МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

МОСКОВСКИЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ

А.Б. СОРОКИН, О.В. ПЛАТОНОВА

ИСКУССТВЕННЫЕ НЕЙРОННЫЕ СЕТИ ПРЯМОГО РАСПРОСТРАНЕНИЯ

УЧЕБНО-МЕТОДИЧЕСКОЕ ПОСОБИЕ

Москва — 2018

УДК 004.032.26
ББК 32.818
С 65

Печатается по решению редакционно-издательского совета
Московского технологического университета (МИРЭА)

Рецензенты:
Хоботов Евгений Николаевич
д.т.н., профессор, профессор кафедры компьютерные системы
автоматизации производства МГТУ им. Н.Э.Баумана
Коваленко Сергей Михайлович
к.т.н., доцент, заведующий кафедрой вычислительной техники
Московского технологического университета (МИРЭА)

С 65 **Сорокин А.Б.**
Искусственные нейронные сети прямого распространения: учебно-методическое пособие / А.Б. Сорокин А.Б., Платонова О.В. – М.: Московский технологический университет (МИРЭА), 2018. – 64 с.

В пособии рассматриваются необходимые теоретические сведения о моделях искусственных нейронных сетей прямого распространения, которые в последние годы активно используется в различной практической деятельности. Приводится описание основных структур и алгоритмов обучения нейронных сетей данного класса. Подробно описаны примеры численного решения конкретных задач. Приводятся тексты программных кодов на языках Java и Python.
Предназначено для студентов 3-го курса квалификации бакалавр, обучающихся по направлению 09.03.04 «Программная инженерия» по профилю «Системы поддержки принятия решений».

УДК 004.032.26
ББК 32.818

ISBN 000-0-0000-0000-0

Оглавление

ВВЕДЕНИЕ

В учебно-методическом пособии рассматриваются искусственные нейронные сети прямого распространения. При этом каждая глава определена соответствующим классом нейротехнологий этого направления.

В первой главе рассматривается необходимые знания о функционировании биологического нейрона. Показывается как на основе этих знаний создана математическая модель – искусственный нейрон МакКаллока – Питтса. В главе большое внимание уделяется численным расчетам логических операций «И», «ИЛИ» и «НЕТ», которые являются основой для понимания дальнейшего изложения.

Вторая глава посвящена однослойным нейронным сетям прямого распространения, которые представлены элементарным персептроном Розенблатта. Для данной модели рассматриваются алгоритмы детерминированного обучения по правилам Хебба, дельта-правилу и правилу Видроу-Хоффа. Представлены не только численные расчеты для понимания правил обучения, но также программная реализация распознавания цифр на языке Java.

Третья глава определена одним из самых значимых классов в направлении нейросетевых технологий – многослойный персептрон. Это представление тесно связано с новым направлением в искусственном интеллекте, так называемым «глубоким обучением». В главе рассмотрен наиболее популярный алгоритм обучения – метод обратного распространения ошибки. Представлены его реализации в численном и программном виде. Большое внимание уделено описанию программной реализации нейронной сети на языке Java, которое позволяет выбрать действие из доступного списка на основании того, в какой окружающей среде находится персонаж компьютерной игры.

В последней главе рассматриваются нейронные сети на основе радиальных базисных функций, в которых математическую основу их функционирования составляет теорема Ковера о разделимости образов. Приведен алгоритм и пример численного расчета параметров сети при гауссовой функции активации на основе метода псевдообратных матриц. Также рассмотрен программный код простейшего аппроксиматора на языке Python с использованием библиотек NumPy и matplotlib.

Надеемся, что данное пособие позволит заинтересованному читателю сделать первый шаг в изучении всего многообразия искусственных нейронных сетей и дополнит учебную литературу по данному направлению искусственного интеллекта.

От всей души желаем Вам успехов!

Авторы

1. МАТЕМАТИЧЕСКИЙ НЕЙРОН

Математический (искусственный) нейрон является структурной единицей искусственной нейронной сети и представляет собой аналог биологического нейрона. С математической точки зрения искусственный нейрон представляют, как некоторую нелинейную функцию от единственного аргумента, который является линейной комбинации всех входных сигналов.

1.1. Биологические представления о нейроне

Нервная система человека, построенная на основе элементов, называемых нейронами, имеет ошеломляющую сложность. Около 1011 нейронов участвуют в примерно 1015 передающих связях, имеющих длину метр и более. Каждый нейрон обладает многими свойствами, общими с другими органами тела, но ему присущи абсолютно уникальные способности: принимать, обрабатывать и передавать электрохимические сигналы по нервным путям, которые образуют коммуникационную систему мозга.

Нейрон– структурно-функциональная единица нервной системы, представляющая собой электрически возбудимую клетку, которая обрабатывает, хранит и передает информацию посредством электрических и химических сигналов (рис. 1.1) [1].

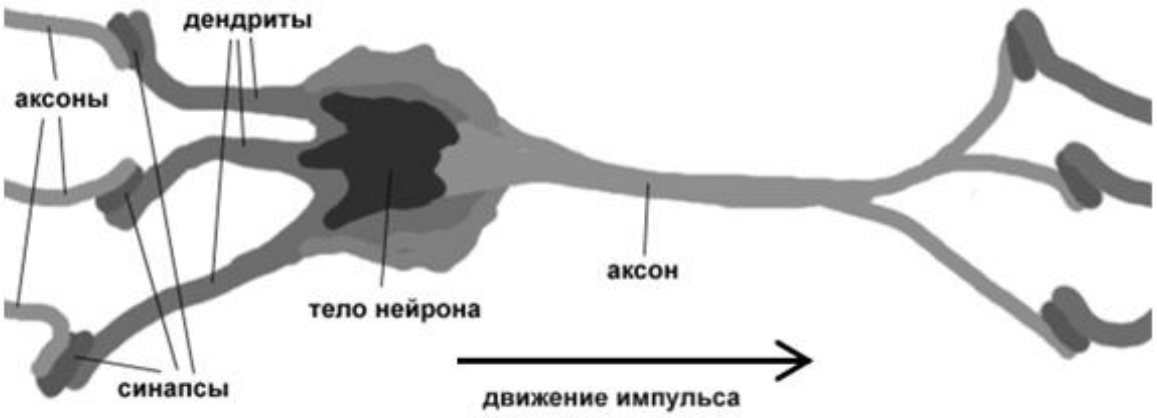

Рис. 1.1. Упрощенная структура биологического нейрона

Аксон – это нейрит (длинный цилиндрический отросток нервной клетки), по которому нервные импульсы идут от тела клетки (сомы) к другим нервным клеткам [2].

Дендрит – разветвлённый отросток нейрона, который получает информацию через химические (или электрические) синапсы от аксонов других нейронов и передаёт её через электрический сигнал телу нейрона [3].

Синапс – место контакта между двумя нейронами [2].

Дендриты идут от тела нервной клетки к другим нейронам, где они принимают сигналы в точках соединения, называемых синапсами. Принятые синапсом

входные сигналы передаются 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, посылая по аксону сигнал другим нейронам. У этой основной функциональной схемы много усложнений и исключений, тем не менее, большинство искусственных нейронных сетей моделируют лишь эти простые свойства.

1.2. Модель МакКаллока – Питтса

Первой работой, которая заложила теоретический фундамент для создания интеллектуальных устройств, моделирующих человеческий мозг на самом низшем структурном уровне, принято считать опубликованную в 1943 г. статью Уоррена Мак-Каллока и Вальтера Питтса «Идеи логических вычислений в нервной деятельности» [1]. Они предложили математическую модель нейрона мозга человека, назвав ее математическим (модельным) нейроном (рис. 1.2).

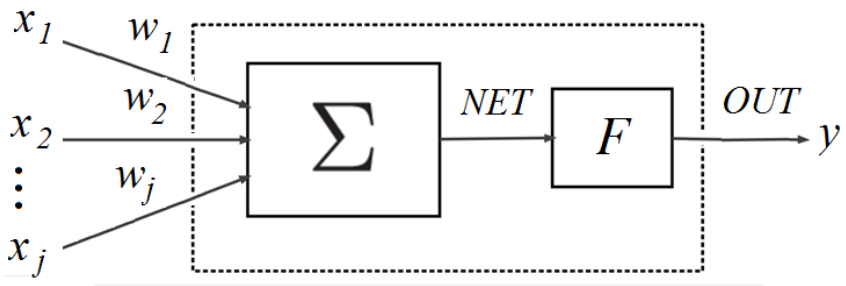

Рис. 1.2. Математический нейрон МакКаллока – Питтса

Математический нейрон принимает входные сигналы $x_1, x_2, \ldots, x_j$ и суммирует их, умножая каждый входной сигнал на некоторый весовой коэффициент w_j [3]. Выход из суммирующего блока обозначается NET.

$$S = NET = \sum_{j=1}^{J} w_j x_j \qquad (1.1)$$

После выполнения операции суммирования математический нейрон формирует с помощью активационной функцией F выходной сигнал y (OUT) [3].

$$OUT = F(NET)$$

Таким образом, активационная функция нейрона определяет нелинейное преобразование, осуществляемое нейроном и вычисляющая выходной сигнал формального нейрона. Выбор активационной функции определяется спецификой поставленной задачи либо ограничениями, накладываемыми некоторыми алгоритмами обучения [4]. В модели МакКаллока – Питса это пороговая функция вида (функция Хевисайда).

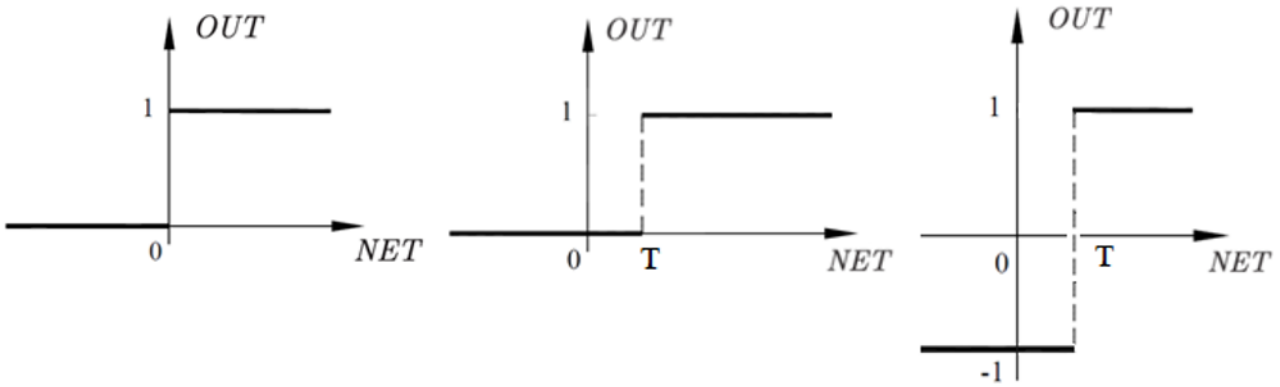

Рис. 1.3. Виды пороговых функций

Математическую запись пороговой функции выглядит следующим образом [5]:

$$y = OUT = \begin{cases} 1, \text{если } NET \geq T \\ 0, \text{если } NET < \text{T} \end{cases} \quad (1.2)$$

Таким образом, математический нейрон существует в двух состояниях. Если взвешенная сумма входных сигналов S меньше порога чувствительности T, то его выходной сигнал y равен нулю. В этом случае говорят, что нейрон не возбуждён. Если входные сигналы достаточно интенсивны и их взвешенная сумма S достигает порога чувствительности T, то нейрон переходит в возбуждённое состояние, и на его выходе, согласно правилу, образуется сигнал $y = 1$ [5]. При этом каждый математический нейрон имеет свое определенное значение порога чувствительности T.

С помощью математического нейрона с пороговой функцией можно моделировать различные логические операции. Например, бинарные операции: логического умножения «И» («AND» – конъюнкция), логического сложения «ИЛИ» («OR» – дизъюнкция -) и логического отрицания «НЕТ» («NOT»), истинность которых представлена в таблице 1.1.

Таблица 1.1. *Истинность бинарных операций.*

И ($\wedge$)		
x_1	x_2	$x_1 \wedge x_2$
0	0	0
0	1	0
1	0	0
1	1	1

ИЛИ ($\vee$)		
x_1	x_2	$x_1 \vee x_2$
0	0	0
0	1	1
1	0	1
1	1	1

НЕТ($\neg$)	
x	$\neg x$
0	1
1	0

Представим математические нейроны для логических операций умножения «И», сложения «ИЛИ» и логического отрицания «НЕТ» (рис. 1.4).

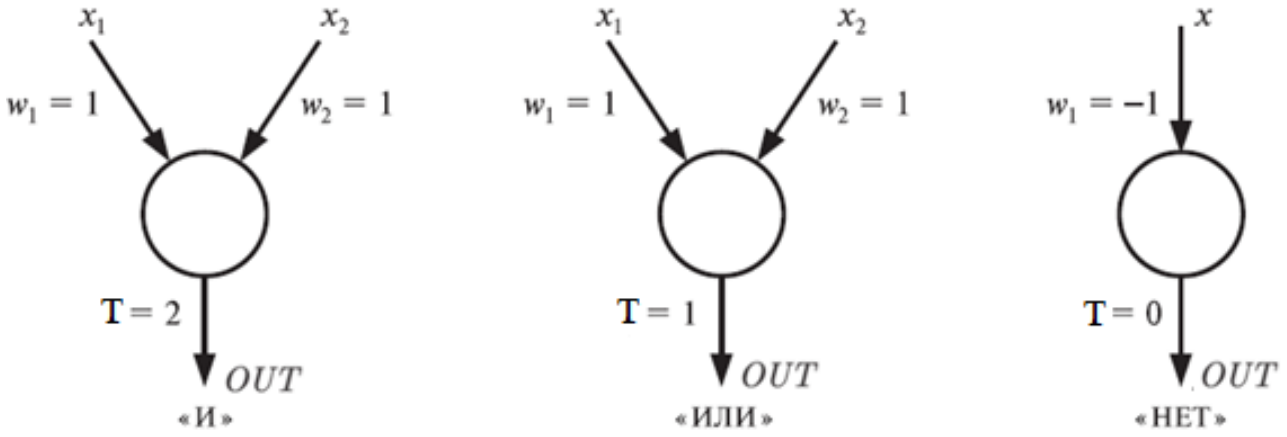

Рис. 1.4. Структура нейронов для логических операций «И», «ИЛИ», «НЕТ»

Рассмотрим пример расчета нейрона для логической операций умножения «И». Рассчитываем сигнал NET, согласно формуле 1.1:

При $x_1 = 0, x_2 = 0 \rightarrow x_1 * w_1 + x_2 * w_2 = 0 * 1 + 0 * 1 = NET_1 = 0$;

Соответственно: при $x_1 = 0, x_2 = 1 \rightarrow NET_2 = 1$; при $x_1 = 1, x_2 = 0 \rightarrow NET_3 = 1$; при $x_1 = 1, x_2 = 1 \rightarrow NET_4 = 1$;

Устанавливаем пороговую функцию в соответствии с рис.1.4 и согласно формуле 1.2

$$OUT = \begin{cases} 1, \text{если } NET \geq 2 \\ 0, \text{если } NET < 2 \end{cases}$$

Тогда на выходе математического нейрона будут следующие значения:

$$OUT_1 = 0, OUT_2 = 0, OUT_3 = 0, OUT_4 = 1$$

Аналогично рассчитываются нейрон для логической операции сложения «ИЛИ». Сигналы NET может такие же как у логического «И», но пороговая иная:

$$OUT = \begin{cases} 1, \text{если } NET \geq 1 \\ 0, \text{если } NET < 1 \end{cases}$$

Тогда на выходе математического нейрона будут следующие значения:

$$OUT_1 = 0, OUT_2 = 1, OUT_3 = 1, OUT_4 = 1$$

Расчет нейрона для операции логического «НЕТ» производится для двух сигналов NET.

При $x = 0 \rightarrow x * w = 0 * (-1) = NET_1 = 0$:

Тогда при $x = 1 \rightarrow NET_2 = -1$.

Устанавливаем пороговую функцию для логического отрицания.

$$OUT = \begin{cases} 1, \text{если } NET \geq 0 \\ 0, \text{если } NET < 0 \end{cases}$$

На выходе математического нейрона: $OUT_1 = 1, OUT_2 = 0$

Таким образом, при расчетах логических операций использовались различные пороговые функции (см. рис. 1.3). Чтобы прийти к единой функции с пороговой чувствительностью $T = 0$ вводят понятия смещения b, которое отличается

от T только знаком: $b = -T$. Нейронное смещение b рассматривается как вес w_0 некоторого дополнительного входного сигнала $x_0 = 1$, величина которого всегда равна единице.

$$NET = S = \sum_{j=1}^{J} w_j x_j + w_0 x_0 \quad (1.3)$$

Таким образом, дополнительный вход x_0 и соответствующий ему вес w_0 используются для инициализации нейрона. Под инициализацией подразумевается смещение активационной функции нейрона по горизонтальной оси, то есть формирование порога чувствительности нейрона.

Приведем примеры расчетов со смещением с пороговой функцией

$$OUT = \begin{cases} 1, \text{если } NET > 0 \\ 0, \text{если } NET \leq 0 \end{cases}$$

Установим $x_0 = 1$ и $w_0 = -3/2$ для расчета математическим нейронном (рис. 1.5) операции логического «И».

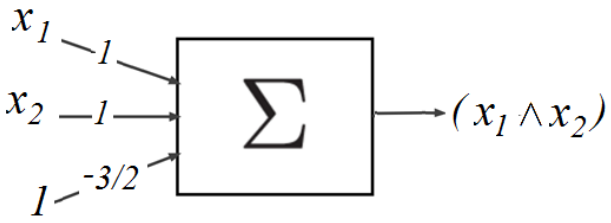

Рис. 1.5. Структура нейрона для логических операций «И»

Тогда согласно формуле 1.3.

При $x_1 = 0, x_2 = 0 \rightarrow NET_1 = x_1 * w_1 + x_2 * w_2 + x_0 * w_0 = 0 * 1 + 0 * 1 - 1 * 3/2 = -1{,}5$.

Соответственно: при $x_1 = 0, x_2 = 1 \rightarrow NET_2 = -0{,}5$; при $x_1 = 1, x_2 = 0 \rightarrow NET_3 = -0{,}5$; при $x_1 = 1, x_2 = 1 \rightarrow NET_4 = 0{,}5$;

Зная пороговую функцию определяем выходной сигнал математического нейрона для логической операции «И».

$$OUT_1 = 0, OUT_2 = 0, OUT_3 = 0, OUT_4 = 1$$

Установим $x_0 = 1$ и $w_0 = -1/2$ для расчета математическим нейронном (рис. 1.6) операции логического «ИЛИ».

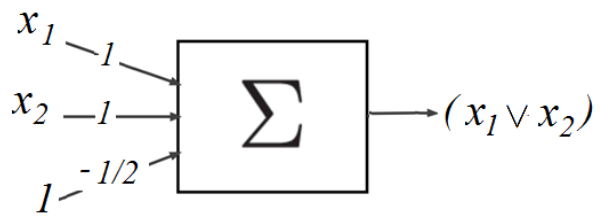

Рис. 1.6. Структура нейрона для логических операций «ИЛИ»

Согласно формуле 1.3.

При $x_1 = 0, x_2 = 0 \rightarrow NET_1 = x_1 * w_1 + x_2 * w_2 + x_0 * w_0 = 0 * 1 + 0 * 1 - 1 * 1/2 = -0{,}5$.

