Государственное бюджетное общеобразовательное учреждение гимназия № 402 Колпинского района Санкт-Петербурга

Школьный проект

Автоматизация идентификации инфекционных заболеваний лёгких по рентгеновскому снимку при помощи нейронной сети

Тихонов Александр Станиславович

10 Класс

Тьютор: Рулёва Татьяна Борисовна

Колпино

2021-2022

Оглавление

[Оглавление 1](#_Toc89799926)

[Введение 3](#_Toc89799927)

[1. Введение в глубокое обучение 5](#_Toc89799928)

[1.1. Машинное обучение 5](#_Toc89799929)

[1.1.1. Обучение с “учителем” 5](#_Toc89799930)

[1.1.2. Обучение без “учителя” 5](#_Toc89799931)

[1.1.3. Параметрическое и непараметрическое обучение 6](#_Toc89799932)

[1.2. Выбор алгоритма обучения 7](#_Toc89799933)

[2. Принцип работы нейронных сетей 7](#_Toc89799934)

[2.1. Принцип прямого распространения в полносвязных слоях 9](#_Toc89799935)

[3. Инструменты работы с нейронной сетью 10](#_Toc89799936)

[3.1. Выбор изначальных весовых коэффициентов 10](#_Toc89799937)

[3.2. Функции активации 11](#_Toc89799938)

[3.3. Даунсемплинг 11](#_Toc89799939)

[3.4. Прямое распространение 13](#_Toc89799940)

[3.5. Обратное распространение ошибок 15](#_Toc89799941)

[3.6. Обновление коэффициентов весов 19](#_Toc89799942)

[4. Создание нейронной сети на Python 21](#_Toc89799943)

[4.1. Подготовка тренировочных данных 21](#_Toc89799944)

[4.2. Архитектура создаваемой модели 23](#_Toc89799945)

[4.3. Программирование нейронной сети 24](#_Toc89799946)

[4.3.1. Скелет кода 24](#_Toc89799947)

[4.3.2. Биасы 25](#_Toc89799948)

[4.3.3. Инициализация весовых коэффициентов 25](#_Toc89799949)

[4.3.4. Функция активации и её производная 26](#_Toc89799950)

[4.3.5. Метод максимального объединения 27](#_Toc89799951)

[4.3.6. Прямое распространение по сети 27](#_Toc89799952)

[4.3.7. Высчитывание ошибок полносвязных слоёв и биасов 29](#_Toc89799953)

[4.3.8. Обновление весовых коэффициентов 30](#_Toc89799954)

[4.4. Обучение нейронной сети 31](#_Toc89799955)

[4.5. Анализ лучшей модели 31](#_Toc89799956)

[5. Заключение 34](#_Toc89799957)

[Ссылки на источники 35](#_Toc89799958)

Введение

Всё чаще можно увидеть применение искусственного интеллекта в различных сферах, ведь благодаря нему можно автоматизировать абсолютно любую задачу. Медицина также не стала исключением. Нейронные сети в медицине используется в разнообразных целях:

* Предсказание падения артериального давления
* Распознавание рака кожи
* Выявление патологий по УЗИ и МРТ, точность которых превышает 90%
* Помощь в разработке новых лекарственных препаратов
* Мониторинг состояния пациента
* Сфера протезирования
* Лечение раковых заболеваний (стартапы)

Врачи нередко используют КТ и Рентген для выявления различных опухолей, новообразований, абсцессов, а также пневмонии у пациентов. Так как при пневмонии в лёгких образуется инфильтрат (отечный воспалённый очаг), то с помощью рентгеновских снимков можно изучить состояние лёгких человека, что несомненно очень актуально в период эпидемии COVID-19. А поскольку рентгеновские аппараты имеются в наличии практически в каждой клинике, то процесс обнаружения COVID-19 (или другой острой респираторной инфекции) становиться куда проще и быстрее.

Однако, основной проблемой данного метода является потребность в специалистах-рентгенологах, а также время, требуемое на анализ снимка. Именно поэтому необходимо создать систему для автоматизации данного процесса.

**Цель проекта —** создание нейронной сети, которая способна автоматизировать процесс определения поражения лёгких человека инфекционным заболеванием COVID-19, а также SARS, MERS-CoV, Varicella, Influenza, Herpes и др. при помощи рентгеновских снимков.

