СОДЕРЖАНИЕ

## Введение

На протяжении тысячелетий человечество пыталось разгадать тайну работы мозга и создать устройства, способные мыслить. Мы изобрели кремниевые зажигалки, с помощью которых можем в любой момент получить огонь, блоки для поднятия тяжестей и даже калькуляторы, способные выполнять для нас расчеты, но все эти простые механические и электронные устройства, облегчающие нашу жизнь, нас уже не удовлетворяют. Теперь мы хотим автоматизировать более сложные задачи, такие как группировка схожих фотографий, отделение больных клеток от здоровых и даже управление автомобилями. По-видимому, для решения таких задач требуется человеческий интеллект или по крайней мере некие загадочные возможности человеческой психики, которых вы не найдете в простых устройствах типа калькуляторов. С появлением первых компьютеров люди всерьез задумались об искусственном интеллекте, который бы закрывал эти потребности. Логично предположить, что ИИ (искусственный интеллект) предполагалось создать по образу человеческого мозга. Уже тогда люди догадывались, что мозг – это огромная сеть из нейронов, которая помогает человеку обучаться и запоминать что-либо.

Сейчас нейронные сети окружают нас практически везде. Они умеют распознавать текст, предметы, лица, рисовать картины по описанию, улучшать качество видео, водить автомобиль и многое другое, о чем мы также поговорим в этом докладе. Нейронные сети являются частью современного искусственного интеллекта и машинного обучения. Для того, чтобы подробнее разобраться в нейронных сетях и вообще понять, что же это такое, мы должны разобраться в основных терминах и понятиях, о чем будет рассказано далее. А пока что начнем с истоков, с истории возникновения.

История создания нейронных сетей

- Все началось в 1943 году, когда Уоррен Маккалох и Уолтер Питтс, американские нейрофизиологи, представили первую модель нейронной сети, которая описывала систему логики, способную анализировать сложные проблемы.
- Затем в 1949 году канадский психолог Дональд Хебб разработал «хеббовскую» теорию обучения, согласно которой, когда два или более нейрона соединяются клеточной связью, их способность выполнять задачу усиливается.
- В 1952 году Алан Тьюринг, британский кибернетик, предположил, что компьютеры должны иметь возможность использовать обучающие алгоритмы, что стало важной концепцией в современном искусственном интеллекте и машинном обучении.
- После, в 1957 году Фрэнк Розенблатт, американский психолог, создал первую компьютерную нейронную сеть, известную как «Персептрон». Эта машина была способна обрабатывать простые логические задачи. Персептрон был основан на классическом механизме нейронов, которые принимали только бинарные входные данные либо 0, либо 1. У него также было ограничение в том, что он мог решать проблему классификации только в линейной функции, а это означало, что он не мог решать несколько уровней проблем.
- В 1960 году Бернард Уидроу совместно со своим студентом Хоффом на основе дельта-правила (*формулы Уидроу*) разработали Адалин, который сразу начал использоваться для задач предсказания и адаптивного управления. Адалин был построен на базе созданных ими же новых элементах — мемисторах.
- Также к 1960-м годам Марвин Мински, ученый-компьютерщик, разработал способы использования компьютеров для моделирования

нейронных сетей. Он также разработал модель разумного поведения, которую назвал Обществом Разума.

- В 1965 году Джозеф Вайценбаум создал ELIZA — первого чат-бота, который имитировал работу психотерапевта и мог общаться с человеком на естественном языке.
- В 1974 Пол Дж. Вербос и Галушкин А. И. Одновременно изобретают алгоритм обратного распространения ошибки для обучения многослойных перцептронов.
- А в 2007 Джеффри Хинтоном в университете Торонто созданы алгоритмы глубокого обучения многослойных нейронных сетей. Хинтон при обучении нижних слоёв сети использовал ограниченную машину Больцмана. По Хинтону необходимо использовать много примеров распознаваемых образов. После обучения получается готовое быстро работающее приложение, способное решать конкретную задачу, например, осуществлять поиск лиц на изображении).

Нейронные сети прошли долгий путь с первых дней теории вычислений, и работа влиятельных исследователей, таких как Фрэнк Розенблатт, Дональд Хебб и Марвин Мински, сыграла жизненно важную роль в развитии этой области. Их работа не только продвинула наше понимание того, как нейронные сети можно использовать в вычислениях, но и их вклад сыграл важную роль в развитии ИИ и машинного обучения.

## Основы нейронных сетей

### Определение нейронных сетей. Простая прогнозирующая машина

Стоит начать с того, что компьютеры, по своей сущности, это обычные калькуляторы, которые способны с очень большой скоростью выполнять арифметические операции. Подобные задачи для человека окажутся невероятно сложны и трудоемки, близки к невозможным. Зато, к примеру, при задаче классификации изображений (Что изображено на картинке?) человек сможет с легкостью определить, что на фотографии (Рис. 1) находится дерево, а особо образованный и внимательный человек заметит дуб из романа Л.Н. Толстого «Война и мир». Эта же задача для компьютера окажется непосильно сложной. Таким образом, можно сделать вывод, что при распознавании образа требуется человеческий интеллект, который отсутствует у машины. Но в наше время существует возможность наделить компьютер искусственным разумом, о чем я и расскажу в данном проекте.

Рис.1 – фотография дуба из романа «Война и Мир»

**Нейронные сети** – это метод в искусственном интеллекте, который учит компьютеры обрабатывать данные таким же способом, как и человеческий мозг, используя математические модели. Нейросети, в отличие от других алгоритмов ИИ, не программируются на выполнение конкретных задач, а просто обучаются на основе большого количества данных и находят в них закономерности. Стоит заметить, что каждая нейронная сеть является ИИ, но не каждый ИИ является нейросетью. (Рис. 2)


Рис. 2 – визуализация нахождения методов обучения в ИИ

У всех полезных компьютерных систем имеются каналы ввода и вывода, между которыми над данными выполняются некоторые вычисления. В случае нейронных сетей это не так. Если точные принципы функционирования какой-либо системы нам неизвестны, то мы пытаемся получить представление о том, как она работает, используя модель с регулируемыми параметрами. Рассмотрим простую прогнозирующую машину (предиктор). Она получает некоторую информация на вход, затем совершает определенные вычисления и выдает данные на выход. Давайте

разберем этот процесс на реальном примере, чтобы достичь максимального понимания. Допустим, мы хотим, чтобы наша машина преобразовывала километры в мили, но формулы для этого преобразования мы не знаем, зато знаем, что изменения связаны линейной зависимостью, то есть чем больше километров, тем больше миль (логично!). Также нам предоставлена таблица истинных данных или же «data set» (Рис. 3). Теперь мы можем сказать, что наша формула будет выглядеть следующим образом: «мили = километры * x», где x – это некий коэффициент, который нам предстоит узнать. Далее подставим вместо X случайное число, допустим 0,5 и получим выходные данные (Рис. 4). Заметим, что наши полученные данные «50» не сходятся с таблицей истинности. Проанализируем ошибку и сделаем правки. Для начала посчитаем величину нашей ошибки, она считается по формуле: «ошибка = истинные данные - выходные». Теперь мы знаем величину ошибки и то, что наше значение меньше истинного. На основе этой информации мы меняем наш коэффициент X и заново прогоняем алгоритм. К примеру, возьмем 0,6 и увидим, что на этот раз наше значение оказалось больше желаемого (Рис. 5). Таким образом мы подгоняем наши выходные данные под истинные путем изменения коэффициента X.

| Истинный пример | Километры | Мили |
|---|---|---|
| 1 | 0 | 0 |
| 2 | 100 | 62,37 |

Рис. 3 – таблица значения для задачи «перевод километров в мили»