Соответственно: при $x_1 = 0, x_2 = 1 \rightarrow NET_2 = 0{,}5$; при $x_1 = 1, x_2 = 0 \rightarrow NET_3 = 0{,}5$; при $x_1 = 1, x_2 = 1 \rightarrow NET_4 = 1{,}5$;

Зная пороговую функцию определяем выходной сигнал математического нейрона для логической операции «ИЛИ».

$$OUT_1 = 0, OUT_2 = 1, OUT_3 = 1, OUT_4 = 1$$

Аналогично рассчитываем математический нейрон (рис. 1.7) для операции логического отрицания при $x_0 = 1$ и $w_0 = -1/2$

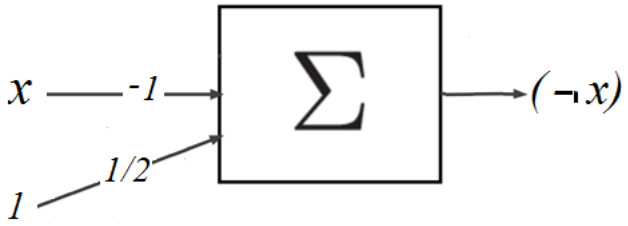

Рис. 1.7. Структура нейрона для логических операций «НЕТ»

Согласно формуле 1.3.

При $x = 0 \rightarrow NET_1 = x * w + x_0 * w_0 = 0 * (-1) + 1 * 1/2 = 1/2$:

Тогда при $x = 1 \rightarrow NET_2 = -1/2$.

Соответственно, на выходе математического нейрона будут сигналы

$$OUT_1 = 1 \text{ и } OUT_2 = 0.$$

Таким образом, с математическим нейроном МакКаллока – Питтса связана классическая задача линейного (однозначного) разделения элементов множества на два класса. К первому классу относятся входные вектора с выходным сигналом $OUT = 1$, а ко второму классу, если выходной сигнал $OUT = 0$.

Тогда существует критическое условие классификации (уравнение разделяющей гиперплоскости)

$$(W, X) = \sum_{j=0}^{N} w_j x_j = 0$$

В N -мерном пространстве (пространстве входных сигналов) разделяющая гиперплоскость перпендикулярна вектору $W = (w_1, \ldots, w_N)$. Вектор входных сигналов $X = (x_1, \ldots, x_N)$ дает выход «1», если его проекция $X_w = (X, W)/\|W\|$ на вектор W больше, чем расстояние $-w_0/\|W\|$ от нуля до гиперплоскости.

В двухмерном пространстве входных сигналов уравнение гиперплоскости имеет вид [6]:

$$x_1 * w_1 + x_2 * w_2 + w_0 = 0 \qquad (1.4)$$

Рассмотрим пример определения гиперплоскости для логического «И» (рис. 1.8)

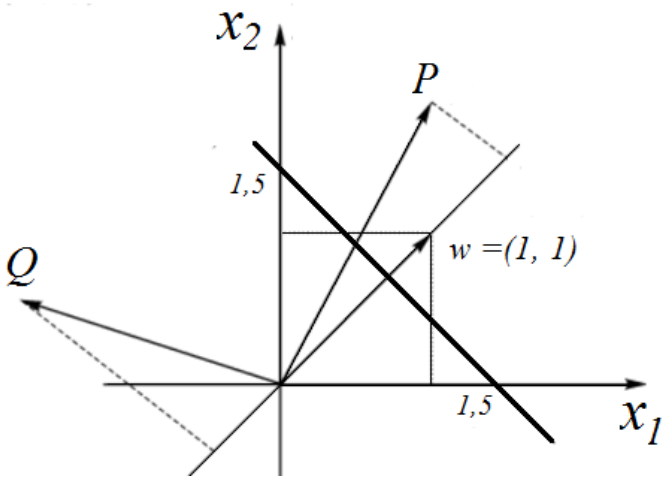

Рис. 1.8. Гиперплоскость для логической операции «И»

Тогда используя формулу 1.4 при $w_1 = w_2 = 1$ и $w_0 = -1{,}5$ получаем уравнение $x_1 + x_2 - 1{,}5 = 0$ гиперплоскости, которая представлена на рис.1.8 жирной линией, пересекающей оси координат в точках (1.5, 0) и (0, 1.5) соответственно. Здесь:$w = (1, 1)$ – нормаль к разделяющей гиперплоскости; P – вектор, относящийся к первому классу с выходным сигналом $OUT = 1$, поскольку проекция $(w,\ P)$ вектора P на нормаль w больше $-w_0/\|w\|$; Q – вектор, относящийся ко второму классу с выходным сигналом $OUT = 0$, поскольку $(w,\ Q) < -w_0/\|w\|$ [7].

Аналогично можно представить гиперплоскости $x_1 + x_2 - 0{,}5 = 0$ для логической «ИЛИ» и $x - 0{,}5 = 0$ для логического «НЕТ» (рис.1.9)

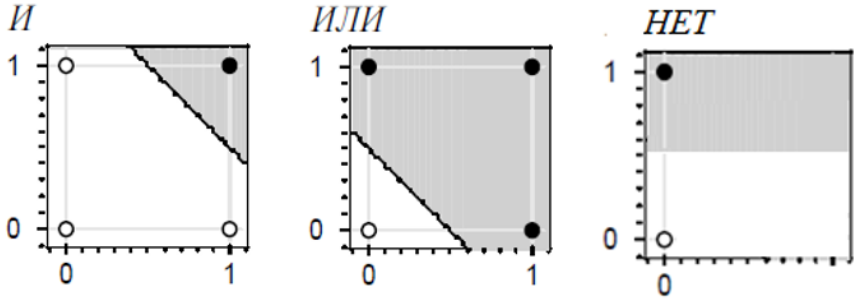

Рис. 1.9. Гиперплоскость для логических операций «И», «ИЛИ» и «НЕТ»

Однако имеется обширный класс функций, не реализуемых математическим нейроном. Эти функции являются линейно неразделимыми: они-то и накладывают определенные ограничения на возможности однослойных сетей. Примером служит логическая операция «ИСКЛЮЧАЮЩЕЕ ИЛИ» [6].

2. ОДНОСЛОЙНАЯ НЕЙРОННАЯ СЕТЬ

Хотя один нейрон и способен выполнять простейшие процедуры распознавания, но для серьезных нейронных вычислений необходимо соединять нейроны в сети. Сеть, в которой все входные элементы соединены непосредственно с выходными элементами, называется однослойной нейронной сетью, или сетью персептрона. Персептрон (от лат. perceptio – восприятие) – кибернетическая модель восприятия информации мозгом человека [2].

2.1. Однослойный персептрон прямого распространения

Простейшая нейронная сеть состоит из одного слоя нейронов (рис.2.1).

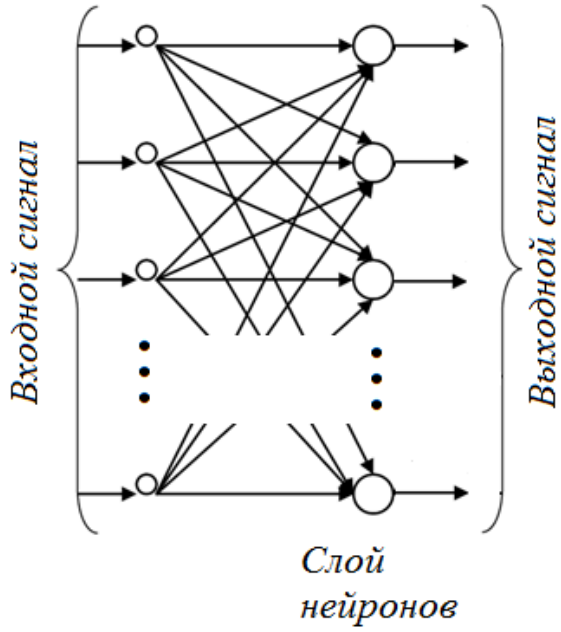

Рис. 2.1 Структура однослойного персептрона.

Слой нейронов – это набор нейронов, на которые в каждый момент времени параллельно поступает входные сигналы. Отметим, что на рис. 2.1 вершины слева от слоя нейронов служат лишь для распределения входных сигналов. Они не выполняют каких-либо вычислений, и поэтому не считаются слоем [8].

Таким образом, каждый элемент из множества входов X отдельным весом соединен с каждым искусственным нейроном, а каждый нейрон выдает взвешенную сумму входов в NET. При этом многие соединения могут отсутствовать.

Веса связей удобно считать как элементы матрицы W. Пусть матрица имеет m строк и n столбцов, где m – число входов, а n – число нейронов.

$$W = \begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1m} \\ w_{21} & w_{22} & \cdots & w_{2m} \\ \cdots & \cdots & \cdots & \cdots \\ w_{n1} & w_{n2} & \cdots & w_{nm} \end{pmatrix}$$

Например, w_{23} – это вес, связывающий второй вход с третьим нейроном. Таким образом, вычисление выходного вектора Y, компонентами которого являются выходы OUT нейронов, сводится к матричному умножению $Y = XW$, где Y и X – векторы-строки.

Разновидностью однослойной сети является элементарный персептрон Розенблатта, который в физическом воплощении соответствуют светочувствительным клеткам сетчатки глаза (рис. 2.2) [1].

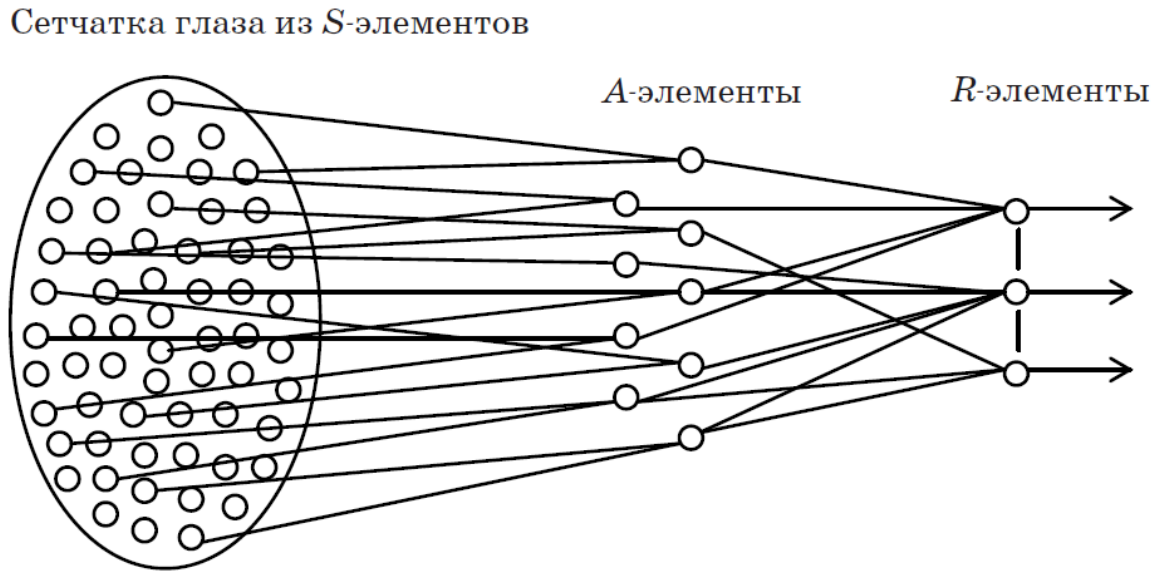

Рис. 2.2 Структура модели зрительной системы

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

Сенсорные нейроны S возбуждаются от воздействия энергии света при превышении некоторого порога. Поэтому предназначены для формирования входных сигналов и выполняют распределительную функцию. Каждый сенсорный нейрон связан с одним или несколькими нейронами следующего слоя (ассоциативными нейронами) [2].

Тогда ассоциирующие A-элементы возбуждаются только тогда, когда возбуждено достаточное число S-элементов, связанных с ними. -элементы предназначены для непосредственной обработки входных сигналов. Выходы ассоциативных нейронов соединены с входами нейронов третьего реагирующего слоя.

Реагирующий R-элемент возбуждается, если на его входе возникает определенная комбинация A-элементов. R-нейроны предназначены для передачи сигналов к другим нейронам или сетям. Нейроны этого слоя имеют несколько входов (дендритов) и один выход (аксон), который возбуждается, если суммарная

величина входных сигналов превосходит порог срабатывания (функция активации нейронов – пороговая) [4].

Таким образом, -элементы и R-элементы подсчитывают сумму значений входных сигналов, помноженных на веса w_i и считаются одним слоем. Соответственно, -элементы не выполняют вычислительных функций и фактически не являются слоем.

Задача обучения персептрона заключается в возбуждении его только при предъявлении определенных входных векторов и отсутствии реакции на другие входные векторы. Обучение элементарного персептрона состоит в изменении весовых коэффициентов w_i связей $A - R$. Веса связей $S - A$ принимают следующие значения; $\{-1;\ 0;\ +1\}$, а значения порогов А-элементов выбираются случайным образом. Таким образом, при обучении персептрона решается задача бинарной классификации – путем настройки весов персептрона строится гиперплоскость, разделяющая все входные векторы на два класса [3].

2.2. Обучение искусственной нейронной сети на основе правил Хебба

Самым важным свойством нейронных сетей является их способность обучаться на основе данных окружающей среды и в результате обучения повышать свою производительность. Обучение нейронной сети происходит посредством интерактивного процесса корректировки синаптических весов и порогов. В идеальном случае нейронная сеть получает знания об окружающей среде на каждой итерации процесса обучения.

Обучение – это процесс, в котором свободные параметры нейронной сети настраиваются посредством моделирования среды, в которую эта сеть встроена. Тип обучения определяется способом подстройки этих параметров [2].

Несложно догадаться, что не существует универсального алгоритма обучения, подходящего для всех архитектур нейронных сетей. Существует лишь набор средств, представленный множеством алгоритмов обучения, каждый из которых имеет свои достоинства и недостатки.

Существуют два концептуальных подхода к обучению нейронных сетей: обучение с учителем и обучение без учителя.

Обучение нейронной сети с учителем предполагает, что для каждого входного вектора из обучающего множества существует требуемое значение выходного вектора, называемого целевым. Эти вектора образуют обучающую пару. Веса сети изменяют до тех пор, пока для каждого входного вектора не будет получен приемлемый уровень отклонения выходного вектора от целевого [9].

Обучение нейронной сети без учителя является намного более правдоподобной моделью обучения с точки зрения биологических корней искусственных

нейронных сетей. Обучающее множество состоит лишь из входных векторов. Алгоритм обучения нейронной сети подстраивает веса сети так, чтобы получались согласованные выходные векторы, т.е. чтобы предъявление достаточно близких входных векторов давало одинаковые выходы [9].

Обучение Хебба (название метода в честь его создателя) является самым известным среди всех правил обучения и основывается на двух правилах [2]:

Первое правило – Если сигнал персептрона неверен и равен нулю, то необходимо увеличить веса тех входов, на которые была подана единица.

Второе правило — Если сигнал персептрона неверен и равен единице, то необходимо уменьшить веса тех входов, на которые была подана единица.

По существу, было предположено, что синаптическое соединение двух нейронов усиливается, если оба эти нейрона возбуждены. Это можно представить, как усиление синапса в соответствии с корреляцией уровней возбужденных нейронов, соединяемых данным синапсом.

Пусть -ая клетка с выходным сигналом y_i связана с j-ой клеткой с выходным сигналом y_j связью с весом w_{ij}, то на значение веса влияют выходные сигналы y_i и y_j (рис. 2.3) [5].

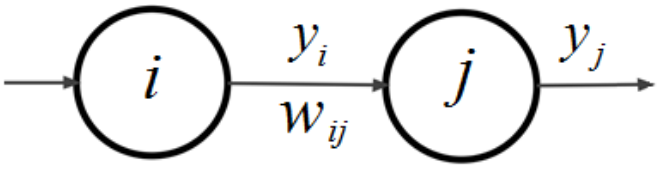

Рис. 2.3. Структурная схема обучаемого нейрона

В соответствии с правилом Хебба вес нейрона изменяется по формуле:

$$\Delta w_{ij} = \eta y_i y_j \,, \tag{2.1}$$

где Δw_{ij} – изменение сила синапса (веса) от нейрона i к нейрону j; η – это коэффициент обучения, значения которого выбираются из интервала (0,1); $y_i = NET_i$ – уровень возбуждения пресинаптического нейрона; $y_j = NET_j$ – уровень возбуждения постсинаптического нейрона.

Правила Хебба могут применяться при обучении «с учителем» и «без учителя». При обучении с учителем выходные сигналы сравниваются с эталонными значениями. При использовании алгоритма без учителя: реальные сигналы с выхода нейронной сети используются для кластеризации входных образов.

Алгоритм обучения с учителем по правилу Хебба сводится к следующей последовательности действий [2]:

1. Инициализация весовых коэффициентов и порогов случайными значениями, близкими к нулю (чтобы сеть сразу не могла войти в насыщение).

2. Подача на вход нейронной сети очередного входного образа.

3. Вычисление значения выхода.

4. Если значение выхода не совпадает с эталонным значением, то происходит модификация коэффициентов в соответствии с формулой при 2.1 η =1

$$w_{ij}(t+1) = w_{ij}(t) + y_j d_i \quad (2.2)$$

где d_i – ожидаемая реакция нейрона.

$$T(t+1) = T(t) \ - d_i \quad (2.3)$$

где T – смещение нейрона.

В противном случае осуществляется переход к пункту 5.

5. Если не все вектора из обучающей выборки были поданы на вход НС, то происходит переход к пункту 2. Иначе, переход к пункту 6.

6. Останов.

Рассмотрим пример реализации обучения нейронной сети с учителем по правилу Хебба для операции логического «ИЛИ», используя биполярную кодировку двоичных сигналов (1, -1). Представим однослойную сеть в виде синоптических связей между нейронами (рис. 2.4).

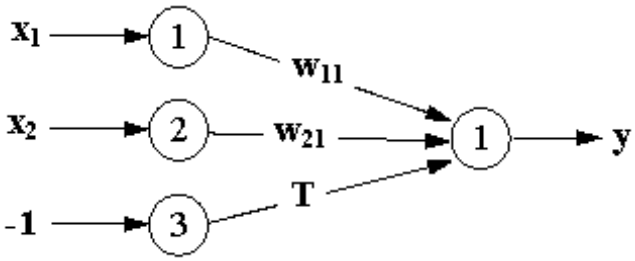

Рис. 2.4. Структурная схема обучаемого нейрона

Входное значение, подаваемое на дополнительный нейрон, равно -1, взвешенная сумма определяется как:

$$S = w_{11}\, x_1 + w_{21}\, x_2 - T$$

При функции активации

$$OUT = \begin{cases} 1, \text{если } NET > 0 \\ -1, \text{если } NET \leq 0 \end{cases} \quad (2.4)$$

Имеем следующую обучающую выборку (табл.2.1).

Таблица 2.1. *Истинность биполярной операций «ИЛИ»*

L	x_1	x_2	d
1	-1	-1	-1
2	1	-1	1
3	-1	1	1
4	1	1	1

Общее количество входных образов $L = 4$, подаваемых на нейронную сеть.

Считая $w_{11} = 0$, $w_{21} = 0$, $T = 0$ при $k = 0$ (число итераций). По 2, 3 и 4 образу нет соответствия.