**Задачи проекта**:

* Изучить принцип работы нейронных сетей
* Изучить технологии машинного обучения, которые помогут достигнуть цели
* Подготовить данные для обучения и тестирования сети
* Определить архитектуру нейронной сети
* Обучить нейросеть

В качестве инструмента будет выступать язык программирования Python с библиотеками: numpy, matplotlib, pillow, pickle, tqdm.

1. Введение в глубокое обучение

Глубокое обучение — подмножество методов машинного обучения, основанных на искусственных нейронных сетях, области изучения и создания машин, способных обучаться.

Процесс обучения называется глубоким, так как алгоритм содержит множество скрытых слоёв с искусственными нейронами. Таким образом каждый слой сообщает следующему сведения, полученные при помощи содержащихся в них единиц, способных преобразовать данные определённым образом.

# 1.1. Машинное обучение

Машинное обучение — это область ИКТ, в которой машины учатся решать задачи, для которых не были запрограммированы непосредственно. То есть машины выявляют закономерности и пытаются прямо или косвенно их имитировать.

## 1.1.1. Обучение с “учителем”

Между прямой и косвенной имитациями имеется разница, ведь существует два основных вида машинного обучения: с учителем и без учителя. Машинное обучение с учителем — это прямая имитация выявленных закономерностей между двумя наборами данных (входной информации и её результата). Такие модели называют предикторами. Например:

* Использование истории просмотренных книг для подбора рекомендаций для пользователя.
* Использование новостей для предсказаний котировок на бирже.

## 1.1.2. Обучение без “учителя”

Машинное обучение без учителя также получает некий набор входных данных, но не имеет целевого. Основная задача в таком случае — найти закономерности в полученных данных и сообщить об этом.

Ярким примером обучения сети без учителя: кластеризация набора данных на группы. Во время процесса кластеризации машина ищет общие черты в информации и отделяет по общим признакам в группы.

## 1.1.3. Параметрическое и непараметрическое обучение

Помимо деления на вид обучения с учителем и без, машинное обучение делится на параметрическое и непараметрическое обучения.

Основным отличием параметрического и непараметрического обучения заключается в количестве параметров, которые влияют на процесс обучения модели. Таким образом в параметрических моделях имеется фиксированное количество параметров, которые по мере обучения регулируются для достижения правильного предсказания, в непараметрических моделях их количество не предопределено и зависит от объёма входных данных.

Для полного понимания разницы моделей следует рассмотреть их отдельно:

* Параметрическое обучение с учителем — алгоритм, обладающий фиксированным количеством параметров (параметрическая часть модели), обучение которой происходит путём их коррекции. Данный способ можно назвать: “метод проб и ошибок”, ведь во время обучения коррекция может привести как к улучшению результатов, так и к их ухудшению.
* Параметрическое обучение без учителя — алгоритм, в который также обладает фиксированным количеством параметров, равному числу групп, ведь её целью является группировка данных (с некоторыми исключениями). Обучение также проходит путём коррекции параметров “методом проб и ошибок”, для определения вероятности принадлежности данных к каждой группе.
* Непараметрическое обучение — алгоритм, число параметров которого зависит от выявленных признаков во входных данных (т.е. неопределенно).

# 1.2. Выбор алгоритма обучения

Для создания нейронной сети, предсказывающей поражение лёгких инфекционными заболеваниями, было принято решение использовать алгоритм глубокого параметрического обучения с учителем. Данный алгоритм полностью соответствует требованиям к нейросети: входные данные имеют одинаковый объём, т.к. будут предварительно обработаны, а также выход будет являться предсказанием, т.е. будет содержать в себе шанс поражения лёгких.

