1. ОСОБЕННОСТИ НЕЙРОСЕТЕВЫХ МЕТОДОВ ОБРАБОТКИ ИНФОРМАЦИИ

В классическом программировании люди вводят правила (программу) и данные для обработки в соответствии с этими правилами и получают ответы (рис. 1.1). В машинном обучении люди вводят данные и ответы, соответствующие этим данным, а на выходе получают правила. Эти правила затем можно применить к новым данным для получения оригинальных ответов. В машинном обучении система обучается, а не программируется явно. Ей передаются многочисленные примеры, имеющие отношение к решаемой задаче, а она находит в этих примерах статистическую структуру, которая позволяет системе выработать правила для автоматического решения задачи.

Рис. 1.1 Классическое программирование и машинное обучение

Машинное обучение тесно связано с математической статистикой, но имеет несколько важных отличий от нее. Машинное обучение, в отличие от статистики, обычно имеет дело с большими и сложными наборами данных к которым практически невозможно применить классические методы статистического анализа, такие как байесовские методы. Как результат, машинное обучение и в особенности ИНС не имеют мощного математического фундамента и основываются почти исключительно на эвристических решениях. Это дисциплина для решения практических задач, в которой идеи чаще доказываются эмпирически, а не теоретически.

Для машинного обучения нужны три составляющие:

- Эталонные входные данные — например, если решается задача распознавания речи, такими входными данными могут быть файлы с записью речи разных людей; если решается задача классификации изображений, такими данными могут быть изображения.
- Примеры ожидаемых результатов — в задаче распознавания речи это могут быть транскрипции звуковых файлов, составленные людьми; в задаче классификации изображений ожидаемым результатом может быть отнесение их к определенной категории, такие как «самолет», «автомобиль» и др.
- Способ оценки качества работы алгоритма — для вычисления количественной меры, насколько отличаются результаты, возвращаемые алгоритмом, от ожидаемых. Эта оценка используется как сигнал обратной связи для корректировки работы алгоритма. Этап корректировки собственно и называется обучением.

Обучение ИНС –это раздел машинного обучения, делающий упор на изучение последовательных слоев (или уровней) значимых представлений данных. Такие многослойные представления изучаются с использованием моделей, называемых ИНС. Следует еще раз почеркнуть, что хотя термин нейронная сеть заимствован из нейробиологии, но модели ИНС не являются моделями мозга. В них используются простейшие вычислительные элементы – формальные (искусственные) нейроны, не являющиеся моделями биологических нейронов.

В ранних классических моделях ИНС рассматривались структуры, состоящие из небольшого количества слоев формальных нейронов, как правило, не более трех. В настоящее время с такими сетями связывают понятие поверхностного обучения. Структуры современных ИНС содержат десятки и даже сотни последовательных слоев — и все они автоматически модифицируются под воздействием обучающих данных. К этим сетям применяют понятие глубокого обучения. Под глубиной в глубоком обучении

не подразумевается более глубокое понимание, достигаемое этим подходом. Смысловое содержание этого термина заключается лишь в многослойном представлении исходных данных и автоматическом выделении в них отличительных признаков. Количество слоев, на которые делится модель данных, называют глубиной модели.

Таким образом, суть обучения ИНС заключается в преобразовании входных данных (например, изображений) в ответ сети (например, в подпись к изображению - «самолет»), которое выявляется путем исследования множества эталонных примеров входных данных и соответствующих им результатов.

Процесс обучения заключается в изменении параметров слоев сети. Этими параметрами являются весовые коэффициенты межнейронных связей. Веса фактически являются набором чисел, также их называют параметрами слоя. Под обучением подразумевается поиск набора значений весов всех слоев в ИНС, при котором она будет правильно отображать эталонные входные данные в соответствующие им ответы сети. Чтобы управлять процессом обучения ИНС, нужно иметь возможность измерять, насколько этот ответ далек от ожидаемого. Для этого нужна функция потерь сети, которую также называют целевой функцией.

Функция потерь в качестве аргументов принимает значение на выходе ИНС и истинное значение, которое сеть должна была бы вернуть, и вычисляет оценку расстояния между ними, отражающую, насколько хорошо ИНС справилась с данным эталонным примером. Эта оценка используется для корректировки значений весов с целью уменьшения потерь для текущего примера (рис. 1.2). Выполнение корректировки является задачей оптимизатора, который реализует так называемый алгоритм обратного распространения ошибки.

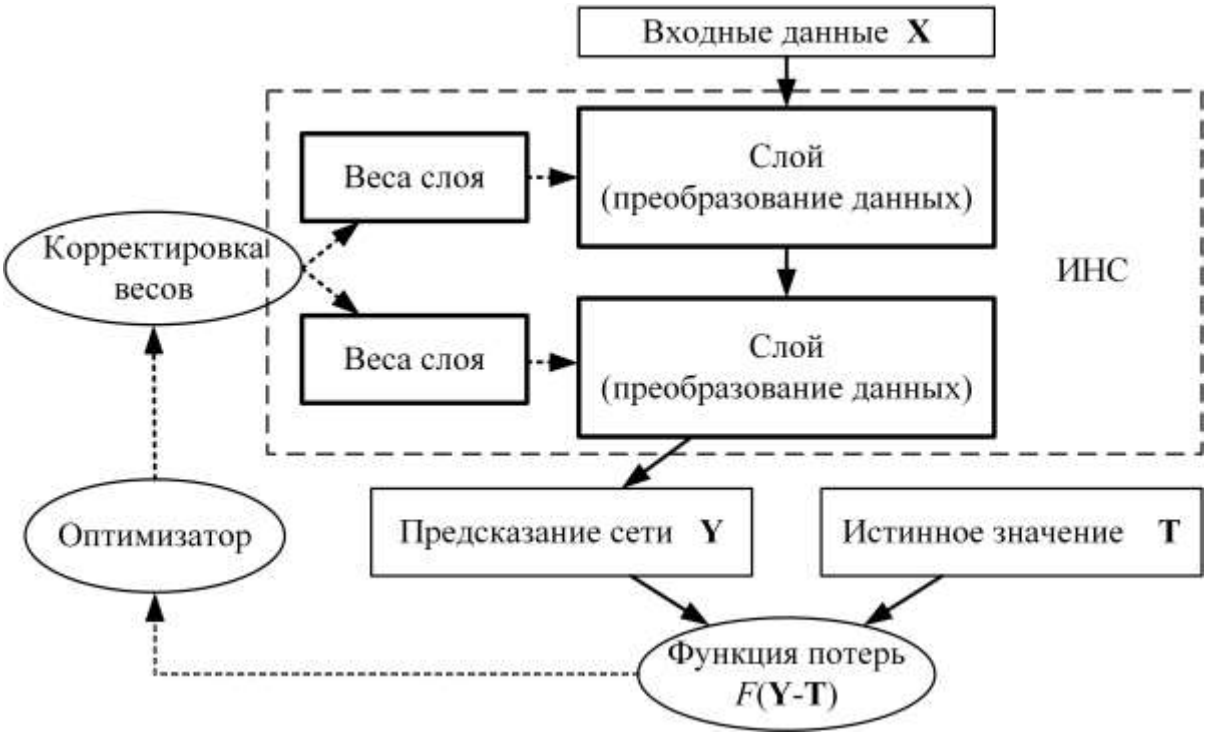


Рис. 1.2 Реализация процесса обучения ИНС

Первоначально весам сети присваиваются случайные значения, то есть фактически сеть реализует последовательность случайных преобразований. Естественно, получаемый ею ответ далек от идеала, и оценка потерь, как правило высока. Но с каждым примером, обрабатываемым сетью, веса корректируются в нужном направлении, и оценка потерь уменьшается. Этот цикл обучения, который повторяется многократно (обычно десятки и сотни итераций с множеством эталонных примеров) и порождает весовые значения, минимизирующие функцию потерь. ИНС с минимальными потерями, возвращающая ответы, близкие к истинным, называется обученной сетью.

Необходимо отметить, что рассмотренный алгоритм обучения принято называть «обучением с учителем», поскольку он предполагает знание истинного ответ, задаваемого человеком – «учителем». Существуют задачи, которые не предполагают знание истинного ответа, например, разделение множества эталонных примеров на группы похожих. В этом случае функция потерь формируется только на основании текущих ответов сети. Этот вид обучения называют «обучением без учителя».

Типичной задачей, для которой применяется алгоритм обучения «с учителем», является классификация – присвоение метки класса каждому примеру из массива входных данных. На входе классификатора обычно имеется вектор признаков объекта, а на выходе – класс, к которому принадлежит этот объект. Если есть только два класса объектов (например, свой/чужой), то такой алгоритм обучения называют бинарным классификатором, иначе он называется многоклассовым классификатором (например, в случае разделения всех рукописных цифр на 10 классов).

Метки классов являются дискретными переменными, ограниченными диапазоном возможных значений. Эти переменные могут быть двух типов: ординальными и номинальными. Значения ординального типа можно упорядочить. Например, значения чисел от 0 до 9 являются ординальными, так как целые числа можно сравнивать между собой. Объекты из набора фруктов {банан, яблоко, апельсин, …} не могут быть упорядочены естественным образом. Объекты из этих наборов называются номинальными, потому что они могут быть описаны только по именам. Простой метод представления номинальных объектов в обучающей выборке состоит в присваивании каждой метке номера. Другой способ представления номинальных классов состоит в добавлении фиктивных переменных для каждой номинальной переменной. Каждая переменная {банан, яблоко, апельсин, …} принимает значение 0 или 1 в зависимости от класса, для которого это значение является истиной. Этот способ часто называют прямым кодированием.

2. АРХИТЕКТУРА ИСКУССТВЕННЫХ НЕЙРОННЫХ СЕТЕЙ

Формальный нейрон

Искусственный нейрон (или формальный нейрон) является элементарным функциональным модулем, из которых строятся ИНС. Он представляет собой модель биологического нейрона, только лишь в смысле осуществляемых им преобразований, а не способа функционирования. Известны логические, непрерывные и импульсные модели нейрона. Логические и импульсные модели нейрона активно исследовались в 60-70-х годах, но не получили должного развития.

Непрерывная модель нейрона имитирует в первом приближении свойства биологического нейрона и включает в себя три основных элемента:

1. Набор синапсов или связей, каждая из которых характеризуется своим весом (*weight*). В частности, сигнал x_i на входе синапса i, умножается на на вес w_i. Положительные значения весов w_i. соответствуют возбуждающим синапсам, отрицательные значения – тормозным.
2. Сумматор вычисляет взвешенную, относительно соответствующих синапсов, сумму входных сигналов нейрона.
3. Функция активации ограничивает амплитуду выходного сигнала нейрона. Эта функция также называется функцией сжатия. Обычно нормализованный диапазон амплитуд выхода нейрона лежит в интервале [0,1] или [-1,1].

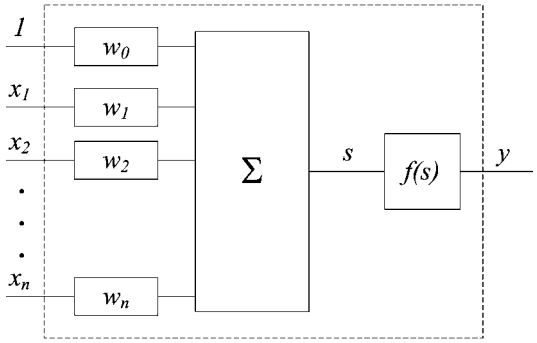


Рис. 2.1. Схема формального нейрона

Математическую модель формального нейрона можно описать уравнением:

$$s = \sum_{i=1}^{n} w_i x_i + w_0$$

$$y = f(s), \text{ где} \qquad (2.1)$$

y – выходной сигнал нейрона;

$f(s)$ – функция активации нейрона;

w_i – весовой коэффициент синаптической связи i –го входа;

x_i – i-й входной сигнал нейрона;

w_0 – начальное состояние (возбуждение) нейрона;

$i = 1,2 \ldots n$ –номера входов нейрона;

n – число входов.

