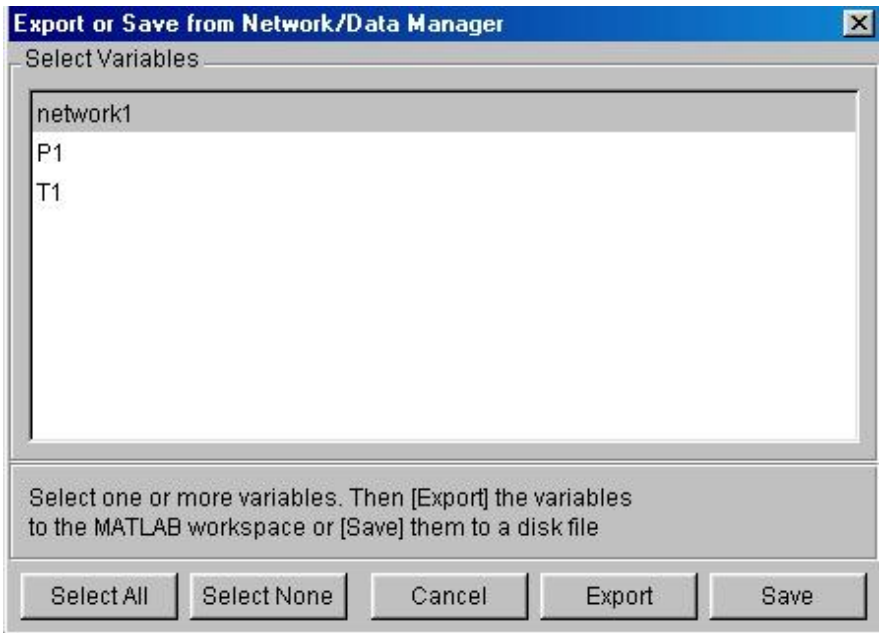

Рис. 6. Окно экспорта или сохранения данных

В данном случае выбрана переменная network1, которая принадлежит к классу network object и описывает нейронную сеть. После того как эта переменная экспортирована в рабочую область, можно, например, построить модель нейронной сети в системе Simulink с помощью оператора gensim.

Диалоговая панель Network показана на рис. 7.

.

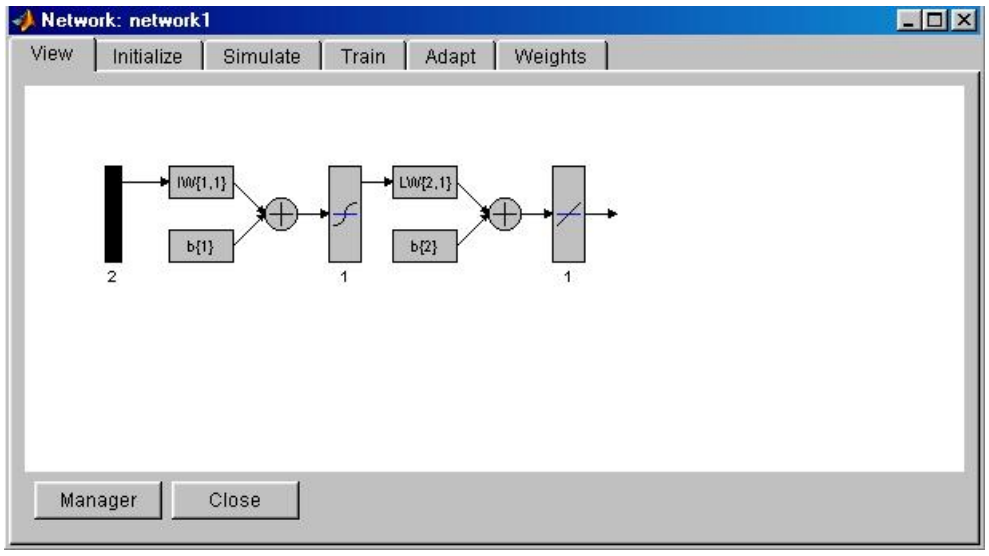

Рис. 9. Структура сети

Она открывается только в том случае, когда в окне Network/Data Manager выделена созданная сеть и становятся активными кнопки View, Initialize, Simulate, Train, Adapt. Панель имеет 6 опций:

View *(*Просмотреть*)* – структура сети;

Initialize (Инициализация) – задание начальных весов и смещений;
Simulate (Моделирование) – моделирование сети;
Train (Обучение) – обучение сети;
Adapt (Адаптация) – адаптация и настройка параметров сети;
Weights (Веса) – просмотр установленных весов и смещений.

Библиографический список

1. Ануфриев И.Е. MATLAB 7. Наиболее полное руководство в подлиннике. / И.Е. Ануфриев, А.Б. Смирнов, Е.Н. Смирнова. – СПб.: БХВ-Петербург, 2005. – 1104 с.

2. Комарцова Л.Г. Нейрокомпьютеры: учебное пособие для вузов. – 2-е изд. / Л.Г. Комарцова, А.В. Максимов. – Изд-во МГТУ им. Н.Э. Баумана, 2004. – 400 с.

3. Медведев В.С. Нейронные сети. MATLAB 6 / В.С. Медведев, В.Г. Потемкин. – М.: ДИАЛОГ-МИФИ, 2002. – 496 с.

4. Уоссермен Ф. Нейрокомпьютерная техника: теория и практика. / Ф. Уоссермен. – М.: Мир, 1992. – 118 с.

5. Ясницкий Л.Н. Искусственный интеллект. Элективный курс: учебное пособие / Л.Н. Ясницкий. – М.: БИНОМ, Лаборатория знаний, 2011. – 200 с.

6. http://matlab.exponenta.ru/neuralnetwork/

7. http://www.mathworks.com/products/neural-network/

Содержание

Учебное издание

Николаева Светлана Глебовна

НЕЙРОННЫЕ СЕТИ. РЕАЛИЗАЦИЯ В MATLAB

Учебное пособие по дисциплине
«Интеллектуальные системы»

Кафедра инженерной кибернетики КГЭУ

Редактор издательского отдела *С.Н. Касимова*
Компьютерная верстка *Ю.Ф. Мухаметшина*

Подписано в печать 15.04.15.
Формат 60×84/16. Бумага ВХИ. Гарнитура «Times». Вид печати РОМ.
Усл. печ. л. 5,34. Уч.-изд. л. 5,93. Тираж 500 экз. Заказ № 01/эл.

Редакционно-издательский отдел КГЭУ,
420066, Казань, Красносельская, 51