2. Принцип работы нейронных сетей

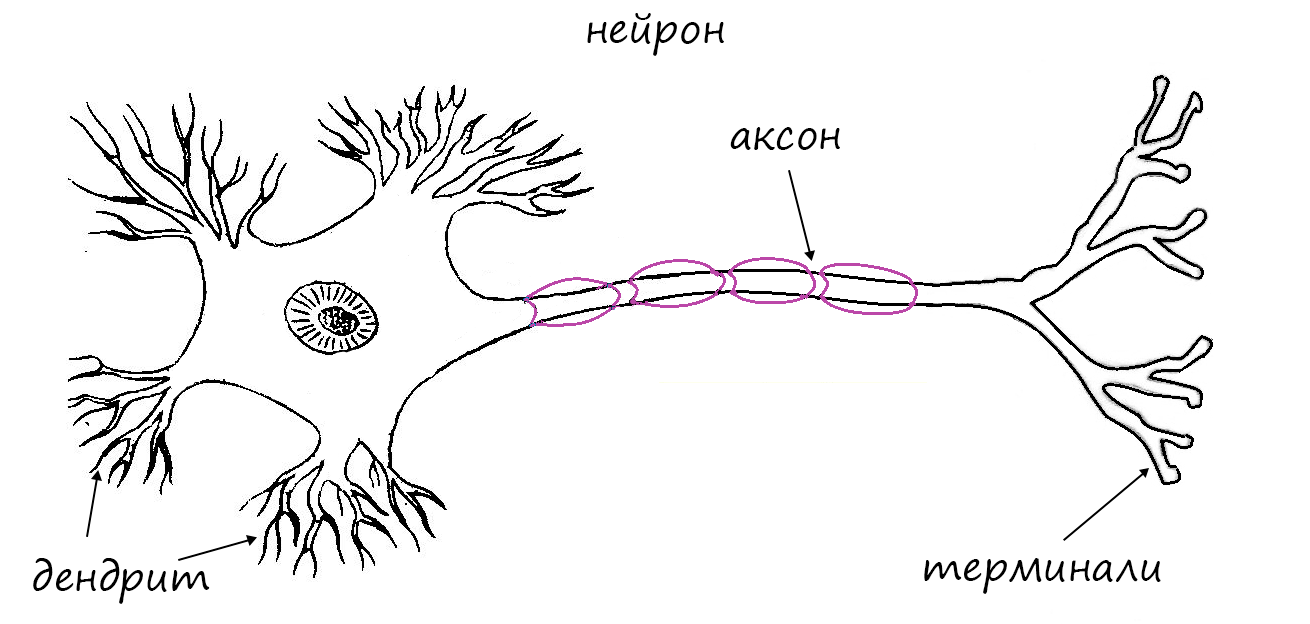
Для того, чтобы лучше понять принцип работы нейросетей необходимо рассмотреть строение базовой структурно-функциональной единицы биологического мозга — *нейрона* (рис. 1).

Рисунок 1 – Строение нейрона

Несмотря на разнообразие возможных форм нейронов, все они передают электрические сигналы через себя к другим нейронам — от дендритов через аксон до терминалей. Таким образом человек, например, способен воспринимать звук, свет, прикосновения и т.п. Сигналы проходят от рецепторных нейронов по нервной системе до мозга, который в основном состоит из нейронов.

Таким образом в головной коре человека содержится порядка 90 млрд нейронов, благодаря чему мы способны производить сложнейшие операции. Ещё совсем недавно компьютер не имел достаточной вычислительной мощности для имитации такого количества нейронов, однако в июне 2020 года исследовательская лаборатория искусственного интеллекта OpenAI создала языковую модель, обладающую 175 миллиардами параметров машинного обучения [1]. На данный момент это самая большая нейронная сеть в мире. Основной целью данной языковой модели является создание текста, похожего на человеческий.

К сожалению, даже такая мощная нейросеть не обладает биологическим сознанием, которое даже для самого человека остаётся загадкой, но несмотря на это, мы знаем о нейронах достаточно много для того, чтобы можно было предложить способы выполнения вычислений, схожих с происходящими между нейронами.

Согласно результатам наблюдений нейроны не реагируют на входной сигнал немедленно, а подавляют его до тех пор, пока он не достигнет уровня, достаточного для запуска генерации выходного сигнала. Это можно сравнить с водой в стакане, которая не сможет выплеснуться, пока чашка не заполнится полностью.

Функция, принимающая входной сигнал и генерирующая выходной исходя из порогового значения называется функцией активации. В математике существует множество таких функций: ступенчатая, логистическая (сигмоидальная), гиперболическая и т.д.

Подводя итог, принцип работы нейросетей можно описать так: дендриты принимают электрические сигналы, комбинируют их, формируя более сильный сигнал, после чего, если этот сигнал превысил порог, нейрон возбуждается, и сигнал проходит через аксон к терминалям, чтобы поступить на дендриты следующих нейронов.