Рис. 4 – результат алгоритма для задачи перевода километров мили при X = 0,5

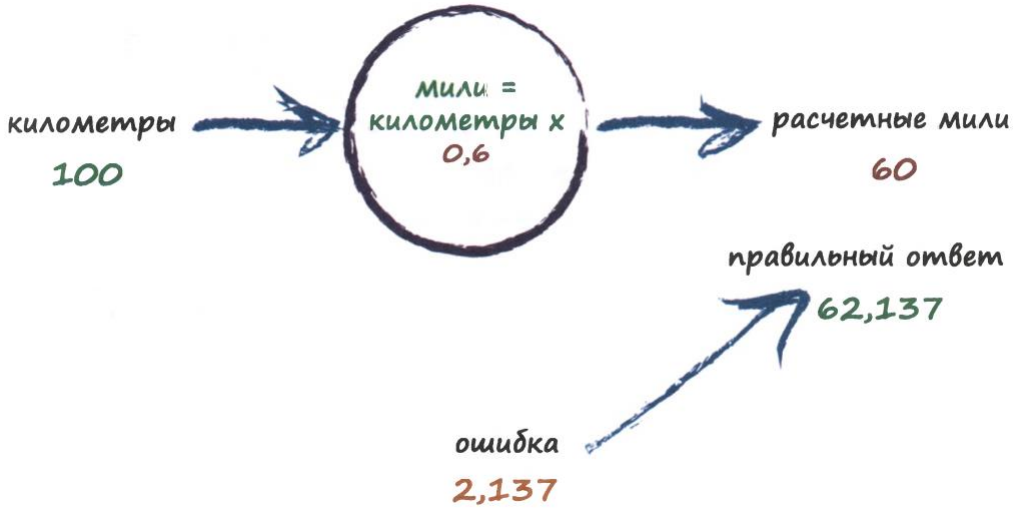


Рис. 5 – результат алгоритма для задачи перевода километры в мили при X = 0,6

## Работа нейрона и его активация

Ранее мы говорили о том, что нейронные сети были созданы на примере мозга живых существ. Но долгое время ученые не понимали как маленький мозг животных может быть быстрее компьютера. Тогда они обратили внимание на архитектуру. Пока компьютер выполняет команды последовательно и не имеет неопределенности, мозг живого организма обрабатывает сигналы параллельно и имеет неопределенность. Строение базовой структурно-функциональной единицы биологического мозга – нейрона. В головном мозге человека насчитывается около 100 миллиардов нейронов. Рассмотрим, как работает нейрон (Рис. 6). Он принимает поступающий электрический сигнал и вырабатывает другой электрический сигнал. Это очень напоминает работу моделей классификатора или предиктора, которые получают некоторые входные данные, выполняют определенные вычисления и выдают результат. Нейрон отличен от функции

тем, что он не воспринимает слабые сигналы, но как только этот сигнал превысит пороговое значение, нейрон дает ответ (Рис. 7). Можно провести аналогию с человеческим мозгом, где такой процесс называется – возбуждение.

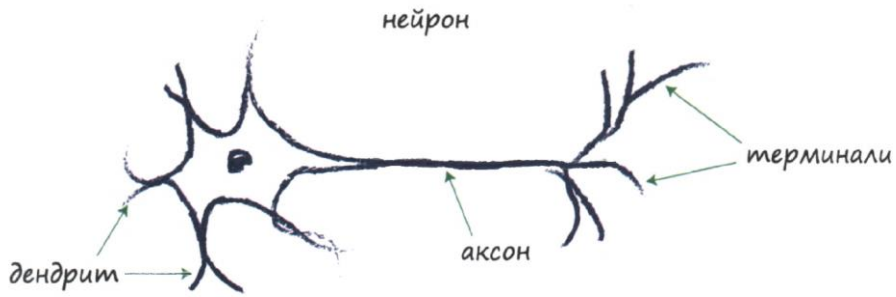


Рис. 6 – схематичное строение нейрона человека

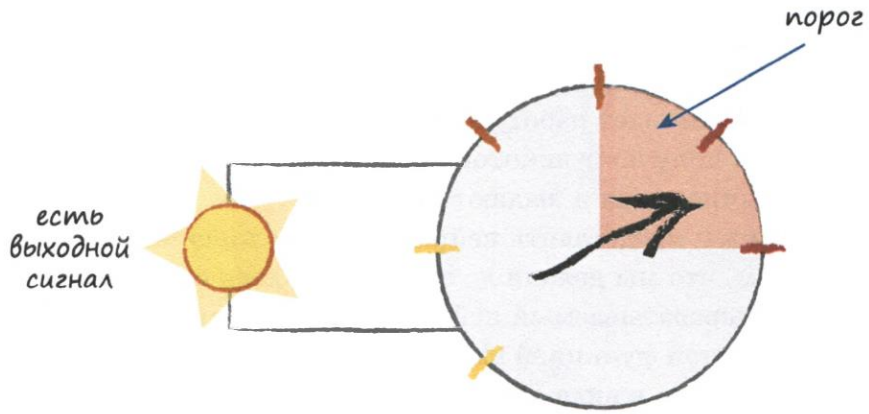


Рис. 7 – пример активного нейрона

Теперь разберемся с принципом работы искусственного нейрона. Каждый вход умножается на некоторые веса, дальше все суммируется, прогоняется через нелинейную функцию, результат выдается на выход — все, это один нейрон. Коробочка, которая по выходным данным пропускает сигнал дальше, или не пропускает. Но что же «активирует» наш нейрон? С этим нам помогает пороговое значение, которое называется функцией активации. Таких функций существует огромное множество. Как пример можно привести ступенчатую функцию (Рис. 8). По рисунку можно

заметить, что слабые значения равны нулю, но как только они переходят через определенный порог, сразу становятся активными.


Рис. 8 – ступенчатая функция активации

Но наш мир не любит острых углов, поэтому даже эту функцию мы сгладим. Такая функция будет называться «сигмоида». Она является более естественной и точной, а также чаще всего используется в создании нейронной сети прямого распространения.

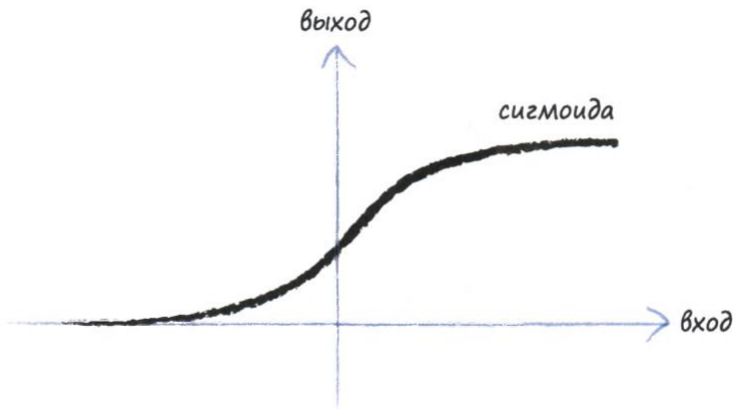


Рис. 9 – сигмоидальная функция активации

## Принцип работы многослойной нейронной сети

В жизни входной нейрон принимает значения от нескольких других и при возбуждении, соответственно, передает сигнал многим другим. Один из способов воспроизведения подобного эффекта в искусственной модели является создание многослойных нейронных структур. Мы будем суммировать входные значения на входе, а затем прогонять через функцию активации (Рис.10).

