Содержание

Введение

С давних времен человечество старалось разгадать загадки устройства нашего мозга, понять как он работает, как приходит к выводу о том, какие объекты относятся к тому или иному классу и создать нечто, что могло бы анализировать информацию и принимать решения, как это присуще человеку.

Большие успехи в решении этой проблемы произошли во второй половине XX века, когда были разработаны различные алгоритмы машинного обучения. Суть этих алгоритмов заключается в анализе большого количества данных (обучающая выборка) и способности на их основе делать прогноз для новых данных (тестовая выборка).

В этой же работе будет рассматриваться такой принцип машинного обучения как нейронная сеть, которая лучшим образом приближена к реальной модели человеческого мозга.

Прежде, чем начинать изучение нейронных сетей, рассмотрим саму концепцию машинного обучения. Для этого, на примере линейного классификатора, разберем процесс обучения алгоритма по данным и дадим графические иллюстрации, демонстрирующие состояние его параметров.

Затем, изучив нейрон как отдельный элемент, рассмотрим, как устроена сеть из нейронов, как они между собой связаны, как обучить нейтронную сеть методом обратного распространения ошибок, как подготовить все параметры и входные данные перед обучением нейронной сети.

Изучив все эти аспекты, построим нейтронную сеть, задача которой будет заключена в распознавании рукописного текста, а именно цифр.

1. Анализ предметной области

Рассмотрим линейный классификатор, являющийся одним из наиболее известных и простых для понимания алгоритмов машинного обучения. Его суть заключается в том, что бы построить некую разделяющую плоскость (прямую, если речь идет о двумерном пространстве) и определить, к каким классам относятся объекты, лежащие в этом пространстве.

Предположим, что необходимо классифицировать грузовые и легковые автомобили по их размеру и средней скорости передвижения. Это значит, что, зная эти параметры, есть возможность соотнести тот или иной автомобиль к классу легковых или же к классу грузовых автомобилей. Выбор всего лишь двух признаков обусловлен тем, что они лучшим образом помогают отличить один класс от другого и с ними можно работать в двумерном пространстве.

Для наглядности построим диаграмму (рис. 1.1), иллюстрирующую расположение объектов по этим признакам. Красным отмечены грузовые автомобили, а зеленым - легковые.

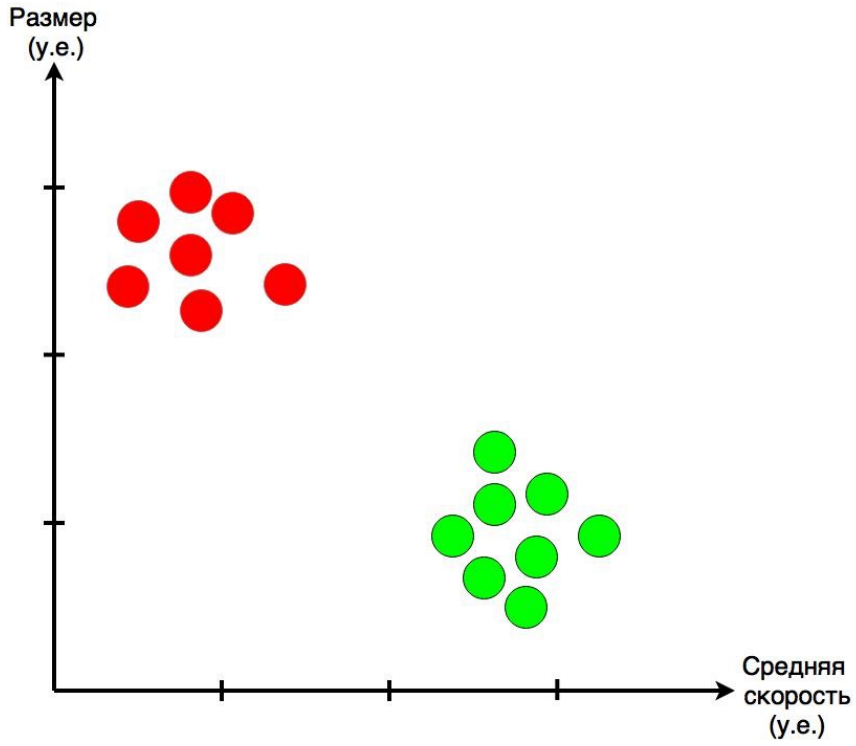

Рис. 1.1. Параметры автомобилей

Как видно из иллюстрации, автомобили, имеющие меньший размер и большую среднюю скорость отнесены к легковым (зеленые кружки), и, наоборот, автомобили с большим размером, но меньшей средней скоростью принадлежат классу грузовых машин (красные кружки).

Логично предположить, что границу между классом грузовых и легковых автомобилей необходимо провести таким образом, что бы она находилась ровно между объектами обоих классов (рис. 1.2). По факту, эта прямая и будет являться линейным классификатором, где объекты, лежащие выше нее принадлежат классу грузовых автомобилей, а ниже - легковых.

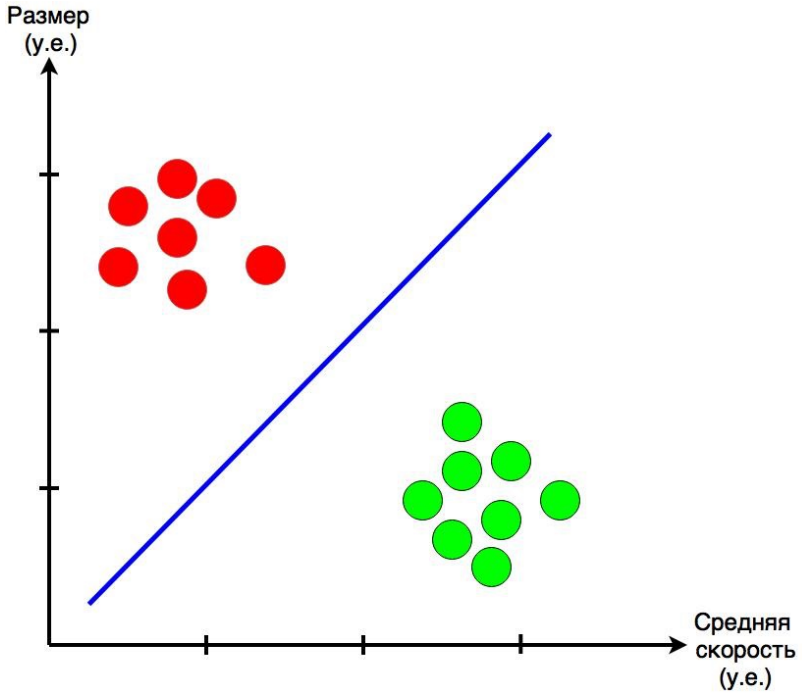

Рис. 1.2. Классификация автомобилей

Говоря об алгоритмах машинного обучения, важно понимать, как происходит их обучение.