# 2.1. Принцип прямого распространения в полносвязных слоях

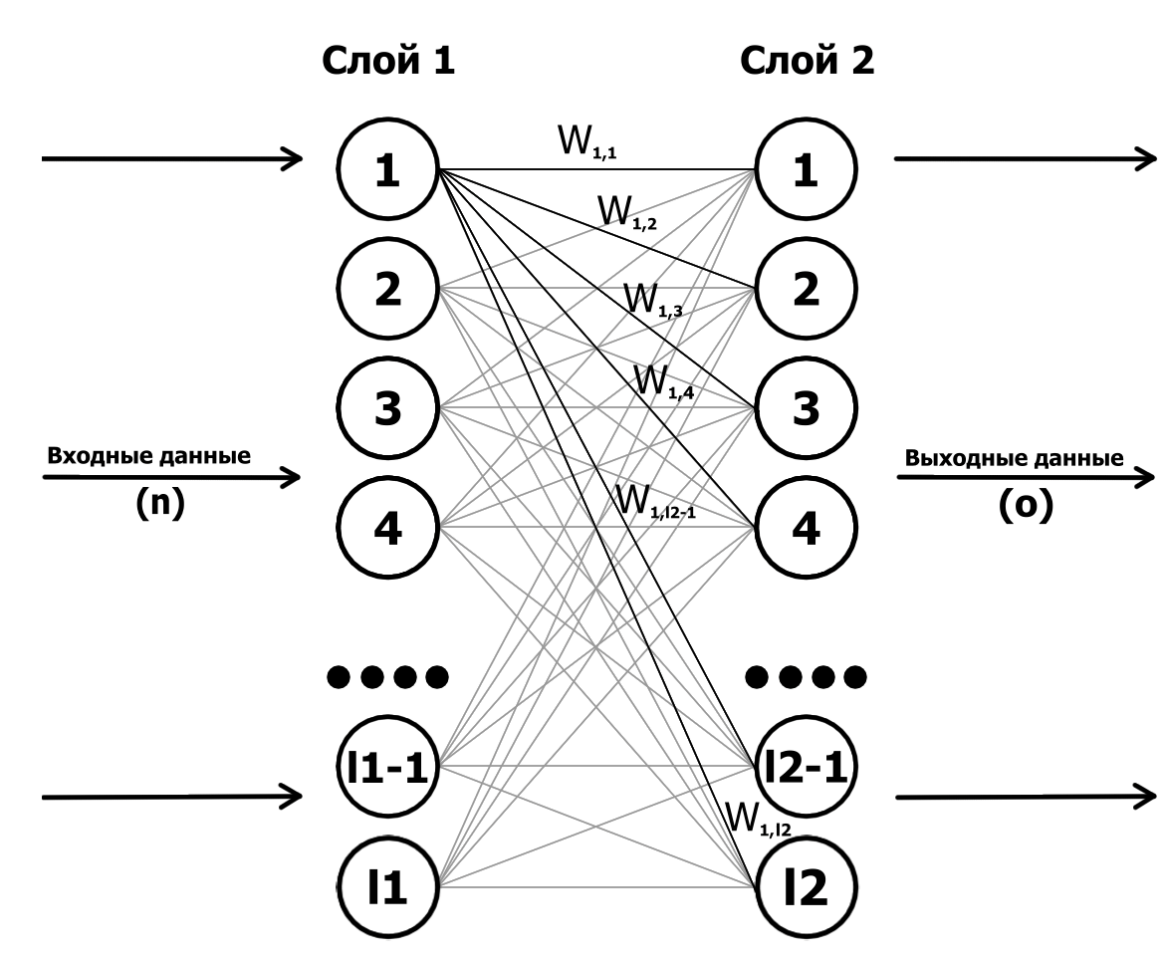
Принцип прямого распространения состоит в том, чтобы получить выходные данные путём преобразований входных данных, используя полученные весовые коэффициенты по итогу обучения модели. Т.е. входные данные умножаются на коэффициенты и результат передаётся на следующий слой, где данная операция повторяется, пока не достигнет выходного слоя.

Рисунок 2 – Полносвязные слои

Для того, чтобы лучше понять принцип прямого распространения выше приведена картинка (рис. 2), на которой:

* Количество входных данных — n
* Количество нейронов 1 слоя — l1
* Количество нейронов 2 слоя — l2
* Количество выходных данных — o

Однако знание о количестве искусственных нейронов не поможет выполнить прямое распространение. На картинке (рис. 2) видно, что каждый следующий нейрон связан с каждым нейроном предыдущего слоя. Именно регулировка силы связей нейронов между собой и придаёт нейронной сети способность обучаться.

Обозначением силы связи соединения нейронов выступает определённый вес. Чем ниже значение веса, тем ниже сила сигнала между нейронами, выше — сильнее. На рисунке 2 веса обозначены коэффициентом W и у каждого веса есть индексы. Они обозначают номера связанных нейронов в текущем слое и следующем.