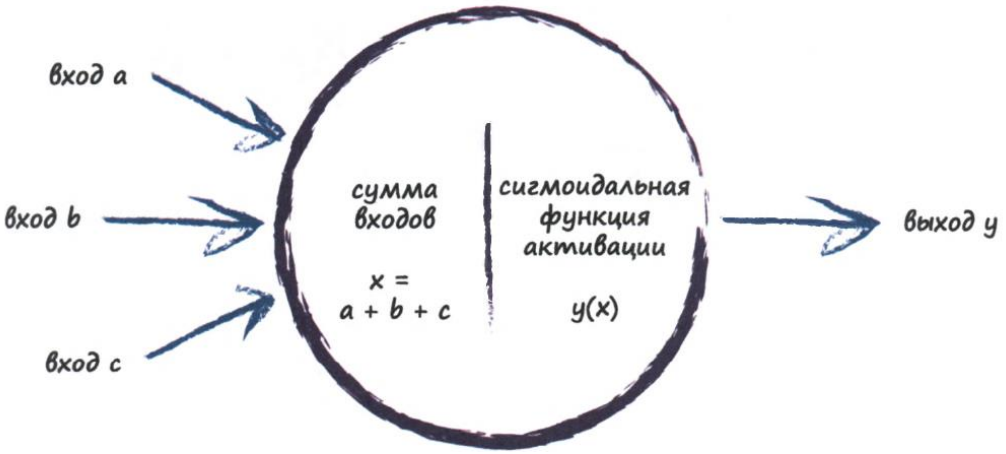


Рис. 10 – схема, отражающая принцип работы нейронной сети

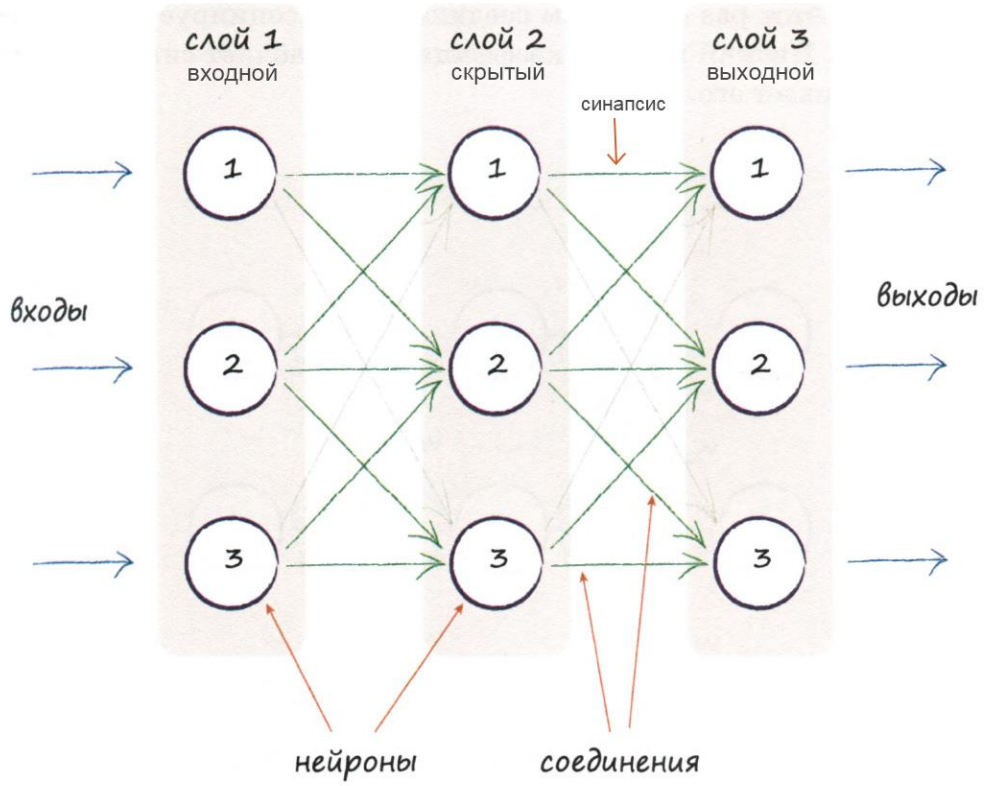


Рис. 11 – схема работы многослойной нейронной сети

Теперь мы готовы переходить к самому интересному: как же работает многослойная нейронная сеть? В качестве рассматриваемого примера (Рис. 11) мы опять возьмем самую простую схему, состоящую всего из 3 слоев (входного, скрытого, выходного) и 3 нейрона, но важно знать, что сети бывают и больших размеров. То есть скрытых слоев может быть бесконечно много, как и узлов (нейронов) и все они будут связаны между собой соединением (синапсисами). Если в нашей нейросети скрытых слоев больше, чем один, то она будет называться глубокой, в обратном случае, как легко догадаться, - не глубокой.

Для обучения нашей сети мы будем регулировать силу узла. А каждый узел будет иметь определенный вес, который будет изменяться. Низкий весовой коэффициент ослабляет сигнал, высокий, что логично, - усиливает его. Стоит отметить, что каждая связь нейронов имеет свой вес. Изначально вес, как и в случае предиктора, выбирается случайным образом, чаще всего в диапазоне от 0 до 1. Как говорилось ранее, нейросеть представляет собой систему слоёв:

- Входной слой, куда подается входной сигнал. Это его единственная функция. К этому слою функция активации не применяется;
- Выходной слой, откуда снимается результат работы;
- Некоторое количество скрытых слоёв между ними. На данном этапе нам потребуется произвести некоторые вычисления. Выходные значения предыдущего узла умножаются на вес синапсиса и складываются. Теперь нам нужно рассчитать выходной сигнал, то есть применить функцию активации. Выходное значение, как привило лежит в диапазоне от 0 до 1. Таким образом мы считаем выходной сигнал для каждого узла второго слоя. Вообще подобные вычисления делаются с помощью матриц, которые сильно упрощают процесс. Пример работы изображен на Рис. 12.

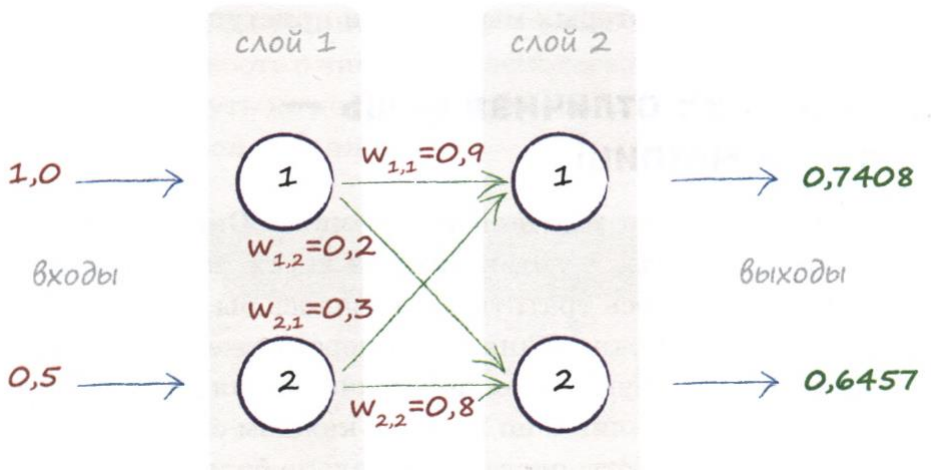


Рис. 12 – пример рассчитанных выходных сигналов сети

А теперь посчитаем выходные значения для нашей первоначальной модели (Рис. 11). До второго слоя мы выполняем те же самые действия, а для третьего, как ни странно, все те же операции! Нам известна величина входных сигналов, получаемых третьим слоем, также у нас есть весовые коэффициенты связей между узлами, ослабляющие сигналы, а это значит, что данных у нас достаточно. В случае, если скрытых слоев оказывается больше, алгоритм действий не меняется.

## Обновление весов. Обратное распространение ошибки

Отлично, мы сделали половину работы, но есть совсем другая задача — обучить нейрон. Обучение заключается в том, чтобы найти правильные веса. Обучение построено на простой идее, если мы на выходе нейрона знаем, какой должен быть ответ, и знаем, какой он получился, нам становится известна эта разница – ошибка. Эту ошибку можно отправить обратно ко всем входам нейрона и понять, какой вход насколько сильно повлиял на эту ошибку, и соответственно, пересчитать вес на этом входе так, чтобы ошибку уменьшить или убрать вовсе.

Теперь нам нужно разобраться, как же пересчитать эти веса. Существует несколько способов, один из них – распределить эту ошибку между всеми узлами в зависимости от внесенной доли, веса (Рис. 13). То есть, при коэффициентах связей 3,0 и 1,0 мы будем распределять ошибку пропорционально весам. На больший вес уйдет большая часть ошибки, на меньший вес – меньшая часть. Если бы узлов было больше, мы бы распределили эту же ошибку между всеми узлами.