Для четырех входных образов (см. таблицу 2.1) получаем последовательно следующие значения согласно формулам 2.2 и 2.3:

Для первого образа $k = 1, x_1 = -1, x_2 = -1, d = -1.$

$w_{11}(1) = w_{11}(0) + x_1 d = 0 + (-1) * (-1) = 1;$

$w_{21}(1) = w_{21}(0) + x_2 d = 0 + (-1) * (-1) = 1;$

$T(1) = T(0) - d = 0 - (-1) = 1.$

По 2 и 3 образу нет совпадения.

Переходим ко второму образу $k = 2, x_1 = 1, x_2 = -1, d = 1.$

$w_{11}(2) = w_{11}(1) + x_1 d = 1 + 1 * 1 = 2;$

$w_{21}(2) = w_{21}(1) + x_2 d = 1 + (-1) * 1 = 0;$

$T(2) = T(1) - d = 1 - 1 = 0.$

По 3 образу нет совпадения.

Переходим к третьему образу $k = 3, x_1 = -1, x_2 = 1, d = 1.$

$w_{11}(3) = w_{11}(2) + x_1 d = 2 + (-1) * 1 = 1;$

$w_{21}(3) = w_{21}(2) + x_2 d = 0 + 1 * 1 = 1;$

$T(3) = T(2) - d = 0 - 1 = -1.$

По 2 и 3 образу нет совпадения.

Переходим к четвертому образу $k = 4, x_1 = 1, x_2 = 1, d = 1.$

$w_{11}(4) = w_{11}(3) + x_1 d = 1 + 1 * 1 = 2;$

$w_{21}(4) = w_{21}(3) + x_2 d = 1 + 1 * 1 = 2;$

$T(4) = T(3) - d = -1 - 1 = -2.$

Совпадение по всем образам.

Обучение прекращается при подаче на вход нейронной сети всех входных образов. Разделительная линия будет следующая: $2x_1 + 2x_2 + 2 = 0 \rightarrow x_1 + x_2 = -1$ (рис. 2.5).

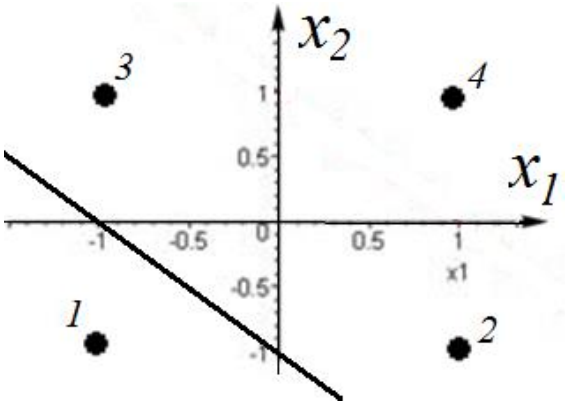

Рис. 2.5. Геометрическая интерпретация обученного нейрона для операции логической «ИЛИ»

Систематизируя правила Хебба, выделяют следующие модели модификации синаптических связей: хеббовскую, антихеббовскую и нехеббовскую. Следуя этой схеме, можно сказать, что синаптическая связь по модели Хебба усиливается при положительной корреляции предсинаптических и постсинаптических сигналов и ослабляется в противном случае. И, наоборот, согласно антихеббовской модели, синаптическая связь ослабляется при положительной корреляции предсинаптического и постсинаптического сигналов и усиливается в противном случае [2].

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

Нехеббовская модель не связана с механизмом модификации, предложенным Хеббом.

2.3. Обучение искусственной нейронной сети на основе дельта-правила

Дельта-правило основано на идее постоянного варьирования синаптических весов w_{ij} с целью уменьшения разности между значениями существующего $w_{ij}(t)$ и требуемого выходного сигнала нейрона $w_{ij}(t+1)$.

$$w_{ij}(t+1) = w_{ij}(t) + \alpha x_i e_j, \quad (2.5)$$

где w_{ij} – весовой коэффициент связи нейрона j и входа i; t – номер текущей итерации обучения; e_j – величина ошибки нейрона, определяется как разность между ожидаемым d_i и реальным значением y_i выходного сигнала нейрона; x_i – i-я компонента входного вектора; α – коэффициент скорость обучения [8].

Из формулы 2.5 видно, что дельта-правило является общей формой математической записи правила Хебба. Дельта-правило сводится к следующей последовательности действий [7]:

1. Инициализация весовых коэффициентов и порогов значениями, близкими к нулю.

2. Подача на вход нейронной сети очередного входного образа, взятого из обучающей выборки, и вычисление суммарного сигнала по всем входам для каждого нейрона ***j***:

$$S_j = \sum_{i=1}^{n} x_i w_{ij}$$

3. Вычисление значения выхода каждого нейрона:

$$OUT = \begin{cases} y = -1, \text{если } S_j > b_j \\ y = 1, \text{если } S_j \le b_j \end{cases}$$

4. Вычисление значения ошибки обучения для каждого нейрона

$$e_j = d_j - y_j$$

5. Проводится модификация весового коэффициента связи по формуле

$$w_{ij}(t+1) = w_{ij}(t) + \alpha x_i e_j$$

6. Повторение пунктов 2 – 5 до тех пор, пока ошибка сети e_j не станет меньше заданной $e_{зад}$.

$$e_j < e_{зад}$$

Рассмотрим пример, обучения нейрона с помощью дельта-правила для воспроизведения логической функции «И». Таблица истинности для биполярных сигналов как результат обучения нейросети представлен в таблице 2.2.

Таблица 2.2. *Истинность биполярной операций «И»*

L	x_1	x_2	d
1	1	1	1
2	1	−1	−1
3	−1	1	−1
4	−1	−1	−1

Общее количество входных образов $L = 4$, подаваемых на нейронную сеть. Функция активации нейрона согласно формуле 2.4. Структурная схема обучаемого нейрона представлена на рис. 2.4.

Задаём коэффициент обучения $\alpha = 1$, ошибка $e = 0$. Инициализируем работу нейронной сети: $w_{11} = 0$, $w_{21} = 0$, $w_0 = -1$ $T_0 = 0$.

Рассматриваем первый образ таблицы 2.2 $x_{11} = 1$, $x_{21} = 1$.

Определяем ошибку для первого образа.

$NET_1 = x_{11}w_{11} + x_{21}w_{21} - w_0T_0 = 1 * 0 + 1 * 0 - 1 * 0 = 0$

Тогда $y_1 = -1$, а должно быть $d_1 = 1$.

Вычисляем значение коэффициентов на каждом шаге:

$w_{11}(t+1) = w_{11}(t) + \alpha(d_1 - y_1)x_{11} = 0 + 1 * (1 - (-1)) * 1 = 2$

$w_{21}(t+1) = w_{21}(t) + \alpha(d_1 - y_1)x_{21} = 0 + 1 * (1 - (-1)) * 1 = 2$

$T(t+1) = T(t) + \alpha(d_1 - 1)x_0 = 0 + 1(1 - (-1)) - 1 = -2$

Второй образ равен: $x_{12} = 1$, $x_{22} = -1$.

Определяем ошибку для второго образа.

$NET_2 = x_{12}w_{11} + x_{22}w_{21} - w_0T(t+1) = 1 * 2 - 1 * 2 + 1 * 2 = 2$

Тогда $y_2 = 1$, а должно быть $d_2 = -1$.

Вычисляем значение коэффициентов на каждом шаге:

$w_{11}(t+2) = w_{11}(t+1) + \alpha(d_2 - y_2)x_{12} = 2 + 1 * (-1 - 1) * 1 = 0$

$w_{21}(t+2) = w_{21}(t+1) + \alpha(d_2 - y_2)x_{22} = 2 + 1 * (-1 - 1) * -1 = 4$

$T(t+2) = T(t+1) + \alpha(d_2 - y_2)x_0 = -2 + 1(-1 - 1) * -1 = 0$

Третий образ равен: $x_{13} = -1, x_{23} = 1.$

Определяем ошибку для третьего образа.

$NET_3 = x_{13}w_{11} + x_{22}w_{21} - w_0T(t+2) = 1*0+1*4-1*0 = 4$

Тогда $y_3 = 1$, а должно быть $d_3 = -1.$

Вычисляем значение коэффициентов на каждом шаге:

$w_{11}(t+3) = w_{11}(t+2) + \alpha(d_3 - y_3)x_{13} = 0 + 1*(-1-1)*-1 = 2$

$w_{21}(t+3) = w_{21}(t+2) + \alpha(d_3 - y_3)x_{23} = 4 + 1*(-1-1)*1 = 2$

$T(t+3) = T(t+2) + \alpha(d_3 - y_3)x_0 = 0 + 1(-1-1)*-1 = 2$

Четвертый образ равен: $x_{14} = -1, x_{24} = -1.$

Определяем ошибку для четвертого образа.

$NET_4 = x_{14}w_{11} + x_{24}w_{21} - w_0T(t+3) = -1*2-1*2-1*2 = -6$

Тогда $y_3 = d_3 = -1$. Ошибка$e = d_3 - y_3 = 0.$

Обучение закончено, разделительная линия равна уравнению $2x_1 + 2x_2 - 2 = 0 \rightarrow x_1 + x_2 = -1$ (рис. 2.6).

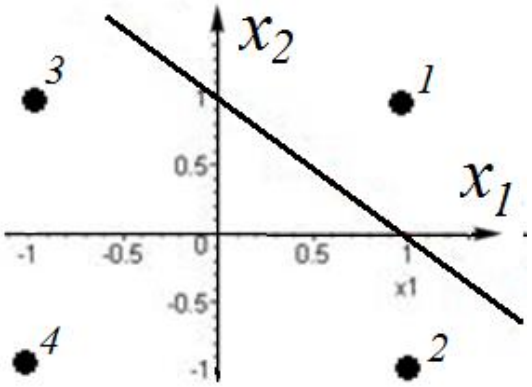

Рис. 2.6. Геометрическая интерпретация обученного нейрона для операции логической «И»

Обобщенное правило обучения «Дельта» известно также под названием Видроу-Хоффа, названному так в честь его создателей. Оно предполагает минимизацию среднеквадратичной ошибки ε нейронной сети, которая для I входных образов определяется по формуле:

$$\varepsilon = 0{,}5\sum_{i=1}^{I}(d_i - y_i)^2,$$

где d_i – требуемый (желаемый) выход -го нейрона, а y_i – выход, который получился в результате вычислений персептрона.

Квадратичная ошибка обучения персептрона ε зависит от того, какими являются весовые коэффициенты w_{ij}. Другими словами, ε является функцией от множества весовых коэффициентов: $\varepsilon = f(w_{ij})$. Если при рассмотрении правила Хебба персептрон обучают методом «поощрения—наказания», то теперь

задача обучения персептрона—это задача оптимизации (минимизации) функции-ошибки персептрона. При этом решение задачи базируется на методах безусловной оптимизации первого порядка, так называемых градиентных методах.

Напомним, что градиент функции представляет собой вектор, проекциями которого на оси координат являются частные производные от функции ε по этим координатам $\partial\varepsilon/\partial w_{ij}$. Градиент функции всегда направлен в сторону ее наибольшего возрастания [3].

Однако задача состоит в отыскании минимума функции, тогда надо опускаться по поверхности ошибок, что обеспечивается движением в сторону, антиградиента. Движение в сторону, противоположную градиенту, будет осуществляться, если на каждой эпохе к координатам текущей точки w_{ij} согласно формуле $w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}$. Тогда приращение величины, прямо пропорциональную частной производной по координате w_{ij}, взятую с противоположным знаком:

$$\Delta w_{ij} = -\eta \, {\partial\varepsilon}/{\partial w_{ij}} \tag{2.6}$$

где η – положительная константа влияющая на скорость обучения, обычно задаваемая в пределах от 0,05 до 1,5 [5].

Квадратичная ошибка ε является сложной функцией, зависящей от выходных сигналов персептрона y_i, которые, в свою очередь, зависят от w_{ij}, По правилу дифференцирования сложной функции

$${\partial\varepsilon}/{\partial w_{ij}} = {\partial\varepsilon}/{\partial y_i} \, {\partial y_i}/{\partial w_{ij}} \tag{2.7}$$

Выходные сигналы нейронов y_i вычисляются с помощью сигмоидных активационных функций $y_i = f_\sigma(S_j)$, аргументом которых являются суммы $S_j = \sum_{i=1}^{n} x_i \, w_{ij}$ [5]. Следовательно

$${\partial\varepsilon}/{\partial w_{ij}} = {\partial f_\sigma(S_j)}/{\partial S_i} \, {\partial S_i}/{\partial w_{ij}} = f_\sigma{}'(S_j)x_j \tag{2.8}$$

Выполним дифференцирование ε по y_i, получаем:

$${\partial\varepsilon}/{\partial w_{ij}} = -(d_i - y_i) \tag{2.9}$$

Подставив (2.8) и (2.9) в (2.7) и затем полученное выражение в (2.6), окончательно будем иметь

$$\Delta w_{ij} = -\eta\big(-(d_i - y_i) f_\sigma{}'(S_j) x_j\big) = \eta \, (d_i - y_i) f_\sigma{}'(S_j) x_j$$

Из этой формулы видно, что в качестве активационной функции при использовании обобщенного дельта-правила функция активации нейронов должна быть непрерывно дифференцируемой на всей оси абсцисс [5].

Таким образом, было рассмотрено обобщенное дельта-правило, которое по

сравнению с обычным дельта-правилом состоит в более быстрой сходимости, в возможности более точной обработки входных и выходных непрерывных сигналов т. е. в расширении круга решаемых персептронами задач. Продолжением обобщенного дельта-правила, является алгоритм обратного распространения ошибки. Однако следует заметить, что персептрон обученный по правилам Хебба или дельта может выдавать только бинарные результаты

2.4. Программная реализация нейронной сети для распознавания цифр

Для реализации распознавания цифр от 0 до 9 будем применять правило Вудроу-Хоффа (дельта-правило). Требуется обучить персептрон распознавать цифры по их изображениям. Эталонное изображение каждой цифры разбивается на сегменты с заданной степенью подробности. Если в пределы сегмента попадает часть изображения, то ему соответствует единичный сигнал, в противном случае – сигнал нулевой (рис.2.7) [1].

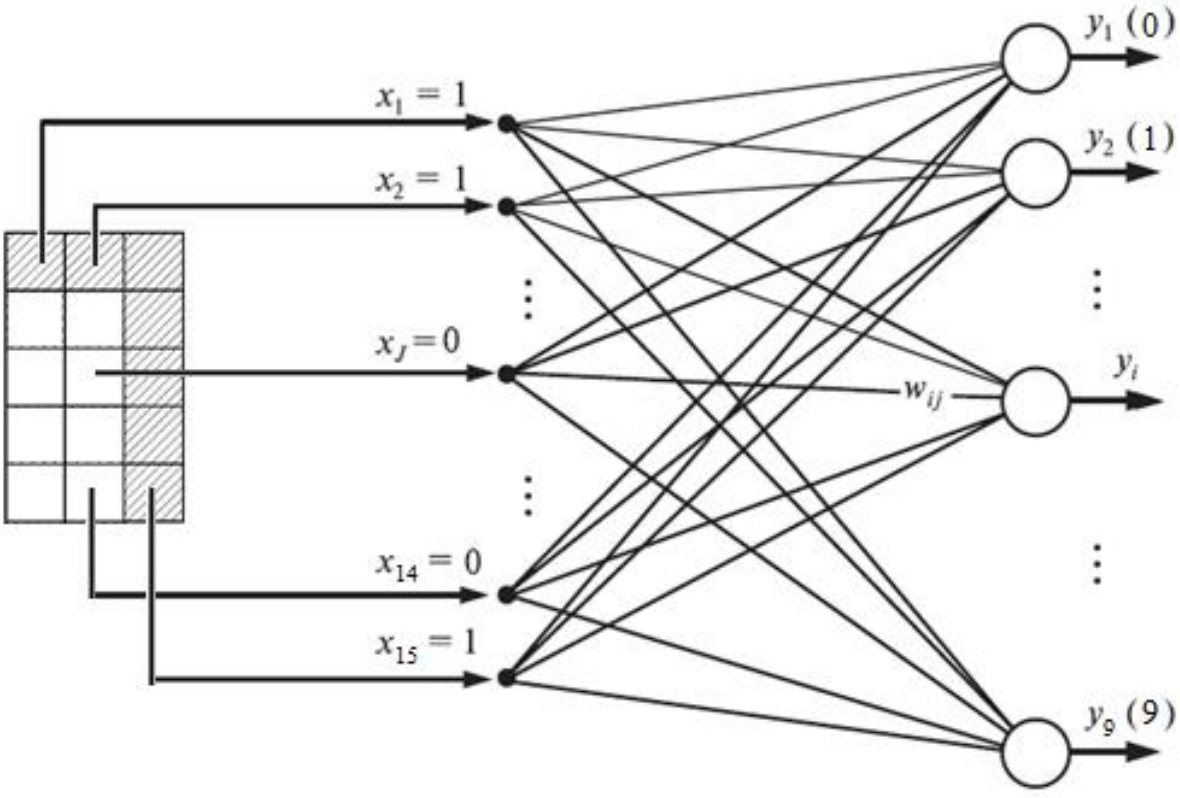

Рис. 2.7. Однослойный персептрон, предназначенный для распознавания цифр

Смысл использования персептрона заключается в том, что после обучения на его вход могут поступать уже не эталонные, а искаженные и зашумленные векторы. Реакция нейросети должна быть правильной до достижения определенного максимального уровня шумов.

Рассмотрим реализацию программного кода на языке java.