Таким образом, основной целью нейронной сети является улучшение своих выходных значений путём уточнения весовых коэффициентов связей внутри сети, т.е. какие-то веса будут стремится к нулю, другие будут расти. Это значит, что по мере обучения некоторые связи не будут оказывать влияния на результат, поскольку из-за маленького весового коэффициента их сигналы не будут передаваться.

Также существуют рекуррентные нейронные сети, которые обладают внутренней памятью, что позволяет нечто целое, разбитое на части, лучше анализировать. Однако в целях проекта кратковременная память не нужна, поэтому будет использован принцип прямого распространения

3. Инструменты работы с нейронной сетью

В предыдущей главе был рассмотрен принцип работы нейронных сетей и, в частности, прямого распространения. Однако для создания сети данных знаний недостаточно. В этой главе будут описаны применённые в создаваемой нейросети технологии.

# 3.1. Выбор изначальных весовых коэффициентов

В работе было принято решение использовать случайно сгенерированные веса для каждой связи в интервале (0, 1). Данное решение было принято из-за того, что к данным весовым коэффициентам будет применяться функция активации sigmoid, которая при принятии большого значения будет выдавать 1.0, что перенасытит нейросеть, ведь при умножении на 1 получится то же число.

# 3.2. Функции активации

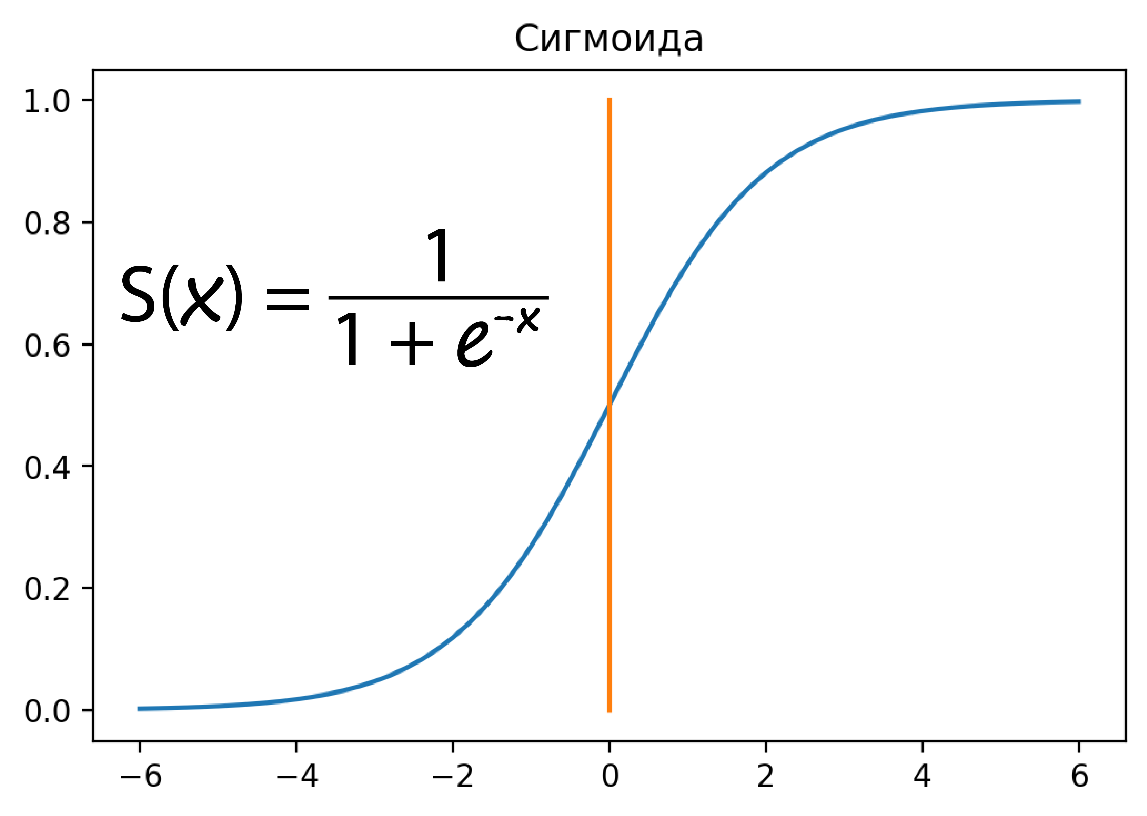
****В разрабатываемой нейронной сети проекта было принято решение использовать логистическую функцию активации (рис. 3)

Рисунок 3– Сигмоидальная функция

У S-образной сигмоидальной функции границы сглажены, что делает её выходы более естественными. Она очень популярна и проста, а также имеет на выходе значения в интервале (0, 1) из-за чего отлично подходит для двоичной классификации.