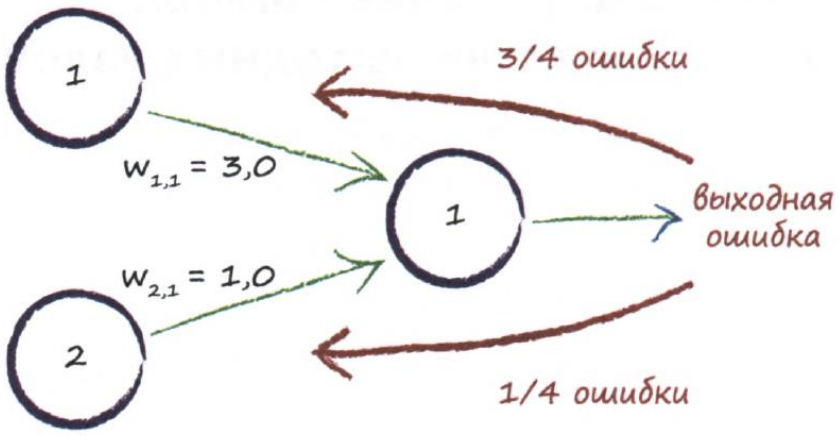


Рис. 13 – иллюстрация распределения ошибки

Такой способ обучение получил вполне логичное и прямое название – **метод обратного распространения ошибки.** При наличии большего количества слоев, мы вновь повторяем эту процедуру для каждого слоя, двигаясь от последнего слоя к первому. Но как посчитать ошибку, если на нейрон приходит несколько синапсисов? В этом случае мы суммируем ошибки из приходящих синапсисов (Рис. 14). Заметим, что ошибка на верхний нейрон складывается из приходящих ошибок из выходного слоя, 0,32 с верхнего и 0,1 с нижнего, что в сумме равняется 0,42. Между входным и скрытым слоем происходит все тоже самое.

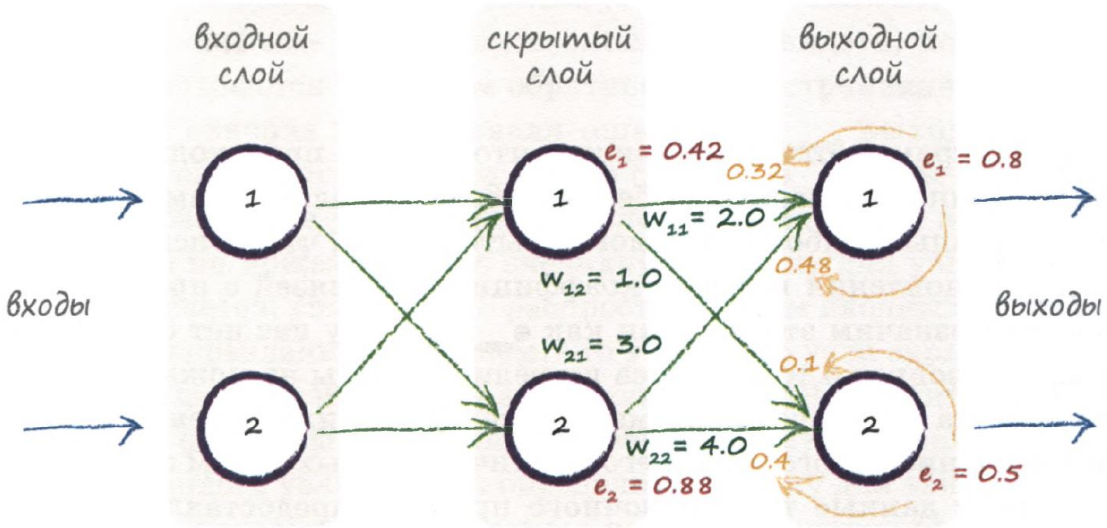


Рис. 14 – метод обратного распространения ошибки в трехслойной сети

Мы почти разобрались с обучением, но все еще не обновили веса связей. Полученные нами ошибки показывают, как должны измениться веса связей. Для решений нашей задачи мы можем просто перебирать случайные сочетания весовых коэффициентов, что называется **методом грубой силы**, но этот способ безумно трудоемкий и долгий. Но оптимальный способ все – таки есть, и называется он **методом градиентного спуска**. Это численный метод нахождения локального минимума функции с помощью движения вдоль градиента, один из основных численных методов современной

оптимизации. Основная идея этого метода заключается в эффективном вычислении частных производных функции сети по всем элементам настраиваемого вектора весов для данного входного вектора. В этом, собственно, и заключается основная вычислительная мощность данного метода. Можно привести простой пример. Представьте, что вы находитесь в горной местности с очень сложным рельефом, вокруг кромешная темнота и вы ничего не видите, но у вас есть фонарик, который освещает небольшую зону вокруг вас. Поэтому вы осматриваетесь и выбираете место, где можно пройти, затем делаете остановку и вновь осматриваетесь, происходит это до тех пор, пока вы не спуститесь в самый низ склона. Таким образом, вы спустились вниз, не зная местности и маршрута движения. Посмотрим на график, который иллюстрирует метод градиентного спуска, который применен к функции, имеющей два параметра (Рис. 15). Также стоит отметить, что возможно и попадание в ложный минимум. То есть минимальная точка в данной области, но не на всем графике. Данный способ хорош тем, что используются достаточно простые и быстрые дифференциальные исчисления.

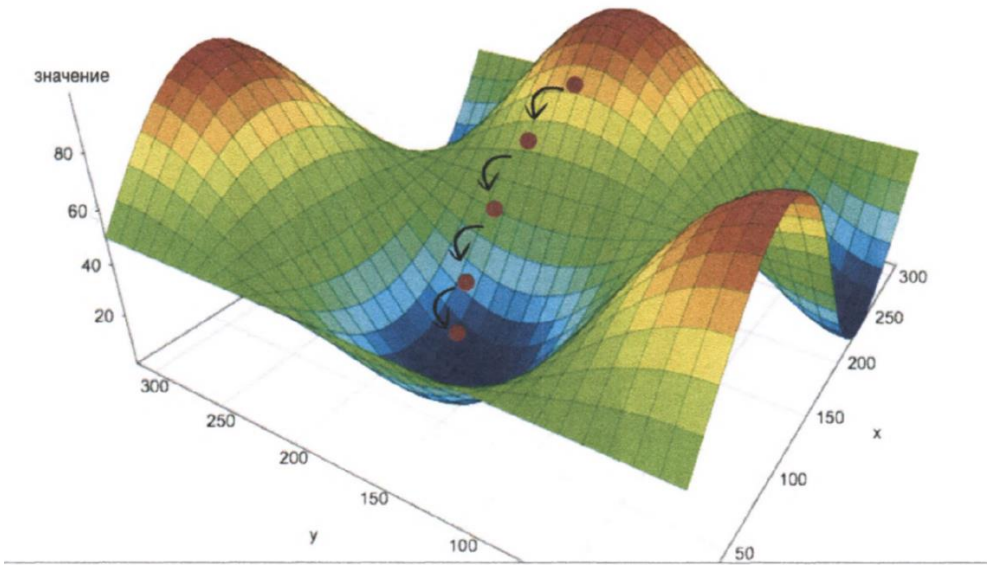

Рис. 14 – график функции, зависящей от двух параметров.

Отлично, мы полностью познакомились с обучением, но считаю своим долгом кратко рассказать еще о некоторым понятиях. Распространенной проблемой обучения является **насыщение** сети. Ситуация, когда большое количество сигналов с большими весами, приводят к сигналам, значение которых близко к нулю. В результате снижается способность сети обучаться на хороших весах. Еще одной проблемой являются нулевые сигналы или веса. Они должны быть не большими, но не нулевыми. Входные и выходные сигналы должны находиться в диапазоне от 0,01 до 0,99.

## Архитектуры нейронных сетей

Архитектур нейронных сетей существует огромное множество, и чтобы рассказать про каждую не хватит и жизни. Все они представлены в статье “The Neural Network Zoo” опубликованной ASIMOV INSTITUTE [1]. Я же лишь вкратце расскажу про три основных и самых популярных архитектуры.