Листинг.2.1. *Neuron.java - класс, описывающий отдельный нейрон.* Включает в себя веса, количество входов, значение порога и смещения, а также методы по вычислению выхода нейрона и модификации весовых коэффициентов.

```
public class Neuron {
  private String name;  //название нейрона
  private int nInput; //количество входов
  private int weights[]; //веса
  int T; //смещение
  int b; //порог
//Выход из суммирующего блока для нейрона y
  private int S(int[] input) {
     int S = 0;
     for (int i = 0; i < nInput; i++) {
        S += weights[i] * input[i];
     }
     S -= T;
     return S;
  }
//Активационная функция
  private int OUT(int[] input) {
     return S(input) > b ? 1 : -1;
  }
//Результат
  public int y(int[] input) {
     return OUT(input);
  }
//изменение весов по дельта-правилу
  public void modification(int[] input, int e, int alpha) {
     for (int i = 0; i < nInput; i++) {
        weights[i] = weights[i] + alpha * input[i] * e;
     }
     T = T + (-1) * alpha * e;
  }
//инициализация нейрона
  public Neuron(String _name, int _n, int[] w, int _T, int _b){
     name = _name;
     nInput = _n;
```

```
      weights = w;
      T = _T;
      b = _b;
   }
}
```

Листинг.2.2. *DeltaRule.java - класс, реализующий сеть, состоящую из нейронов, и методы по ее обучению и нахождению выбранной цифры.*

```
public class DeltaRule{
   private Neuron[] neurons; //нейроны сети
   private int n; //количество нейронов
   private int[][] num = {  //цифры
         {1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1},
         {0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1},
         {1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1},
         {1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1},
         {1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1},
         {1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1},
         {1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1},
         {1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1},
         {1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1},
         {1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1}
   };
   private int alpha; //коэффициент обучения
   private int err; //заданная ошибка
//инициализация сети
   public DeltaRule(int _n, int _alpha, int _err) {
      n = _n;
      alpha = _alpha;
      err = _err;
      neurons = new Neuron[n];
      for (int i = 0; i < n; i++) {
         neurons[i] = new Neuron(String.valueOf(i), 15, new int[]{0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0}, 0, 0);
      }
   }
//вычисление текущей ошибки для i-го нейрона, на вход которому подается
цифра r
```

```
  private int e(int[] input, int i, int r) {
     int d = r == i ? 1 : -1; //результат, который должен быть на выходе
     return d - neurons[i].y(input); //начение ошибки
  }
//Распознать цифру: является ли n0 цифрой n?
  public boolean guess(int n0, int n) {
     return !(neurons[n].y(num[n0]) == -1);
  }
//Обучение
  public void learn() {
     int[] e = new int[10];
     for(int i=0; i<10; i++) e[i]=0;
     boolean f = false;
//Для каждой цифры
     for (int i = 0; i < 10; i++) {
        do {
           f = false;
           for(int j=0; j<10; j++) {
              e[j] = e(num[j], i, j); //текущая ошибка
              neurons[i].modification(num[j], e[j], alpha); //изменение весов
              if (e[j] > err) f = true;
           }
        } while (f); // пока текущая ошибка больше заданной
     }
  }
}
```

Листинг.2.3. *Реализация класса Main.java*

```
import java.util.Scanner;
public class Main {
  public static void main(String[] args) {
     Scanner in = new Scanner(System.in);
//создание нейронной сети
     DeltaRule d = new DeltaRule(10,1,0);
     d.learn(); //обучение сети по дельта-правилу
     System.out.print("1. Распознать число\n2. Выход\n\t>>");
     int ans = in.nextInt();
     while (ans != 2) {
```

```
            System.out.print("\nВаше число = ");
            ans = in.nextInt();
            for (int i = 0; i < 10; i++) {
               System.out.print("Это " + i + "? ");
               if (d.guess(i, ans)) System.out.println("Да!");
               else  System.out.println("Нет..");
            }
            System.out.print("\n\n1. Распознать число\n2. Выход\n\tПункт меню>>");
            ans = in.nextInt();
        }
    }
}
```

Результат работы программы представлен на рисунке 2.8.

```
Ваше число = 3
Это 0? Нет..
Это 1? Нет..
Это 2? Нет..
Это 3? Да!
Это 4? Нет..
Это 5? Нет..
Это 6? Нет..
Это 7? Нет..
Это 8? Нет..
Это 9? Нет..
```

Рис. 2.8. Результат работы программы распознавания цифр

Дальнейшее развитие идеи персептрона и алгоритмов обучения по дельта-правилу связано с усложнением его структуры и функциональных свойств. Например, может быть реализована программа распознавания букв русского алфавита. В отличие от предыдущей схемы, такой персептрон имеет 33 выходных нейрона: каждой букве алфавита соответствует свой выходной нейрон. Полагается, что сигнал первого выходного нейрона y_1 должен быть равен единице, если персептрону предъявлена буква «А», и равен нулю, если предъявляется любая другая буква. И так далее до буквы «Я».

3. МНОГОСЛОЙНАЯ НЕЙРОННАЯ СЕТЬ

Наиболее часто используемой архитектурой нейросети является многослойный персептрон (multilayer percetron – MLP), который представляет собой обобщение однослойного персептрона. Однако многослойные сети обладают большими возможностями, чем однослойные. Поэтому данная архитектура успешно применяется для решения разнообразных сложных задач. При этом обучение с учителем выполняется с помощью такого популярного алгоритма, как алгоритм обратного распространения ошибки [2].

3.1. Многослойный персептрон прямого распространения

Многослойная нейронная сеть состоит из множества входных узлов, которые образуют входной слой; одного или нескольких скрытых слоев вычислительных нейронов и одного выходного слоя (рис.3.1). Входной сигнал распространяется по сети в прямом направлении от слоя к слою [1].

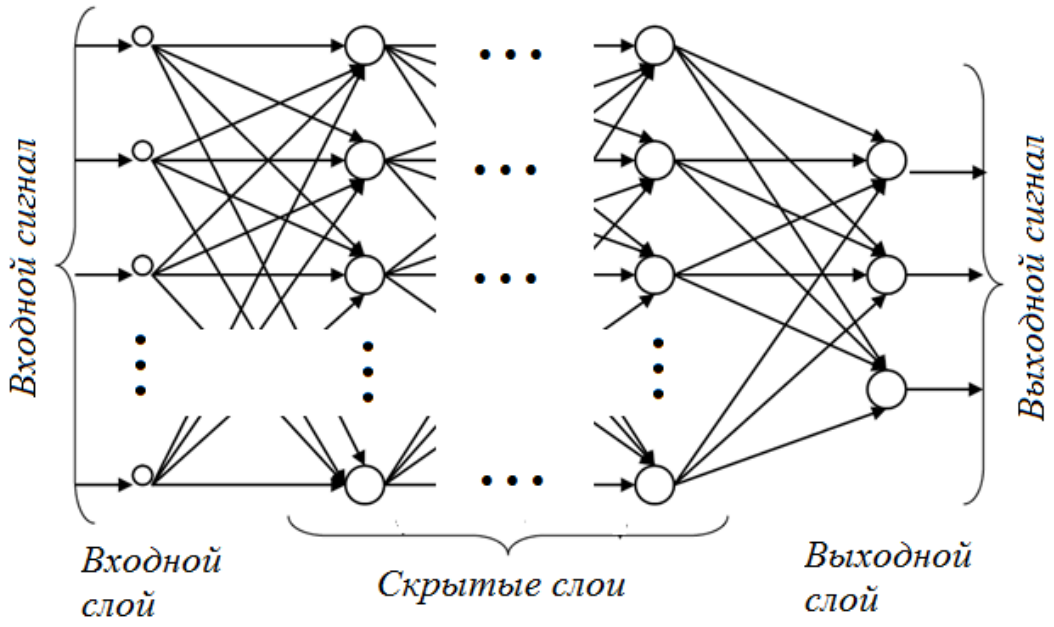

Рис. 3.1. Структура многослойного персептрона.

Таким образом, многослойная сеть содержит один или несколько слоев скрытых нейронов. Эти нейроны позволяют сети обучаться решению сложных задач, последовательно извлекая наиболее важные признаки из входного вектора. При этом данная структура сети обладает высокой степенью связности, реализуемой посредством синаптических соединений.

Необходимо отметить, что многослойные сети не могут привести к увеличению вычислительной мощности по сравнению с однослойной сетью, если активационная функция между слоями линейна**.** Поэтому каждый нейрон имеет нелинейную функцию активации. Соответственно, активационная функция

должна быть полностью дифференцируемой. Самой популярной функцией активации является сигмоидальная функция [9].

Если блок F (см. рис.1.2) сужает диапазон изменения величины NET так, что при любых значениях NET значения OUT принадлежат некоторому конечному интервалу, то F называется «сжимающей» функцией. В качестве «сжимающей» функции часто используется «сигмоидальная» (S-образная) функция, показанная на рис. 3.2.

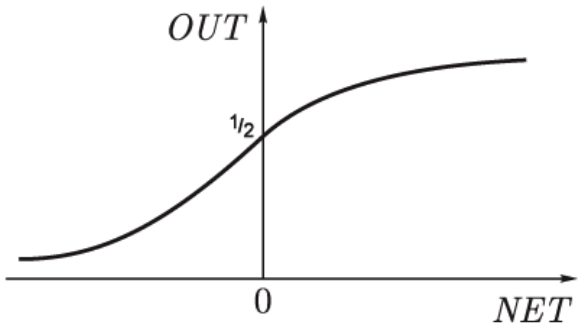

Рис. 3.2. Сигмоидальная функция

Сигмоидальная функция (сигмоид) математически выражается как

$$F(x) = {}^{1}/_{1 + e^{-x}}$$

Таким образом, в значениях NET и OUT будет:

$$OUT = {}^{1}/_{1 + e^{-NET}}$$

Одна из причин, по которой сигмоид используется в нейронных сетях, это простое выражение его производной через саму функцию (которое и позволило существенно сократить вычислительную сложность метода обратного распространения ошибки, сделав его применимым на практике).

$${}^{\partial OUT}/_{\partial NET} = OUT(1 - OUT)$$

Для слабых сигналов (т.е. когда OUT близко к нулю) кривая вход-выход имеет сильный наклон, дающий большое усиление. Когда величина сигнала становится больше, усиление падает. Таким образом, большие сигналы воспринимаются сетью без насыщения, а слабые сигналы проходят по сети без чрезмерного ослабления. Другими словами, центральная область логистической функции, имеющая большой коэффициент усиления, решает проблему обработки слабых сигналов, в то время как области с падающим усилением на положительном и отрицательном концах подходят для больших возбуждений [7].

Другой широко используемой активационной функцией является гиперболический тангенс. В качестве активационной функции искусственной нейронной сети она записывается следующим образом [7]:

$$OUT = th(x)$$

Подобно сигмоиду гиперболический тангенс является S-образной функцией, но он симметричен относительно начала координат, и в точке $NET = 0$ значение выходного сигнала OUT равно нулю (рис.3.3).

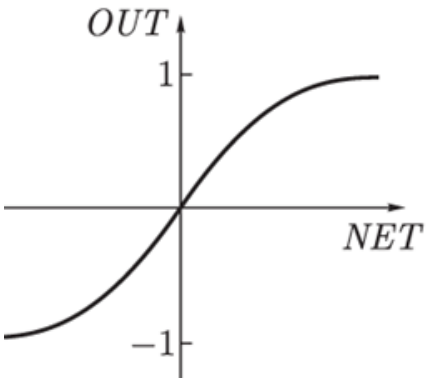

Рис. 3.3. Функция гиперболического тангенса

В отличие от сигмоидальной функции, гиперболический тангенс принимает значения различных знаков, и это его свойство применяется для целого ряда сетей.

3.2. Решение задачи XOR

Для того чтобы продемонстрировать ограниченные возможности однослойных персептронов при решении задач прибегают к рассмотрению так называемой задачи XOR – «исключающего ИЛИ» [2].

Напомним, что функция XOR от двух аргументов, каждый из которых может быть нулем или единицей. Она принимает значение единицы, когда один из аргументов равен единице, но не оба сразу (табл.3.1).

Таблица 3.1. *Истинность операций «XOR»*

L	x_1	x_2	d
1	0	0	0
2	1	0	1
3	0	1	1
4	1	1	0

Из таблицы 3.1 видно, что какие бы значения ни приписывались весам и порогу, сеть неспособна воспроизвести соотношение между входом и выходом, требуемое для представления функции XOR. Очевидно, что при использовании всего одного нейрона с двумя входами граница решений в пространстве входов будет линейной. Для всех точек, расположенных по одну сторону этой линии, выход нейрона будет равен единице, а для остальных точек – нулю [1].

Для решения этой задачи соединим структурные схемы нейронов для логических операций «И» (рис.1.5) и «ИЛИ» (рис.1.6). В результате получим структурную схему нейронной сети со скрытым слоем (рис.3.4).

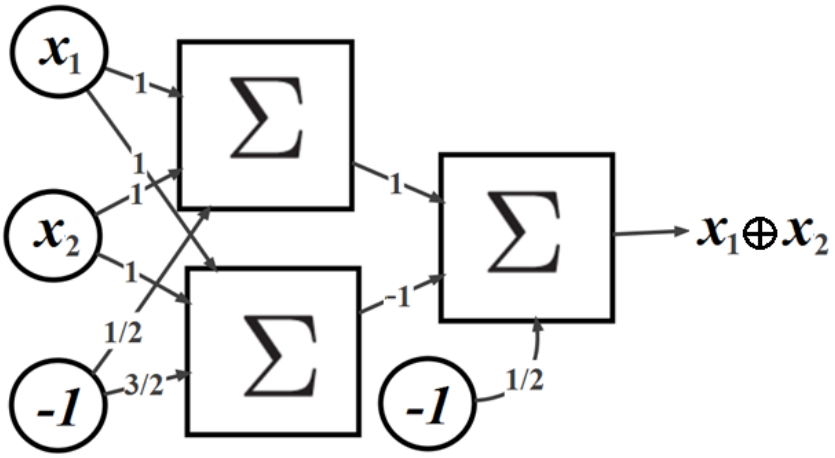

Рис. 3.4. Многослойный персептрон для решения задачи XOR.

Тогда верхний нейрон скрытого слоя будет характеризоваться операцией «ИЛИ», а нижний логической операцией «И». Наклоны гиперплоскостей для этих операций показаны на рис. 1.9.

Зная выходные значения верхнего и нижнего нейронов получаем:

$$NET_j = OUT_i^{\text{или}} * 1 - OUT_i^{\text{и}} * 1 - 1 * 0{,}5$$

Тогда

$NET_1 = -0{,}5 \rightarrow OUT_1 = 0;$

$NET_2 = 0{,}5 \rightarrow OUT_2 = 1;$

$NET_3 = 0{,}5 \rightarrow OUT_3 = 1;$

$NET_4 = -0{,}5 \rightarrow OUT_4 = 0;$

Это соответствует рис. 3.5.

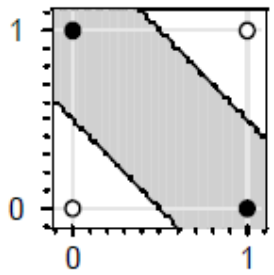

Рис. 3.5. Графическое решение задачи XOR.

Функцией выходного нейрона является построение линейной комбинации границ решений, сформированных двумя скрытыми нейронами. Верхний скрытый нейрон соединен возбуждающей (положительной) связью с выходным нейроном, в то время как нижний скрытый нейрон – тормозящей (отрицательной). Если оба скрытых нейрона заторможены (что соответствует входному образу (0, 0)), выходной нейрон также остается неактивным. Если оба скрытых нейрона возбуждены (что соответствует входному образу (1, 1)), выходной нейрон остается неактивным, так как тормозящее влияние большого отрицатель-

ного веса (нижний скрытый нейрон) преобладает над возбуждающим воздействием с меньшим весом (верхний скрытый нейрон). Когда нижний скрытый нейрон находится в заторможенном состоянии, а верхний – в возбужденном (что соответствует входным образам (0,1) и (1, 0)), выходной нейрон переходит в возбужденное состояние, так как возбуждающий сигнал с положительным весом приходит от верхнего скрытого нейрона.

3.3. Обучение методом обратного распространения ошибок

При обучении однослойного персептрона правильные выходные состояния нейронов, тогда настройка весов идет в направлении, минимизирующем ошибку на выходе сети. Очевидно, что в случае многослойной сети правильные значения имеются только для нейронов выходного слоя. Остальные как правило, не известны, многослойный персептрон уже невозможно обучить, руководствуясь только величинами ошибок на выходах нейронной сети [2].

Для обучения многослойной сети в 1986 году Д. Руммельхартом и Р. Хинтоном был предложен алгоритм обратного распространения ошибок (back propagation), который стал значимой вехой в развитии нейросетевых технологий. Идея метода состоит распространении сигналов ошибки от выходов нейросети к ее входам, в направлении, обратном прямому распространению сигналов в обычном режиме работы.

Таким образом, входные сигналы двигаются в прямом направлении, в результате реализуется выходной сигнал, из которого определяется значение ошибки. Величина ошибки двигается в обратном направлении, в результате происходит корректировка весовых коэффициентов связей сети.

Общая структура алгоритма опирается на обобщение дельта-правила с усложнением формул подстройки весов. Соответственно, а качестве активационной функции используется сигмоидальная активационная функция $OUT = 1/(1 + e^{-NET})$. Напомним, что производная этой функции равна: $\partial OUT/\partial NET = OUT(1 - OUT)$.

Рассмотрим алгоритм обратного распространения ошибки, который требует выполнения следующих шагов [3]:

Шаг 1. Выбрать очередную обучающую пару из обучающего множества; подать входной вектор на вход сети. Берется пример входного сигнала с соответствующим правильным значением выхода.

Шаг 2. Вычислить выход сети. При этом нейроны последовательно от слоя к слою функционируют по следующим формуле:

$$u_j = f\left(\sum_{i=1}^{n} x_i\, w_{ij}\right)$$

где $u_j = OUT_j$ – выходное значение рассматриваемого слоя j; $\sum_{i=1}^{n} x_i\, w_{ij} = NET_j$ – сумма входных сигналов слоя $\{x_1, x_2, \ldots, x_i\}$ и совокупность весов $\{w_{1j}, w_{2j}, \ldots, x_{ij}\}$; $f(NET)$ - сигмоидальная функция

Операции, выполняемые шагами 1 и 2, сходны с теми, которые выполняются при функционировании уже обученной сети, – подается входной вектор и вычисляется получающийся выход. Вычисления выполняются послойно. На шаги 1 и 2 можно смотреть как на «движение вперед», так как сигнал распространяется по сети от входа к выходу

Шаг 3. Рассчитывается прямое распространение ошибки через сеть. Берется, функционал суммарной квадратичной ошибки сети для одного выходного образа имеет вид:

$$E = 0{,}5 \sum_{n} (d_j - y_j)^2,$$

где $d_j = OUT^*$ – целевое значение выходного сигнала; $y_j = OUT$ – реальное значение выходного сигнала.

Шаг 4. Начиная с выходов, выполняется обратное движение через ячейки выходного и промежуточного слоя, при этом программа рассчитывает значения:

Для выходной ячейки:

$$\delta_0 = y_j(1 - y_j)(d_j - y_j),$$

где $y_j(1 - y_j) = OUT(1 - OUT)$ – производная от сигмоидальной активационной функции.

Выход нейрона слоя, вычитаемый из целевого значения, дает сигнал ошибки. Он умножается на производную сжимающей функции $y_j(1 - y_j)$, вычисленную для этого нейрона, давая, таким образом, величину обратной ошибки в узле.

Для скрытых ячеек

$$\delta_j = y_j(1 - y_j) \sum_{k} \omega_{kj}\, \delta_k,$$

где δ_j – ошибка элемента с индексом j; k – индекс, соответствующий слою, который посылает ошибку «обратно»;

$\sum_k \omega_{kj}\, \delta_k$ – обозначает все ячейки, связанные со скрытым слоем, ω_{ij} – заданный вектор веса в скрытом слое, δ_k – обратная ошибка от слоя, который ее посылает.

Шаг 5. Рассчитываем величину, на которую необходимо изменить значения весовых коэффициентов. При этом используется дельта-правило.

Тогда для весов соединений между скрытым слоем и выходом

$$w_{ij}^{*} = w_{ij} + \alpha\ \delta_0 f(U_i)$$

где i – номер нейрона в выходном слое;

j – номер нейрона в скрытом слое; α – коэффициент обучения; $f(U_i)$ – выходной сигнал; δ_0 – ошибка обучения на выходе.

Для весов соединений между скрытым слоем и входом

$$w_{jv}^{*} = w_{jv} + \alpha\ \delta_j f\left(U_j\right)$$

где j – номер нейрона в скрытом слое; v – номер нейрона в входном слое;

α – коэффициент обучения; $f\left(U_j\right)$ – сигнал от ячейки скрытого слоя; δ_j – ошибка обучения в скрытом слое.

Для весов смещений

$$w_{i0}^{*} = w_{i0} + \alpha\ \delta_0 f(U_0)$$

где i – номер нейрона; 0 – смещение; $f(U_0)$ – сигнал смещения δ_0 – ошибка обучения в смещении.