Выражению (2.1) может быть поставлена в соответствие схема формального нейрона, представленная на рис. 2.1. В общем виде работа формального нейрона заключается в следующем. Перед началом работы на блок сумматора подают сигнал начального состояния (возбуждения) нейрона – w_0. На каждый i-й вход нейрона поступают сигналы x_i либо от других нейронов, либо с устройства ввода исходного образа. Каждый i-й входной сигнал x_i умножается на коэффициент синаптической связи w_i. В блоке сумматора взвешенные входные сигналы и начальное возбуждение w_0 алгебраически складываются. Результат суммирования (взвешенная сумма) s подается на блок функционального преобразования $f(s)$. Эта модель нейрона, является классической. В некоторых случаях используются и другие модели формальных нейронов. К таким нейронам относятся нейроны с квадратичным сумматором, нейрон со счетчиком совпадений и др. Выбор модели нейронов определяется прежде всего характером решаемой задачи.

Функции активации

Функция активации нейрона $f(s)$ определяет нелинейное преобразование, осуществляемое нейроном. Существует несколько видов

активационных функций, среди которых можно выделить основные (табл. 2.1).

Функции активации нейронов *Таблица 2.1*

N	Название	Формула	Область значений
1	Линейная	$f(s)= ks$	$(-\infty, \infty)$
2	Полулинейная ReLU (rectified linear unit)	$f(s)= \begin{cases} s, s \geq 0 \\ 0, s < 0 \end{cases}$	$(0, \infty)$
3	Логистическая (сигмоидная)	$f(s) = \frac{1}{1+e^{-as}}$	$(0, 1)$
4	Симметричная сигмоидная (гиперболический тангенс)	$f(s) = \frac{e^{as} - e^{-as}}{e^{as} + e^{-as}}$	$(-1, 1)$
5	Рациональная сигмоидная	$f(s) = \frac{s}{a+\lvert s \rvert}$	$(-1, 1)$
6	Пороговая	$f(s)= \begin{cases} 1, s \geq \theta \\ 0, s < \theta \end{cases}$	$(0, 1)$
7	Сигнатурная	$f(s)= \begin{cases} 1, s \geq \theta \\ -1, s < \theta \end{cases}$	$(-1, 1)$
8	SoftPlus	$f(s)=\log (1+e^s)$	$(-1, 1)$

Наиболее распространенными функциями активации являются пороговая, полулинейная и сигмоидные – логистическая и симметричная (гиперболический тангенс).

Пороговая и сигнатурная (симметричная пороговая) функции относятся к классу дискретных функций (рис 2.2). Основным недостатком нейронов с пороговыми функциями активации является отсутствие достаточной гибкости при обучении и настройке ИНС на решаемую задачу. Если значение вычисляемого скалярного произведения $w_i x_i$ не достигает заданного порога, то выходной сигнал не формируется, и нейрон “не срабатывает”. Это означает, что теряется интенсивность выходного сигнала данного нейрона и, следовательно, формируется невысокое значение уровня на взвешенных входах в следующем слое нейронов. По этой причине пороговые функции активации обычно используются для представления нейросетями функций логического типа. Нейроны с пороговыми функциями активации используются также в сетях, обучаемых по методу конкуренции. Нейрон с

наибольшим значением вычисляемого скалярного произведения $w_i x_i$ (победитель) получает право иметь на выходе значение y=1, формируемое пороговой функцией.

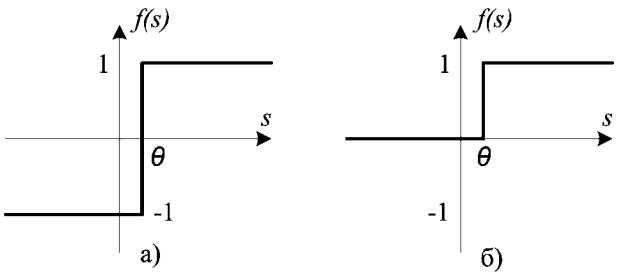


Рис. 2.2. Графики функций активации
а) сигнатурная б) пороговая

Сигмоидные функции активации относятся к классу непрерывных функций (рис. 2.3). Эти функции удовлетворяют основным условиям, предъявляемым к непрерывным активационным функциям нейронов, используемым в обучаемых ИНС: непрерывность, монотонное возрастание и дифференцируемость. Можно отметить, что с уменьшением коэффициента наклона a сигмоиды становятся более пологими, а при a->∞ превращаются в пороговую и сигнатурную функции соответственно. В числе их достоинств следует также упомянуть относительную простоту и непрерывность производных и свойство усиливать слабые сигналы лучше, чем большие. Рациональная или «упрощенная» сигмоида представляет интерес в силу простоты ее программной реализации. Сигмоидные функции в силу своей нелинейности позволяют выделять в поисковом пространстве области сложной формы, в том числе невыпуклые.

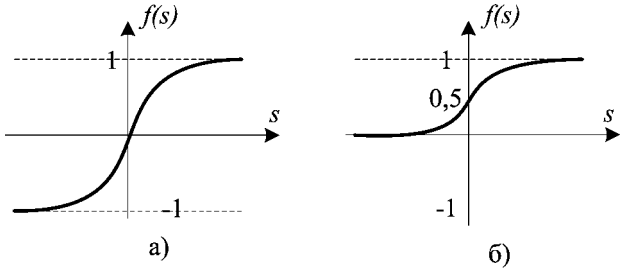


Рис. 2.3. Графики сигмоидных функций активации
а) симметричная б) логистическая

Линейные функции активации также относятся к классу непрерывных функций (рис. 2.4). Линейный участок такой функции активации позволяет оперировать с непрерывными сигналами. Зоны нечувствительности определяются физической реализуемостью этих функций.

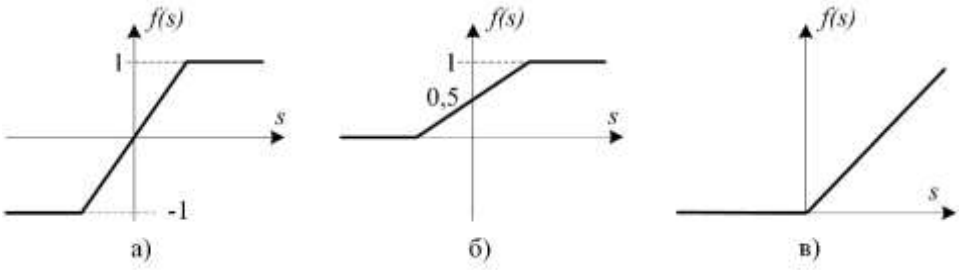


Рис. 2.4. Графики линейных функций активации с насыщением
а) линейная б) полулинейная в) ReLU

Выбор функции активации определяется спецификой задачи, удобством реализации (программным способом, в виде электрической схемы или другим способом) и алгоритмом обучения: некоторые алгоритмы накладывают ограничения на вид функции активации, их нужно учитывать. Удачный выбор функции активации может сократить время обучения в несколько раз.

Формальный нейрон фактически представляет собой процессор с очень ограниченной специальной системой команд (нейросетевым базисом). Формальные нейроны по способу представления информации бывают аналоговые и цифровые. И те и другие выполняют единообразные вычислительные действия и не требуют внешнего управления. Большое число параллельно работающих процессоров обеспечивают высокое быстродействие.

Рассмотренная простая модель формального нейрона игнорирует многие свойства своего биологического двойника:

1. Вычисления выхода нейрона предполагаются мгновенными, не вносящими задержки. Моделировать динамические системы, имеющие "внутреннее состояние", с помощью таких нейронов нельзя.

2. В модели отсутствуют нервные импульсы. Нет модуляции уровня

сигнала плотностью импульсов, как в нервной системе. Не появляются эффекты синхронизации, когда скопления нейронов обрабатывают информацию синхронно, под управлением периодических волн возбуждения-торможения.

3. Нет четких алгоритмов для выбора функции активации.

4. Нет механизмов, регулирующих работу сети в целом (пример - гормональная регуляция активности в биологических нервных сетях).

5. Чрезмерно формализованы понятия: "порог", "весовые коэффициенты". В реальных нейронах нет числового порога, он динамически меняется в зависимости от активности нейрона и общего состояния сети. Весовые коэффициенты синапсов тоже не постоянны. "Живые" синапсы обладают пластичностью и стабильностью: весовые коэффициенты настраиваются в зависимости от сигналов, проходящих через синапс.

6. Существует большое разнообразие биологических синапсов. Они встречаются в различных частях клетки и выполняют различные функции. Тормозные и возбуждающие синапсы реализуются в данной модели в виде весовых коэффициентов противоположного знака, но разнообразие синапсов этим не ограничивается. Дендро-дендритные, аксо-аксональные синапсы не реализуются в модели формального нейрона.

По этой причине формальный нейрон зачастую называют просто узлом нейронной сети.

Классификация моделей нейронных сетей

Можно выделить три основных признака, положенных в основу классификации ИНС: архитектура межнейронных связей, тип обучения, класс решаемых задач. Выбор архитектуры нейронной сети обусловлен типом решаемой задачи и определяет способ ее обучения.

Формализованная модель ИНС представляет собой направленный граф со взвешенными ребрами, в котором формальные нейроны являются узлами. По архитектуре связей ИНС могут быть сгруппированы в два класса: сети

прямого распространения, в которых графы не имеют петель, и рекуррентные сети, или сети с обратными связями.

К первому классу сетей относятся многослойные сети прямого распространения или ацикличные сети. В этих сетях нейроны располагаются по слоям. В такой сети существует входной слой узлов источника сигнала (не являющихся нейронами), информация от которого передается на слой нейронов (вычислительных узлов), и далее распространяется по промежуточным (скрытым) слоям сети до ее выхода. Узлы источника сигнала формируют вектор распознаваемого образа, поступающий на нейроны первого (скрытого) слоя. Выходные сигналы первого слоя используются в качестве входных для второго слоя и т.д. Обычно нейроны каждого из слоев сети используют в качестве входных сигналов только выходные сигналы нейронов предыдущего слоя. Вектор сигналов нейронов выходного слоя сети определяет общий отклик ИНС на данный входной образ. ИНС прямого распространения с m входами, h_1 нейронами первого скрытого слоя, h_2 нейронами второго скрытого слоя и q нейронами выходного слоя называется сетью с архитектурой m-h_1-h_2-q. На рис. 2.5 показана сеть 3-4-3 (3 входных узла, 4 скрытых и 3 выходных нейрона).

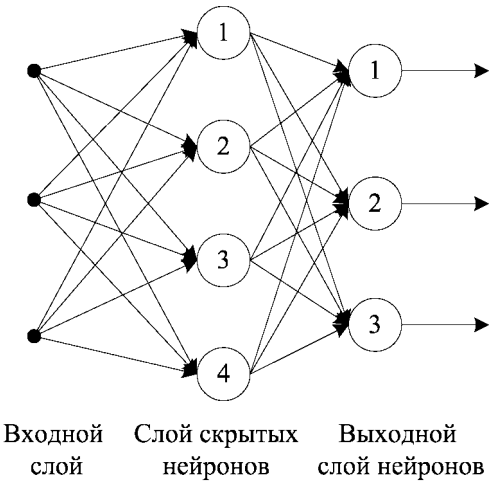


Рис. 2.5 Сеть прямого распространения с одним скрытым слоем

ИНС прямого распространения считается полносвязной в том смысле, что все узлы каждого слоя соединены со всеми узлами смежных слоев. Если

некоторые из синаптических связей отсутствуют, такая сеть называется сетью с прореженными связями. В таких прямонаправленных ИНС нейроны одного слоя связаны только с определенной частью нейронов следующего слоя. Эта архитектура ИНС широко используются для решения задач классификации образов, аппроксимации данных и управления. Для ее обучения используется метод обучения «с учителем».

Рекуррентные ИНС отличаются от сетей прямого распространения наличием по крайней мере одной обратной связи. Рекуррентная сеть может состоять из одного или нескольких слоев нейронов, каждый из которых направляет свой выходной сигнал на входы всех остальных нейронов слоя. Такие сети, как правило, используются для решения задач кластеризации – разделения большого массива исходных данных на группы (кластеры) похожих данных и выявления закономерностей в этих массивах. Для обучения рекуррентных ИНС используется метод обучения «без учителя».

Сети прямого распространения являются статическими в том смысле, что на входной образ они вырабатывают одну совокупность выходных значений, не зависящих от предыдущего состояния сети. Рекуррентные сети являются динамическими, так как в силу обратных связей в них модифицируются входы нейронов, что приводит к изменению состояния сети.