# 3.3. Даунсемплинг

Для ускорения процесса обучения было принято решение добавить первым слоем слой, производящий даунсемплинг входных данных. Существует несколько способов сделать это. В проекте будет использоваться технология максимального объединения (max-pooling).

Суть максимального объединения заключается в том, чтобы применить некий фильтр к матрице, в данном случае такой, который из видимого им окна будет выбирать максимальное число, записывать его в результирующую матрицу, и двигаться дальше по входной матрице с заданным шагом. Таким образом чем больше шаг перемещения фильтра, тем сильнее будут ужаты исходные данные.

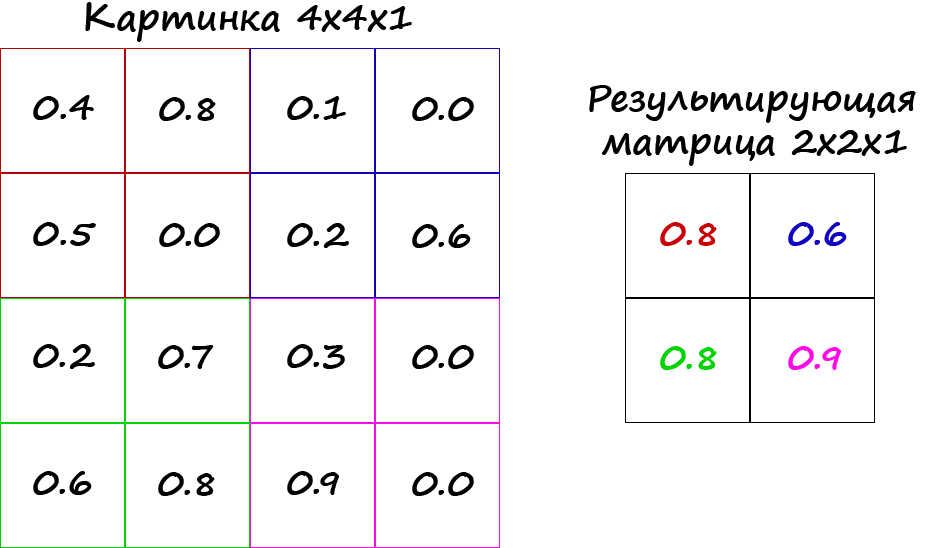
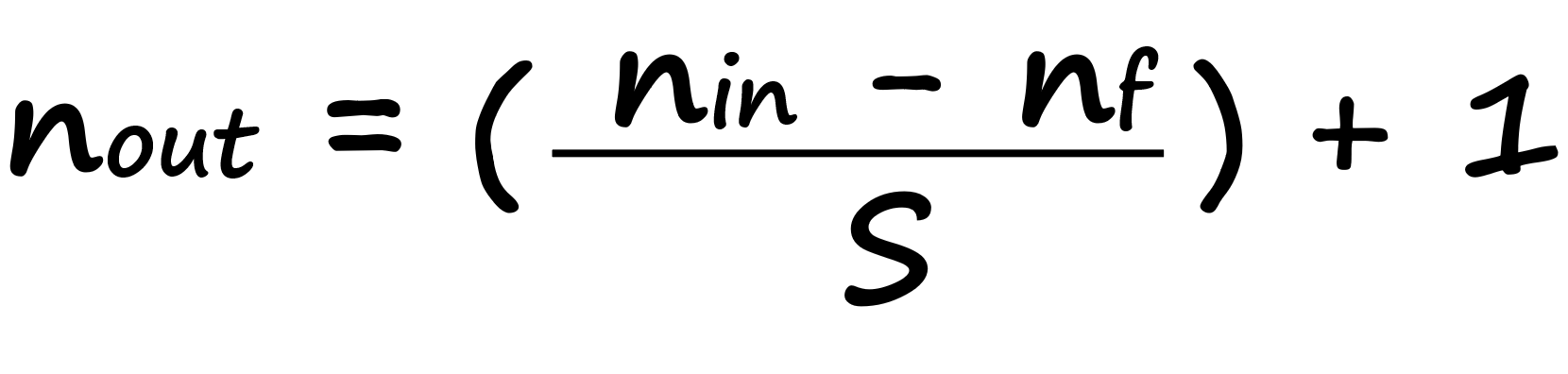
Для лучшего понимания ниже приведён пример применения max-pooling с фильтром 2x2 и шагом 2, на картинке размером 4x4 (рис. 4).