Шаги 4 и 5 составляют «обратный проход», здесь вычисляемый сигнал ошибки распространяется обратно по сети и используется для настройки весов

Шаг 6. Повторять шаги с 2 по 4 для каждого вектора обучающего множества до тех пор, пока ошибка на всем множестве не достигнет приемлемого уровня.

Весь смысл обратного распространения ошибки состоит в том, что для ее оценки для нейронов скрытых слоев можно принять взвешенную сумму ошибок последующего слоя. При этом коэффициент обучения минимизирует процент изменения, которое может произойти с весами. Хотя при небольшом коэффициенте процесс может занять больше времени, мы минимизируем возможность пропуска правильной комбинации весов. Если коэффициент обучения слишком велик, сеть может никогда не сойтись, то есть не будут найдены правильные веса связей.

О сходимости необходимо сделать несколько дополнительных замечаний. Во-первых, практика показывает, что сходимость метода обратного распространения весьма медленная. Невысокий темп сходимости является «генетической болезнью» всех градиентных методов, так как локальное направление градиента отнюдь не совпадает с направлением к минимуму. Во-вторых, подстройка весов выполняется независимо для каждой пары образов обучающей выборки. При этом улучшение функционирования на некоторой заданной паре может, вообще говоря, приводить к ухудшению работы на предыдущих образах. В этом смысле, нет достоверных (кроме весьма обширной практики применения метода) гарантий сходимости [10].

Рассмотрим пример функционирования сети (рис. 3.6) в процессе обучения.

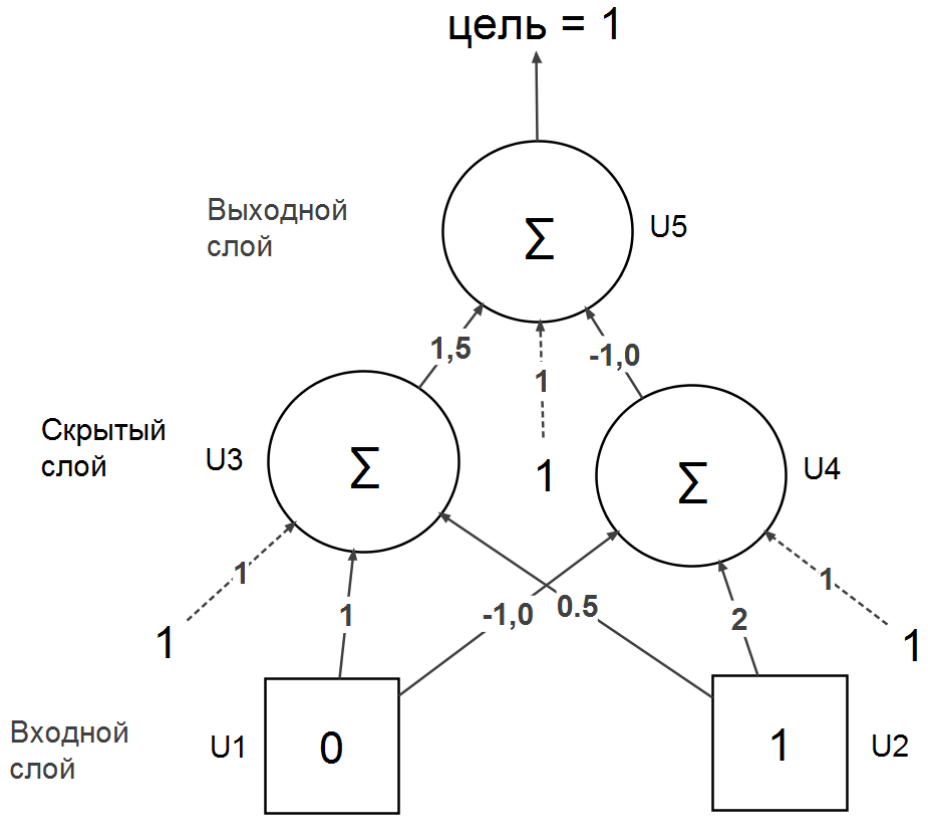

Рис. 3.6. Пример двухслойной сети.

Движение вперед.

Сначала выполняется расчет движения входного сигнала по сети [10]. Рассмотрим значения для скрытого слоя:

$U_3 = f(W_{31}U_1, + W_{32}U_2 + W_0X_0)$

U_3=f(l*0 + 0,5*l + l*l) = f(l,5)

f(x) – является активационной функцией, то есть сигмоидальной функцией

$$f(U_3) = {}^{1}/_{(1+e^{-NET})} = {}^{1}/_{(1+e^{-1,5})} = {}^{1}/_{(1+0,22313)} = 0,81757$$

$U_4 = f(W_{41}U_1, + W_{42}U_2 + W_0X_0)$

U_4=f(-l*0 + 2*l + l*l) = f(3)

Тогда $f(U_4) = {}^{1}/_{(1+e^{-3})} = {}^{1}/_{(1+0,04978)} = 0,95274$

Теперь сигнал дошел до скрытого слоя. Конечный шаг заключается в том, чтобы переместить сигнал из скрытого слоя в выходной слой и рассчитать значение на выходе из сети:

$U_5 = f(W_{51}U_1, + W_{52}U_2 + W_0X_0)$

U_5= f(l,5*0,81757 + (-1,0*0,952574) + l*l) = f(l,2195)

Тогда $f(U_5) = 0,78139$

Правильной реакцией нейронной сети на тестовый входной сигнал является 1,0 (цель). Однако выходное значение, рассчитанное сетью, составляет $OUT = 0{,}78139$. Это не так уж и плохо, но можно уменьшить значение ошибки, применив для сети алгоритм обратного распространения ошибки.

Для коррекции весовых коэффициентов в сети используется суммарная квадратичная ошибка. Поэтому ошибка составляет:

$$E = 0{,}5\sum_{n}(d_j - y_j)^2 = 0{,}5(1 - 0{,}78139)^2 = 0{,}023895$$

Алгоритм обратного распространения для ошибки

Применим обратное распространение, начиная с определения ошибки в выходном и скрытых узлах [10].

Рассчитаем ошибку в выходном узле.

$\delta_5 = y_j(1 - y_j)(d_j - y_j) = 0{,}78139\,(1 - 0{,}78139\,)(1 - 0{,}78139\,)$;

$\delta_5 = 0{,}0373$

Теперь следует рассчитать ошибку для двух скрытых узлов.

$\delta_{U3} = OUT(1 - OUT)(\omega_{53}\delta_5) = f(U_3)\big(1 - f(U_3)\big)(\omega_{53}\delta_5) =$

$= 0{,}81757(1 - 0{,}81757)(1{,}5 * 0{,}0373) = 0{,}0083449$

$\delta_{U4} = OUT(1 - OUT)(\omega_{54}\delta_5) = f(U_4)\big(1 - f(U_4)\big)(\omega_{54}\delta_5) =$

$= 0{,}95274(1 - 0{,}95274)(-1{,}0 * 0{,}0373) = -0{,}0016851$

Когда рассчитаны значения ошибок для выходного и скрытого слоев, можно с помощью уравнений дельта-правил изменить веса. Используем коэффициент обучения (α), равный 0,5.

Сначала следует обновить веса для соединений между выходным и скрытым слоями:

$$\mathrm{w}_{\mathrm{ij}}^{*} = \mathrm{w}_{\mathrm{ij}} + \alpha\,\delta_0\,f(U_i)$$

Рассматриваем связь между выходом U_5 и элементом скрытого слоя U_4.

Тогда $\mathrm{w}_{54} = -1$, $\delta_5 = 0{,}0373$, $f(U_4) = 0{,}95274$

$\mathrm{w}_{54}^{*} = \mathrm{w}_{54} + \alpha\,\delta_5 f(U_4) = -1 + \big(0{,}5 * 0{,}0373 * 0{,}95274\ \big) = -0{,}9882$

Рассматриваем связь между выходом U_5 и элементом скрытого слоя U_3.

Тогда $\mathrm{w}_{53} = 1{,}5$, $\delta_5 = 0{,}0373$, $f(U_3) = 0{,}81757$

$\mathrm{w}_{53}^{*} = \mathrm{w}_{53} + \alpha\,\delta_5 f(U_3) = 1{,}5 + \big(0{,}5 * 0{,}0373 * 0{,}81757\ \big) = 1{,}51525$

Теперь нужно обновить смещение для выходной ячейки U_5:

$$\mathrm{w}_{\mathrm{ij}}^{*} = \mathrm{w}_{\mathrm{ij}} + \alpha\,\delta_0\,f(U_0)$$

$\mathrm{w}_{50}^{*} = \mathrm{w}_{50} + \alpha\,\delta_5 f(U_{50}) = 1 + \big(0{,}5 * 0{,}0373 * 1\ \big) = 1{,}01865$

Таким образом, для w_{54}^{*} и w_{53}^{*} вес увеличен. Смещение, было обновлено для повышения возбуждения.

Теперь нужно показать изменение весов для скрытого слоя (для входа к скрытым ячейкам)

$$w_{ij}^{*} = w_{ij} + \alpha\,\delta_i f(U_i)$$

Рассматриваем связь между входом U_2 и элементом скрытого слоя U_4.

Тогда $\mathrm{w}_{42} = 2$, $\delta_{U4} = -0{,}0016851$, $f(U_2) = 1$

$$\mathrm{w}_{42}^{*} = \mathrm{w}_{42} + \alpha\,\delta_{U4} f(U_2) = 2 + (0{,}5 * -0{,}0016851 * 1) = 1{,}99916$$

Рассматриваем связь между входом U_1 и элементом скрытого слоя U_4.

Тогда $\mathrm{w}_{41} = -1$, $\delta_{U4} = -0{,}0016851$, $f(U_1) = 0$

$$\mathrm{w}_{41}^{*} = \mathrm{w}_{41} + \alpha\,\delta_{U4} f(U_1) = -1 + (0{,}5 * -0{,}0016851 * 0) = \mathbf{-1{,}0}$$

Рассматриваем связь между входом U_2 и элементом скрытого слоя U_3.

Тогда $\mathrm{w}_{32} = 0{,}5$, $\delta_{U3} = 0{,}0083449$, $f(U_2) = 1$

$$\mathrm{w}_{32}^{*} = \mathrm{w}_{32} + \alpha\,\delta_{U3} f(U_2) = 0{,}5 + (0{,}5 * 0{,}0083449 * 1) = 0{,}50417$$

Рассматриваем связь между входом U_1 и элементом скрытого слоя U_3.

Тогда $\mathrm{w}_{31} = 1$, $\delta_{U3} = 0{,}0083449$, $f(U_1) = 0$

$$\mathrm{w}_{31}^{*} = \mathrm{w}_{31} + \alpha\,\delta_{U3} f(U_1) = 1 + (0{,}5 * 0{,}0083449 * 0) = 1{,}0$$

Последний шаг - это обновление смещений для ячеек:

$$\mathrm{w}_{\mathrm{ij}}^{*} = \mathrm{w}_{\mathrm{ij}} + \alpha\,\delta_0\, f(U_0)$$

$$\mathrm{w}_{40}^{*} = \mathrm{w}_{40} + \alpha\,\delta_{U4} f(U_0) = 1 + \left(0{,}5 * -0{,}0016851 * 1\right) = 0{,}99915$$

$$\mathrm{w}_{30}^{*} = \mathrm{w}_{30} + \alpha\,\delta_{U3} f(U_0) = 1 + \left(0{,}5 * 0{,}0083449 * 1\right) = 1{,}00417$$

Обновление весов для выбранного обучающего сигнала завершается. Проверим, что алгоритм действительно уменьшает ошибку на выходе, введя тот же обучающий сигнал еще раз:

$$\mathrm{U}_3^{*} = \mathrm{f}(\mathrm{w}_{31}^{*}\mathrm{U}_1 + \mathrm{w}_{32}^{*}\mathrm{U}_2 + \mathrm{w}_{30}^{*}\,\mathrm{X}_0)$$
$$= \mathrm{f}(\mathrm{l}, 0 * 0 + 0{,}50417 * \mathrm{l} + 1{,}00417 * \mathrm{l}) = \mathrm{f}(\mathrm{l}, 50834)$$
$$\mathrm{U}_3^{*} = {}^{1}\!/_{(1 + e^{-1{,}50834})} = 0{,}8188$$
$$\mathrm{U}_4^{*} = \mathrm{f}(\mathrm{w}_{41}^{*}\mathrm{U}_1 + \mathrm{w}_{42}^{*}\mathrm{U}_2 + \mathrm{w}_{40}^{*}\,\mathrm{X}_0)$$
$$= \mathrm{f}(-\mathrm{l}, 0 * 0 + 1{,}99916 * \mathrm{l} + 0{,}99915 * \mathrm{l}) = \mathrm{f}(2{,}99831)$$
$$\mathrm{U}_4^{*} = {}^{1}\!/_{(1 + e^{-2{,}99831})} = 0{,}952497$$
$$\mathrm{U}_5^{*} = \mathrm{f}(\mathrm{w}_{53}^{*}\mathrm{U}_3^{*} + \mathrm{w}_{54}^{*}\mathrm{U}_4^{*} + \mathrm{w}_{50}^{*}\,\mathrm{X}_0)$$
$$= \mathrm{f}(1{,}51525 * 0{,}8188 - 0{,}9882 * 0{,}952497 + 1{,}01865 * \mathrm{l})$$
$$= \mathrm{f}(1{,}32379)$$
$$\mathrm{U}_5^{*} = {}^{1}\!/_{(1 + e^{-1{,}32379})} = 0{,}7898$$

Ошибка составляет:

$$E = 0{,}5\sum_{n}(d_j - y_j)^2 = 0{,}5(1 - 0{,}7898)^2 = 0{,}022$$

Начальная ошибка была равна *0,023895*. Текущая ошибка составляет *0,022*, а это значит, что одна итерация алгоритма обратного распространения позволила уменьшить среднюю ошибку на *0,001895*.

3.4. Программная реализация многослойной нейронной сети

Алгоритм обратного распространения применяется при создании нейронной сети для персонажей компьютерных игр [11]. Задействуем нейронную сеть, чтобы выбрать действие из доступного списка на основании того, в какой окружающей среде находится персонаж игры. Для решения этой задачи необходимо обучить нейронную сеть на ограниченном количестве примеров (то есть образцов поведения в зависимости от обстановки), а затем позволить ей самостоятельно генерировать поведение во всех прочих ситуациях [10].

Для программной реализации будет использоваться многослойная нейронная сеть, построенная по принципу «победитель получает все». Такие архитектуры полезны в том случае, если выходы должны быть разделены на несколько классов (рис.3.7).

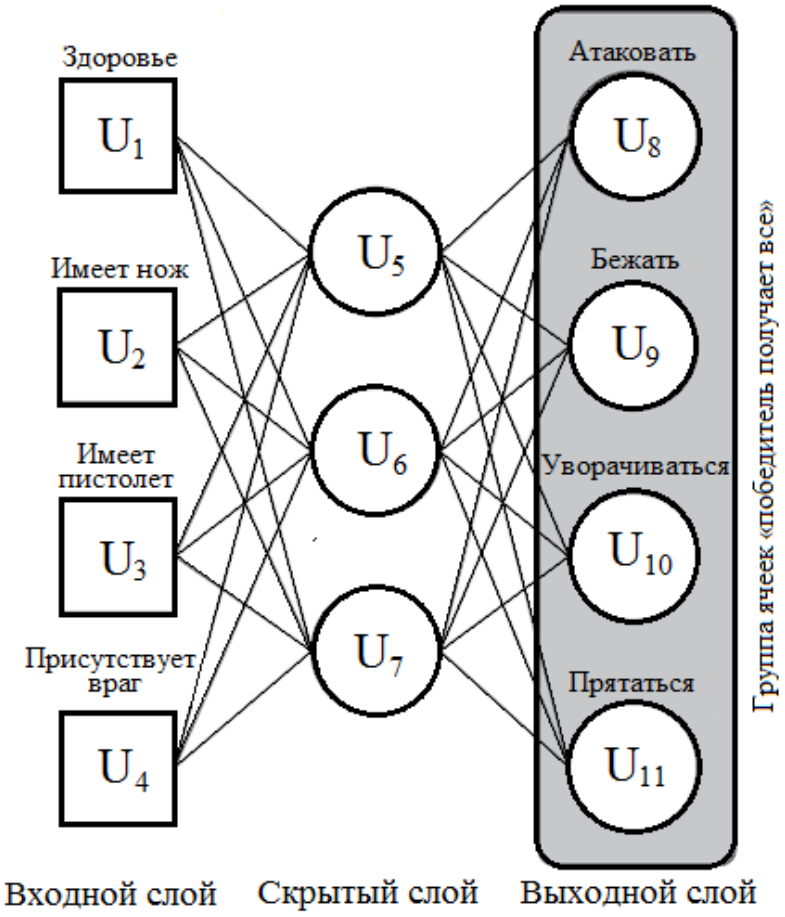

Рис. 3.6. Структурная схема для многослойной сети «победитель получает все».

В сети, созданной по принципу «победитель получает все», выходная ячейка с большей суммой весов является «победителем» группы и допускается к действию. В рассматриваемом приложении каждая ячейка представляет определенное поведение, которое доступно для персонажа в игре. В качестве примеров поведения можно назвать такие действия, как выстрелить из оружия, убежать, уклониться и др. Срабатывание ячейки в группе по принципу «победитель получает все» приводит к тому, что агент выполняет определенное действие. Когда агенту вновь позволяется оценить окружающую среду, процесс повторяется. На рис. 3.6 представлена сеть, которая использовалась для тестирования архитектуры и метода выбора действия. Четыре входа обозначают «здоровье персонажа» (0 – плохое, 2 – хорошее), «имеет нож» (1, если персонаж имеет нож, 0 – в противном случае), «имеет пистолет» (1, если персонаж имеет пистолет, 0 – в противном случае) и «присутствует враг» (количество врагов в поле зрения). Выходы определяют поведение, которое выберет персонаж. Действие «атаковать» приводит к тому, что персонаж атакует врагов в поле зрения, «бежать» вынуждает персонаж убегать, «уворачиваться» приводит к произвольному движению персонажа, а «прятаться» вынуждает персонаж искать укрытие. Это высокоуровневые образы поведения, и предполагается, что подсистема поведения будет выбирать действие и следовать ему [10].

Обучение нейросети состоит в предоставлении обучающих примеров из небольшой группы желательных действий. Затем следует выполнение алгоритма обратного распространения с учетом желаемого результата и действительного результата. Например, если персонаж имеет пистолет, здоров и видит одного врага, желаемое действие – атаковать. Однако если персонаж здоров, имеет нож, но видит двух врагов, то правильное действие – спрятаться [11].

Сеть должна рассчитывать реакцию на входы и выполнять действие, которое будет похожим на обучающие сценарии. Примеры, которые использовались для обучения сети, представлены в табл. 3.2.

Таблица 3.2. Примеры для обучения нейронной сети

Здоровье	Имеет нож	Имеет пистолет	Враги	Поведение
2	0	0	0	Уворачиваться
2	0	0	1	Уворачиваться
2	0	1	1	Атаковать
2	0	1	2	Атаковать
2	1	0	2	Прятаться
2	1	0	1	Атаковать
1	0	0	0	Уворачиваться

1	0	0	1	Прятаться
1	0	1	1	Атаковать
1	0	1	2	Прятаться
1	1	0	2	Прятаться
1	1	0	1	Прятаться
0	0	0	0	Уворачиваться
0	0	0	1	Прятаться
0	0	1	1	Прятаться
0	0	1	2	Бежать
0	1	0	2	Бежать
0	1	0	1	Прятаться

Примечание: здоровье: хорошее – 2, не полностью здоров – 1, плохое – 0; имеет нож: да – 1, нет – 0; имеет пистолет: да – 1, нет – 0; враги – имеется в виду их количество врагов в приделах видемости от 0 до 2.