Предположим, что перед нами нет готового линейного классификатора, но при этом имеются размеченные данные в виде некой таблицы, в которой указаны размер, средняя скорость и метка класса, показывающая, к какому классу относится данный автомобиль. Такая форма данных называется обучающей выборкой. Для более серьезных задач необходимы достаточно

большие обучающие выборки, но для простоты нашего примера остановимся на двух (таблица 1).

Таблица 1.

«Обучающая выборка»

№	Размер (условные единицы)	Скорость (условные единицы)	Класс
1	3.0	1.0	Грузовой автомобиль
2	1.0	3.0	Легковой автомобиль

Изобразим эти данные на диаграмме (рис. 1.3).

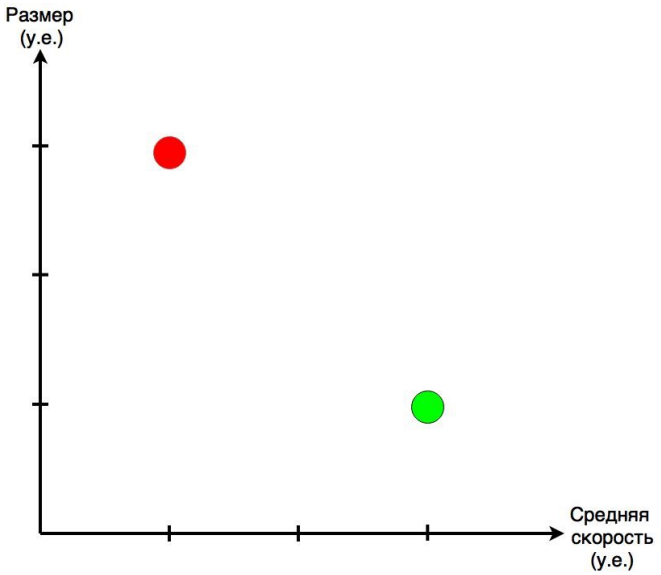

Рис. 1.3. Объекты обучающей выборки

Из курса математики известно, что уравнение прямой выглядит следующим образом (1.1):

$$y = Ax + B \quad (1.1)$$

В нашем примере положим параметр B равный нулю, так как это довольно сильно затруднит построение разделяющей линии и не имеет практического применения (конктретно для этого примера), так как основная задача данного раздела - изучить основную идею того, как обучается алгоритм.

С чего же начать построение разделяющей линии? В первую очередь дадим некое случайное значение параметра A (например 0.25), которое в последствии и будем изменять (рис. 1.4).

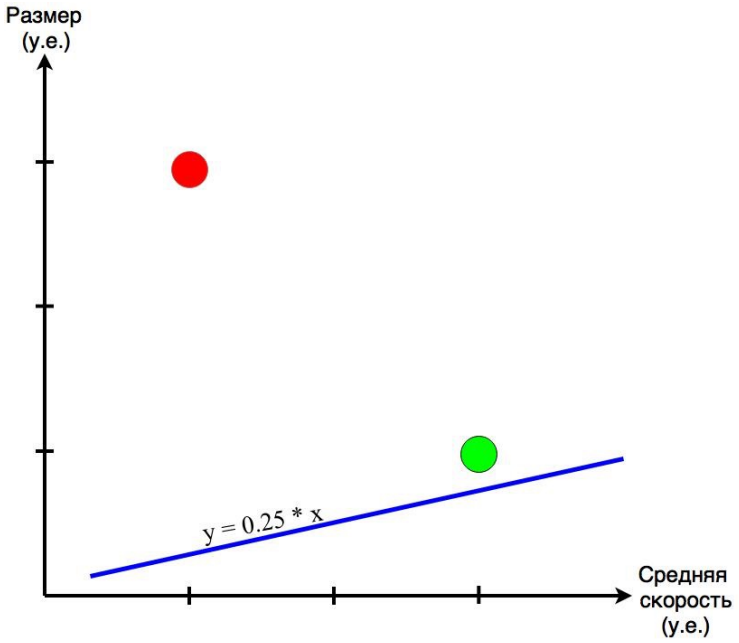

Рис. 1.4. Разделяющая плоскость при случайном значении A

Из диаграммы видно, что данная разделяющая плоскость плохо выполняет свою задачу. Для того, что бы исправить это, воспользуемся обучающей выборкой.

Пусть x - средняя скорость автомобиля, а y - его размер. С точки зрения математики ничего не изменилось, просто с такими обозначениями удобней работать в дальнейшем.

Возьмем первый объект и сравним то значение y, которое даст уравнение (1.1) при A = 0.25.

$$y = 0.25 * 3.0 = 0.75$$

В результате, значение y далеко от необходимого значения скорости для данного объекта, равного единице. Таким образом, при A = 0.25 была получена ошибка, равная 0.25 (то, что она получилась равной значению параметра A - чистое совпадение).

Используем полученную ошибку для обновления коэффициента A. Ошибка сама по себе представляет разницу между истинным значением функции и тем, что дало уравнение с данными параметрами.

Предположим, что мы хотим, чтобы разделяющая линия проходила над первым объектом и положим истинное значение функции равным 1.1, и тогда ошибка будет равна 0.35 (рис. 1.5).

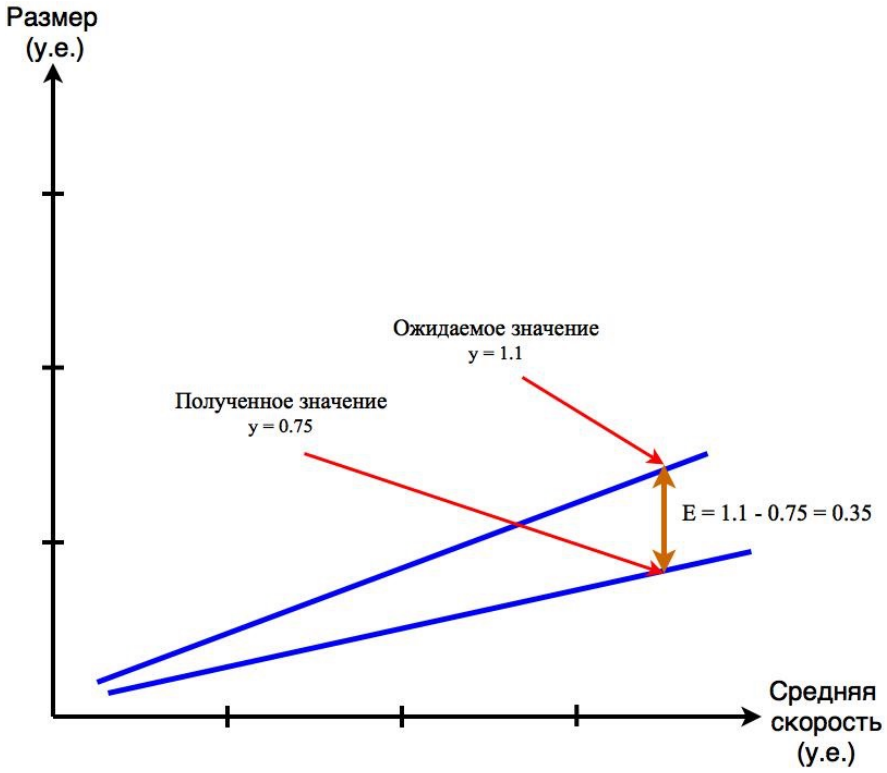

Рис. 1.4. Ошибка между ожидаемым и полученным значениями

Обозначим ошибку буквой E. и разберем, из чего она из чего она состоит.