3. ОБУЧЕНИЕ МЕТОДОМ ОБРАТНОГО РАСПРОСТРАНЕНИЯ ОШИБКИ

Процедура обучения

Процесс обучения ИНС, в большинстве случаев, является настройкой ее весовых коэффициентов с целью эффективного выполнения решаемой задачи. Настройка весовых коэффициентов осуществляется по заданной обучающей выборке. Функционирование сети должно улучшаться по мере выполнения итераций этой настройки.

Существует множество определений термина «обучение», однако применительно к ИНС наиболее подходит следующее [1]: «*Обучение — это процесс, при котором свободные параметры нейронной сети адаптируются в результате ее непрерывной стимуляции внешним окружением. Тип обучения определяется тем способом, которым производятся изменения параметров».*

Это определение процесса обучения предполагает следующую последовательность событий:

1. В нейронную сеть поступают стимулы из внешней среды.
2. В результате этого изменяются свободные параметры нейронной сети
3. После изменения внутренней структуры нейронная сеть отвечает на возбуждения уже иным образом.

Кроме термина «обучение» также используются равноправные понятия «тренировка сети» и «настройка параметров сети». Можно выделить два основных вида обучения: контролируемое обучение (*supervised learning*) и самообучение (*self-organized learning*). Первый вид подразумевает наличие «учителя», который наблюдает реакцию сети и направляет изменения ее параметров. Во втором случае сеть самоорганизуется под действием внешней среды и изучает ее самостоятельно, без помощи «учителя».

Широко используемым в настоящее время методом контролируемого обучения многослойных прямонаправленных ИНС является алгоритм

обратного распространения ошибки (*backpropagation algorithm*).

Многослойная полносвязная ИНС прямого распространения является одной из наиболее известных структур нейронных сетей. Как было сказано выше, она состоит из множества сенсорных элементов (входных узлов или узлов источника), одного или нескольких скрытых слоев (*hidden layer*) вычислительных нейронов и одного выходного слоя (*output layer*) нейронов.. Такие сети еще называют многослойными персептронами (рис. 3.1).

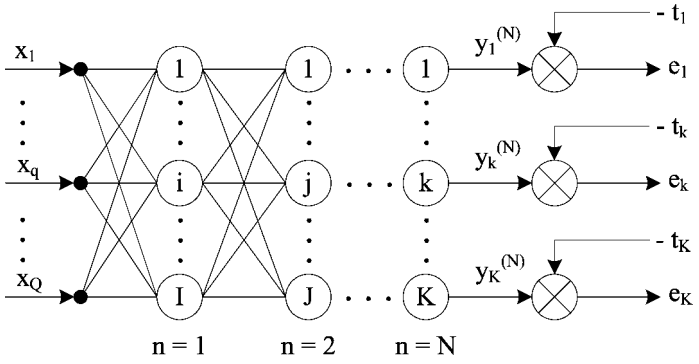


Рис. 3.1. Структура обучаемой сети

Многослойные персептроны имеют три отличительные признака:

1. Каждый нейрон имеет нелинейную функцию активации. Эта функция является гладкой (т.е. всюду дифференцируемой), в отличие от пороговой функции, используемой в персептроне Розенблатта. Чаще всего используется логистическая функция, удовлетворяющая этому требованию. Наличие нелинейности играет очень важную роль, так как в противном случае сеть можно свести к обычному однослойному персептрону.

2. Сеть содержит один или несколько слоев скрытых нейронов, не являющихся частью входа или выхода сети. Эти нейроны позволяют сети обучаться решению сложных задач, последовательно извлекая наиболее важные признаки из входного образа (вектора).

3. Сеть обладает высокой степенью связности, реализуемой посредством синаптических соединений.

Под обучением многослойного персептрона понимается процесс адаптации сети к предъявленным эталонным образцам путем модификации

весовых коэффициентов связей между нейронами. В однослойной сети алгоритм обучения с учителем очевиден, так как желаемые выходные состояния нейронов единственного выходного слоя t_k заведомо известны. Подстройка синаптических связей идет в направлении минимизации ошибки E на выходе сети. В многослойных сетях желаемое значение выходов нейронов всех внутренних слоев, кроме последнего, неизвестны. Поэтому многослойный персептрон уже невозможно обучить, руководствуясь только величинами ошибок на выходе ИНС.

Обучение многослойного персептрона методом обратного распространения ошибки предполагает два прохода по всем слоям сети: прямого и обратного. При прямом проходе образ (входной вектор) подается на сенсорные узлы сети, после чего распространяется по сети от слоя к слою. В результате генерируется набор выходных сигналов, который и является фактической реакцией сети на данный входной образ. Во время прямого прохода все синаптические веса сети фиксированы. Во время обратного прохода все синаптические веса сети настраиваются в соответствии с правилом коррекции ошибок, а именно: фактический выход сети вычитается из желаемого отклика, в результате чего формируется сигнал ошибки. Этот сигнал впоследствии распространяется по сети в направлении, обратном направлению синаптических связей. Отсюда и название – «алгоритм обратного распространения ошибки» или упрощенно – «алгоритм обратного распространения».

Для многослойного персептрона выделяют два типа сигналов:

1. Функциональный сигнал поступает в сеть в виде входного вектора признаков (стимула) и передается в прямом направлении от узла к узлу. Этот сигнал достигает последнего слоя сети в виде выходного вектора, то есть сеть реализует некоторую функциональную зависимость выхода от стимула.

2. Сигнал ошибки берет свое начало на выходе сети и распространяется в обратном направлении от слоя к слою. Он получил свое название благодаря

тому, что вычисляется каждым узлом сети на основе некоторой функции ошибки.

Каждый нейрон многослойного персептрона может выполнять два типа вычислений:

1. Вычисление функционального сигнала на выходе нейрона, реализуемого в виде непрерывной нелинейной функции от входного сигнала и синаптических весов, связанных с данным нейроном.

2. Вычисление оценки вектора градиента ошибки по синаптическим весам на входе данного нейрона. Эти значения необходимы для обратного прохода через сеть.

Минимизируемой целевой функцией ошибки ИНС является величина:

$$E(w) = \sum_{p=1}^{P} \sum_{k=1}^{K} (y_{k,p}^{(N)} - t_{k,p})^2 ,$$

где $y_{k,p}^{(N)}$ - выходное состояние нейрона k выходного слоя $n = N$ при подаче на вход образа номер p из обучающей выборки. Структура обучающей выборки имеет вид:

Обучающая выборка *Таблица 3.1*

Номер образа	Вход						Желаемый выход					
	x_1	x_2	…	x_q	…	x_Q	t_1	t_2	…	t_k	…	t_K
1												
…												
p												
…												
P												

где P – количество эталонных образов; Q- размерность векторов входных образов сети; K – количество нейронов в выходном слое сети.

Минимизация целевой функции ведется методом градиентного спуска, что означает подстройку весовых коэффициентов сети согласно дельта-правилу:

$$\Delta w_{ij}^{(n)} = -\eta \frac{\partial E}{\partial w_{ij}}, \text{ где}$$

$\Delta w_{ij}^{(n)}$ - приращение весового коэффициента синаптической связи, соединяющей *i*-й нейрон слоя *n-1* с *j*-м нейроном слоя *n*, η - скорость обучения (коэффициент), причем $0 < \eta < 1$.

Доказано, что $\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial y_j} \cdot \frac{dy_j}{ds_j} \cdot \frac{\partial s_j}{\partial w_{ij}}$, где s_j - взвешенная сумма входных сигналов нейрона *j*, т.е. аргумент активационной функции. Так как множитель dy_j/ds_j является производной этой функции по ее аргументу, из этого следует, что производная активационной функция должна быть определена на всей оси абсцисс. В связи с этим функция единичного скачка и прочие активационные функции с разрывами не подходят для рассматриваемых ИНС. В них применяются такие гладкие функции, как гиперболический тангенс или классическая сигмоидная функция (рис. 3.2)

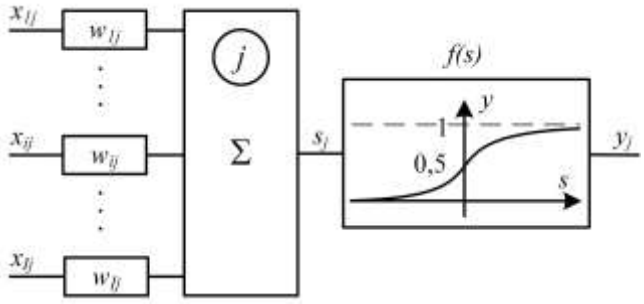


Рис. 3.2. Нейрон j промежуточного слоя

Третий множитель $\partial s_j/\partial w_{ij} = y_i^{(n-1)} = x_{ij}^{(n)}$, где

$y_i^{(n-1)}$ – выход нейрона *i* предыдущего слоя *n*, т.е. вход нейрона *j* слоя *n*.

Доказано, что первый множитель легко раскладывается следующим образом:

$$\frac{\partial E}{\partial y_j} = \sum_k \frac{\partial E}{\partial y_k} \cdot \frac{dy_k}{ds_k} \cdot \frac{\partial s_k}{\partial y_j} = \sum_k \frac{\partial E}{\partial y_k} \cdot \frac{dy_k}{ds_k} \cdot w_{jk}^{(n+1)}$$

Суммирование по *k* выполняется среди нейронов следующего слоя *n+1*. Введем новую переменную:

$$\delta_j^{(n)} = \frac{\partial E}{\partial y_j},$$

и получим рекурсивную формулу для расчетов величин $\delta_j^{(n)}$ слоя n из величин $\delta_k^{(n+1)}$ более старшего слоя $n+1$:

$$\delta_j^{(n)} = \sum_k \delta_k^{(n+1)} \cdot w_{jk}^{(n+1)} \tag{3.1}$$

Для выходного слоя :

$$\delta_k^{(N)} = (y_k^{(N)} - t_k) \tag{3.2}$$

Теперь можно записать в раскрытом виде :

$$\Delta w_{ij}^{(n)} = -\eta \cdot \delta_j^{(n)} \cdot \frac{dy_j}{ds_j} y_i^{(n-1)} \tag{3.3}$$

Чем меньше параметр скорости обучения η, тем меньше корректировка синаптических весов, осуществляемая на каждой итерации, и тем более гладкой является траектория в пространстве весов. Однако это улучшение происходит за счет замедления процесса обучения. С другой стороны, если увеличить параметр η для повышения скорости обучения, то результирующие большие изменения синаптических весов могут привести систему в неустойчивое состояние. Простейшим способом повышения скорости обучения без потери устойчивости является изменение дельта-правила (5.3) за счет добавления к нему коэффициента инерционности.

$$\Delta w_{ij}^{(n)}(t) = -(\mu \cdot \Delta w_{ij}^{(n)}(t-1) + \eta \cdot \delta_j^{(n)}(t) \cdot \frac{dy_j}{ds_j} \cdot y_i^{(n-1)}(t)), \tag{3.4}$$

где μ – коэффициент инерционности, t – номер текущего шага обучения. Уравнение (3.4) называют обобщенным дельта-правилом, при μ=0 оно вырождается в обычное дельта-правило.

Таким образом, полный алгоритм обучения ИНС с помощью процедуры обратного распространения ошибки строится так:

1. Подать на входы сети один из возможных образов и в режиме обычного функционирования ИНС, когда сигналы распространяются от входов к выходам, рассчитать значения последних:

$$s_j^{(n)} = \sum_{i=0}^{I} y_i^{(n-1)} \cdot w_{ij}^{(n)},$$

где I – число нейронов в слое n-1 с учетом того, что

$y_i^{(n-1)}=x_{ij}^{(n)}$ – i-ый вход нейрона j слоя n;

$y_j^{(n)} = f(s_j^{(n)})$, где $f()$ – активационная функция нейрона;

$y_q^{(0)}=x_q$, где x_q – q-ая компонента вектора входного образа.

2. Вычислить $\delta_k^{(N)}$ для всех нейронов выходного слоя по формуле (3.2). Рассчитать по формуле (3.3) изменения весовых коэффициентов $\Delta w_{jk}^{(n)}$ слоя $n=N$.

3. Вычислить по формулам (3.1) и (3.3) величины $\delta_j^{(n)}$ и $\Delta w_{ij}^{(n)}$ для всех остальных слоев $n<N$ $(n=N-1,...1)$.