Данные, приведенные в табл. 3.2, должны быть переданы сети в произвольном порядке во время обучения с помощью алгоритма обратного распространения. Если предложить нейронной сети сценарий, в котором персонаж имеет хорошее здоровье, владеет оружием одного вида, например, ножом и видит двух врагов (то есть 2:1:0:2), то нейросеть выберет действие «прятаться». В большинстве случаев сеть успешно проходит обучение на всех представленных примерах. При увеличении количества скрытых ячеек обучение проходит идеально. Однако в этом случае для работы нейронной сети требуется намного больше компьютерных ресурсов. Поэтому были выбраны три скрытые ячейки с несколькими циклами обучения [10].

Рассмотрим исходный код на языке программирования java для реализации обучения нейронной сети на основе алгоритма обратного распространения ошибки и ее тестирования.

Глобальные константы и переменные показаны в листинге 3.1.

Листинг 3.1. *Глобальные константы и переменные для нейронной сети и алгоритма обратного распространения*

```
public final class Const {
   public static final int INPUT_NEURONS = 4;
   public static final int HIDDEN_NEURONS = 3;
   public static final int OUTPUT_NEURONS = 4;
   public static final double LEARN_RATE = 0.2; // Коэффициент обучения
   public static final int MAX_SAMPLES = 18;
}
```

```
// Веса
   // Смещение последняя строка
   // Вход скрытых ячеек (со смещением)
   double wih[][] = new double[Const.INPUT_NEURONS + 1]
[Const.HIDDEN_NEURONS];
   // Вход выходных ячеек (со смещением)
   double who[][] = new double[Const.HIDDEN_NEURONS + 1]
[Const.OUTPUT_NEURONS];
   // Активаторы
   // Массив inputs определяет значение входных ячеек
   double inputs[] = new double[Const.INPUT_NEURONS];
   // Массив hidden содержит выход для скрытых ячеек
   double hidden[] = new double[Const.HIDDEN_NEURONS];
   // Массив target предоставляет желаемое значение сети для заданных входов
   double target[] = new double[Const.OUTPUT_NEURONS];
   // Массив actual отображает реальный результат работы сети
   double actual[] = new double[Const.OUTPUT_NEURONS];
   // Ошибки
   // Массив erro хранит ошибку для каждой входной ячейки
   double erro[] = new double[Const.OUTPUT_NEURONS];
   // Массив errh содержит ошибки скрытых ячеек
   double errh[] = new double[Const.HIDDEN_NEURONS];
```

Веса определяются как веса соединений между входным и скрытым (wih), а также между скрытым и выходным слоями (who). Вес соединения между U_5 и U_1 (рис. 3.6) является весом входа в скрытый слой, представленный wih[0][0] (так как U_1 – это первая входная ячейка, а U_5 – первая скрытая ячейка, начиная с нуля). Данный вес обозначается как $w_{5,1}$. Вес $w_{11,7}$ (соединение между ячейкой U_{11} выходного слоя и U_7 скрытого слоя) равен who[2][3]. Веса смещения занимают последнюю строку в каждой таблице и идентифицируются с помощью значения +1 в массивах wih и who.

Значения сигналов хранятся в четырех массивах. Массив inputs определяет значение входных ячеек, массив hidden содержит выход для скрытых ячеек, массив target предоставляет желаемое значение сети для заданных входов, а массив actual отображает реальный результат работы сети.

Ошибки сети предоставляются в двух массивах. Массив erro хранит ошибку для каждой входной ячейки. Массив errh содержит ошибки скрытых ячеек.

Веса произвольно выбираются в диапазоне (от –0,5 до 0,5). Коэффициент обучения задается как 0,2. Диапазон весов и коэффициент обучения могут быть изменены в зависимости от проблемы и требуемой точности решения.

Существуют три вспомогательные функции, которые используются, чтобы задавать произвольные веса для сети, а также для работы алгоритма. Они приведены в листинге 3.2.

Листинг 3.2. *Вспомогательные функции для алгоритма обратного распространения*

```
//заполнение матриц весов
   void assignRandomWeights() {
      int hid, inp, out;
      for (inp = 0; inp < Const.INPUT_NEURONS + 1; inp++) {
         for (hid = 0; hid < Const.HIDDEN_NEURONS; hid++) {
            // Произвольный выбор весов в диапазоне от -0.5 до 0.5
            wih[inp][hid] = Math.random() - 0.5;
         }
      }
      for (hid = 0; hid < Const.HIDDEN_NEURONS + 1; hid++) {
         for (out = 0; out < Const.OUTPUT_NEURONS; out++) {
            who[hid][out] = Math.random() - 0.5;
         }
      }
   }
   //Сигмоида
   double sigmoid(double val) {
      return (1.0 / (1.0 + Math.exp(-val)));
   }
   //Сигмоида обратная
   double sigmoidDerivative(double val) {
      return (val * (1.0 - val));
   }
```

Функция assignRandomWeights произвольно задает вес для всех соединений сети (включая все смещения). Функция sigmoid определяет значение функции сжатия (сигмоида), которая используется при прямом вычислении. Функция sigmoidDerivative устанавливает значение производной функции sigmoid и используется при обратном распространении ошибки.

Следующая функция реализует фазу прямого вычисления алгоритма, которая представлена в листинге 3.3.

Листинг 3.3. Алгоритм прямого распространения

```
   //Алгоритм прямого распространения
   void feedForward() {
```

```
    int inp, hid, out;
    double sum;
    /* Вычислить вход в скрытый слой */
    for (hid = 0; hid < Const.HIDDEN_NEURONS; hid++) {
       sum = 0.0;
       for (inp = 0; inp < Const.INPUT_NEURONS; inp++) {
          sum += inputs[inp] * wih[inp][hid];
       }
       /* Добавить смещение */
       sum += wih[Const.INPUT_NEURONS][hid];
       hidden[hid] = sigmoid(sum);
    }
    /* Вычислить вход в выходной слой */
    for (out = 0; out < Const.OUTPUT_NEURONS; out++) {
       sum = 0.0;
       //Скрытые нейроны
       for (hid = 0; hid < Const.HIDDEN_NEURONS; hid++) {
          sum += hidden[hid] * who[hid][out];
       }
       /* Добавить смещение */
       sum += who[Const.HIDDEN_NEURONS][out];
       actual[out] = sigmoid(sum);
    }
 }
```

Алгоритм прямого распространения начинает свою работу с расчета активации для скрытых слоев с учетом входов из входного слоя. Смещение добавляется в ячейку до вычисления функции сигмоида. Затем аналогичным образом рассчитывается выходной слой. Обратите внимание, что сеть может иметь одну или несколько выходных ячеек. Поэтому обработка выходных ячеек помещена в цикл, чтобы рассчитать все необходимые активации на выходе. Алгоритм обратного распространения показан в листинге 3.4.

Листинг 3.4. *Алгоритм обратного распространения*

```
 //Алгоритм обратного распространения
 void backPropagate() {
    int inp, hid, out;
    /* Вычислить ошибку выходного слоя (шаг 3 для выходных ячеек) */
    for (out = 0; out < Const.OUTPUT_NEURONS; out++) {
```

```
            erro[out] = (target[out] - actual[out]) * sigmoidDerivative(actual[out]);
        }
        /* Вычислить ошибку скрытого слоя (шаг 3 для скрытого слоя) */
        for (hid = 0; hid < Const.HIDDEN_NEURONS; hid++) {
            errh[hid] = 0.0;
            for (out = 0; out < Const.OUTPUT_NEURONS; out++) {
                errh[hid] += erro[out] * who[hid][out];
            }
            errh[hid] *= sigmoidDerivative(hidden[hid]);
        }
        /* Обновить веса для выходного слоя (шаг 4 для выходных ячеек) */
        for (out = 0; out < Const.OUTPUT_NEURONS; out++) {
            for (hid = 0; hid < Const.HIDDEN_NEURONS; hid++) {
                who[hid][out] += (Const.LEARN_RATE * erro[out] * hidden[hid]);
            }
            /* Обновить смещение */
            who[Const.HIDDEN_NEURONS][out] += (Const.LEARN_RATE *
erro[out]);
        }
        /* Обновить веса для скрытого слоя (шаг 4 для скрытого слоя) */
        for (hid = 0; hid < Const.HIDDEN_NEURONS; hid++) {
            for (inp = 0; inp < Const.INPUT_NEURONS; inp++) {
                wih[inp][hid] += (Const.LEARN_RATE * errh[hid] * inputs[inp]);
            }
            /* Обновить смещение */
            wih[Const.INPUT_NEURONS][hid] += (Const.LEARN_RATE * errh[hid]);
        }
    }
```

Данная функция следует алгоритму, описанному в разделе «Обучение методом обратного распространения ошибок». Сначала рассчитывается ошибка выходной ячейки (или ячеек) с использованием действительного и желаемого результатов. Далее определяются ошибки для скрытых ячеек. Наконец, обновляются веса для всех соединений в зависимости от того, находятся они во входном или выходном скрытом слое. Здесь важно отметить, что в алгоритме не рассчитывается вес для смещения, а просто применяется ошибка к самому смещению. В алгоритме прямого распространения смещение добавляется без использования для него веса.

Данный алгоритм позволяет рассчитать выход многослойной нейронной сети и ее изменения с помощью алгоритма обратного распространения. Рассмотрим исходный код примера, в котором этот алгоритм используется для реализации нейросети.

В листинге 3.5 приведена структура, задачей которой является отображение примеров для обучения. Структура задает входы (здоровье, нож, пистолет, враг), а также значение желаемого результата (массив out) и включает инициализированные данные для обучения сети.

Листинг 3.5. *Отображение данных для обучения нейронной сети.*

```
public class Element {
   double health;
   double knife;
   double gun;
   double enemy;
   double out[];

   Element(double health, double knife, double gun, double enemy, double out[]){
      this.health = health;
      this.knife = knife;
      this.gun = gun;
      this.enemy = enemy;
      this.out = new double[4];
      for(int i = 0; i < out.length && i < this.out.length; i++) {
         this.out[i] = out[i];
      }
   }
}
Element samples[];
   public void setSamples(){
      samples = new Element[Const.MAX_SAMPLES];
      samples[0] = new Element(2.0, 0.0, 0.0, 0.0, new double[] { 0.0, 0.0, 1.0, 0.0 });
      samples[1] = new Element(2.0, 0.0, 0.0, 1.0, new double[] { 0.0, 0.0, 1.0, 0.0 });
      samples[2] = new Element(2.0, 0.0, 1.0, 1.0, new double[] { 1.0, 0.0, 0.0, 0.0 });
      samples[3] = new Element(2.0, 0.0, 1.0, 2.0, new double[] { 1.0, 0.0, 0.0, 0.0 });
      samples[4] = new Element(2.0, 1.0, 0.0, 2.0, new double[] { 0.0, 0.0, 0.0, 1.0 });
      samples[5] = new Element(2.0, 1.0, 0.0, 1.0, new double[] { 1.0, 0.0, 0.0, 0.0 });
      samples[6] = new Element(1.0, 0.0, 0.0, 0.0, new double[] { 0.0, 0.0, 1.0, 0.0 });
```

```
    samples[7] = new Element(1.0, 0.0, 0.0, 1.0, new double[] { 0.0, 0.0, 0.0, 1.0 });
    samples[8] = new Element(1.0, 0.0, 1.0, 1.0, new double[] { 1.0, 0.0, 0.0, 0.0 });
    samples[9] = new Element(1.0, 0.0, 1.0, 2.0, new double[] { 0.0, 0.0, 0.0, 1.0 });
    samples[10] = new Element(1.0, 1.0, 0.0, 2.0, new double[] { 0.0, 0.0, 0.0, 1.0 });
    samples[11] = new Element(1.0, 1.0, 0.0, 1.0, new double[] { 0.0, 0.0, 0.0, 1.0 });
    samples[12] = new Element(0.0, 0.0, 0.0, 0.0, new double[] { 0.0, 0.0, 1.0, 0.0 });
    samples[13] = new Element(0.0, 0.0, 0.0, 1.0, new double[] { 0.0, 0.0, 0.0, 1.0 });
    samples[14] = new Element(0.0, 0.0, 1.0, 1.0, new double[] { 0.0, 0.0, 0.0, 1.0 });
    samples[15] = new Element(0.0, 0.0, 1.0, 2.0, new double[] { 0.0, 1.0, 0.0, 0.0 });
    samples[16] = new Element(0.0, 1.0, 0.0, 2.0, new double[] { 0.0, 1.0, 0.0, 0.0 });
    samples[17] = new Element(0.0, 1.0, 0.0, 1.0, new double[] { 0.0, 0.0, 0.0, 1.0 });
}
```

Вход «здоровье» может иметь три значения, входы «нож» и «пистолет» являются булевыми, а вход «враг» показывает количество врагов в пределах видимости (см. примечание к табл. 3.2). Действия также являются булевыми, где ненулевое значение показывает выбранное действие.

Поскольку сеть построена по принципу «победитель получает все», следует закодировать простую функцию, которая будет определять выходную ячейку с самой большой суммой весов. Она выполняет поиск по вектору максимального значения и возвращает строку, которая отображает нужное действие (листинг 3.6). Возвращенное значение затем может использоваться в качестве индекса в массиве строк strings и позволяет вывести текст, показывающий реакцию.

Листинг 3.6. *Функция для сети «победитель получает все».*

```
String strings[] = { "Атаковать", "Бежать", "Уворачиваться", "Прятаться" };
int action(double[] vector) {
    int index, sel;
    double max;
    sel = 0;
    max = vector[sel];
    for (index = 1; index < Const.OUTPUT_NEURONS; index++) {
        if (vector[index] > max) {
            max = vector[index]; sel = index;
        }
    }
    return(sel);
}
```

В листинге 3.7 показана функция main, которая выполняет обучение и тестирование нейронной сети.

Листинг 3.7. *Пример функции main, которая используется для обучения и тестирования нейронной сети.*

```
//dip
public class Main {
   public static void main(String[] args) {
      Neiro neiro = new Neiro();
      neiro.assignRandomWeights();
      neiro.feedForward();
      neiro.setSamples();
      double err;
      int sample = 0;
      int sum = 0;
      /* Обучить сеть */
      for (int i = 0; i < 100000; i++) {
         if (++sample == Const.MAX_SAMPLES)
            sample = 0;
         neiro.inputs[0] = neiro.samples[sample].health;
         neiro.inputs[1] = neiro.samples[sample].knife;
         neiro.inputs[2] = neiro.samples[sample].gun;
         neiro.inputs[3] = neiro.samples[sample].enemy;
         neiro.target[0] = neiro.samples[sample].out[0];
         neiro.target[1] = neiro.samples[sample].out[1];
         neiro.target[2] = neiro.samples[sample].out[2];
         neiro.target[3] = neiro.samples[sample].out[3];
         //прямое распространение
         neiro.feedForward();
         //обратное распространение
         neiro.backPropagate();
      }
      /* Проверить сеть */
      for (int i = 0 ; i < Const.MAX_SAMPLES ; i++) {
         neiro.inputs[0] = neiro.samples[i].health;
         neiro.inputs[1] = neiro.samples[i].knife;
         neiro.inputs[2] = neiro.samples[i].gun;
         neiro.inputs[3] = neiro.samples[i].enemy;
```

```
            neiro.target[0] = neiro.samples[i].out[0];
            neiro.target[1] = neiro.samples[i].out[1];
            neiro.target[2] = neiro.samples[i].out[2];
            neiro.target[3] = neiro.samples[i].out[3];
            neiro.feedForward();
            if (neiro.action(neiro.actual) != neiro.action(neiro.target)) {
               System.out.println(neiro.inputs[0] + " " + neiro.inputs[1] + " " +
neiro.inputs[2] + " " + neiro.inputs[3] + " " +
                     neiro.strings[neiro.action(neiro.actual)] + " " +
neiro.strings[neiro.action(neiro.target)]);
            } else {
               sum++;
            }
         }
         System.out.println("Сеть работает на " + ((float)sum / (float)
Const.MAX_SAMPLES) * 100.0 + "%");
         /* Выполнение тестов */
         /* Здоровье Нож Пистолет Враг*/
         neiro.setInputs(new double[] {2.0, 1.0, 1.0, 1.0});
         neiro.feedForward();
         System.out.println("Жизни Пистолет Нож Враги");
         System.out.println("  2      1      1    1   Действие: " +
neiro.strings[neiro.action(neiro.actual)]);
         neiro.setInputs(new double[] {1.0, 1.0, 1.0, 2.0});
         neiro.feedForward();
         System.out.println("  1      1      1    2   Действие: " +
neiro.strings[neiro.action(neiro.actual)]);
         neiro.setInputs(new double[] {0.0, 0.0, 0.0, 0.0});
         neiro.feedForward();
         System.out.println("  0      0      0    0   Действие: " +
neiro.strings[neiro.action(neiro.actual)]);
         neiro.setInputs(new double[] {0.0, 1.0, 1.0, 1.0});
         neiro.feedForward();
         System.out.println("  0      1      1    1   Действие: " +
neiro.strings[neiro.action(neiro.actual)]);
         neiro.setInputs(new double[] {2.0, 0.0, 1.0, 3.0});
         neiro.feedForward();
```

```
        System.out.println("  2      0      1    3   Действие: " +
neiro.strings[neiro.action(neiro.actual)]);
        neiro.setInputs(new double[] {2.0, 1.0, 0.0, 3.0});
        neiro.feedForward();
        System.out.println("  2      1      0    3   Действие: " +
neiro.strings[neiro.action(neiro.actual)]);
        neiro.setInputs(new double[] {.0, 1.0, .0, 3.0});
        neiro.feedForward();
        System.out.println("  0      1      0    3   Действие: " +
neiro.strings[neiro.action(neiro.actual)]);
    }
}
```

После инициализации генератора случайных чисел с помощью функции srand будут произвольно сгенерированы веса соединений сети. Затем для обучения сети выполняется цикл while. Примеры рассматриваются последовательно, а не в произвольном порядке. Выполняется расчет реакции сети на входной вектор, а затем – проверка средней ошибки. Наконец, запускается алгоритм обратного распространения, чтобы изменить веса соединений сети. После выполнения ряда итераций программа выходит из цикла, и сеть тестируется на точность на основе значений, заданных для обучения. После стандартного тестирования выполняется тестирование сети с примерами, которые не входили в начальную группу при обучении (рис.3.7).

```
Сеть работает на 100.0%
Жизни Пистолет Нож Враги
  2       1      1     1    Действие: Атаковать
  1       1      1     2    Действие: Прятаться
  0       0      0     0    Действие: Уворачиваться
  0       1      1     1    Действие: Прятаться
  2       0      1     3    Действие: Прятаться
  2       1      0     3    Действие: Прятаться
  0       1      0     3    Действие: Бежать
```

Рис. 3.7. Результат работы программы

Таким образом, нейронная сеть правильно генерирует действие из заданного набора в ответ на новую обстановку. Хотя сеть не обучали конкретно для этих примеров, она способна правильно на них реагировать. В этом заключается процесс тестирования нейронной сети [10].