Рисунок 4– Пример работы метода max-pooling

Количество получаемых значений после применения max-pooling можно рассчитать по формуле 1. В этой формуле **n**in — количество входных пикселов, **n**f — количество пикселов в фильтре, **S** — шаг перемещения фильтра.

Формула 1 — Формула для расчёта количества выходных пикселов после применения max-pooling

Помимо оптимизации сети операция максимального объединения также способствует выделению весомых коэффициентов изображения, игнорируя менее значимые детали, а также не даёт сети быстро переобучаться.

# 3.4. Прямое распространение

Для перемножения массивного количества значений существует компактный подход, который предполагает использование матриц. То есть входные данные представляются (за частую) в виде одномерной матрицы, где все данные располагаются в одном столбце. Матрица весовых коэффициентов в таком случае представлена двухмерной, связывающей входные данные с каждым нейроном следующего слоя.

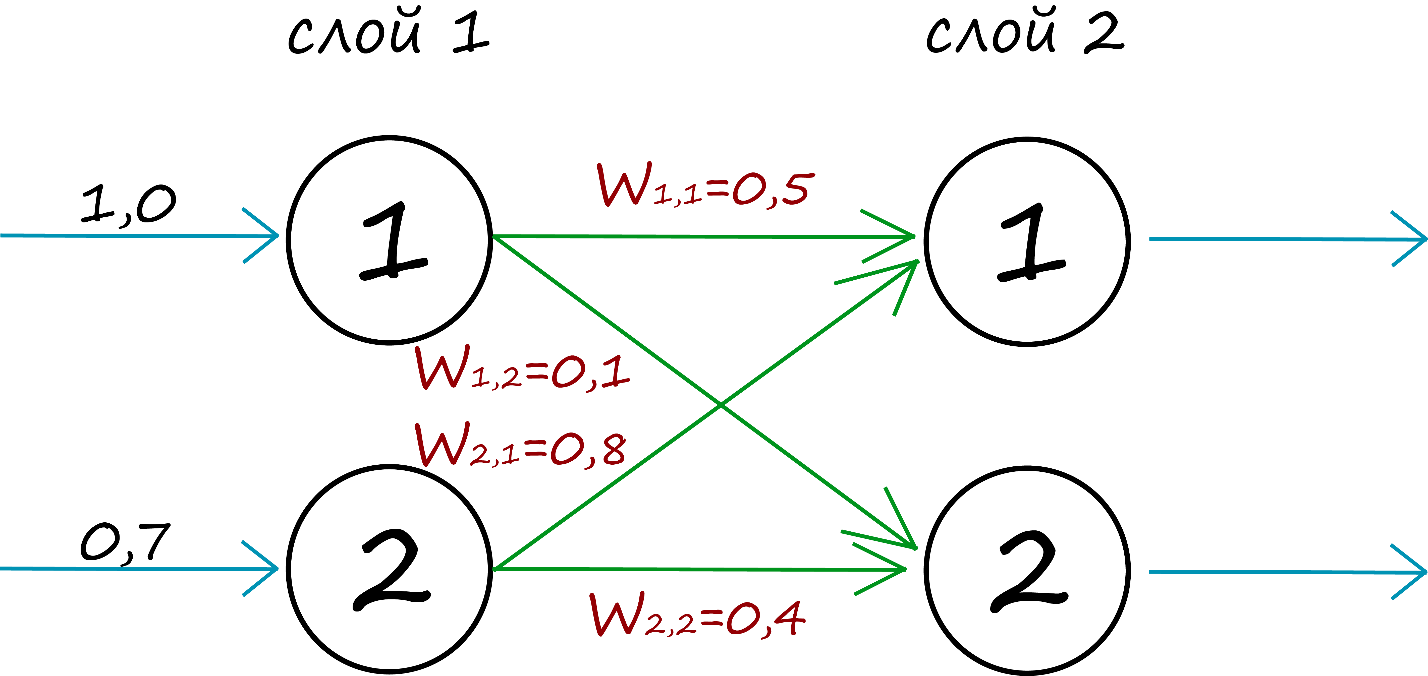
Для понимания того, почему принято решение использовать матрицы приведём пример (рис. 5)