4. Скорректировать все весовые коэффициенты ИНС

$$w_{ij}^{(n)}(t) = w_{ij}^{(n)}(t-1) + \Delta w_{ij}^{(n)}(t)$$

5. Если ошибка сети, вычисленная по всем P эталонным образцам существенна, перейти на шаг 1, в противном случае – «конец».

Нейронной сети на шаге 1 попеременно в случайном порядке предъявляются все тренировочные образы, чтобы ИНС «не забывала» одни по мере запоминания других. Алгоритм расчета значений $\delta_k^{(N)}$ в выходном слое представлен на рисунке 3.3.

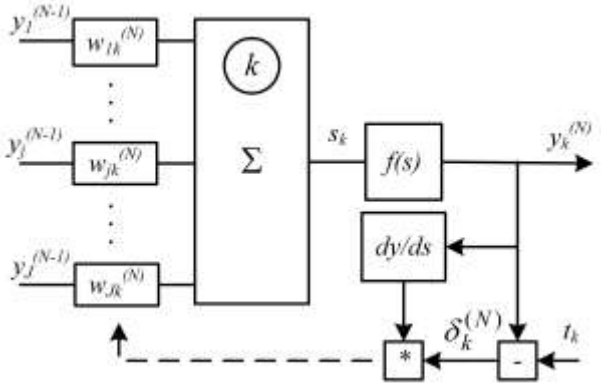


Рис. 3.3. Схема расчета сигнала обратного распространения в выходном слое ($n=N$)

Алгоритм расчета значений $\delta_j^{(n)}$ при обратном распространении в

промежуточных слоях показан на рисунке 3.4.

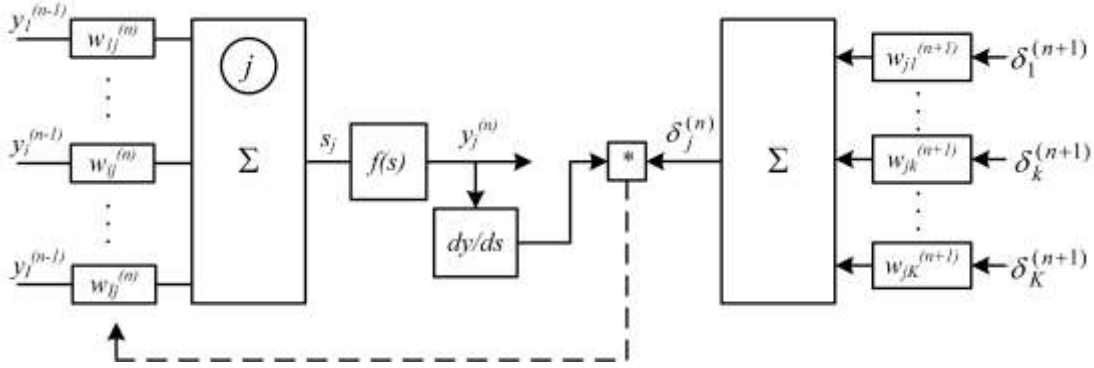


Рис. 3.4. Схема расчета сигнала обратного распространения в промежуточном слое ($n < N$)

Комментарии к алгоритму обратного распространения

Пример расчета ошибки

Для обновления весовых коэффициентов связей в скрытых слоях ИНС необходимо знать величины ошибок для нейронов этих слоев. Ошибка — это разность между желаемым выходным значением нейрона и его фактическим выходным значением (3.2). В обучающей выборке есть только целевые значения для нейронов последнего, выходного слоя (табл. 3.1). Желаемые выходные значения для нейронов скрытых слоев неизвестны. Поэтому для нейронов скрытых слоев сети нужно иметь правило расчета их ошибок.

Расчет ошибок внутренних слоев осуществляется путем реализации обратной связи по ошибке выходного слоя и распространения ее в обратном направлении — от выхода ко входу сети (рис 3.4). При этом исходят из того, что ошибка на выходе нейрона скрытого слоя представляет собой сумму ошибок, распределенных по всем связям, исходящим из этого нейрона в прямом направлении. Рассмотрим процесс обратного распространения ошибки на примере трехслойной ИНС с двумя выходными нейронами (рис 3.5).

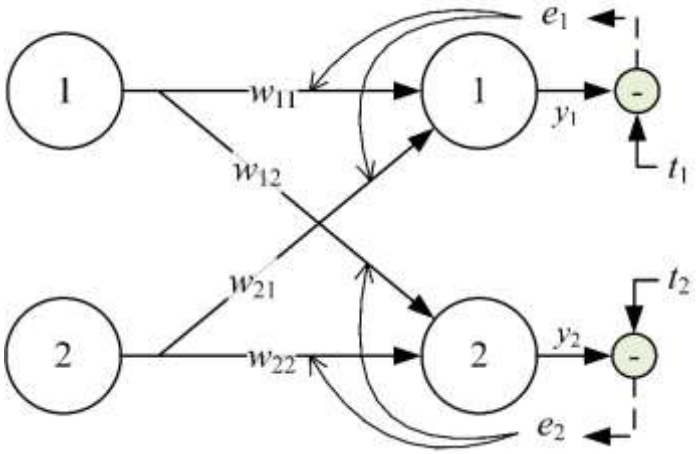


Рис 3.5 Пропорциональное распределение ошибки

Ошибка может формироваться на обоих выходных нейронах $e_k=y_k-t_k$, где k=1,2. Для расчета ошибки в скрытых слоях ИНС используется правило распределения ошибки e_k между связанными нейронами предыдущего слоя пропорционально весовым коэффициентам соответствующих связей. Это простое правило заключается в том, что нейроны, сделавшие больший вклад в ошибочный ответ, получают большую долю ошибки, а нейроны, сделавшие меньший вклад, получают меньшую долю. Учитывая, что связи одного выходного нейрона не зависят от связей другого, расчет ошибок можно осуществлять отдельно для каждого e_k.

Тогда, доля ошибки e_1 в расчете поправки весов w_{11} и w_{21} будет определяться следующим образом:

$\delta_{11}= e_1 \cdot w_{11}/(w_{11}+w_{21})$, $\delta_{21}= e_2 \cdot w_{21}/(w_{11}+w_{21})$.

Выражения в знаменателе этих формул играют роль нормирующего множителя, поэтому для упрощения записи им пренебрегают. При этом изменяется лишь масштабирующий коэффициент ошибки, передаваемый по обратной связи:

$\delta_{11}= e_1 \cdot w_{11}$, $\delta_{21}= e_2 \cdot w_{21}$.

Расчет значений δ_{12} и δ_{22} для поправок весовых коэффициентов w_{12} и w_{22} не зависит от предыдущих расчетов:

$\delta_{12}= e_2 \cdot w_{12}$, $\delta_{22}= e_2 \cdot w_{22}$.

Увеличение количества выходных нейронов принципиально не усложняет задачу расчета соответствующих значений δ_{jk}. Таким образом, имеется некоторая доля выходной ошибки e_1 приписываемая связи с весом w_{11}, и некоторая доля выходной ошибки e_2, приписываемая связи с весом w_{12}. Тогда ошибка на выходе первого нейрона скрытого слоя будет определяться суммой этих долей:

$\delta_1 = \delta_{11} + \delta_{12} = e_1 \cdot w_{11} + e_2 \cdot w_{12}$

Аналогично ошибка на выходе второго нейрона скрытого слоя:

$\delta_2 = \delta_{21} + \delta_{22} = e_1 \cdot w_{21} + e_2 \cdot w_{22}$

На рис. 3.6 приведен пример расчета значений ошибки на выходе нейронов скрытого слоя.

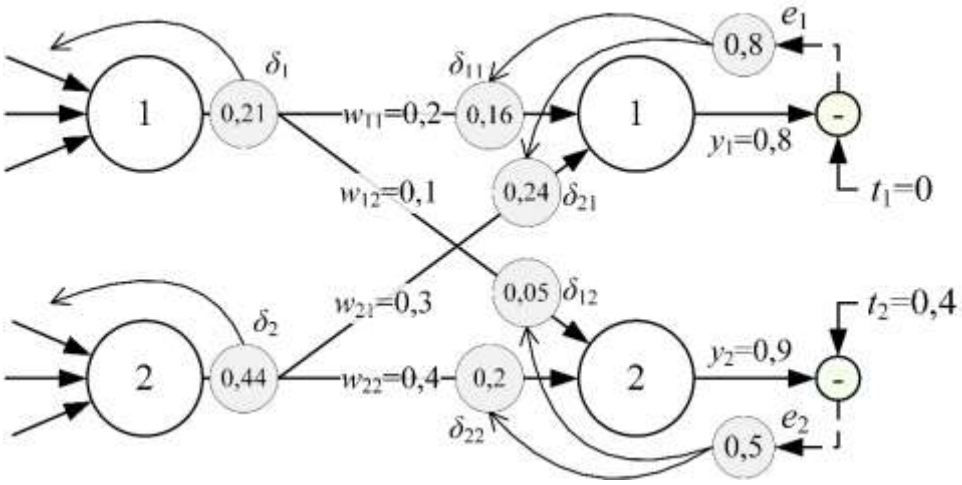


Рис 3.6 Пример расчета ошибок в скрытом слое ИНС

Расчет ошибок нейронов предыдущих слоев ИНС осуществляется по этой методике вплоть до входного слоя сети.

Для трехслойной ИНС с двумя входами, и распределением нейронов по слоям 3-2-2 схема расчета функционального сигнала показана на рис.3.7.

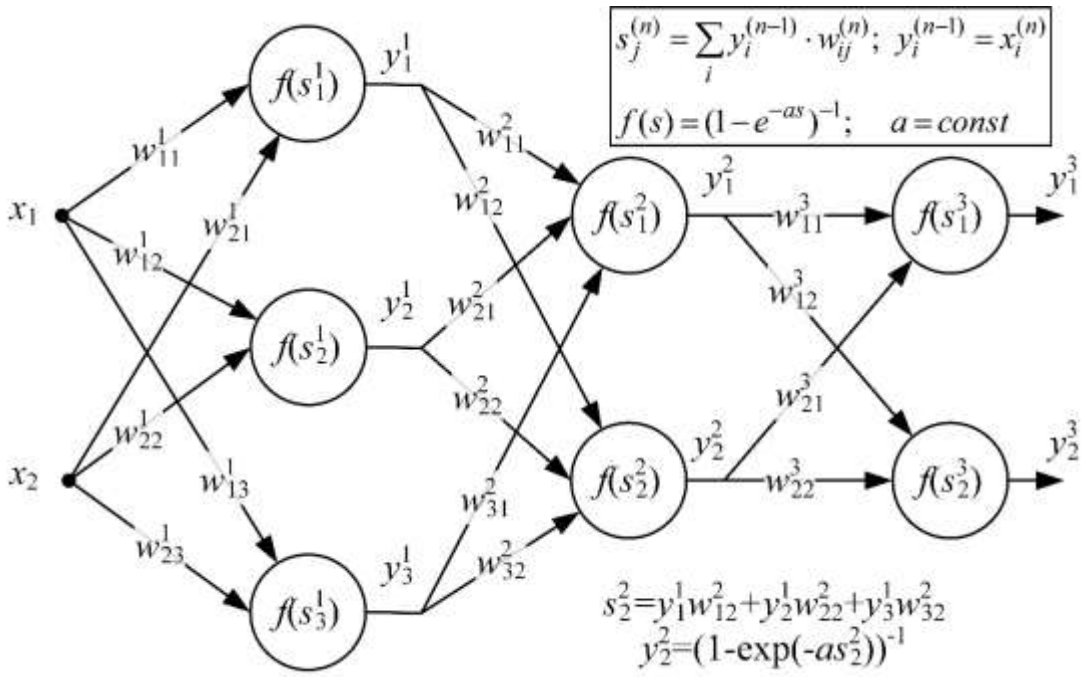


Рис. 3.7 Расчет функционального сигнала

Ведем следующие обозначения:

$\mathbf{X} = (x_1\ x_2)$, $\mathbf{S}_1 = \left(s_1^1\ s_2^1\ s_3^1\right)$, $\mathbf{Y}_1 = f\left(s_1^1\ s_2^1\ s_3^1\right) = \left(y_1^1\ y_2^1\ y_3^1\right)$, $\mathbf{S}_2 = \left(s_1^2\ s_2^2\right)$,

$\mathbf{Y}_2 = f\left(s_1^2\ s_2^2\right) = \left(y_1^2\ y_2^2\right)$, $\mathbf{S}_3 = \left(s_1^3\ s_2^3\right)$, $\mathbf{Y}_3 = f\left(s_1^3\ s_2^3\right) = \left(y_1^3\ y_2^3\right)$

$$\mathbf{W}_1 = \begin{pmatrix} w_{11}^1 & w_{12}^1 & w_{13}^1 \\ w_{21}^1 & w_{22}^1 & w_{23}^1 \end{pmatrix},\ \mathbf{W}_2 = \begin{pmatrix} w_{11}^2 & w_{12}^2 \\ w_{21}^2 & w_{22}^2 \\ w_{31}^2 & w_{32}^2 \end{pmatrix},\ \mathbf{W}_3 = \begin{pmatrix} w_{11}^3 & w_{12}^3 \\ w_{21}^3 & w_{22}^3 \end{pmatrix}.$$

Тогда в матричном виде расчет функционального сигнала можно записать следующим образом:

$\mathbf{S}_i = \mathbf{Y}_{i-1}\mathbf{W}_i$, $\mathbf{Y}_0 = \mathbf{X}$, $\mathbf{Y}_i = f(\mathbf{S}_i)$, где i=1, 2, 3.