### Прямые нейронные сети

Нейронные сети прямого действия или Feed-forward Neural Network (Рис. 15) и перцептроны (P) очень просты: они передают информацию спереди назад (вход и выход, соответственно). Обычно FFNN обучают методом обратного распространения, давая сети парные наборы данных «что входит» и «что мы хотим получить на выходе». Это называется контролируемым обучением, в отличие от неконтролируемого обучения, когда мы только даем ей входные данные и позволяем сети заполнить пробелы. Хорошая сеть, многие задачи классификации успешно решаются, но у нее есть две проблемы:

- Много параметров. Например, если взять нейросеть из 3 скрытых слоев, которой нужно обрабатывать картинки 100*100 пикселей, это значит, что на входе будет 10 000 пикселей, и они заводятся на 3 слоя. Поэтому требуется огромное количество данных для обучения. Кроме того, сеть, у которой много параметров, имеет дополнительную склонность переобучаться.

- Затухающие градиенты. Когда в нейросети много слоев, от градиента (обратного распространения) в самом конце может остаться очень-очень маленькая часть.

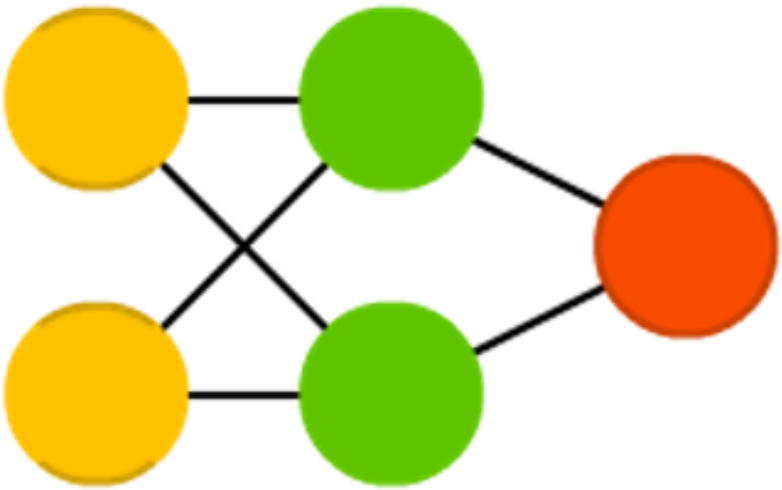

Рис. 15 – макет сети прямого распространения

## Рекуррентные нейронные сети

Рекуррентные нейронные сети (Recurrent Neural Networks) – это FFNN с некой изюминкой: они не являются статичными; у них есть связи между проходами, связи во времени (Рис. 16). Такие сети активно применяют для работы с последовательными данными. Нейроны получают информацию не только от предыдущего слоя, но и от самих себя с предыдущего прохода. Это означает, что порядок подачи входных данных и обучения сети имеет значение. За счет этого образуется некий эффект «памяти».

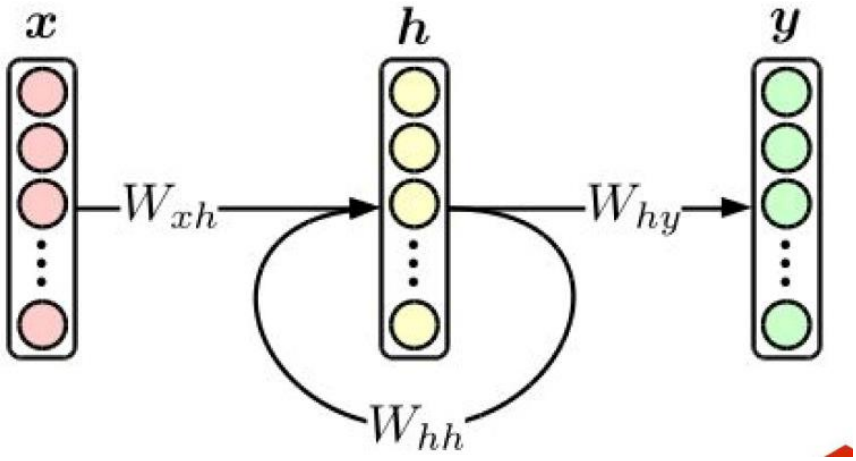


Рис. 16 – макет рекуррентной нейросети

## Сверточные нейронные сети

Сверточные нейронные сети (Convolutional Neural Networks) сильно отличаются от большинства других сетей (Рис. 17). Они в основном используются для классификации изображений, но могут применяться и для других типов входных данных, например аудио. Типичный случай использования CNN – это когда вы подаете сети изображения, а сеть классифицирует данные, например, выдает «кошка», если вы даете ей изображение кошки, и «собака», если вы даете ей изображение собаки. CNN, как правило, начинают с входного «сканера», который не предназначен для анализа всех обучающих данных за один раз. Например, для ввода изображения размером 200 x 200 пикселей вам не нужен слой с 40 000 узлов. После того как вы прошли этот входной слой (и, возможно, использовали его для обучения), вы подаете на него следующие 20 x 20 пикселей: вы перемещаете сканер на один пиксель вправо. Затем эти входные данные проходят через сверточные слои вместо обычных слоев, где не все узлы соединены со всеми узлами. Каждый узел имеет дело только с близкими соседними ячейками. Эти сверточные слои имеют тенденцию уменьшаться (Рис. 18) по мере углубления, в основном на легко делимые коэффициенты входного сигнала (так, 20, вероятно, попадет в слой 10, а затем в слой 5). Чтобы применить CNN для работы со звуком, вы, по сути,

подаете на вход аудиоволны и дюймы по длине клипа, сегмент за сегментом. Реальные реализации CNN часто добавляют в конец FFNN для дальнейшей обработки данных, что позволяет получить крайне нелинейные абстракции.