Пусть t - истинное или искомое значение функции. Тогда, значение, на которое необходимо изменить параметр A, для получения t обозначим как ΔA. Учитывая все эти обозначения получим формулу (1.2).

$$t = (A + \Delta A) * x \qquad (1.2)$$

Вычтем из уравнения (1.2) уравнение (1.1) и определим, чему равно значение ошибки E.

$$t - y = (A + \Delta A) * x - A * x$$

$$t - y = A * x + \Delta A * x - A * x$$

$$t - y = \Delta A * x$$

Учитывая, что по определению, ошибка - это разность между искомым и полученными значениями, то

$$E = t - y.$$

$$E = \Delta A * x$$

$$\Delta A = E / x \qquad (1.3)$$

Таким образом, было получено выражение (1.3) для нахождения ΔA, на которое необходимо изменить параметр A в уравнении (1.2).

Вернемся к нашему примеру и рассчитаем новый параметр A.

$$\Delta A = 0.35 / 3.0 = 0.1167$$

$$A = 0.25 + 0.1167 = 0.3367$$

Итак, при рассмотрении первого объекта обучающей выборки было получено значение параметра A, равное 0.3367. Изобразим новую разделяющую линию на диаграмме (рис. 1.6). Теперь первый объект находится под прямой, что, чисто в теории, правильно.

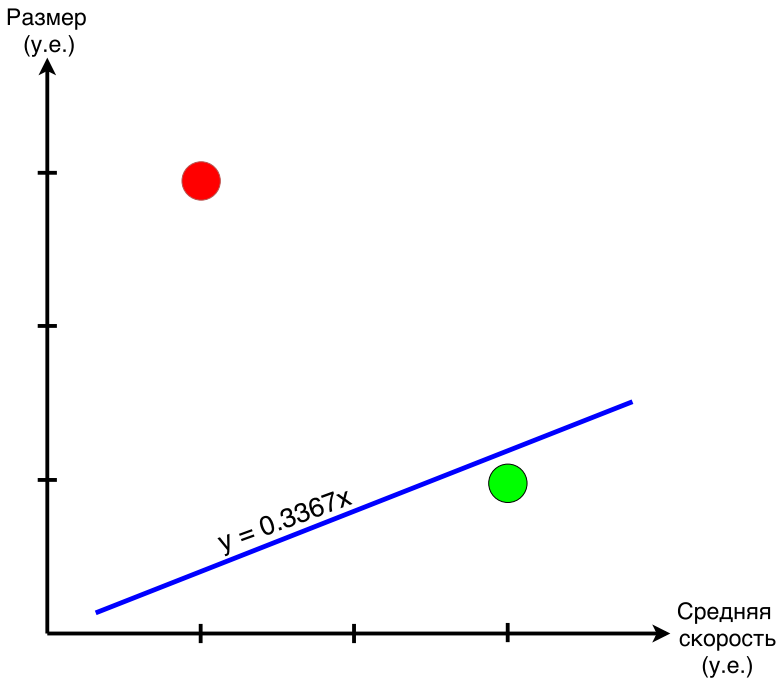

Рис. 1.6. Состояние классификатора после обработки первого объекта

Возьмем второй объект обучающей выборки и рассчитаем значение y, используя обновленные параметры.

$$y = 0.3367 * 1.0 = 0.3367$$

В итоге получился результат, сильно отличающийся от ожидаемого ответа 3.0. Предположим, что ожидаемое значение не 3.0, а 2.9, исходя из тех же рассуждений, что и в предыдущем случае. На этот раз необходимо, что бы линия была под данным объектом. Найдем ошибку для этого случая.

$$E = 2.9 - 0.3367 = 2.5333$$

Аналогичным образом найдем ΔA и рассчитаем новый параметр A.

$$\Delta A = 2.5333 / 1.0 = 2.5333$$

$$A = 0.3367 + 2.5333 = 2.9$$

Изобразим прямую, описывающую уравнение (1.2) с новым параметром A (рис. 1.7).

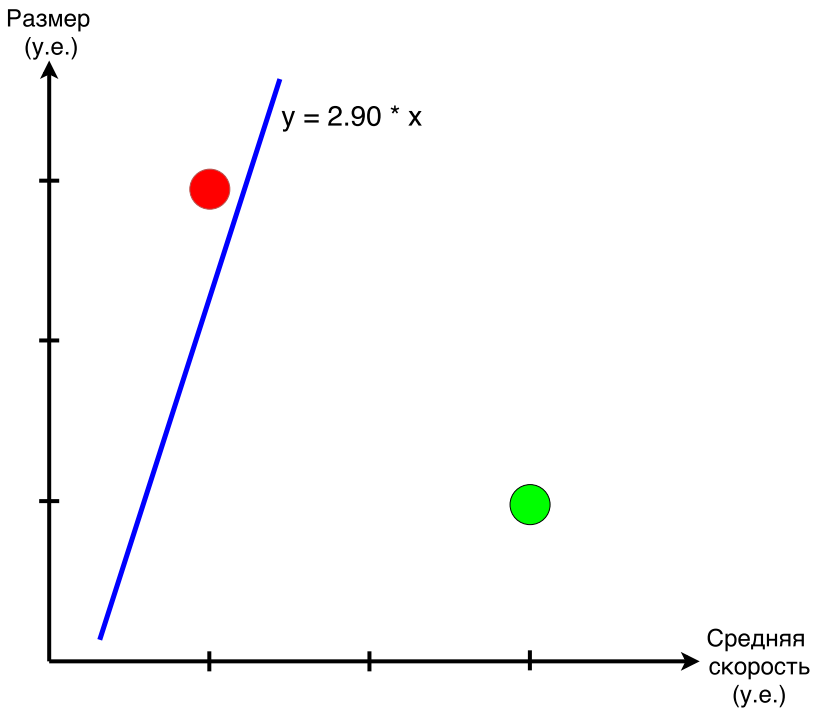

Рис. 1.7. Состояние классификатора после обработки второго объекта

Можно заметить, что какая бы обучающая выборка не была рассмотрена, разделяющая прямая будет находиться вблизи последнего объекта этой выборки.