Расчет значений ошибки во внутренних слоях для этой ИНС показан на рис. 3.8.

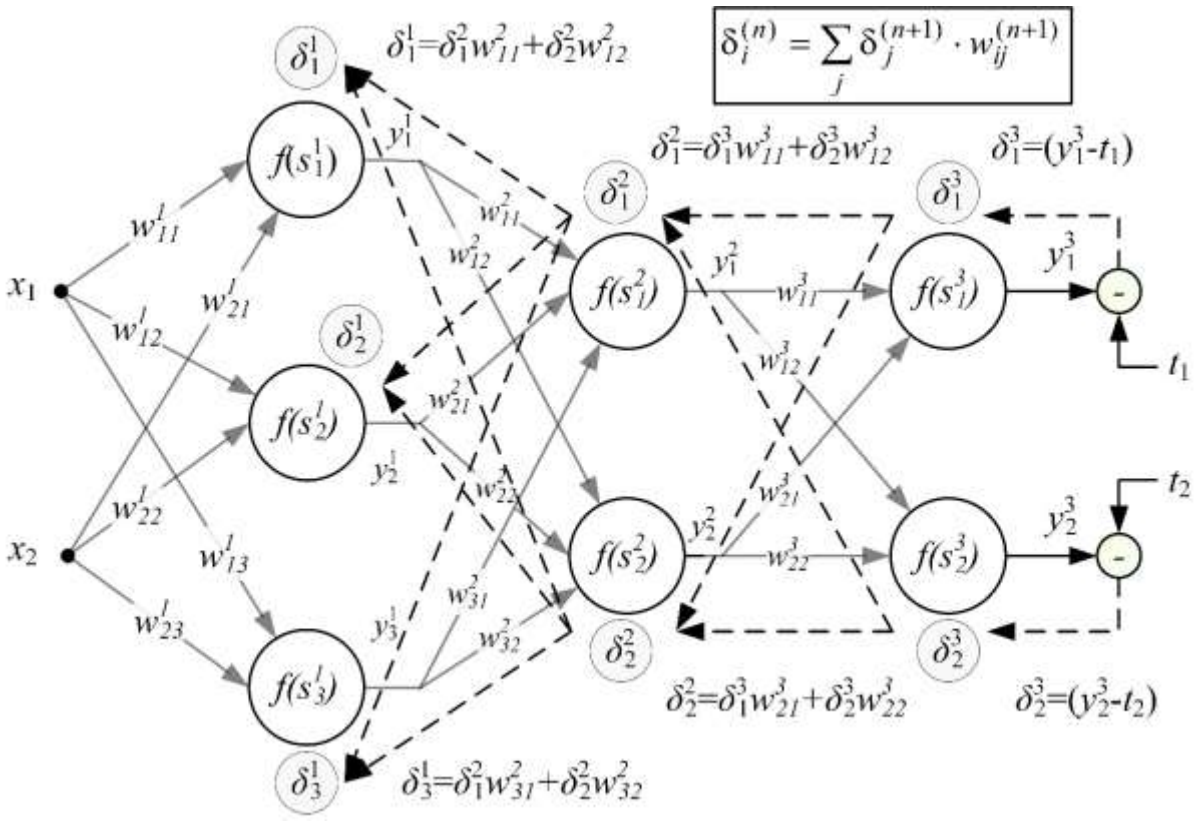


Рис 3.8. Расчет сигнала ошибки

Ведем следующие обозначения:

$\boldsymbol{\delta}_1 = \left(\delta_1^1\ \delta_2^1\ \delta_3^1\right)$, $\boldsymbol{\delta}_2 = \left(\delta_1^2\ \delta_2^2\right)$, $\boldsymbol{\delta}_3 = \left(\delta_1^3\ \delta_2^3\right)$.

Тогда в матричном виде расчет сигнала ошибки можно записать следующим образом:

$\boldsymbol{\delta}_{i-1} = \boldsymbol{\delta}_i W_i^{\mathrm{T}}$, $\boldsymbol{\delta}_3 = (e_1\ e_2)$, где $e_k = y_k^3 - t_k$, где i=1, 2, k=1, 2.

Расчет корректирующих значений для весовых коэффициентов во всех трех слоях ИНС показан н рис. 3.8

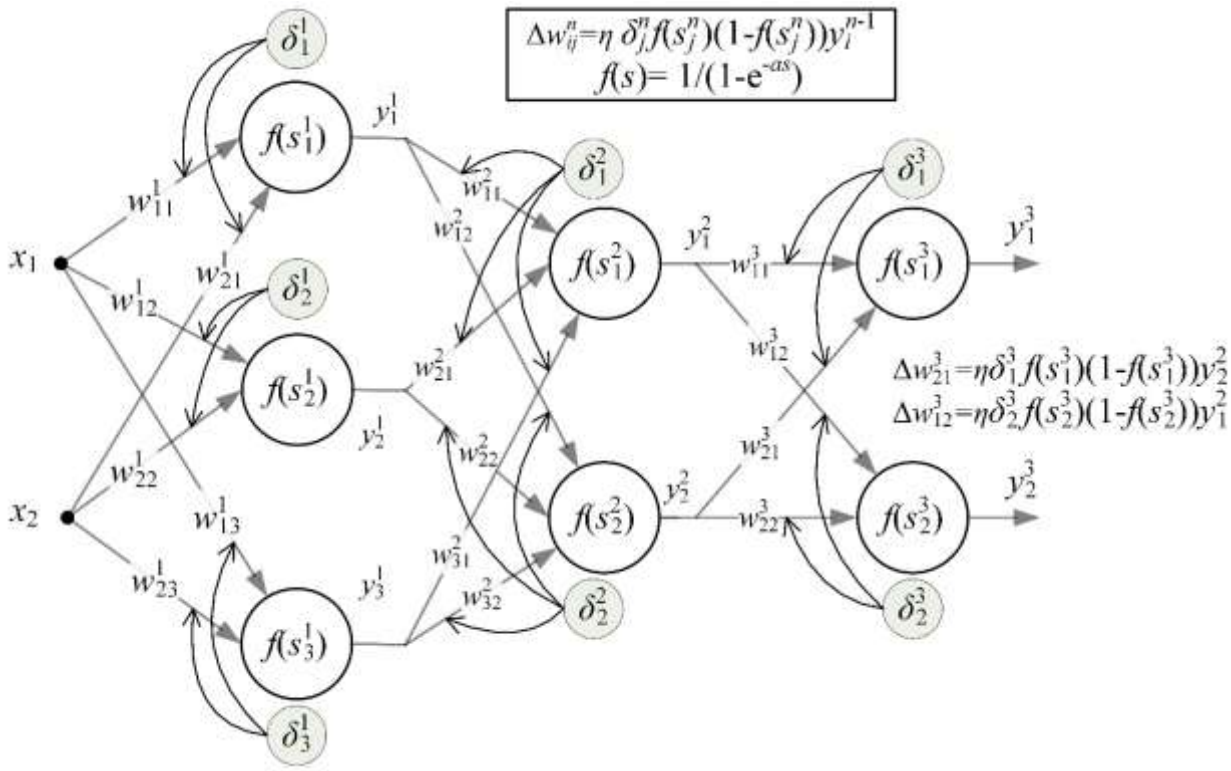


Рис. 3.8 Расчет приращений весовых коэффициентов

Выбор значения скорости обучения

Целью обучения ИНС является уменьшение заданной функции ошибки. Это означает, что нужно решить задачу оптимизации: по заданной функции найти аргументы (весовые коэффициенты), при которых эта функция минимизируется. Задача усложняется тем, что для большой ИНС целевая функция — это очень сложная композиция других функций.

Для выпуклых функций задача локальной оптимизации автоматически превращается в задачу глобальной оптимизации — в нахождение точки, в которой достигается наименьшее значение функции, то есть самой глубокой «ямы». В этой точке производная функции ошибки обращается в ноль. У многослойных ИНС, функция ошибки может иметь очень сложный ландшафт с большим числом локальных максимумов и минимумов. Формально это значит, что производная от функции ошибки (градиент) обращается в ноль много раз в разных точках.

Если гарантировать точное решение задачи оптимизации оказывается

невозможно, то можно попытаться найти наилучший вариант из доступных - выбрать не первый попавшийся, а «приемлемый» минимум функции. Для этого предназначен метод оптимизации, называемый градиентным спуском. Почти все современные методы оптимизации невыпуклых функций представляют собой те или иные его варианты. Градиентный спуск — это движение в направлении по той касательной к поверхности функции, которая наклонена вниз сильнее всего. Аналогия со скатывающимся вниз шариком заключается в том, что шарик будет катиться именно в сторону наибольшего наклона поверхности. Цель такого движения состоит в том, чтобы скатиться по поверхности функции к некоторому локальному минимуму, точке экстремума оптимизируемой функции. Реализация дискретной алгоритмической процедуры градиентного спуска не обеспечит точного попадания в минимум, но процедуру можно остановить в тот момент, когда изменения градиента станут совсем маленькими.

Важный параметр градиентного спуска — это его скорость обучения. Этот параметр регулирует размер шага, с которым совершается движение в направлении градиента. Выбор скорости обучения сильно влияет на результат оптимизации. В частности, могут возникнуть две нежелательные ситуации (рис. x):

• если шаги будут слишком маленькими, то обучение будет слишком долгим, и повышается вероятность застрять в небольшом локальном минимуме функции;

• если шаги будут слишком большими, можно бесконечно «прыгать» через искомый минимум взад-вперед, но так и не прийти в самую нижнюю точку.

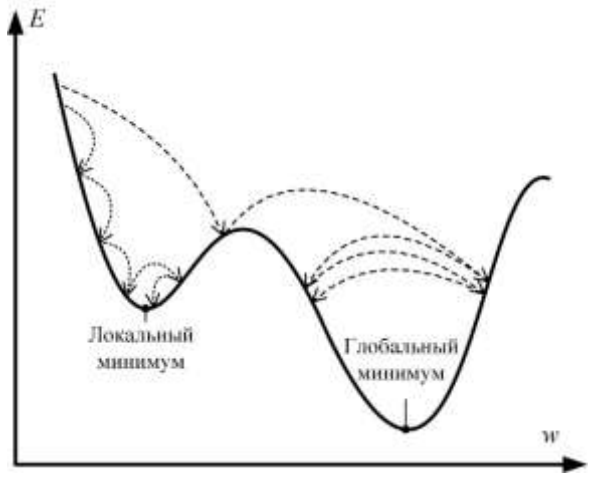


Рис. 3.9 Влияние скорости обучения на результат оптимизации функции ошибки *E*.

Рассмотрим, как метод градиентного спуска применяется для решения задачи обучения многослойной ИНС. Предположим, что существует обучающая выборка (табл. 3.1). Модель ИНС с весами w_{ij} на этих данных делает некоторые предсказания – y_k и задана функция ошибки e_k, которую можно вычислить на каждом примере. Это может быть квадрат или модуль отклонения (y_k-t_k) или перекрестная энтропия.