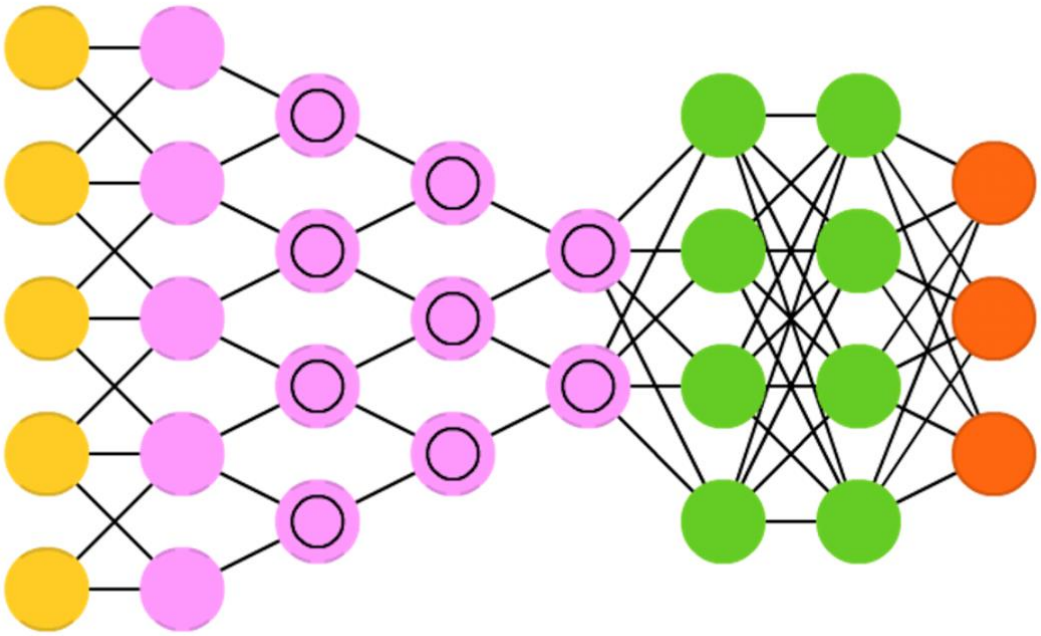

Рис. 17 – макет сверточной сети

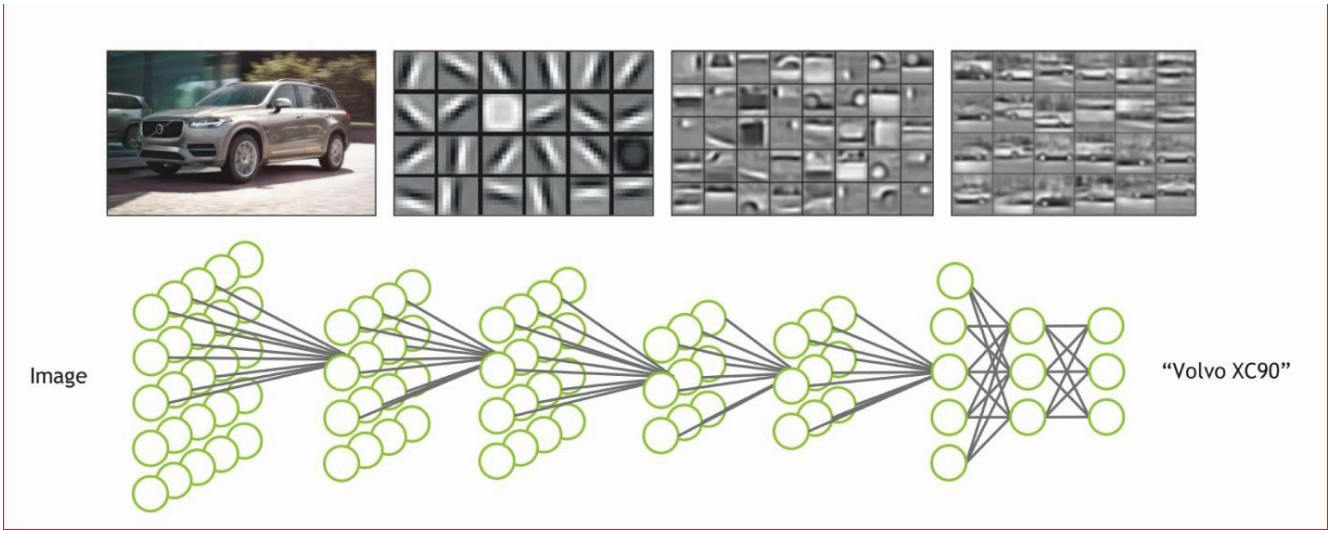


Рис. 18 – визуализация работы сверточной нейросети

## Платформы и инструменты

Настало время поговорить о том, где и с помощью каких инструментов пишут нейросети. Стоит начать с того, что чаще всего их пишут на языке программирования Python, так как он очень прост в освоении, интуитивно понятен и на него написано большое количество библиотек. Популярной оболочкой для написания нейросетей является Jupyter Notebook. Это интерактивный блокнот, первоначально являвшийся веб-реализацией и развитием IPython, ставший самостоятельным проектом. Поддерживает не только Python, но и R, Julia, Scala и ряд других языков программирования. Также в процессе работы я нашел альтернативу Jupyter – Google Colab, он очень схож по интерфейсу и функционалу, но, как по мне, немного проще в настройке и освоении. Хороши эти оболочки тем, что код пишется в отдельных ячейках и, если разработчик хочет посмотреть на график или формулу, он пишет нужную команду в соответствующей ячейке. Такой подход экономит время и помогает избежать ошибок. Также их используют специалисты по data science и начинающие программисты на Python. Далее рассмотрим библиотеки и прочие инструменты для создания нейросетей. Опять же их существует огромное множество, но мы вкратце рассмотрим лишь некоторые из них, самые популярные.

- TenserFlow - библиотека от Google, разработанная специально для обучения нейросетей. Для обработки больших объёмов данных использует многоуровневую систему узлов, что расширяет сферу применения далеко за научную область.
- PyTorch - библиотека для машинного и глубинного обучения от энтузиастов. Она имеет большое количество поклонников, отличается

полной документацией и удобным инструментарием, хорошо масштабируется.

- SciPy - это бесплатная библиотека с открытым исходным кодом, основанная на NumPy. Она особенно полезна для работы с большими наборами данных, способна выполнять научные и технические вычисления. SciPy также поставляется со встроенными модулями для оптимизации массивов и линейной алгебры, как и NumPy. Является одной из основополагающих библиотек Python благодаря своей роли в научном анализе и инженерии.
- MatPlotLib - это библиотека для визуализации данных на Python. Она широко используется для создания графиков и диаграмм, а также отлично подходит для визуализации данных. Matplotlib обладает широким спектром инструментов для создания различных типов графиков и часто используется в сочетании с другими библиотеками, такими как Pandas, для исследования данных.
- Pandas - это библиотека для обработки и анализа данных на Python. Она широко используется для работы со структурированными данными и отлично подходит для очистки, преобразования и анализа данных. Pandas имеет широкий спектр инструментов для работы с данными, включая объекты dataframe и series, которые похожи на таблицы и столбцы в SQL.
- NumPy - это библиотека для численных вычислений на Python. Она широко используется для работы с массивами и матрицами и отлично подходит для выполнения математических операций с данными. NumPy часто используется в сочетании с другими библиотеками, такими как SciPy и Pandas, для обработки и анализа данных.

## Метрики оценки нейронных сетей

В каждой задаче машинного обучения ставится вопрос оценки результатов моделей и поэтому были придуманы специальные способы, метрики, с помощью который стало возможно оценить работу нейросети. Почти наверняка наша модель будет ошибаться на некоторых объектах: будь она даже идеальной, шум или выбросы в тестовых данных всё испортят. При этом разные модели будут ошибаться на разных данных и в разной степени. Задача специалиста по машинному обучению – подобрать подходящий критерий, который позволит сравнивать различные модели. Метрики качества (Metrics) — это функции, по которым определяется качество обученной модели, но не происходит непосредственной оптимизации. Используются для контроля переобучения и для сравнения различных моделей. Метрик существует большое количество и сейчас мы попробуем разобраться какие они вообще бывают и как выбрать определенную метрику под конкретный случай.

Метрики делятся на два типа: offline и online. Online метрики вычисляются по данным, собираемым с работающей системы (например, медианная длина сессии). Offline метрики могут быть измерены до введения модели в эксплуатацию, например, по историческим данным или с привлечением специальных людей, асессоров. Перейдём к обзору метрик и начнём с самой простой разновидности классификации – бинарной, а затем постепенно будем наращивать сложность. В случае бинарной классификации метки принято обозначать как «+» и «-». Рассматриваемые далее метрики основаны на использовании следующих исходов: истинно положительные (TP), истинно отрицательные (TN), ложно положительные (FP) и ложно отрицательные (FN).

Первым критерием качества является **accuracy** – доля объектов, для которых мы правильно предсказали класс, описывается он формулой (Рис. 19). Однако познакомившись чуть внимательнее с этой метрикой, можно заметить, что у неё есть несколько недостатков:

- она не учитывает дисбаланс классов (когда какой-то из классов соответствует очень редко наблюдаемым или диагностируемым явлениям);
- А также не учитывает цену ошибки на объектах разных классов.

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Рис. 19 – формула для расчета оценки качества «accuracy»

Следующая метрика называется **precision**. Эта точность показывает количество истинно положительных исходов из всего набора положительных меток и считается по следующей формуле (Рис. 20). Важность этой метрики определяется тем, насколько высока для рассматриваемой задачи «цена» ложно положительного результата.

$$precision = \frac{TP}{TP + FP}$$

Рис. 20 – формула для расчета оценки качества «precision»

Если же мы рассмотрим долю правильно найденных положительных объектов среди всех объектов положительного класса, то мы получим метрику, которая называется **полнотой (recall)**. Получается она по

следующей формуле (Рис. 21). Интуитивно метрика показывает долю найденных документов из всех релевантных. Чем меньше ложно отрицательных срабатываний, тем выше recall модели.

$$\text{Recall} = \frac{TP}{TP + FN}$$

Рис. 21 - формула для расчета оценки качества «recall»

Как мы уже знаем, модели очень удобно сравнивать, когда их качество выражено одним числом. В случае пары Precision-Recall существует популярный способ скомпоновать их в одну метрику - взять их среднее гармоническое. Данный показатель эффективности носит название **F1-меры (F1-measure).** Стоит иметь в виду, что F1-мера предполагает одинаковую важность Precision и Recall. Вычисляется по формуле (Рис. 22).

$$F_\beta = (1 + \beta^2) \cdot \frac{precision \cdot recall}{(\beta^2 \cdot precision) + recall}$$

Рис. 22 - формула для расчета оценки качества «F-1 мера»

Следующая метрика ROC (receiver operating characteristic) – график (Рис. 23), показывающий зависимость верно классифицируемых объектов положительного класса от ложно-положительно классифицируемых объектов негативного класса. С помощью ROC — кривой, можно сравнить модели, а также их параметры для поиска наиболее оптимальной

комбинации. Чем лучше классификатор разделяет два класса, тем больше площадь под ROC-кривой – и мы можем использовать её в качестве еще одной метрики. Эта метрика называется AUC (area under curve) и она работает благодаря следующему свойству ROC-кривой. AUC равен доле пар объектов, которые алгоритм верно упорядочил, т.е. предсказание классификатора на первом объекте больше. Стоит отметить, что в любой задаче, где нам важна не метка сама по себе, а правильный порядок на объектах, имеет смысл применять AUC.