4. СЕТЬ РАДИАЛЬНО-БАЗИСНЫХ ФУНКЦИЙ

Нейронная сеть радиально-базисных функций рассматривается как задача аппроксимации кривой по точкам в пространстве высокой размерности. В соответствии с такой точкой зрения обучение эквивалентно нахождению такой поверхности в многомерном пространстве, которая наиболее точно соответствует данным обучения. При этом критерий «наилучшего соответствия» выбирается в не котором статистическом смысле. Таким образом, обобщение эквивалентно использованию этой многомерной поверхности для интерполяции данных тестирования. Такой подход лежит в основе метода радиальных базисных функций, состоящего в традиционной интерполяции в многомерном пространстве [2].

4.1. RBF-сеть прямого распространения

Радиальные базисные нейронные сети (Radial Basis Function – RBF) или RBF-сети – это двухслойные сети прямого распространения. Базовая архитектура сетей на основе RBF-сетей, предполагает наличие трех слоев, выполняющих совершенно различные функции (рис.4.1).

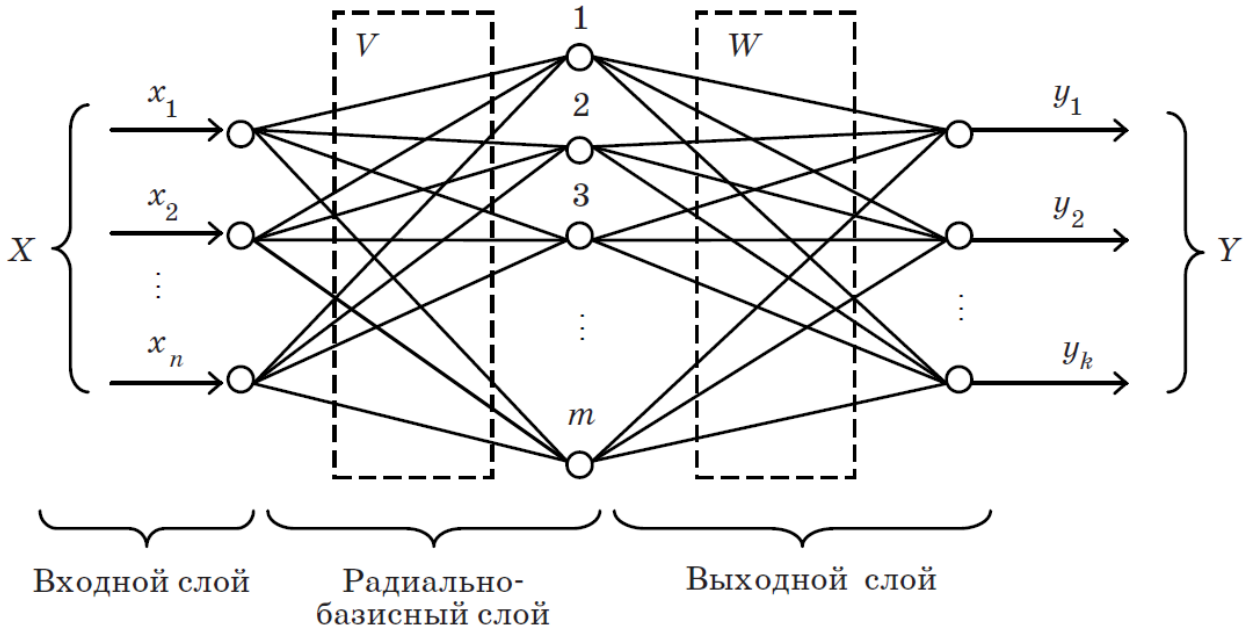

Рис. 4.1. Структура RBF-сети

Входной слой состоит из сенсорных элементов, которые связывают сеть с внешней средой. Второй слой является единственным скрытым (hidden) слоем сети. Он выполняет нелинейное преобразование входного пространства в скрытое. В большинстве реализаций скрытое пространство имеет более высокую размерность, чем входное [2].

В задачах классификации данных в пространстве более высокой размерности с большей вероятностью удовлетворяет требованию линейной разделимости.

Поэтому в RBF-сетях размерность скрытого слоя, как правило, существенно превышает размерность входного слоя. Также важно отметить тот факт, что размерность скрытого пространства непосредственно связана со способностью сети аппроксимировать гладкое отображение «ВХОД-ВЫХОД». Чем выше размерность скрытого слоя, тем более высокой будет точность аппроксимации [4].

Математическую основу функционирования радиальных сетей составляет теорема Т. Ковера о разделимости образов, которая утверждает следующее:

Нелинейное преобразование сложной задачи классификации образов в пространство более высокой размерности повышает вероятность линейной разделимости образов [2].

Теорема Ковера о разделимости образов базируется на двух моментах [4]:

1. Определение нелинейной скрытой функции $\varphi_i(x)$, где x– входной вектор, а $i = 1,2, \dots, K$ – размерность скрытого пространства;

2. Высокая размерность скрытого пространства по сравнению с размерностью входного. Эта размерность определяется значением, присваиваемым K (то есть количеством скрытых нейронов)/

Если вектор радиальных функций $\varphi(x) = [\varphi_1(x), \varphi_2(x), \dots, \varphi_K(x)]^T$ в N-мерном входном пространстве обозначить $\varphi(x)$, то это пространство является нелинейно φ-разделяемым на два пространственных класса X^+и X^- тогда, когда существует такой вектор весов w, что

$$w^T\varphi(x) > 0, x \in X^+$$
$$w^T\varphi(x) < 0, x \in X^-$$

Граница между этими классами определяется уравнением $w^T\varphi(x) = 0$

Ковер доказал, что каждое множество образов, случайным образом размещенных в многомерном пространстве, является -разделяемым с вероятностью 1 при условии большой размерности K этого пространства.

На практике это означает, что применение достаточно большого количества скрытых нейронов, реализующих радиальные функции $\varphi_i(x)$, гарантирует решение задачи классификации при построении всего лишь двухслойной сети. При этом скрытый слой должен реализовать вектор $\varphi_i(x)$, а выходной слой может состоять из единственного линейного нейрона, выполняющего суммирование выходных сигналов от скрытых нейронов с весовыми коэффициентами, заданными вектором w.

Приведем три примера (рис. 4.2) φ-разделимых дихотомий (*деление на два взаимоисключающих понятия*) для различных множеств из пяти точек в двумерном пространстве: линейно-разделимая дихотомия (а); сферически разделимая дихотомия (6); квадратично-разделимая дихотомия (в) [5].

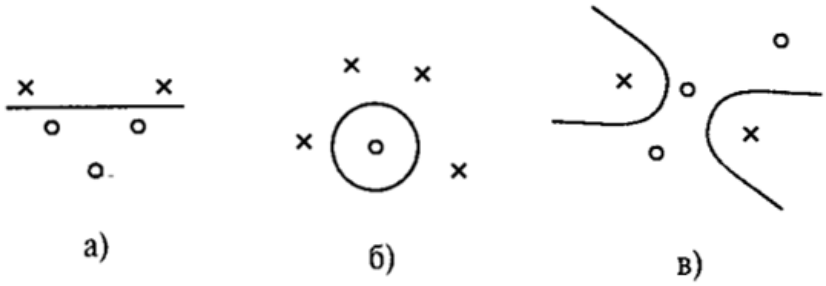

Рис. 4.2. Примеры -разделимых дихотомий.

Главное отличие RBF-сетей от обычных многослойных сетей прямого распространения состоит в функции нейронов скрытого слоя. В обычной многослойной сети каждый нейрон рабочего слоя реализует в многомерном пространстве гиперплоскость (рис. 4.2а), а RBF-нейрон – гиперсферу (рис. 4.2б, 4.2в).

Скрытый слой выполняет нелинейное отображение, реализуемое нейронами с базисными радиальными функциями, параметры которых уточняются в процессе обучения. Таким образом, все веса радиально-базисного слоя (скрытого слоя) полагаются равными единице, и работу -го нейрона RBF-слоя можно описать формулой [2]:

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

где C_i – вектор центра активационной RBF-функции нейрона: $X, C \in R^n$. Таким образом, входной вектор и вектор центра имеют одинаковую размерность.

В качестве радиальной базисной функции φ обычно используется гауссова функция (рис.4.3)

$$\varphi(\|X - C_i\|) = exp\left(\frac{\|X - C_i\|^2}{2{\sigma_i}^2}\right)$$

где σ_i – ширина «окна» активационной функции.

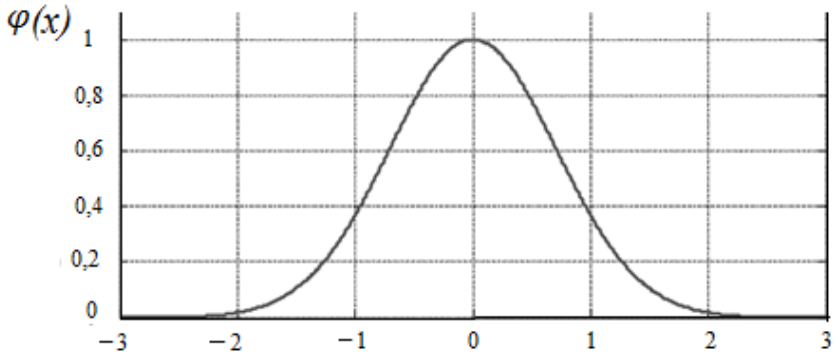

Рис. 4.3. Функция Гаусса при $c = 0$ и $\sigma = 1$

Проблему подбора параметров радиальных функций и значений весов w_i сети можно свести к минимизации целевой функции, которая при использовании метрики Эвклида записывается в форме [2]:

$$\|X - C_i\| = \sqrt{(x_1 - c_{i1})^2 + (x_2 - c_{i2})^2 + \cdots + (x_n - c_{in})^2}$$

Таким образом, i-й нейрон скрытого слоя определяет сходство между входным вектором X и эталонным вектором C_i.

Как следует из рис. 4.3, функция активации RBF-нейрона принимает большие значения лишь в тех случаях, когда входной образ находится вблизи центра нейрона. Вне области, «покрытой» образами обучающей последовательности, сеть формирует лишь малые значения на своих выходах.

Возможны и другие варианты активационной функции. Например, $\varphi(\|X - C_i\|) = \|X - C_i\|$ – линейная функция, $\varphi(\|X - C_i\|) = \|X - C_i\|^3$ – кубическая функция, $\varphi(\|X - C_i\|) = (\|X - C_i\|^2 + \sigma^2)^{1/2}$ – мультиквадратическая функция.

Нейроны выходного слоя имеют линейную активационную функцию. Их роль сводится исключительно к взвешенному суммированию сигналов, генерируемых нейронами рабочего слоя:

$$y_j = \sum_{i=1}^{m} w_{ij} f_i(X), j = \overline{1,k}$$

Число нейронов выходного слоя определяется характером представления выходных данных.

4.2. Расчет параметров RBF сети

Рассмотрим простой вариант определения весов RBF-сети. Пусть RBF-сеть имеет k входов и один выход (рис. 4.4) [4].

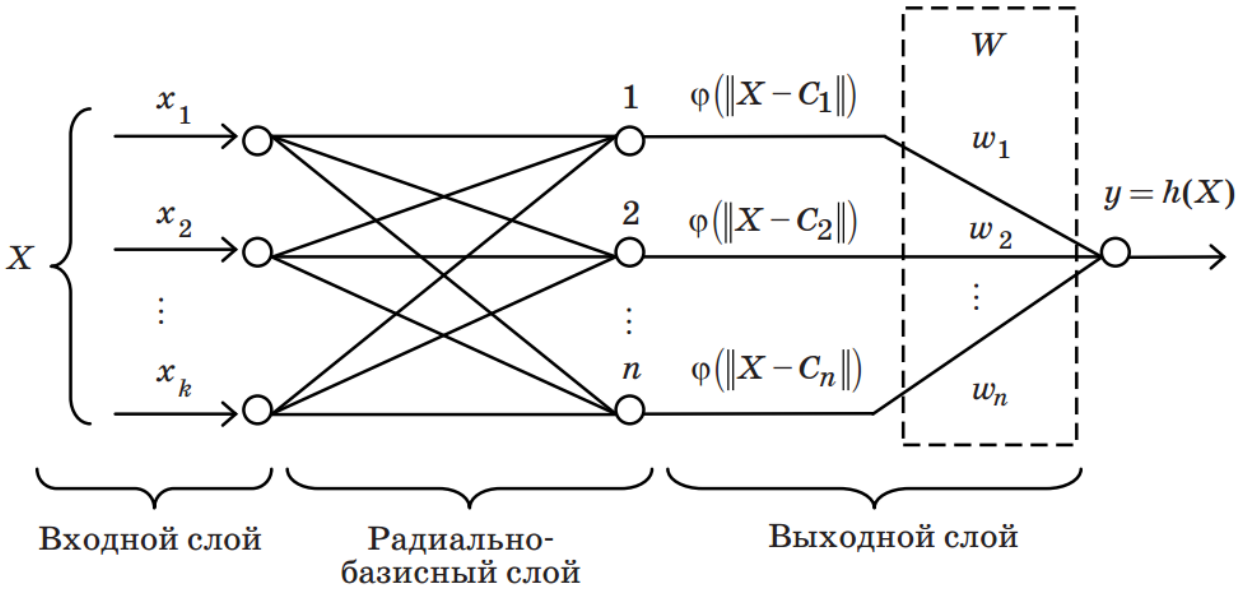

Рис. 4.4. Структура RBF-сети со скалярным выходом

Выберем число рабочих нейронов m = n, где n – число обучающих пар, заданных набором $\{(X_1, y_1), (X_2, y_2), \ldots, (X_n, y_n)\}$, где $Xi = [x_{i1}, x_{i2}, \ldots, x_{ik}]^T$.

Для того чтобы каждый нейрон реагировал на «свой» вектор из обучающего набора, полагаем:

$$C_i = X_i$$

Окна активационной функции σ выбирают достаточно большими, но так, чтобы они не перекрывались в пространстве входных сигналов. Требуется найти такие весовые коэффициенты W, чтобы для каждого входного вектора из обучающего набора выполнялось

$$h(X_i) = y_i$$

Для первого входного вектора из обучающего набора можно записать

$$h(X_i) = \sum_{i=1}^{n} w_i\, f_i(X_1) = w_1 f_1(X_1) + w_2 f_2(X_1) + \cdots + w_n f_n(X_1)$$

Для всех n входных векторов при правильном выборе W должно выполняться

$$\begin{bmatrix} f_1(X_1) & f_2(X_1) & & f_m(X_1) \\ f_1(X_2) & f_2(X_2) & \dots & f_m(X_2) \\ \dots & \dots & & \dots \\ f_1(X_n) & f_2(X_n) & & f_m(X_n) \end{bmatrix} \times \begin{bmatrix} w_1 \\ w_2 \\ \dots \\ w_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{bmatrix}$$

Вводим обозначения для матриц [2]:

$$FW = Y$$

Тогда

$$W = F^{-1}Y \tag{4.1}$$

Формула 4.1 позволяет рассчитать веса RBF-сети с одним выходом при числе нейронов скрытого слоя, равном числу обучающих пар. Аналогичный результат может быть получен при произвольном числе нейронов выходного слоя RBF-сети, если число обучающих пар равно числу нейронов скрытого слоя.

Процесс подбора приближенного значения весов может рассматриваться как задача минимизации целевой функции, описывающей ошибку выхода сети. Для оптимального выбора коэффициентов RBF-сети может быть использован метод наименьших квадратов. Рассмотрим RBF-сеть с одним выходным и m скрытыми нейронами [2]:

$$y = h(X) = \sum_{i=1}^{m} w_i\, f_i(X) \tag{4.2}$$

Пусть необходимо аппроксимировать зависимость, заданную множеством вход-выходных данных (обучающая выборка):

$$\{(X_1, y_1), (X_2, y_2), \dots, (X_n, y_n)\}$$

Для качественной аппроксимации требуется минимизировать ошибку выхода сети, заданную формулой

$$E = \sum_{i=1}^{n} (h(X_i) - y_i)^2 \tag{4.3}$$

Рассмотрим производную формулы 4.3.

$$\frac{\partial E}{\partial w_j} = 2\sum_{i=1}^{n}(h(X_i) - y_i)\frac{\partial h(X_i)}{\partial w_j}$$

В соответствии с формулой 4.2.

$$\frac{\partial h(X_i)}{\partial w_j} = f_j(X_i)$$

следовательно,

$$\frac{\partial E}{\partial w_j} = 2\sum_{i=1}^{n}(h(X_i) - y_i)f_j(X_i)$$

В точке оптимума

$$\frac{\partial E}{\partial w_j} = 2\sum_{i=1}^{n}(h(X_i) - y_i)f_j(X_i) = 0$$

или

$$\sum_{i=1}^{n}(h(X_i) - y_i) = \sum_{i=1}^{n} f_j(X_i)y_i$$

Обозначим

$$F_j = \begin{bmatrix} f_j(X_1) \\ f_j(X_2) \\ \dots \\ f_j(X_n) \end{bmatrix}, H = \begin{bmatrix} h(X_1) \\ h(X_2) \\ \dots \\ h(X_n) \end{bmatrix}, Y = \begin{bmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{bmatrix}$$

Тогда

$$F_j^T H = F_j^T Y, j = \overline{1,m}$$

$$\begin{bmatrix} F_1^T H \\ F_2^T H \\ \dots \\ F_m^T H \end{bmatrix} = \begin{bmatrix} F_1^T Y \\ F_2^T Y \\ \dots \\ F_m^T Y \end{bmatrix}$$

$$F^T H = F^T Y$$

Поскольку

$$H = \begin{bmatrix} h(X_1) \\ h(X_2) \\ \dots \\ h(X_n) \end{bmatrix} = \begin{bmatrix} \sum_{j=1}^{m} w_j f_j(X_1) \\ \sum_{j=1}^{m} w_j f_j(X_2) \\ \dots \\ \sum_{j=1}^{m} w_j f_j(X_n) \end{bmatrix} = \begin{bmatrix} f_1(X_1) & f_2(X_1) & \dots & f_m(X_1) \\ f_1(X_2) & f_2(X_2) & \dots & f_m(X_2) \\ \dots & \dots & \dots & \dots \\ f_1(X_n) & f_2(X_n) & \dots & f_m(X_n) \end{bmatrix} \times \begin{bmatrix} w_1 \\ w_2 \\ \dots \\ w_m \end{bmatrix} = FW$$

Тогда можно записать

$$F^T F W = F^T Y$$

и окончательно [2]

$$W = (F^T F)^{-1} F^T H = F^+ H$$

где F^+ – псевдообратная матрица *(псевдоинверсия* прямоугольной матрицы F).
H – вектор ожидаемых значений выходного сигнала сети.

Рассмотрим пример расчета радиально-базисной сети. Пусть задан набор из трех пар точек ($p = 3$): $\{(0{,}9;\ 1), (2{,}1;\ 1{,}9), (3{,}1;\ 3)\}$.