В любом случае общая функция ошибки E будет суммой ошибок на каждом тренировочном примере. Тогда приращения весовых коэффициентов ИНС на шаге t пропорциональны градиенту ошибки ΔE:

$$w_{ij}(t)= w_{ij}(t-1)+\eta\Delta E$$

Получается, чтобы сделать один шаг градиентного спуска, нужно, перебрать все обучающее множество, а оно может быть очень большим в реальных задачах. Поэтому на практике используется стохастический градиентный спуск, в котором ошибка подсчитывается и веса подправляются не после прохода по всему обучающему множеству, а после каждого примера:

$$w_{ij}(t)= w_{ij}(t-1)+\eta\Delta e$$

Преимущество стохастического градиентного спуска состоит в том, что ошибка на каждом шаге считается быстрее, а веса меняются сразу же, что

ускоряет процесс обучения. Содержательное преимущество стохастического варианта градиентного спуска заключается в переборе большего количества вариантов параметров w_{ij} в пространстве поиска, что позволяет надеяться на нахождение минимума функции ошибки E, близкого к оптимальному. На практике используется также вариант стохастического градиентного спуска по мини-батчам (*mini-batch*), когда ошибка E вычисляется по небольшим подмножествам обучающей выборки.

Подготовка обучающей выборки

Подготовка обучающих данных заключается в выборе правил масштабировании значений этих данных, в планировании схемы выходных сигналов и присвоении случайных начальных значений весовым коэффициентам ИНС для обеспечения благоприятных условий процедуре ее обучения.

Выбирая диапазон изменения входных сигналов нужно иметь в виду выражение для корректировки весовых коэффициентов (3.3). Оно зависит от градиента функции активации. Использование небольших значений градиента равносильно ограничению способности сети к обучению. Это называется насыщением (параличом) нейронной сети. Отсюда следует, что для входных сигналов лучше задавать небольшие значения. При этом их нельзя задавать и слишком малыми, поскольку они тоже входят в указанное выражение (3.4). Рекомендуемым решением является масштабирование входных сигналов таким образом, чтобы их значения оставались в интервале [0, 1]. Иногда для входных сигналов вводят небольшое смещение, порядка 0,01 с целью недопущения нулевых значений.

Целевые выходные значения ИНС — это значения в правой части обучающей выборки, которые появляются на нейронах последнего слоя. Логистическая функция активации асимптотически приближается к значения 0 при отрицательных аргументах и к значению 1 при положительных аргументах. Поэтому нужно назначать целевые выходные значения таким

образом, чтобы они были допустимыми при данной функции активации, избегая значений, которые никогда не могут быть достигнуты. Общепринято использовать диапазон значений [0, 1], но некоторые авторы рекомендуют использовать диапазон [0,01 0,99], поскольку при стремлении к достижению целевых значений 0 и 1 можно получить чрезмерно большие значения весовых коэффициентов.

Аналогичная аргументация применима и для выбора начальных значений весовых коэффициентов. Следует избегать больших начальных значений весовых коэффициентов, поскольку использование функции активации в этой области значений может приводить к насыщению сети. Один из возможных вариантов — прибегнуть к выбору случайных числовых значений, распределенных по равномерному закону в некотором диапазоне. Существуют эмпирические правила для задания случайных начальных значений весовых коэффициентов в зависимости от конкретной конфигурации сети и используемой функции активации.

Одно из этих правил заключается в том, что грубая оценка границ диапазона изменения весовых коэффициентов определяется обратной величиной квадратного корня из количества связей, ведущих к нейрону. Например, если к нейрону ведут четыре связи, то начальные значения весов не должны превышать значений ±0,5, а если же нейрон имеет 100 входящих связей, то веса должны находиться в диапазоне [-0,1 0,1]. Понятно, что чем больше связей приходится на нейрон, тем больше складывается весовых коэффициентов. Поэтому правило, которое уменьшает диапазон допустимых значений весов с увеличением количества связей на узел, имеет логическое объяснение. Это правило не является универсальным.

В любом случае нельзя задавать для начальных весов равные значения, особенно нулевые. Равные веса приводят к одинаковым поправкам для всех весовых коэффициентов, что, в свою очередь, вновь приведет к весам, имеющим одинаковые значения. Если правильно обученная сеть должна

иметь разные значения весовых коэффициентов (что характерно для большинства задач), то это состояние никогда не будет достигнуто. Таким образом, общая рекомендация заключается в том, что значения весов внутренних связей должны быть случайными и небольшими, но не нулевыми.

Обобщение и перекрестная проверка

При обучении методом обратного распространения в ИНС подают примеры из обучающей выборки и настраивают ее синаптические веса. При этом предполагают, что обученная сеть будет способна к обобщению. Считается, что ИНС обладает хорошей обобщающей способностью, если отображение входа на выход, осуществляемое ею, является корректным для данных, никогда ранее не «виденных» сетью в процессе обучения. При этом считается, что примеры для проверки обобщающей способности ИНС принадлежат той же совокупности, из которой они брались для обучения.

Основой для решения такой задачи служит стандартный статистический подход, получивший название перекрестной проверки (*cross-validation*). В рамках этого подхода имеющиеся в наличии данные сначала случайным образом разбиваются на обучающее множество (*training set*) и тестовое множество (*test set*).

Смысл этого подхода состоит в проверке качества модели на данных, отличных от использованных для обучения, чтобы избежать переобучения ИНС. Свойство переобученности (*overfitting*) сети заключается в том, что она правильно реагирует на примеры из обучающей выборки, но относительно плохо работает на примерах, не участвовавших в обучении (на примерах из тестовой выборки). Иными словами - ИНС запоминает все возможные примеры вместо того, чтобы научиться подмечать в них закономерности.

4. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ АЛГОРИТМА ОБРАТНОГО РАСПРОСТРАНЕНИЯ ОШИБКИ

Рассмотрим пример программной реализации алгоритма обратного распространения ошибки на языке *Python*. Для начала опишем класс двуслойной ИНС, который должен содержать три основные функции:

• инициализация — задание размерности входа, количества скрытых и выходных нейронов, присвоение начальных значений весовых коэффициентов;

• прямое распространение — получение значений сигналов с выходных нейронов после предоставления значений входящих сигналов.

• обратное распространение — корректировка весовых коэффициентов в процессе предъявления сети тренировочных примеров;

«Скелет» программного кода может иметь следующий вид:

```
# определение класса нейронной сети
class neuralNetwork:
 # инициализировать нейронную сеть
 def _init_():
      pass
 # прямое распространение
 def query():
      pass
 # обратное распространение
 def train():
      pass
```

Инициализация сети

Необходимо задать размерность входного вектора (*inodes*), количество нейронов в скрытом (*hnodes*) и выходном (*onodes*) слоях. Эти данные определяют конфигурацию ИНС. Вместо того чтобы жестко задавать их в коде, предусмотрим установку соответствующих значений в виде параметров во время создания объекта нейронной сети. Благодаря этому можно будет создавать новые экземпляры ИНС. Также предусмотрим еще один параметр - коэффициент скорости обучения (*lr)*:

```
    # инициализировать нейронную сеть
    def _init_(self, inputnodes, hiddennodes, outputnodes,
learningrate):
        # задать количество узлов входа и количество нейронов
      в скрытом и выходном слое
        self.inodes = inputnodes
      self.hnodes = hiddennodes
      self.onodes = outputnodes
      # коэффициент обучения
      self.lr = learningrate
```

Весовые коэффициенты сети

Весовые коэффициенты связей (веса) используются при расчете распространения сигналов в прямом направлении и обратном распространении ошибок, и именно весовые коэффициенты корректируются в процессе обучения сети. Используем матричную форму записи и создадим следующие матрицы весов:

- матрицу весов **Wih** для связей между входным и скрытым слоями, размерностью *hnodes* x *inodes*;
- матрицу весов **Who** для связей между скрытым и выходным слоями, размерностью *onodes* x *hnodes*.

Начальные значения весовых коэффициентов должны быть небольшими и выбираться случайным образом. Функция из библиотеки *numpy* генерирует массив случайных чисел в диапазоне от 0 до 1, где размерность массива равна `rows` x `columns`:

```
numpy.random.rand(rows, columns).
```

Для использования библиотеки *numpy*, необходимо импортировать ее в самом начале программы:

```
import numpy as np
```

Веса должны иметь не только положительные, но и отрицательные значения и изменяться в малых пределах. Вычтем 0,5 из генерируемых случайных значений, перейдя к диапазону [-0,5 +0,5]. Весовые коэффициенты после инициализации должны быть доступными для других методов, таких как функции прямого и обратного распространения.

Ниже приведен программный код, который создает две матрицы весовых коэффициентов **Wih** и **Who**, используя значения переменных `self.inodes`, `self.hnodes` и `self.onodes` для задания соответствующих размеров каждой из них.

```
self.wih=(np.random.rand(self.hnodes, self.inodes)-0.5)
self.who=(np.random.rand(self.onodes, self.hnodes)-0.5)
```

Выходной сигнал каждого нейрона является взвешенной суммой его входных сигналов, преобразованной функцией активации. В качестве функции активации в этом примере используется сигмоида с единичным коэффициентом наклона $y=1/(1-e^{-x})$. Определим ее один раз в объекте ИНС во время его инициализации. После этого мы сможем ссылаться на нее. Библиотека *scipy* в содержит набор специальных функций, в том числе сигмоиду, которая называется *expit* ().

Библиотека *scipy* импортируется точно так же, как и библиотека *numpy*.

```
# библиотека scipy.special содержит сигмоиду expit()
 import scipy.special
```

Ниже приведен программный код, определяющий функцию активации, который будет использован в разделе инициализации нейронной сети.

```
# использование сигмоиды в качестве функции активации
    self.activation_function = lambda x:
 scipy.special.expit(x)
```

Функция создана с использованием короткого способа записи, называемого лямбда-выражением. Функция принимает аргумент *x* и возвращает значение `scipy.special.expit`(), что и есть сигмоида. Эту функцию нужно будет вызвать по ее имени `self.activation_function`().

Тогда программный код функции инициализации ИНС будет иметь вид:

```
        # инициализировать нейронную сеть
    def _init_(self, inputnodes, hiddennodes, outputnodes,
learningrate):
          # задать количество узлов входа, нейронов в скрытом и
```

```
выходном слое
    self.inodes = inputnodes
self.hnodes = hiddennodes
self.onodes = outputnodes

    # Матрицы весовых коэффициентов связей wih  (между #
входным и скрытым слоями)
    # и who  (между скрытым и выходным слоями).
    self.wih=(np.random.rand(self.hnodes, self.inodes)-0.5)
    self.who=(np.random.rand(self.onodes, self.hnodes)-0.5)

# коэффициент обучения
self.lr = learningrate
# использование сигмоиды в качестве функции активации
self.activation_function       =       lambda       x:
scipy.special.expit(x)
pass
```

Прямое распространение сигнала

Функция *query*() принимает в качестве аргумента входные данные нейронной сети и возвращает ее выходные значения. Для этого нужно передать сигналы от входных узлов через скрытый слой нейронов к нейронам выходного слоя. Можно получить взвешенные входные сигналы для нейронов скрытого слоя **hidden_inputs** путем скалярного умножения матрицы весов **Wih** на матрицу-столбец входных сигналов **inputs**:

Ниже представлена инструкция, которая показывает, как применить функцию скалярного произведения библиотеки *numpy* к матрицам весов и входных сигналов:

```
hidden_inputs = numpy.dot(self.wih, inputs)
```

Для получения выходных сигналов нейронов скрытого слоя к каждому из них применяется функция активации:

```
# рассчитать выходные сигналы нейронов скрытого слоя
hidden_outputs                                          =
self.activation_function(hidden_inputs)
```

Таким образом, выходные сигналы скрытого слоя описываются матрицей-столбцом **hidden_outputs**.