Для решения данной проблемы введем понятие коэффициента обучения L. Этот параметр показывает, на какую долю от целой ошибки будет изменено значение A. То есть ΔA можно записать формулой (1.4).

$$\Delta A = L * (E / x) \quad (1.4)$$

Пусть L = 0.5, тогда, выполнив аналогичные вычисления, после работы с первым объектом ΔA = 0.5 * (0.35 / 3.0) = 0.0583 и A = 0.25 + 0.0583 = 0.3083. Конечно, рассмотрев только один объект, результат не будет полностью удовлетворять условию того, что легковые автомобили находятся под разделяющей прямой. Но, рассмотрев второй объект, ΔA будет равняться 1.2958, а сам параметр A = 1.6042. Таким образом, новая линия полностью удовлетворяет условию и правильно разделяет объекты на классы (рис. 1.8)

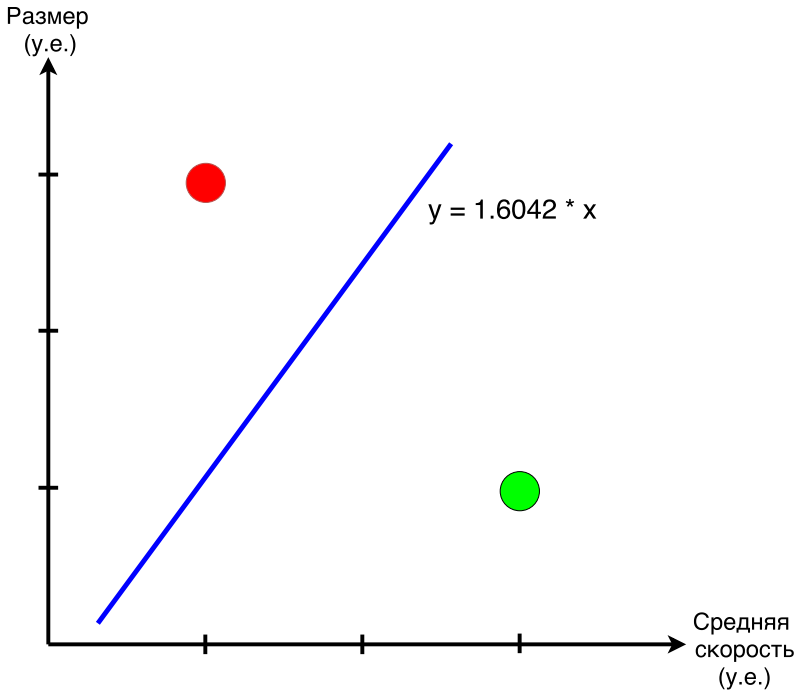

Рис. 1.8. Результат после введения коэффициента обучения

Подведем итог и выделим основные моменты процесса обучения, которые присущи большей части алгоритмов машинного обучения, в том числе и нейтронным сетям.

В первую очередь, необходимо инициализировать начальные параметры алгоритма некоторыми случайными значениями. Иногда эти

значения не совсем случайны и подбираются под конкретную задачу, что позволяет повысить точность алгоритма после обучения.

Затем, перебирая объекты обучающей выборки, необходимо рассмотреть ошибку между ответом алгоритма и ожидаемым значением для каждого объекта, вычислить её и изменить параметры алгоритма на величину, равную значению данной ошибки, умноженной на коэффициент обучения.

2. Введение в нейронные сети

2.1. Изучение нейрона

Нейрофизиологов давно интересовало, на каких принципах основана работа нервной клетки нейрона, и как их можно смоделировать, чтобы в дальнейшем использовать эти представления для создания искусственного интеллекта.

Нервную клетку, можно рассматривать как устройство, которое на входе имеет много дендритов (рис. 2.1). Дендриты - отростки на нервной клетке. Если вокруг них происходит концентрация отрицательных ионов, то они (ионы) переходят внутрь клетки и накапливаются в ней. Когда суммарный заряд, накопившийся внутри клетки, превосходит некоторый порог активации, клетка приходит в возбуждённое состояние и генерирует электрический импульс, который распространяется по длинному отростку, который называется аксоном, на конце которого происходит взаимодействие со следующей нервной клеткой.

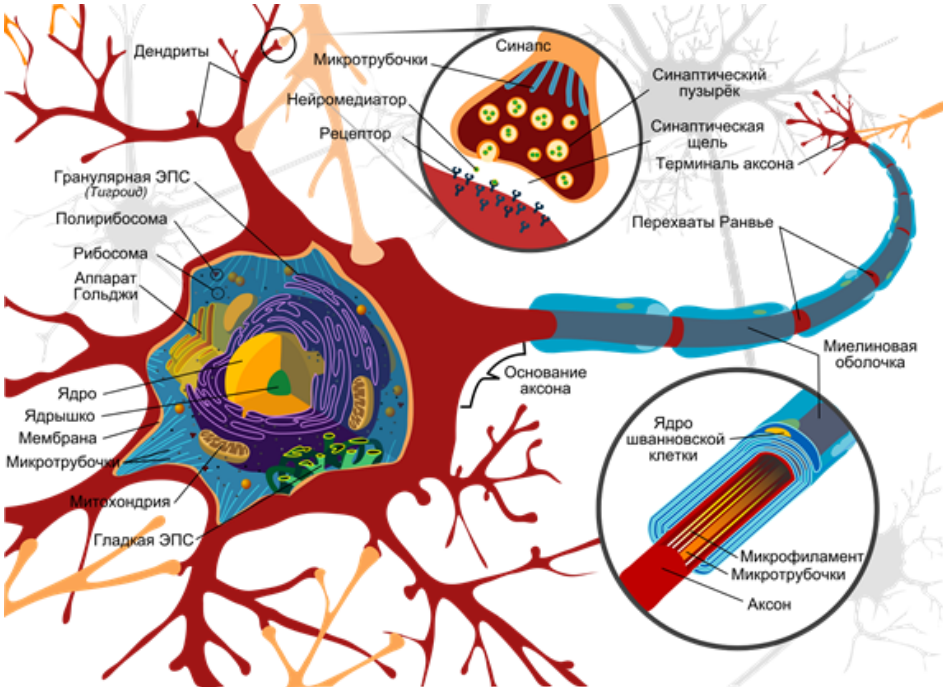

Рис. 2.1. Схема нейрона
(Mariana Ruiz Villarreal)

В компьютерных нейронных сетях используют упрощенную модель нейрона, но основные черты в целом схожи с реальным аналогом.

Одной из наиболее известных моделей нейронной клетки является модель Маккалока — Питтса (рис. 2.2).

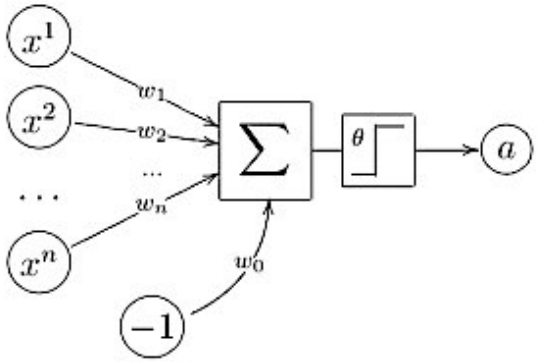

Рис. 2.2. Модель Маккалока — Питтса

На вход такому нейрону подаются сигналы $x_1 \ldots x_n$, причем доля каждого сигнала взвешивается параметром w_n. Затем все взвешенные сигналы суммируются и значение суммы передается функции активации.