Требуется аппроксимировать эту зависимость функцией

$$y(x) = w_1 h_1(x) + w_2 h_2(x),$$

где $h_1(x)$ и $h_2(x)$ – выходы радиально-базисных нейронов, заданных в виде

$$h_1(x) = \exp(-(x - 1{,}5)^2),\ h_2(x) = \exp(-(x - 1{,}5)^2)$$

Центры функции активации $c_1 = {}^{(0{,}9 + 2{,}1)}/_2 = 1{,}5$, соответственно $c_2 = 2{,}5$ (рис. 4.5)

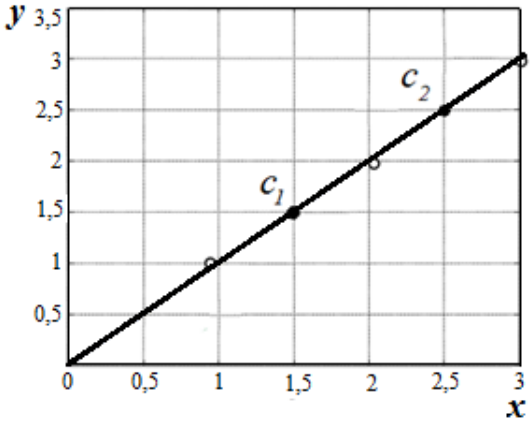

Рис. 4.5. Центры функции активации

Найдем неизвестные коэффициенты w_1 и w_2:

$$h_1(x_1) = \exp(-(x - 1{,}5)^2) = \exp(-(0{,}9 - 1{,}5)^2) = e^{-0{,}36} = 0.6977$$
$$h_1(x_2) = \exp(-(x - 1{,}5)^2) = \exp(-(2{,}1 - 1{,}5)^2) = e^{-0{,}36} = 0.6977$$
$$h_1(x_3) = \exp(-(x - 1{,}5)^2) = \exp(-(3{,}1 - 1{,}5)^2) = e^{-2{,}56} = 0.0773$$
$$h_2(x_1) = \exp(-(x - 2{,}5)^2) = \exp(-(0{,}9 - 2{,}5)^2) = e^{-2{,}56} = 0.0773$$
$$h_2(x_2) = \exp(-(x - 2{,}5)^2) = \exp(-(2{,}1 - 2{,}5)^2) = e^{-0{,}16} = 0.8521$$
$$h_2(x_3) = \exp(-(x - 2{,}5)^2) = \exp(-(3{,}1 - 2{,}5)^2) = e^{-0{,}36} = 0.6977$$

Тогда

$$F = \begin{bmatrix} h_1(x_1) & h_2(x_1) \\ h_1(x_2) & h_2(x_2) \\ h_1(x_3) & h_2(x_3) \end{bmatrix} = \begin{bmatrix} 0{,}6977 & 0{,}0773 \\ 0{,}6977 & 0{,}8521 \\ 0{,}0773 & 0{,}6977 \end{bmatrix}, Y = \begin{bmatrix} 1 \\ 1{,}9 \\ 3 \end{bmatrix}$$

Далее

$$F^T = \begin{bmatrix} 0{,}6977 & 0{,}6977 & 0{,}0773 \\ 0{,}0773 & 0{,}8521 & 0{,}6977 \end{bmatrix}$$

Умножаем строку на столбец

$a_{11} = 0{,}6977^2 + 0{,}6977^2 + 0{,}0773^2 = 0{,}9795$

$a_{12} = 0{,}6977 \times 0{,}0773 + 0{,}6977 \times 0{,}8521 + 0{,}0773 \times 0{,}6977 = 0{,}7024$

и т.д.

Тогда

$$F^T F = \begin{bmatrix} 0{,}9795 & 0{,}7024 \\ 0{,}7024 & 1{,}2188 \end{bmatrix}$$

Найдем обратную матрицу используя формулу

$$(F^T F)^{-1} = \begin{bmatrix} 1{,}7398 & -1{,}0026 \\ -1{,}0026 & 1{,}3982 \end{bmatrix}$$

Далее

$$W = (F^T F)^{-1} F^T Y =$$

$$= \begin{bmatrix} 1{,}7398 & -1{,}0026 \\ -1{,}0026 & 1{,}3982 \end{bmatrix} \times \begin{bmatrix} 0{,}6977 & 0{,}6977 & 0{,}0773 \\ 0{,}0773 & 0{,}8521 & 0{,}6977 \end{bmatrix} \times \begin{bmatrix} 1 \\ 1{,}9 \\ 3 \end{bmatrix} =$$

$$= \begin{bmatrix} 0{,}1244 \\ 3{,}0373 \end{bmatrix}$$

На рис. 4.6 приведен результат аппроксимации. Полученное качество очевидно невысоко по сравнению с линейной функцией, которая обеспечивает в данном случае наилучшую аппроксимацию исходных данных.

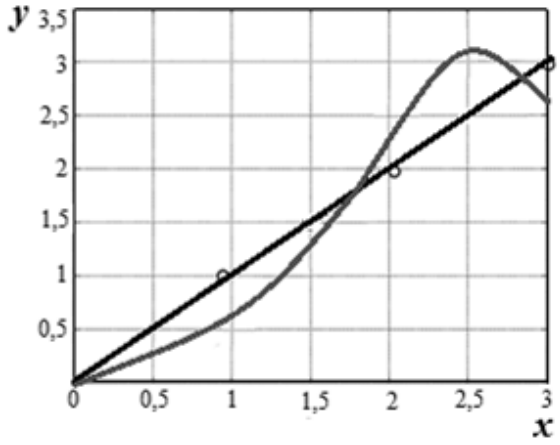

Рис. 4.6. Центры функции активации

Таким образом, если параметры гауссовой функции (центр и радиус) заданы, то задача нахождения весов выходного слоя RBF-сети может быть решена методами линейной алгебры – методом псевдообратных матриц [5].

4.3. Программная реализация задачи аппроксимации.

Рассмотрим программную реализацию простейшего аппроксиматора на языке Python [12]. При этом использовалась библиотека NumPy для вычислений и библиотеку matplotlib для вывода графика на экран.

NumPy – это один из основных пакетов для научных вычислений в Python. Он содержит функциональные возможности для работы с многомерными массивами, высокоуровневыми математическими функциями.

Листинг. *Программную реализация простейшего аппроксиматора на языке Python.*

```
import numpy as np
import matplotlib.pyplot as plt

class RBFN(object):

    def __init__(self, input_shape, hidden_shape, sigma = 1.0):
        #input_shape: измерение входных данных
        #hidden_shape: число скрытых радиально-базисных функций
        self.input_shape = input_shape
        self.hidden_shape = hidden_shape
        self.sigma = sigma
        self.centers = None
        self.weights = None

    def _kernel_function(self, center, data_point):
        return np.exp(-self.sigma*np.linalg.norm(center-data_point)**2)

    def _calculate_interpolation_matrix(self,X):
        """Считает интерполяционную матрицу G с помощью self._kernel_function
        # Arguments
            X: Training data
        # Input shape
            (num_data_samples, input_shape)
        # Returns
            G: Interpolation matrix
        """
        G = np.zeros((X.shape[0], self.hidden_shape))
        for data_point_arg, data_point in enumerate(X):
            for center_arg, center in enumerate(self.centers):
                G[data_point_arg,center_arg] = self._kernel_function(center,
                                                    data_point)
        return G
```

```
    def fit(self,X,Y):
        """ Устанавливает self.weights (веса), используя линейную регрессию
        # Arguments
            X: training samples
            Y: targets
        # Input shape
            X: (num_data_samples, input_shape)
            Y: (num_data_samples, input_shape)
        """
        random_args = np.random.permutation(X.shape[0]).tolist()
        self.centers = [X[arg] for arg in random_args][:self.hidden_shape]
        G = self._calculate_interpolation_matrix(X)
        self.weights = np.dot(np.linalg.pinv(G),Y)

    def predict(self,X):
        """
        # Arguments
            X: test data
        # Input shape
            (num_test_samples, input_shape)
        """
        G = self._calculate_interpolation_matrix(X)
        predictions = np.dot(G, self.weights)
        return predictions

x = np.linspace(0,10,100)
y = np.sin(x)
model = RBFN(input_shape = 1, hidden_shape = 10)
model.fit(x,y)
y_pred = model.predict(x)

plt.plot(x,y,'b-',label='Действительная функция')
plt.plot(x,y_pred,'r-',label='Аппроксимация')
plt.legend(loc='lower right')
plt.title('RBFN')
plt.show()
```

Результат работы программы представлен на рис. 4.7.

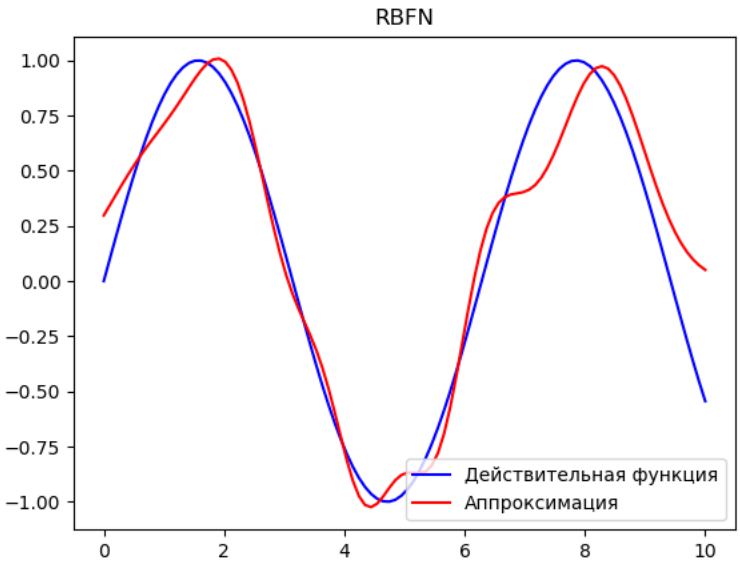

Рис. 4.7. Результат аппроксимации

4.4. Сравнение сетей RBF и MLP

Таким, образом, процесс обучения сети RBF с учетом выбранного типа радиальной базисной функции сводится к двум этапам [2]:

- к подбору центров c_i и параметров σ_i формы базисных функций (обычно используются алгоритмы обучения без учителя);
- к подбору весов нейронов выходного слоя (обычно используются алгоритмы обучения с учителем).

При этом второй этап значительно проще первого, поскольку сводится к вычислению выражения $W = F^+ * H$, где основные вычислительные затраты – расчет псевдоинверсии матрицы F.

Для первого этапа как правило обучение на основе самоорганизации. Его целью является оценка подходящих положений центров радиальных базисных функций. Процесс самоорганизации обучающих данных автоматически разделяет пространство на так называемые области Вороного, определяющие различающиеся группы данных. Данные, сгруппированные внутри кластера, представляются центральной точкой, определяющей среднее значение всех его элементов. Центр кластера отождествляется с центром соответствующей радиальной функции.

Сети на основе радиальных базисных функций (RBF) и многослойный персептрон (MLP) являются примерами нелинейных многослойных сетей прямого распространения. И те, и другие являются универсальными аппроксиматорами. Таким образом, неудивительно, что всегда существует сеть RBF, способная имитировать многослойный персептрон (и наоборот). Однако эти два типа сетей отличаются по некоторым важным аспектам [2]:

1. Сети RBF (в своей основной форме) имеют один скрытый слой, в то время как многослойный персептрон может иметь большее количество скрытых слоев.

2. Обычно вычислительные (computationa1) узлы многослойного персептрона, расположенные в скрытых и выходном слоях, используют одну и ту же модель нейрона. С другой стороны, вычислительные узлы скрытого слоя сети RBF могут в корне отличаться от узлов выходного слоя и служить разным целям.

3. Скрытый слой в сетях RBF является нелинейным, в то время как выходной линейным. В то же время скрытые и выходной слои многослойного персептрона, используемого в качестве классификатора, являются нелинейными. Если многослойный персептрон используется для решения задач нелинейной регрессии, в качестве узлов выходного слоя обычно выбираются линейные нейроны.

4. Аргумент функции активации каждого скрытого узла сети RBF представляет собой *Евклидову норму* (расстояние) между входным вектором и центром радиальной функции. В то же время аргумент функции активации каждого скрытого узла многослойного персептрона – это скалярное произведение входного вектора и вектора синоптических весов данного нейрона.

5. Многослойный персептрон обеспечивает глобальную аппроксимацию нелинейного отображения. С другой стороны, сеть RBF с помощью экспоненциально уменьшающихся локализованных нелинейностей (т.е. функций Гаусса) создает локальную аппроксимацию нелинейного отображения.

Это, в свою очередь, означает, что для аппроксимации нелинейного отображения с помощью многослойного персептрона может потребоваться меньшее число параметров, чем для сети RBF при одинаковой точности вычислений. Линейные характеристики выходного слоя сети RBF означают, что такая сеть более тесно связана с персептроном Розенблатта, чем с многослойным персептроном. Тем не менее сети RBF отличаются от этого персептрона тем, что способны выполнять нелинейные преобразования входного пространства.

Структура сетей RBF является необычной в том смысле, что архитектура скрытых элементов в корне отличается от структуры выходных. Поскольку основой функционирования нейронов скрытого слоя являются радиальные базисные функции, теория сетей RBF тесно связана с теорией радиальных базисных функций, которая в настоящее время является одной из основных областей изучения в численном анализе [2].

Интересным также является тот факт, что настройка линейных весов выходного слоя позволяет обеспечить хорошее качество классификации.

Вопросы для самопроверки

1. Что такое искусственная нейронная сеть?

2. Из каких частей состоит математический нейрон МакКаллока – Питтса?

3. Какие варианты активационной функции могут быть использованы в математическом нейроне МакКаллока – Питтса?

4. Как выглядит искусственный нейрон для реализации логических операций «И», «ИЛИ» и «НЕТ»?

5. На каких формулах основываются расчет сигналов NET и OUT?

6. Понятие смещение. Как прийти к единой пороговой чувствительности?

7. Понятие разделяющей гиперплоскости и ее уравнения для логических операций «И», «ИЛИ» и «НЕТ».

8. Дайте определение однослойной нейронной сети прямого распространения. Как выглядит структура однослойного персептрона?

9. Что такое персептрон? Какое физическое воплощение имеет элементарный персептрон Розенблатта?

10. Как выглядит структура модели зрительной системы? Что такое S-элементы, A-элементы и R-элементы?

11. Как формулируются правила обучения Хебба? Алгоритм обучения с учителем по правилам Хебба?

12. Приведите расчет по правилам Хебба для логических операций «И», «ИЛИ» и «НЕТ».

13. Что такое хеббовская, антихеббовская и нехеббовская модели модификации синаптических связей?

14. Как формулируются дельта-правило для обучения искусственной нейронной сети? Алгоритм обучения с учителем по дельта-правилу?

15. Как вычисляется значение ошибки обучения по дельта-правилу?

16. Как проводится модификация весового коэффициента связи по дельта-правилу?

17. Приведите расчет по дельта-правилу для логических операций «И», «ИЛИ» и «НЕТ».

18. Как формулируются правило Видроу-Хоффа для обучения искусственной нейронной сети? Как рассчитать квадратичную ошибка обучения?

19. Как обучить персептрон распознаванию цифр или букв по дельта-правилу? Какая структура однослойного персептрона, предназначенного для распознавания цифр?

20. Как описывается дельта-правило на языке программирования Java?

21. Как оценить качество обучения нейросети для распознавания цифр?

21. Дайте определение многослойной нейронной сети прямого распространения. Как выглядит структура многослойного персептрона?

22. Какие активационные функции используются в многослойной нейронной сети прямого распространения?

23. Какая структура персептрона, моделирующий функцию «Исключающее ИЛИ»? Нарисуйте ее.

24. С помощью формул, описывающих работу математического нейрона, убедитесь, что нарисованный Вами персептрон действительно моделирует функцию «Исключающее ИЛИ».

25. Объясните, в чем состоит идея алгоритма обратного распространения ошибки? Отражает ли название алгоритма его идею?

26. Какую роль в методе обратного распространения ошибки выполняет коэффициент скорости обучения?

27. Попробуйте численно решить и запрограммировать алгоритм обратного распространения ошибки на каком-либо языке програмирования.

28. Годится ли алгоритм обратного распространения ошибки для обучения персептрона со ступенчатыми активационными функциями?

29. Годится ли правила Хебба для обучения персептрона с нейронами, имеющими сигмоидные функции активации?

30. Годится ли дельта-правило для обучения персептрона с нейронами, имеющими сигмоидные функции активации?

31. Назовите преимущества и недостатки алгоритма обратного распространения ошибки по сравнению со всеми изученными ранее методами обучения нейронных сетей.

32. Дайте определение сети на основе радиальных базисных функций. Как выглядит структура такой сети?

33. Расскажите теорему Ковера о разделимости образов. На чем она базируется?

34. В чем заключается задача подбора параметров радиальных функций и значений весов сети?

35. В чем заключается сущность метода псевдообратной матрицы для расчета сети RBF?

36. Рассмотрите решение задачи XOR с помощью сети RBF с четырьмя скрытыми элементами.

37. Какие существуют различия между сетью на основе радиальных базисных функций и многослойным персептроном?

Список литературы

1. Ясницкий Л.Н. Интеллектуальные системы / Л. Н. Ясницкий – М.: Лаборатория знаний, 2016. – 221 с.

2. Хайкин С. Нейронные сети: полный курс / С. Хайкин; 2-е издание.: Пер. с англ. – М.: Издательский дом «Вильямс», 2006. – 1104 с.

3. Каллан Р. Основные концепции нейронных сетей/ Р. Каллан; Пер. с англ. – М.: Издательский дом «Вильямс», 2001. – 290 с.

4. Бураков, М. В. Нейронные сети и нейроконтроллеры: учеб. пособие / М. В. Бураков. – СПб.: ГУАП, 2013. – 284 с.

5. Круг П. Г. Нейронные сети и нейрокомпьютеры: учеб. пособие / П. Г. Круг – М.: Издательство МЭИ, 2002. – 176 с.

6. Барский А. Б. Нейронные сети: распознавание, управление, принятие решений / А. Б. Барский – М.: Финансы и статистика, 2004. – 176 с.

7.Оссовский С. Нейронные сети для обработки информации / Пер. с пол. И.Д. Рудинского. – М.: Финансы и статистика, 2002. – 344 с.

8. Уоссерман Ф. Нейрокомпьютерная техника: теория и практика / Пер. с англ. Ю.А. Зуев. – М.: Мир, 1992.

9. Рутковская Д., Пилиньский М., Рутковский Л. Нейронные сети, генетические алгоритмы и нечёткие системы: Пер. с польск. И.Д.Рудинского, - М.: Горячая линия – Телеком, 2007. – 452 с.

10. Джонс М. Т. Программирование искусственного интеллекта в приложениях / М. Т. Джонс; Пер. с англ. Осипов А. И. – М.: ДМК Пресс, 2011. – 312 с.

11. Unity 5.x. Программирование искусственного интеллекта в играх: пер. с англ. Р. Н. Рагимова. - М.: ДМК Пресс, 2017. – 272 с.

12. Рашка С. Python и машинное обучение / С. Рашка; Пер. с англ. А. В. Логунова. – М.: ДМК Пресс, 2017. – 418 с.

Учебное издание

Сорокин Алексей Борисович
Платонова Ольга Владимировна

ИСКУССТВЕННЫЕ НЕЙРОННЫЕ СЕТИ ПРЯМОГО РАСПРОСТРАНЕНИЯ

Учебно-методическое пособие

Компьютерная верстка, редактор, корректор или
печатается в авторской редакции

Подписано в печать 00.00.2018. Формат 60×84 1/16.
Физ. печ. л. 4. Тираж 100 экз. Изд. № 00. Заказ № 000.

Московский технологический университет (МИРЭА)
119454, Москва, пр. Вернадского, д. 78