Распространение сигнала от скрытого слоя до выходного ничем

принципиально не отличается от предыдущего случая, поэтому способ расчета остается тем же, а значит, и программный код будет аналогичен предыдущему. Ниже приведен итоговый фрагмент программного кода, объединяющий расчеты прямого распространения сигналов по слоям сети.

```
# прямое распространение
 def query(self, inputs_list):
# преобразовать список входных значений
# в двухмерный массив
inputs = numpy.array(inputs_list, ndmin=2).T

# рассчитать входящие сигналы для скрытого слоя
 hidden_inputs = numpy.dot(self.wih, inputs)
# рассчитать исходящие сигналы для скрытого слоя
hidden_outputs = self.activation_function(hidden_inputs)

# рассчитать входящие сигналы для выходного слоя
final_inputs = numpy.dot(self.who, hidden_outputs)
# рассчитать исходящие сигналы для выходного слоя
final_outputs = self.activation_function(final_inputs)

return final_outputs
```

Для функции *query*() в качестве входных данных требуются только входные сигналы ИНС - **input_list**.

Обратное распространение (тренировка сети)

Задачу тренировки сети можно разделить на две части.

• Первая часть — расчет выходных сигналов для заданного тренировочного примера с помощью функции *query*().

• Вторая часть — сравнение рассчитанных выходных сигналов с желаемым ответом и обновление весов связей между нейронами на основе значений ошибки.

Программный код первой части почти совпадает с кодом функции *query*(), поскольку процесс передачи сигнала от входного слоя к выходному остается одним и тем же.

Единственным отличием является введение дополнительного параметра, передаваемого при вызове функции - `targets_list` – массива

желаемых ответов на каждый предъявленный тренировочный пример:

```
def train(self, inputs_list, targets_list)
```

Обратное распространение начинается с вычисления разности между желаемым выходным значением для тренировочного примера, поданного на вход сети, и фактическим выходным значением сети. Это разность между матрицами-столбцами **targets - final_outputs**, рассчитываемая поэлементно. Соответствующий программный код имеет вид:

```
# ошибка = целевое значение - фактическое значение
    output_errors = targets - final_outputs
```

Затем необходимо рассчитать матрицу столбец ошибок для нейронов скрытого слоя **hidden_errors**. Ошибки распределяются между нейронами пропорционально весам связей **Who**, а затем они рекомбинируются на каждом нейроне скрытого слоя. Эти вычисления можно представить в следующей матричной форме:

$$\mathbf{hidden\_errors = Who^T * output\_errors}$$

Программный код реализует вычисление скалярного произведения матриц с помощью модуля *numpy*.

```
# ошибки скрытого слоя - это ошибки output_errors,
# распределенные пропорционально весовым
# коэффициентам связей
# и рекомбинированные на скрытых нейронах
    hidden_errors = numpy.dot(self.who.T, output_errors)
```

Значения ошибок будут нужны для корректировки весовых коэффициентов в каждом слое. Для корректировки весов связей между скрытым и выходным слоями используются значения **output_errors**. Для корректировки весов связей между входным и скрытым слоями используются только что рассчитанные значения **hidden errors**.

В выражение для обновления веса связи между узлом j и узлом k следующего слоя входит производная функции активации. Выражение для производной сигмоиды $f(s)$ определяется простой формулой: $f'(s)=f(s)(1-f(s))$.

Тогда выражение для корректировки весов преобразуем в программную инструкцию для обновления весов связей между скрытым и

выходным слоями:

```
    # обновить весовые коэффициенты связей между скрытым и
выходным слоями
        self.who += self.lr * np.dot((output_errors *
    final_outputs * (1.0 - final_outputs)),
    np.transpose(hidden_outputs))
```

Операция += означает увеличение переменной, указанной слева от знака равенства, на значение, указанное справа от него. Величина `self.lr` — это коэффициент обучения, матричное умножение выполняется с помощью функции `np.dot()`, `transpose(hidden_outputs)` - транспонированная матрица выходных сигналов предыдущего слоя, символ "*" означает обычное поэлементное умножение.

Программный код для корректировки весов связей между входным и скрытым слоями будет построен по этой же схеме:

```
    # обновить весовые коэффициенты связей между входным и
скрытым слоями
    self.wih += self.lr * np.dot((hidden_errors *
hidden_outputs * (1.0- hidden_outputs)), np.transpose(inputs))
```

Тогда программный код функции тренировки сети будет иметь следующий вид:

```
      # обратное распространение - тренировка нейронной
  сети
      def train(self, inputs_list, targets_list):
          # преобразование списка входных значений
          # в двухмерный массив
      inputs = numpy.array(inputs_list, ndmin=2).T
      targets = numpy.array(targets_list, ndmin=2).T

      # рассчитать входящие сигналы для скрытого слоя
      hidden_inputs = numpy.dot(self.wih, inputs)
      # рассчитать исходящие сигналы для скрытого слоя
      hidden_outputs = self.activation_function(hidden_inputs)

      # рассчитать входящие сигналы для выходного слоя
      final_inputs = numpy.dot(self.who, hidden_outputs)
      # рассчитать исходящие сигналы для выходного слоя
      final_outputs = self.activation_function(final__inputs)

          # ошибки выходного слоя =
      #  (целевое значение - фактическое значение)
      output_errors = targets - final_outputs
          # ошибки скрытого слоя - это ошибки output_errors,
```

```
        # распределенные пропорционально весовым
    коэффициентам связей
    # и рекомбинированные на скрытых узлах
    hidden_errors = numpy.dot(self.who.T,
    output_errors)

        # обновить веса для связей между скрытым и выходным
    слоями
        self, who += self.lr * numpy .dot ((output  errors *
    final_outputs *  (1.0 - final_outputs)),
    numpy.transpose(hidden_outputs))

        # обновить весовые коэффициенты для связей между
        # входным и скрытым слоями
        self.wih += self.lr * numpy.dot((hidden
    errors * hidden_outputs *  (1.0 - hidden_outputs)
    ),  numpy.transpose(inputs))
    pass
```

Обучение нейронной сети распознаванию рукописных цифр

Набор рукописных цифр MNIST

Существует типовая коллекция изображений рукописных цифр, используемых разработчиками алгоритмов распознавания в качестве набора для тестирования своих программных реализаций. Этим тестовым набором является база данных изображений рукописных цифр под названием “MNIST”, собранных Национальным институтом стандартов и технологий США (National Institute of Standards and Technology — часть NIST в аббревиатуре MNIST) и предоставляемая для бесплатного всеобщего доступа.

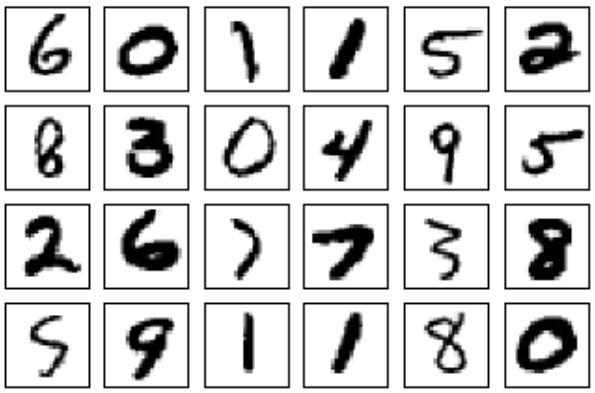


Рис. 4.1 Примеры рукописных цифр из набора MNIST

Базы данных MNIST существует в разных форматах. В этом примере используются данные, в формате *CSV*- файлов, в которых отдельные значения представляют собой обычный текст, разделенный запятыми.

Полный тренировочный набор содержит десятки тысяч размеченных экземпляров, используемых для тренировки НС. Термин «размеченный» означает, что для каждого экземпляра указан соответствующий правильный ответ - маркер. Другой тестовый набор, включающий примерно 10 000 экземпляров, используется для проверки правильности работы алгоритмов распознавания. Он также содержит корректные маркеры, позволяющие увидеть, способна ли обученная ИНС дать правильный ответ. Использование независимых друг от друга наборов тренировочных и тестовых данных гарантирует, что с тестовыми данными ИНС ранее не сталкивалась.

Каждое изображение представлено в виде строки текста, которая содержит числа, разделенные запятыми:

• Первое значение — это маркер, т.е. фактическая цифра, например, «7» или «9», соответствующая данному рукописному изображению. Это правильный ответ, который должна сформировать обученная ИНС.

• Последующие числа, разделенные запятыми, — это значения яркости пикселей изображения рукописной цифры в градациях серого. Массив имеет размерность 28x28, поэтому за каждым маркером следуют 784 целых значений из диапазона (0 – 255).

Для того, чтобы увидеть, каким образом список из 784 значений формирует изображение рукописной цифры нужно преобразовать его в массив, состоящий из 28 строк и 28 столбцов и отобразить его. Используем подмножество набора данных, содержащихся в базе MNIST для пояснения программной реализации алгоритма обучения ИНС. Сформируем два файла в формате *CSV*:

• 100 записей из тренировочного набора данных MNIST: `mnist_train_l00.csv`

• 10 записей из тестового набора данных MNIST: `mnist_test_10.csv`

Открытие файлов и получение их содержимого:

```
data_file=open("mnist_dataset/mnist_train_100.csv", 'r')
data_list = data_file.readlines()
data_file.close()
```

Первая строка открывает файл с помощью функции `open()`. В качестве первого из параметров ей передается имя файла, включая каталог, в котором он находится. Второй параметр (буква `'r'`) означает, что файл открывается только для чтения.

Функция `open()` создает дескриптор (указатель), играющий роль ссылки на открываемый файл, которая присваивается переменной `data_file`. Когда файл открыт, любые последующие действия с ним осуществляются через этот дескриптор.

Далее используется функция `readlines()`, ассоциированная с дескриптором файла `data_file`, для чтения всех строк содержимого файла в переменную `data_list`. Эта переменная содержит список, каждый элемент которого является строкой из прочитанного файла. Можно переходить к любой строке файла так же, как к конкретным записям в списке. Таким образом, `data_list[0]` — это первая запись, а `data_list[9]` —десятая и т.п.

Функция `readlines()` считывает весь файл в память. Эффективнее работать поочередно с каждой строкой, а не считывать в память весь файл целиком. Однако используемые файлы имеют небольшое количество записей, и применение функции `readlines()` значительно упрощает программный код. Последняя строка закрывает файл. Длина полученного списка равна 100.

Чтобы увидеть рукописный вариант цифры нужно преобразовать список чисел, разделенных запятыми, в массив 28x28. Это можно сделать в соответствии со следующей процедурой:

• разбить длинную текстовую строку значений, разделенных запятыми, на отдельные значения, используя символ запятой в качестве разделителя;

• проигнорировать первое значение, являющееся маркером, извлечь следующие 784 значения и преобразовать их в массив, состоящий из 28 строк и 28 столбцов;

• отобразить массив.

Предварительно необходимо импортировать библиотеки расширений *Python* для работы с массивами и графикой.

```
import numpy as np
import matplotlib.pyplot
%matplotlib inline
```

Рассмотрим три следующие инструкции:

```
all_values = data_list[0].split(',')
image_array = np.asfarray(all_values[1:]).reshape((28,28))
matplotlib.pyplot.imshow(image_array, cmap='Greys',
interpolation ='None')
```

В первой строке длинная первая запись `data_list[0]` разбивается на отдельные значения с использованием запятой в качестве разделителя. Это делается с помощью функции `split()`, параметр которой определяет символ-разделитель. Результат помещается в переменную `all_values`. Разбор следующей строки кода начнем с середины. Запись списка в виде `all_values[1:]` указывает на то, что берутся все элементы списка за исключением первого. Тем самым игнорируется первое значение - маркер, и выделяются остальные 784 элемента, `np.asfarray()` — это функция библиотеки *numpy*, преобразующая текстовые строки в числа и создающая массив этих чисел. Последний фрагмент инструкции — `.reshape((28,28))` — означает, что список будет сформирован в виде квадратной матрицы размером 28x28. Результирующий массив такой размерности получает название `image_array`.

Инструкция в третьей строке просто выводит на экран массив

`image_array` с помощью функции `imshow()`. Используется цветовая палитра оттенков серого с помощью параметра `cmap='Greys'`. Графическое изображение цифры «2» соответствует маркеру (рис. 4.2).