Существуют различные функции активации. Это может быть и пороговая функция (рис 2.3.), но чаще используется сигмоидная функция активации (2.1) (рис. 2.4), значение которой можно трактовать как вероятность положительного ответа.

$$y(x) = \frac{1}{1+e^{-x}} \tag{2.1}$$

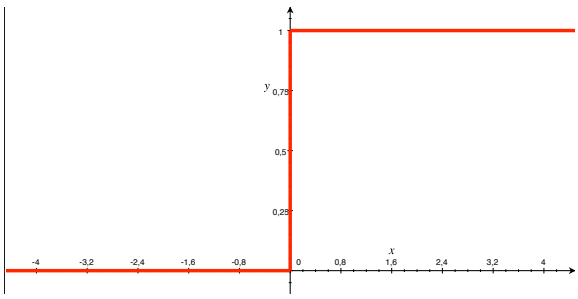

Рис. 2.3 Пороговая функция

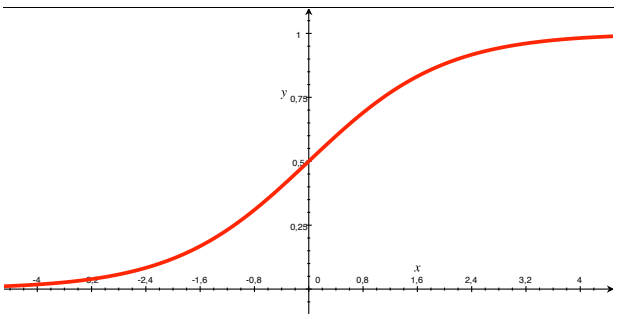

Рис 2.4. Сигмоидная функция

2.2. Устройство нейронной сети

Изучив устройство нейронов, необходимо рассмотреть как они связаны между собой, понять как сигналы от одного нейрона передаются другому и как в целом устроена сеть из них.

Говоря о нейронной сети, важно понимать, какие именно параметры необходимо обучить для получения дальнейших прогнозов. Так как сам нейрон представляет из себя сумматор входящих в него сигналов и некую функцию активации можно сделать вывод, что основными параметрами нейронной сети являются веса w_i, имеющие диапазон значений в промежутке от 0 до 1 и характеризующие степень значимости той или иной связи между нейронами.

Графически подобную нейтронную сеть можно упрощенно представить в виде кружков, обозначающие нейроны и стрелок - связей между ними. На рисунке 2.5 изображена трехслойная нейронная сеть.

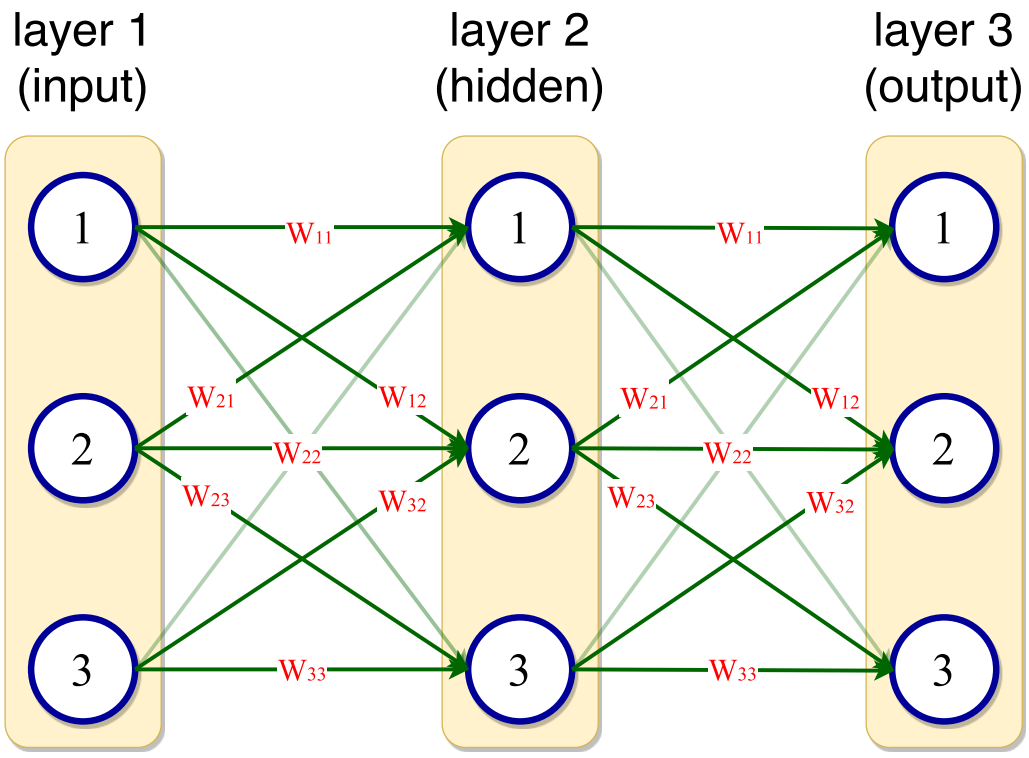

Рис. 2.5. Трехслойная нейронная сеть

Слои нейросети такого вида имеют определенные названия. Входной слой (input) - это слой, принимающий некоторые входные данные и передающий их скрытому слою (hidden). Название второго слоя связано с тем, что пользователь никак с ним не взаимодействует. На самом деле слоев может быть довольно много и обучение такой нейронной сети открывает новый класс алгоритмов машинного обучения под названием «Deep Learning». Третий же слой называется слоем выхода (otput), так как именно из него выходят конечные значения (который можно интерпретировать как ответ) нейронной сети.

Рассмотрим принцип работы нейронной сети. Для примера возьмем более простую сеть, состоящую из всего лишь двух слоев с двумя нейронами на каждой и имеющие сигмоидную функцию активации (рис. 2.4).

Аналогично примеру линейного классификатора, проинициализируем параметры весов случайными величинами: w11 = 0.9, w12 = 0.2, w21 = 0.3, w22 = 0.8 (рис. 2.6).

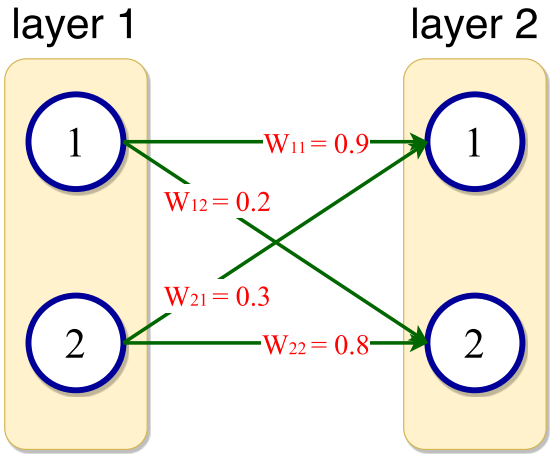

Рис. 2.6. Инициализация весов

Подадим сети значения 1.0 и 0.5. Эти значения будут являться входными данными. Так как задача первого слоя принять и передать входные значения на следующий слой, функция активации для него применяться не будет. В каждый нейрон второго слоя данные будут приходить в форме, наглядно изображенной на рисунке 2.7.