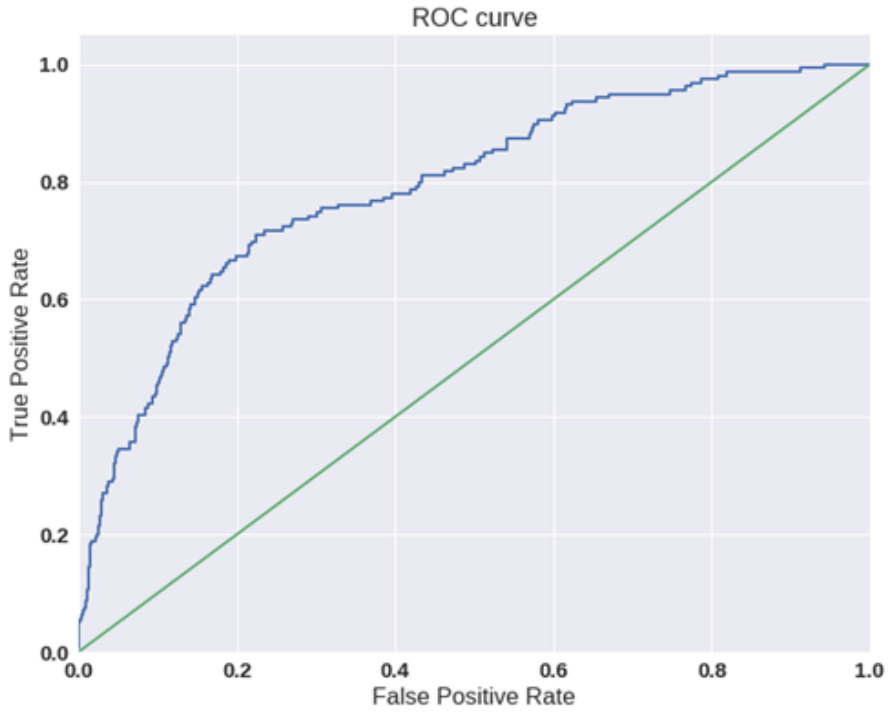


Рис. 23 – график ROC кривой

Если классов становится больше двух, расчёт метрик усложняется. Есть два варианта получения итогового значения метрики из матриц ошибок:

- Усредняем элементы матрицы ошибок (TP, FP, TN, FN) между бинарными классификаторами. Это называют **микроусреднением.**
- Считаем Precision, Recall для каждого классификатора отдельно, а потом усредняем. Это называют **макроусреднением**.

Усреднение первым способом сделает вклад маленького класса в общую метрику незаметным. А при усреднении вторым способом среднее считается уже для нормированных величин, так что вклад каждого класса будет одинаковым.

**Применение нейронных сетей**

Уже сегодня искусственные нейронные сети используются во многих областях. Это распознавание текста и речи, голосовые помощники, автопилоты, игровая индустрия, медицинская диагностика, робототехника, семантический поиск, экспертные системы и системы поддержки принятия решений, предсказание курсов акций, системы безопасности, анализ текстов и многие-многие другие. Рассмотрим несколько особенно ярких и интересных примеров использования нейронных сетей в разных областях.

- В 1996 году фирмой «Accurate Automation Corp» по заказу NASA и «Air Force» был разработан экспериментальный автопилотируемый гиперзвуковой самолет-разведчик LoFLYTE (Low-Observable Flight Test Experiment). Он использовал нейронные сети, позволяющие автопилоту обучаться, копируя приемы пилотирования летчика. Поскольку самолет был предназначен для полетов со скоростью 4-5 махов, то быстрота реакции пилота-человека могла быть недостаточной для адекватного отклика на изменение режима полета. В этом случае на помощь приходили нейронные сети.
- Определение тематики текстовых сообщений — еще один пример успешного использования искусственных нейронных сетей. Так, сервер новостей Convectis был выбран в 1997 году компанией «PointCast» для автоматической рубрикации сообщений по категориям. Определяя значения ключевых слов по контексту, сервер Convectis был способен в реальном времени распознавать тематику и автоматически рубрицировать огромные потоки текстовых сообщений, передаваемых по таким информационным сетям.

Нейросетевой продукт позволял определять область интересов пользователей Интернета и предлагал им рекламу соответствующей тематики.

- В медицинской диагностике нейронные сети нередко используются вместе с экспертными системами. Компанией «НейроПроект» была создана система объективной диагностики слуха у грудных детей. Общепринятая методика диагностики состоит в том, что в процессе обследования регистрируются отклики мозга в ответ на звуковой раздражитель, проявляющиеся в виде всплесков на электроэнцефалограмме. Для диагностики слуха ребенка опытному эксперту-аудиологу необходимо провести около 2 тыс. тестов, нейронная сеть способна с той же достоверностью определить уровень слуха уже по 200 наблюдениям в течение всего нескольких минут, причем без участия специалиста.