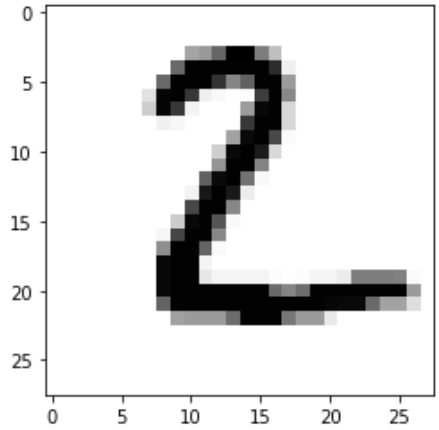


Рис.4.2 Вывод графического изображения рукописной цифры

Подготовка тренировочных данных MNIST для обучения

Для обучения ИНС необходимо продумать структуру тренировочных данных. Во-первых, следует провести нормировку значений яркости пикселей – привести все значения к диапазону от 0 до 1. Причем рекомендуется избегать нулевых значений сигналов и в качестве нижней границы диапазона выбирать малое положительное число, например, 0.01. Это поможет избежать проблем с обновлением весов. Деление исходных значений, изменяющихся в диапазоне 0-255, на 255 приведет их к диапазону 0-1. Последующее умножение этих значений на коэффициент 0.99 и смещение на 0.01, приведет их к желаемому диапазону 0.01-1.0. Эти действия реализуется следующей инструкцией:

```
scaled_input=(np.asfarray(all_values[1:])/255.0*0.99)+0.01
```

Т.о. осуществлена подготовка левой части обучающей выборки - входных данных MNIST путем их масштабирования и сдвига. Теперь можно подавать их на вход ИНС как с целью ее тренировки, так и с целью тестирования.

Далее нужно сформировать правую часть обучающей выборки -

желаемые выходные значения ИНС. Эти значения должны укладываться в диапазон, обеспечиваемый функцией активации. Значения логистической функции (сигмоиды) лежат в диапазоне от 0.0 до 1.0. Причем фактически никогда предельные значения 0.0 или 1.0 не достигаются, поскольку сигмоида лишь асимптотически приближается к ним. Определим структуру правой части обучающей выборки.

В процессе обучения ИНС от нее требуется классифицировать изображение и присвоить ему корректный маркер. Таким маркером может быть одно из десяти чисел в диапазоне от 0 до 9. Это означает, что выходной слой сети должен иметь 10 нейронов, по одному на каждый возможный ответ (маркер). Если ответом является «0», то активизироваться должен первый нейрон, тогда как остальные нейроны должны оставаться пассивными. Если ответом является «9», то активизироваться должен последний нейрон выходного слоя при пассивных остальных нейронах. Следующая иллюстрация поясняет эту схему на нескольких примерах выходных значений.

Пример кодирования значений на выходе ИНС *Таблица* 4.1.

Нейроны выходного слоя	маркер	пример «5»	пример «0»	пример «9»
1	0	0.00	0.99	0.02
2	1	0.01	0.00	0.01
3	2	0.00	0.00	0.01
4	3	0.00	0.01	0.42
5	4	0.01	0.00	0.00
6	5	0.99	0.02	0.01
7	6	0.00	0.00	0.00
8	7	0.00	0.00	0.01
9	8	0.02	0.01	0.02
10	9	0.01	0.02	0.85

Первый пример соответствует случаю, когда сеть распознала входные данные как цифру «5». Наибольший из исходящих сигналов выходного слоя

принадлежит нейрону с меткой «5». Остальные нейроны формируют сигналы, близкие к нулю.

Последний пример иллюстрирует другую ситуацию. Здесь самый большой сигнал генерирует последний нейрон, соответствующий метке «9». Однако и нейрон с меткой «4» дает значимый сигнал средней величины. Обычно ИНС должна принимать решение, основываясь на наибольшем сигнале, но, в данном случае она отчасти считает, что правильным ответом могло бы быть и «4». Такого рода неопределенности встречаются в ответах ИНС, и должны трактоваться как возможность существовании другого ответа. Следовательно, целевые массивы правой части обучающей выборки должны содержать 10 чисел, из которых малы все, кроме одного, соответствующего маркеру примера.

Например, желаемый ответ ИНС для маркера «5» должен представлять массив [0.01, 0.01, 0.01, 0.01, 0.01, 0.99, 0.01, 0.01, 0.01, 0.01].

Инструкции, создающие целевую матрицу для текущей строки входных данных, выглядят следующим образом:

```
# количество выходных нейронов - 10
onodes = 10
targets = np.zeros(onodes) + 0.01
targets[int(all_values[0])] = 0.99
```

Первая строка после комментария устанавливает количество выходных нейронов равным 10, что соответствует нашему примеру с десятью маркерами. Во второй строке с помощью функции `np.zeros()` создается массив, заполненный нулями и к каждому элементу массива добавляется 0.01. Следующая строка выбирает первый элемент записи из текущей строки набора данных MNIST, являющийся целевым маркером тренировочного набора, и преобразует его в целое число, которое используется для того, чтобы установить значение соответствующего элемента массива равным 0.99.

Полностью программный код этого раздела приведен в приложении А.

Тестирование нейронной сети

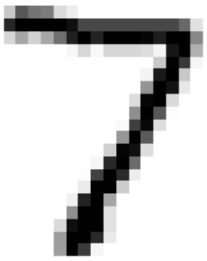
Проверяется работоспособность обученной ИНС на тестовом наборе данных. Загрузка тестовых записей:

```
    # загрузить в список тестовый набор данных CSV-файла набора
MNIST
        test_data_file =
     open("mnist_dataset/mnist_test_10.csv", 'r')
     test_data_list = test_data_file.readlines()
     test_data_file.close()
```

Ниже представлены результаты выполнения одиночного теста путем опроса уже обученной нейронной сети с использованием первой записи тестового набора данных.

```
# тестирование нейронной сети
# получить первую тестовую запись
  all_values = test_data_list[0].split(',')
# вывести маркер
  print(all_values[0])
image_array=numpy.asfarray(all_values[1:]).reshape((28,28))
matplotlib.pyplot.imshow(image_array, cmap='Greys',
interpolation='None')
n.query((numpy.asfarray(all_values[1:])/255.0*0.99)+0.01)
```

```
7
```



```
array([[0.09160385],
       [0.03291914],
       [0.06713181],
       [0.10217017],
       [0.12075188],
       [0.04449184],
       [0.01833258],
       [0.74872497],
       [0.09724058],
       [0.09487999]])
```

Рис. 4.3 Пример одиночного теста обученной ИНС

В качестве маркера первой записи тестового набора сеть определила

символ «7». Графическое отображение пиксельных значений подтверждает, что рукописной цифрой действительно является цифра «7».

В результате опроса обученной сети получен список чисел, являющихся выходными значениями каждого нейрона последнего слоя. Одно из этих значений намного превышает остальные, и этому значению соответствует маркер «7». Это восьмой элемент списка, поскольку первому элементу соответствует маркер «0».

Затем необходимо проверить, насколько хорошо ИНС распознает остальную частью набора данных, и определить количество правильных и ошибочных результатов:

```
# тестирование нейронной сети
# журнал оценок работы сети, первоначально пустой scorecard
= []
# перебрать все записи в тестовом наборе данных
for record in test_data_list:
# получить список значений из записи, используя
# символы запятой (',') в качестве разделителей
 all_values = record.split(',')
# правильный ответ - первое значение
correct_label = int(all_values[0])
print(correct_label, "истинный маркер")
# масштабировать и сместить входные значения
inputs = (np.asfarray(all_values[1:]) / 255.0 * 0.99) + 0.01
# опрос сети
outputs = n.query(inputs)
# индекс наибольшего значения является маркерным значением
label = np.argmax(outputs)
print(label, "ответ сети")
# добавить оценку ответа сети к концу списка
 if (label == correct_label) :
# в случае правильного ответа сети добавить
# к списку значение 1
scorecard.append(1)
else:
# в случае неправильного ответа сети добавить
# к списку значение 0
scorecard.append(0)
pass
pass
```

Пустой список `scorecard` служит журналом оценок работы сети, обновляемым после обработки каждой записи. В цикле извлекаются значения

из текстовой записи, в которой они разделены запятыми. Первое значение, указывающее правильный ответ, сохраняется в отдельной переменной `correct_label`. Остальные значения масштабируются, чтобы их можно было использовать в качестве входных данных для передачи ИНС. Ответ ИНС сохраняется в переменной `outputs`.

Далее функция `np.argmax()` находит среди элементов массива максимальное значение и сообщает его индекс. В последнем фрагменте кода полученное значение сравнивается с правильным ответом, и если они совпадают, то в журнал записывается «1», в противном случае — «0». Ниже представлены результаты выполнения этого программного кода вместе с выведенными записями рабочего журнала.

```
7 истинный маркер
7 ответ сети
2 истинный маркер
0 ответ сети
1 истинный маркер
1 ответ сети
0 истинный маркер
0 ответ сети
4 истинный маркер
4 ответ сети
1 истинный маркер
1 ответ сети
4 истинный маркер
4 ответ сети
9 истинный маркер
4 ответ сети
5 истинный маркер
4 ответ сети
9 истинный маркер
7 ответ сети
```

```
print(scorecard)
```

```
[1, 0, 1, 1, 1, 1, 1, 0, 0, 0]
```

Рис. 4.4 Пример протокола тестирования обученной ИНС

Последняя выведенная строка результатов показывает, что из десяти тестовых записей правильно были распознаны 6. Таким образом, доля правильных результатов составила 60%. Для небольшой размера тренировочного набора это неплохой результат.

Матрица неточностей

Задача распознавания рукописных цифр является типичной задачей многоклассовой классификации. Успешность результатов классификации оценивается по ряду показателей. Правильность (*accuracy*) классификации оценивается по следующей формуле:

правильность = правильные заключения / все заключения

В приведенном выше примере *accuracy*=0,6. Эта формула дает грубую оценку эффективности и не подразумевает разделения результатов на правильные и неправильные для каждого класса. Более детальную оценку эффективности классификации дает матрица неточностей (*confusion matrix*).. В качестве примера рассмотрим бинарный классификатор в виде «детектора пятерок». Этот классификатор распознает среди рукописных цифр только одну цифру – «5», относя все остальные к классу «не пятерка».

Матрица неточностей (рис. 4.5) представляет собой таблицу, которая сравнивает ответ классификатора с правильным ответом. Анализируются результаты отнесения каждой цифры к определенному классу и определяется доля неправильных решений.

Рис. 4.5 Матрица неточностей

В бинарной классификации выходная переменная имеет два возможных значения: T (true) или F (false). Результаты классификации можно разделить на категории:

• Истинно положительные результаты (*TP*). Эго случаи, для которых результат классификации - *T* (объект – «5») совпадает с правильным ответом – *T* (этот объект действительно «5»).

• Истинно отрицательные результаты (*TN*). Эго случаи, для которых результат классификации - *F* (объект – «не 5») совпадает с правильным ответом - *F* (этот объект действительно «не 5»).

• Ложноположительные результаты (*FP*). Эго случаи, для которых результат классификации – *T* (объект – «5») не совпадает с правильным ответом – *F* (этот объект на самом деле «не 5»). Такая ситуация известна как ошибка 1 рода

• Ложноотрицательные результаты (*FN*). Эго случаи, для которых результат классификации – F (объект – «не 5») не совпадает с правильным ответом – *T* (этот объект на самом деле «5»). Такая ситуация известна как ошибка 2 рода.

Эта матрица носит название матрицы неточностей, потому что позволяет оценить, как часто классификатор путает два класса при попытке разделения исходных данных. Безупречный классификатор имел бы только истинно положительные и истинно отрицательные результаты, так что его матрица неточностей содержала бы ненулевые значения только на своей главной диагонали.

Результаты анализа реального классификатора отражаются в следующих ключевых показателях. Отношение истинно положительных результатов к суммарно положительным носит название точности (*precision*). Точность является мерой вероятности того, что положительный ответ окажется правильным:

$$\text{точность} = TP/(TP + FP)$$

Отношение истинно положительных результатов ко всем возможным положительным результатам носит название полноты (*recall*). Это отношение показывает долю истинно положительных решений.

$$\text{полнота} = TP/(TP + FN)$$

Точность и полноту часто объединяют в единый показатель под названием мера F_1 (F_1 score) - это среднее гармоническое (*harmonic mean*) точности и полноты:

F_1= 2/(1/точность+1/полнота) = $FP/(TP+2/(FN+FP))$

Классификатор получает высокое значение меры F_1 только если высокими являются значения и полноты, и точности. Полезно также оценить эффективность классификатора, проверив его для каждого класса.