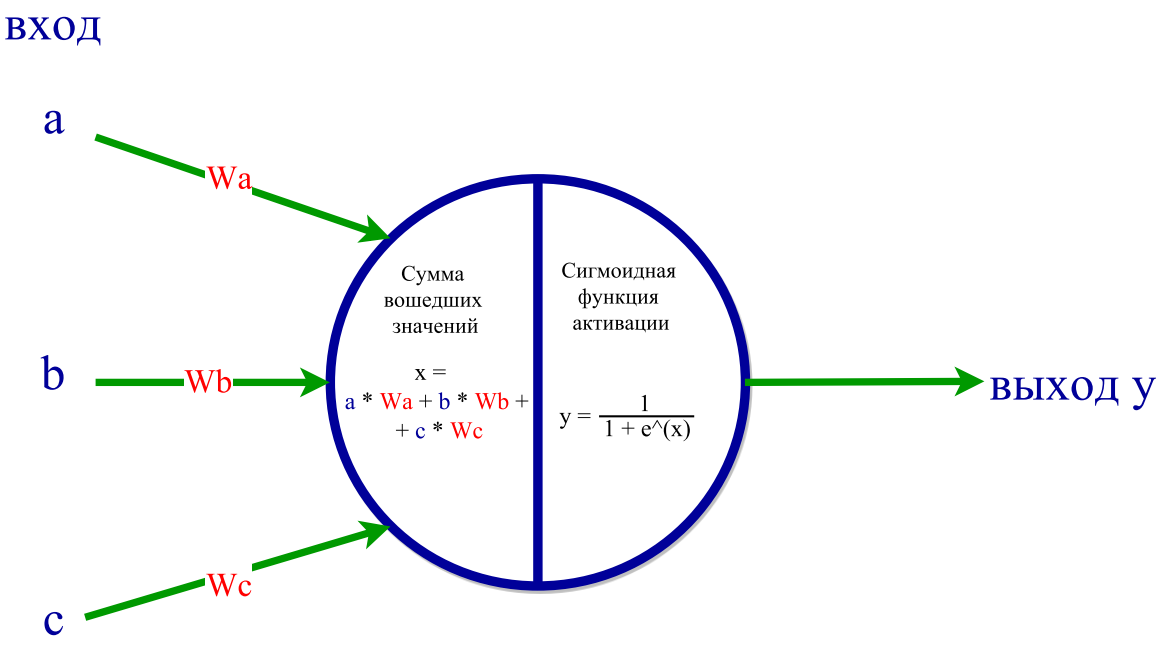

Рис. 2.7. Модель нейрона

Посчитаем сумму значений, входящих в верхний и второй узлы второго слоя:

$$x1 = (1.0 * 0.9) + (0.5 * 0.3),$$

$$x2 = (1.0 * 0.2) + (0.5 * 0.8).$$

Затем применим для них сигмоидную функцию активации:

$$y1 = 1 / (1 + 0.3499) = 1 / 1.3499 = 0.7408,$$

$$y2 = 1 / (1 + 0.5488) = 1/(1.5488) = 0.6457.$$

В итоге нейронная сеть обработала полученные значения и вывела результат (рис. 2. 8).

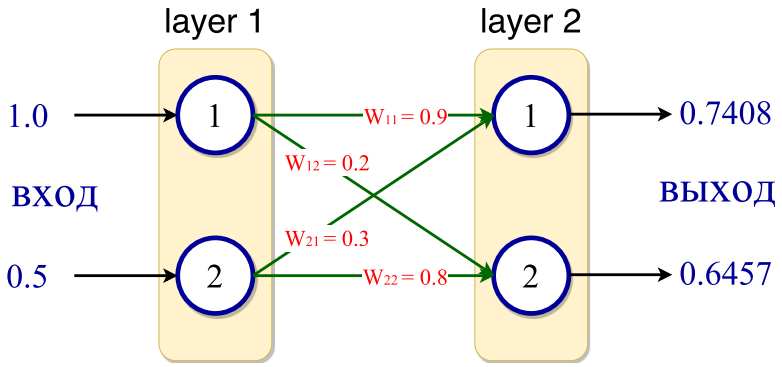

Рис. 2.8. Вывод сети

Как видно, данный способ передачи сигналов от одних нейронов к другим довольно трудоемкий и неудобный.

Для упрощения всех вычислений представим все веса нейронной сети (W) и входящие значения (I) в виде матриц и перемножим их (2.2). Из рисунка видно, что получившаяся матрица значений (X), входящих во второй слой равна выражениям из предыдущего метода.

$$W * I = X. \quad (2.2)$$

$$\begin{pmatrix} W_{11} & W_{21} \\ W_{12} & W_{22} \end{pmatrix} \begin{pmatrix} input\_1 \\ input\_2 \end{pmatrix} = \begin{pmatrix} (W_{11} * input\_1) + (W_{21} * input\_2) \\ (W_{12} * input\_1) + (W_{22} input\_2) \end{pmatrix} = \begin{pmatrix} X1 \\ X2 \end{pmatrix}$$

Таким образом, для передачи сигналов от одного слоя к другому выгодно использовать операции над матрицами.

Затем, для получения выходных (O) значений нейросети, сигналы, вошедшие во второй слой необходимо преобразовать с помощью функции активации , которая в данном примере имеет вид сигмнойдной (sigmoid()). Суть преобразование состоит в том, что для каждого элемента матрицы (X) необходимо применить функцию активации, что на выходе даст матрицу того же размера, что и изначальная матрица. Данное преобразование будет иметь вид (2.3):

$$sigmoid(X) = O. \quad (2.3)$$

$$\begin{pmatrix} sigmoid(X1) \\ sigmoid(X2) \end{pmatrix} = \begin{pmatrix} output\_1 \\ output\_2 \end{pmatrix}$$

2.3. Обратное распространение ошибки

В предыдущем разделе было сказано, что основными параметрами нейронной сети являются веса, влияющие на значение, приходящее в каждый нейрон.

Как и в примере линейного классификатора, будем рассматривать ошибку между искомым значением и тем, что было получено на выходе нейрона.

Рассмотрим простую модель в которой имеется три нейрона: два на входном и один на выходном слое. Вопрос заключается в том, как распределить значение ошибки между весами. Первое, что приходит в голову - разделить значение ошибки пополам и передать их весам (рис. 2.9 (а)). Но более логичным решением было бы разделить ошибку пропорционально отношению весов друг к другу (рис. 2.9 (б)). То есть, для данного примера веса относятся как 1:3, а это значит, что и ошибку необходимо разделить аналогичным образом. Такой подход позволит позволит сильней «штрафовать» тот вес, который делает больший вклад в неправильный ответ.