## **Практическая часть**

Теперь нам хватает знаний, чтобы создать свою модель нейронной сети. В моем случае это будет рекурентная нейронная сеть, которая будет решать проблему классификации изображений, а точнее рукописных знаков. Для этого мне понадобится большой дата сет рукописных цифр, для этого я взял открытый дата сет, который включает в себя около 10000 изображений и делится в соотношение 2:8 на тестовую и обучающую группы. Пример готового кода, который был написан в интерактивном блокноте Jupyter Notebook на языке программирования Python, выглядит так:

```
import numpy
# scipy.special for the sigmoid function expit()
import scipy.special
# library for plotting arrays
import matplotlib.pyplot
# ensure the plots are inside this notebook, not an external window
%matplotlib inline
```

```
# neural network class definition
class neuralNetwork:


    # initialise the neural network
    def __init__(self, inputnodes, hiddennodes, outputnodes, learningrate):
        # set number of nodes in each input, hidden, output layer
        self.inodes = inputnodes
        self.hnodes = hiddennodes
        self.onodes = outputnodes

        # link weight matrices, wih and who
        # weights inside the arrays are w_i_j, where link is from node i to node j in the next layer
        # w11 w21
        # w12 w22 etc
        self.wih = numpy.random.normal(0.0, pow(self.inodes, -0.5), (self.hnodes, self.inodes))
        self.who = numpy.random.normal(0.0, pow(self.hnodes, -0.5), (self.onodes, self.hnodes))

        # learning rate
        self.lr = learningrate

        # activation function is the sigmoid function
        self.activation_function = lambda x: scipy.special.expit(x)

        pass


    # train the neural network
    def train(self, inputs_list, targets_list):
        # convert inputs list to 2d array
        inputs = numpy.array(inputs_list, ndmin=2).T
        targets = numpy.array(targets_list, ndmin=2).T

        # calculate signals into hidden layer
        hidden_inputs = numpy.dot(self.wih, inputs)
        # calculate the signals emerging from hidden layer
        hidden_outputs = self.activation_function(hidden_inputs)

        # calculate signals into final output layer
        final_inputs = numpy.dot(self.who, hidden_outputs)
        # calculate the signals emerging from final output layer
        final_outputs = self.activation_function(final_inputs)

        # output layer error is the (target - actual)
        output_errors = targets - final_outputs
        # hidden layer error is the output_errors, split by weights, recombined at hidden nodes
        hidden_errors = numpy.dot(self.who.T, output_errors)

        # update the weights for the links between the hidden and output layers
        self.who += self.lr * numpy.dot((output_errors * final_outputs * (1.0 - final_outputs)), numpy.transpose(hidden_out

        # update the weights for the links between the input and hidden layers
        self.wih += self.lr * numpy.dot((hidden_errors * hidden_outputs * (1.0 - hidden_outputs)), numpy.transpose(inputs))

        pass


    # query the neural network
    def query(self, inputs_list):
        # convert inputs list to 2d array
        inputs = numpy.array(inputs_list, ndmin=2).T

        # calculate signals into hidden layer
        hidden_inputs = numpy.dot(self.wih, inputs)
        # calculate the signals emerging from hidden layer
        hidden_outputs = self.activation_function(hidden_inputs)

        # calculate signals into final output layer
        final_inputs = numpy.dot(self.who, hidden_outputs)
        # calculate the signals emerging from final output layer
        final_outputs = self.activation_function(final_inputs)

        return final_outputs
```

```
# number of input, hidden and output nodes
input_nodes = 784
hidden_nodes = 200
output_nodes = 10

# learning rate
learning_rate = 0.1

# create instance of neural network
n = neuralNetwork(input_nodes,hidden_nodes,output_nodes, learning_rate)
```

```
# load the mnist training data CSV file into a list
training_data_file = open("data_set\mnist_train.csv", 'r')
training_data_list = training_data_file.readlines()
training_data_file.close()
```

```
# train the neural network

# epochs is the number of times the training data set is used for training
epochs = 5

for e in range(epochs):
    # go through all records in the training data set
    for record in training_data_list:
        # split the record by the ',' commas
        all_values = record.split(',')
        # scale and shift the inputs
        inputs = (numpy.asfarray(all_values[1:]) / 255.0 * 0.99) + 0.01
        # create the target output values (all 0.01, except the desired label which is 0.99)
        targets = numpy.zeros(output_nodes) + 0.01
        # all_values[0] is the target label for this record
        targets[int(all_values[0])] = 0.99
        n.train(inputs, targets)
        pass
    pass
```

```
# load the mnist test data CSV file into a list
test_data_file = open("data_set\mnist_test.csv", 'r')
test_data_list = test_data_file.readlines()
test_data_file.close()
```

```
# test the neural network

# scorecard for how well the network performs, initially empty
scorecard = []

# go through all the records in the test data set
for record in test_data_list:
    # split the record by the ',' commas
    all_values = record.split(',')
    # correct answer is first value
    correct_label = int(all_values[0])
    # scale and shift the inputs
    inputs = (numpy.asfarray(all_values[1:]) / 255.0 * 0.99) + 0.01
    # query the network
    outputs = n.query(inputs)
    # the index of the highest value corresponds to the label
    label = numpy.argmax(outputs)
    # append correct or incorrect to list
    if (label == correct_label):
        # network's answer matches correct answer, add 1 to scorecard
        scorecard.append(1)
    else:
```

Рис. 24 – код написанной нейросети

Мною было проведено два запуска. Первый для оценки работоспособности сети, который показывает, насколько точно выдает ответ наша нейросеть. Результат оказался очень хорошим – 97 %.

In [6]:
```
# load the mnist test data CSV file into a list
test_data_file = open("data_set\mnist_test.csv", 'r')
test_data_list = test_data_file.readlines()
test_data_file.close()
```

In [7]:
```
# test the neural network

# scorecard for how well the network performs, initially empty
scorecard = []

# go through all the records in the test data set
for record in test_data_list:
    # split the record by the ',' commas
    all_values = record.split(',')
    # correct answer is first value
    correct_label = int(all_values[0])
    # scale and shift the inputs
    inputs = (numpy.asfarray(all_values[1:]) / 255.0 * 0.99) + 0.01
    # query the network
    outputs = n.query(inputs)
    # the index of the highest value corresponds to the label
    label = numpy.argmax(outputs)
    # append correct or incorrect to list
    if (label == correct_label):
        # network's answer matches correct answer, add 1 to scorecard
        scorecard.append(1)
    else:
        # network's answer doesn't match correct answer, add 0 to scorecard
        scorecard.append(0)
        pass

    pass
```

In [8]:
```
# calculate the performance score, the fraction of correct answers
scorecard_array = numpy.asarray(scorecard)
print ("performance = ", scorecard_array.sum() / scorecard_array.size)
```

```
performance =  0.9734
```

Рис. 25 – количественный показатель корректной работы сети

Второй с загрузкой тестовых данных вручную, заметим, что нейросеть отлично справилась со своей задачей.

```
# the index of the highest value corresponds to the label
label = numpy.argmax(outputs)
print("network says ", label)
# append correct or incorrect to list
if (label == correct_label):
    print ("match!")
else:
    print ("no match!")
    pass
```

```
[[2.68263804e-03]
 [1.58367502e-03]
 [5.81142444e-02]
 [4.54185024e-04]
 [9.87810696e-01]
 [6.47654066e-03]
 [1.10899521e-03]
 [4.44764937e-03]
 [1.65304935e-04]
 [1.99217027e-04]]
network says  4
match!
```

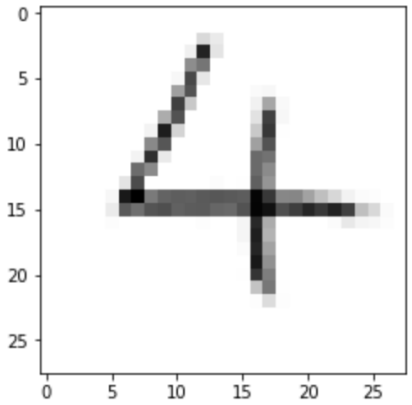


Рис. 26 – пример корректной классификации изображения

## **Заключение.**

В настоящее время искусственные нейронные сети являются важным расширением понятия вычисления. Они уже позволили справиться с рядом непростых проблем и обещают создание новых программ и устройств, способных решать задачи, которые пока под силу только человеку, но самые громкие технологические революции все еще ждут нас в будущем. В рамках своего проекта я постарался максимально широко и доступно раскрыть многие аспекты, связанные с нейронными сетями: историю создания, их работу, архитектуры, инструменты и библиотеки, метрики оценки и применение. А также сделал свою нейронную сеть, которая решает задачи классификации изображений.

**Список используемой литературы:**


1. [1]The ASIMOV INSTITUTE в статье «The Neural Network ZOO»
2. David Kriesel «A Brief Introduction to Neural Networks» - dkriesel.com, 2005, 219 стр.
3. Tariq Rashid «Make your own neural network» - Москва «Диалектика», 2016, 270 стр.
4. Саймон Хайкин «Нейронные сети: полный курс» - Москва «Вильямс», 2006, 1069 стр.
5. Петер Флах «Машинное обучение» - Москва «ДМК», 2015, 400 стр.
6. ИТМО «Обзор библиотек для машинного обучения на Python»
7. Научная статья: Голядкин Р. В., Игнатенко В. А. «Обзор библиотек для работы с нейронными сетями», БГАУ имени В.Я. Горина, 1 п. Майский, Белгородская область // Научная электронная библиотека Elibrary.ru
8. Научная статья: Горячкин Б. С., Китов М. А. «Компьютерное зрение», МГТУ им. Н.Э. Баумана // Научная электронная библиотека «КиберЛенинка»
9. Научная статья: М. В. Сычугов «Моделирование прогноза временных метрик на основе нейронных сетей», известия ЮФУ. Технические науки // Научная электронная библиотека Elibrary.ru
10. Научная статья: Н. О. Турсуков, И. И. Викснин, Е. А. Неверов, Е. Л. Шейнман, С. С. Чупров «Оценка эффективности нейронных сетей на основе критериев выполнения задачи классификации объектов», Санкт-Петербургский государственный электротехнический университет «ЛЭТИ» им. В.И. Ульянова (Ленина) // Научная электронная библиотека Elibrary.ru
11. Научная статья: Фаустова К. И. «Нейронные сети: применение сегодня и перспективы развития», Воронежский экономико-правовой институт // Научная электронная библиотека «КиберЛенинка»

12. Научная статья: Богославский С. Н. «Область применения искусственных нейронных сетей и перспективы их развития», научный журнал КубГАУ // Научная электронная библиотека «КиберЛенинка»
13. Статья: Губко Павел, Горчаков Алексей, Буркина Мария «Метрики классификации и регрессии» // Яндекс образование