Этот метод называют методом обратного распространения ошибки.

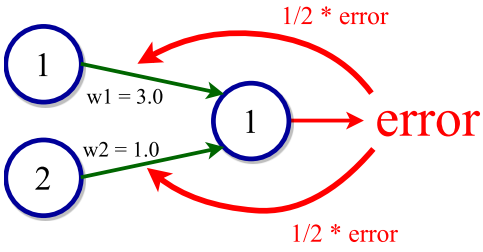

Рис. 2.9 (а)

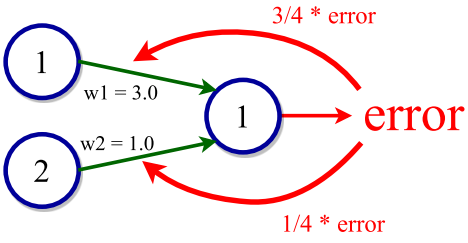

Рис. 2.9 (б)

Конечно же очень редко в один нейрон приходит всего 2 сигнала. В этом случае справедлива формула (2.4), где e_i - ошибка, накладываемая на вес w_i, E - полное значение ошибки. Суммирование идет по всем весам, приходящим в нейрон на данном слое.

$$e_i = E * \frac{w_i}{\sum_k w_k} \tag{2.4}$$

Разберемся, как происходит распространение ошибки в случае, когда в нейросети более двух слоев.

Для решения этой проблемы рассмотрим трехслойную нейтронную сеть (рис. 2.10) и введем ряд обозначений: e_{output} - ошибка нейрона выходного слоя; w_{ho} - веса между скрытым и выходным слоями; e_{hidden} - ошибка нейрона скрытого слоя; w_{ih} - веса между входным и скрытым слоями.

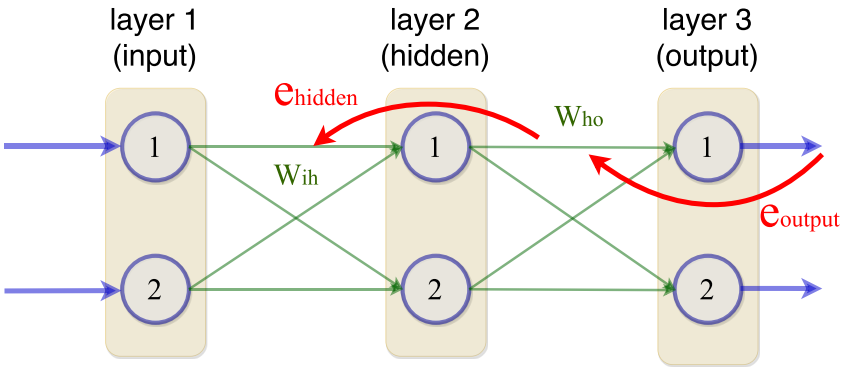

Рис. 2.10. Распространение ошибки

Вопрос заключается в том, что принять за ошибку на спрятанном слое, так как на выходном слое все довольно просто, ошибкой является разница между ожидаемым ответом и полученным. В скрытом слое невозможно выявить ожидаемое значение.

В этом случае за ошибку нейрона на скрытом слое принимается сумма ошибок, распространенные на веса связей, выходящих из данного нейрона (рис. 2.11). Просчитав ошибки всех нейронов на скрытом слое, необходимо распределить их аналогично примеру выше.

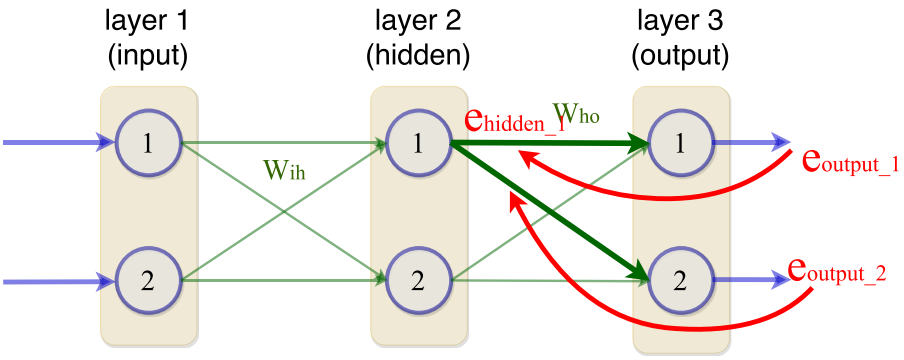

Рис. 2.11. Ошибка скрытого слоя

В предыдущем разделе было продемонстрировано, что использование матриц для работы с весами нейросети сильно упрощает вычисления. Логично предположить, что аналогичный подход позволит упростить процесс распределения ошибок.

В первую очередь, представим ошибки выходного слоя в виде матрицы (2.5).

$$E_{output} = \begin{pmatrix} e_1 \\ e_2 \end{pmatrix} \tag{2.5}$$

Теперь необходимо сконструировать выражение для получения ошибок на скрытом слое. Рассмотрим эту задачу, опираясь на пример выше. Для верхнего нейрона скрытого слоя ошибка будет равна:

$$e_{hidden\_1} = (e_1 * \frac{w_{11}}{w_{11} + w_{21}}) + (e_2 * \frac{w_{12}}{w_{12} + w_{22}})$$

$$e_{hidden\_2} = (e_1 * \frac{w_{21}}{w_{21} + w_{11}}) + (e_2 * \frac{w_{22}}{w_{22} + w_{12}}).$$

Если связать эти выражения, то можно получить их представление в виде перемножения матриц (2.6):

$$E_{hidden} = \begin{pmatrix} \frac{w_{11}}{w_{11} + w_{21}} & \frac{w_{12}}{w_{12} + w_{22}} \\ \frac{w_{21}}{w_{21} + w_{11}} & \frac{w_{22}}{w_{22} + w_{12}} \end{pmatrix} * \begin{pmatrix} e_1 \\ e_1 \end{pmatrix} \tag{2.6}$$

Проанализируем это выражение. Главными его компонентами являются значения ошибок выходного слоя e_n и вес w_{ij}. Чем больше вес, тем большая доля ошибки на него распространяется. Из этого можно сделать вывод, что деление веса w_{ij} на сумму всех весов является чисто нормализующим

фактором, который не влияет на отношению между весами. Исходя из этого, можно упростить выражение (2.6) и получить (2.7).

$$E_{hidden} = \begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{pmatrix} * \begin{pmatrix} e_1 \\ e_1 \end{pmatrix} \tag{2.7}$$

Так же можно заметить, что матрица, содержащая веса - это транспонированная матрица весов, которая была построена ранее (2.2). Это значит, что при реализации программы не будет необходимости строить новую матрицу, а только лишь транспонировать уже имеющуюся.

Запишем получившееся выражение в более абстрактном виде (2.8):

$$E_{hidden} = W^T * E_{output} \tag{2.8}$$

2.4. Обновление весов

В линейном классификаторе, рассмотренном в первой части, для обновления параметра A была введена величина ΔA, вычисляемая по формуле (1.3). Но такая простая формула не пригодна для применения к весам.

Значение весов зависит от многих факторов, поэтому ошибку, которую дает каждое значение w_i, следует задать функциональной зависимостью E(w). Таким образом, необходимо решить задачу минимизации функции ошибки E(w).

Минимум функции лучше всего искать методом градиентного спуска, при этом учитывая квадратичное значение ошибки.

Этот метод на каждой итерации показывает направление наиболее сильного убывания функции, а выбор квадратичного значения ошибки обусловлен легкостью его дифференцирования и нахождения производной.

Для поиска антиградиента функции необходимо продифференцировать функцию по всем переменным. В случае с функцией ошибки E(w), это будет выглядеть следующим образом:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}}(t_k - o_k)^2 \quad ,$$

где o_k - ответ k-го узла выходного слоя, t_k - ожидаемое значение, которое является константой и не зависит от w_{ik}. В виду этого выполним следующее преобразование:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_k} * \frac{\partial o_k}{\partial w_{ij}}$$

и возьмем производную $\partial E/\partial o_k$:

$$\frac{\partial E}{\partial w_{ij}} = -2(t_k - o_k) * \frac{\partial o_k}{\partial w_{ij}} \quad .$$

Теперь выясним, что представляет собой выражение $\partial o_k / \partial w_{ik}$. Так как ответ каждого узла равен сумме произведений входного значения на веса и пропущенных через сигмоидную функцию, перепишем это выражение:

$$\frac{\partial E}{\partial w_{ij}} = -2(t_k - o_k) * \frac{\partial}{\partial w_{ij}} sigmoid(\sum_j w_{jk} * o_j)$$

Здесь o_j является выходом из узла скрытого слоя, а не выходного, как в предыдущем случае.

Теперь необходимо взять производную от сигмоидной функции. Опустим вычисления и запишем результат:

$$\frac{\partial}{\partial x} sigmoid(x) = sigmoid(x) * (1 - sigmoid(x))$$

Затем подставим полученное значение и получим формулу, выражающую направление антиградиента.

$$\frac{\partial E}{\partial w_{ij}} = -2(t_k - o_k) * sigmoid(\sum_j w_{jk} * o_j) * (1 - sigmoid(\sum_j w_{jk} * o_j)) * \frac{\partial}{\partial w_{ij}} (\sum_j w_{jk} * o_j) =$$
$$= -2(t_k - o_k) * sigmoid(\sum_j w_{jk} * o_j) * (1 - sigmoid(\sum_j w_{jk} * o_j)) * o_j$$

Учитывая, что необходимо знать только направление, константа 2, стоящая в начале выражения, может быть отброшена. Так же заметим, что выражение t_k - o_k является значением ошибки e_j. Таким образом, была получена формула для вычисления направления антиградиента (2.9) и формула для обновления веса (2.10), где L - коэффициент обучения.

$$\frac{\partial E}{\partial w_{ij}} = -(e_j) * sigmoid(\sum_j w_{jk} * o_j) * (1 - sigmoid(\sum_j w_{jk} * o_j)) * o_j \quad (2.9)$$

$$new\_w_{jk} = old\_w_{jk} + L * \frac{\partial E}{\partial w_{ik}} \quad (2.10)$$

Рассмотрим, как будут выглядеть эти операции, используя матрицы. Для этого запишем выражение для получения матрицы ΔW.

$$\begin{pmatrix} \Delta w_{11} & \Delta w_{12} & \cdots & \cdots \\ \Delta w_{21} & \Delta w_{22} & \cdots & \cdots \\ \cdots & \cdots & \Delta w_{jk} & \cdots \\ \cdots & \cdots & \cdots & \cdots \end{pmatrix} = \begin{pmatrix} E_1 * S_1(1-S_1) \\ E_2 * S_2(1-S_2) \\ E_3 * S_3(1-S_3) \\ \cdots \end{pmatrix} * \begin{pmatrix} o_1 & \cdots & o_j & \cdots \end{pmatrix}$$

Из этого выражения можно увидеть, что матрица ответов O_j является горизонтальной. Так как изначально была получена только вертикальная матрица, то для получения горизонтальной необходимо провести операцию транспонирования.

Запишем итоговое выражение (2.11) для определения значения, на которое необходимо изменить вес w_{jk}.

$$\Delta w_{jk} = -(E_k) * sigmoid(O_k) * (1 - sigmoid(O_k)) * O_j^T \qquad (2.11)$$

2.5. Подготовка данных

Качество работы нейронной сети зависит от многих факторов: от начальных значений весов, от размера обещающей выборки, от шумов в начальных данных, от самого формата данных и т.д.. Поэтому, перед работой с нейронной сетью необходимо подготовить данные и привести их в формат, пригодный для обработки.

В первую очередь необходимо нормализовать входные значения. Это обусловлено тем, что при использовании сигмоидной функции активации (рис. 2.4) с параметрами, большими 1, её график становится более плоским и не дает качественных различий, даже если входные параметры будут отличаться на порядок. Таким образом, необходимо привести входные данные в диапазон (0, 1).

Так же стоит отметить, что и выходные значения нейронной сети будут лежать в этом же промежутке, так как область значений функции активации лежит в промежутке (0, 1). Это значит, что нормализация требуется и для ожидаемых значений (target), иначе значение ошибки будет мало по сравнению с ожидаемым значением.

Инициализация значений весов так же имеет свои особенности. Если сделать их значение довольно большим, то входящий сигнал будет увеличен, и при использовании сигмоидной функции активации возникнут проблемы, описанные ранее.

Математики давно изучают зависимости начальных значений весов при использовании различных функций активации. В результате был выработан основной метод инициализации. Он заключается в том, что когда в нейрон приходит много сигналов, их сумма с учетом весов не должна выводить аргумент сигмоидной функци за разумные пределы. В итоге было выявлено, что лучшим образом с этой задачей справляются веса, имеющие случайные значение с нормальным распределением (рис. 2.12), лежащие в диапазоне $(-1/\sqrt{n};\ 1/\sqrt{n})$, где n - количество сигналов, приходящих в нейрон. Но при этом

значение веса не должны равняться нулю, так как в этом случае даже большое значение не будет учтено.

3. Тестирование нейронной сети

(тест с обучающей выборкой и собственные рисунки)

(описать графики при различном количестве обучающих выборок)

(протестировать разное количество нейронов на внутреннем слое, протестировать разное количество пробегов входных данных, проверка при поворотах изображения)

Сделать ссылки на обучающую и тестовые выборки

4. Исследование нейронной сети

(внутри нейронной сети, оценка сложности алгоритма обучения)

5. UML- диаграмма класса

6. Блок-схемы работы алгоритма

7. Листинг программы

Заключение

Список литературы