Рисунок 5 — Пример перцептрона

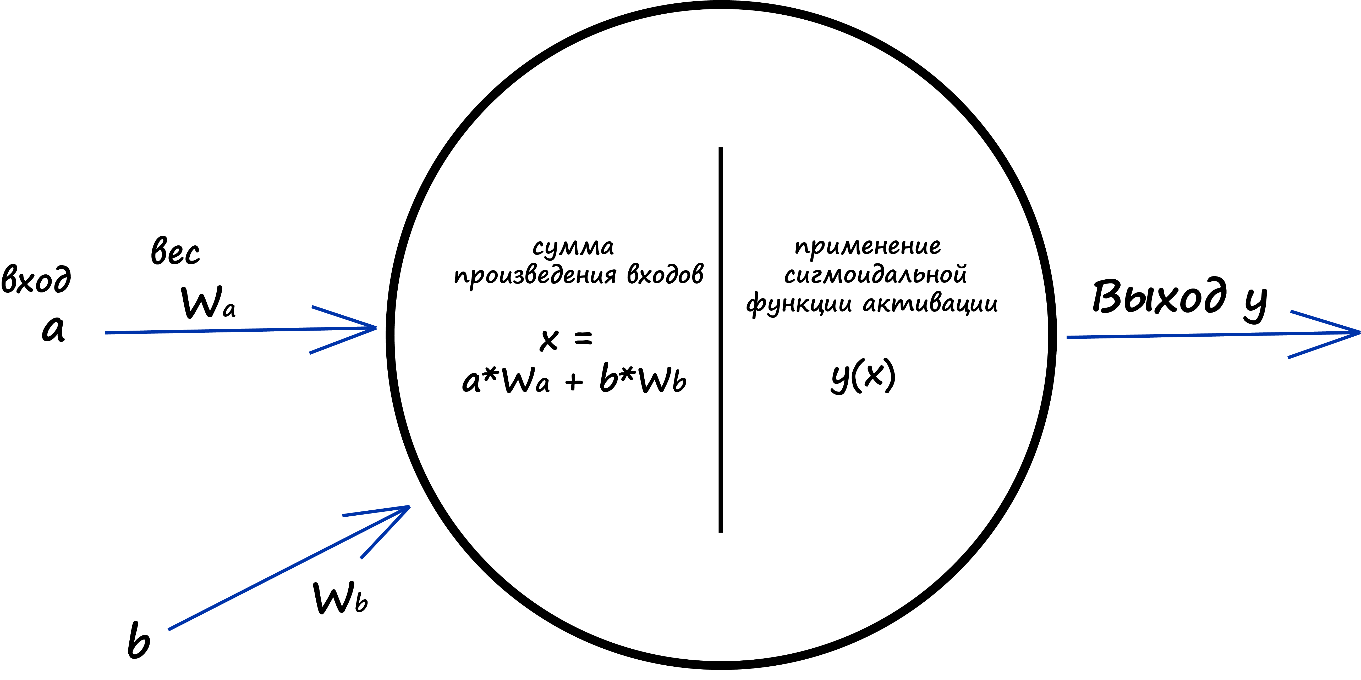
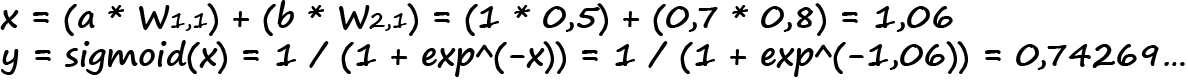
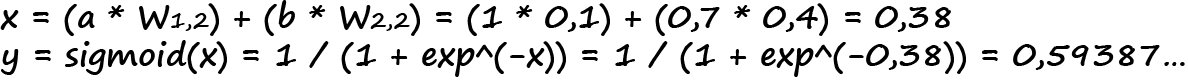
Для того, чтобы получить выходные значения необходимо сложить произведение всех поступающих сигналов и к результату применить функцию активации. Данный процесс подробно изображён на рисунке 6

Рисунок 6 — Описание вычислений “внутри” нейрона

Таким образом получается, что для нахождения выхода 1 и 2 нейрона 2 слоя необходимо выполнить следующие действия (ф-ла 2 и 3):

Формула 2 — Расчёт выходного значения 1 нейрона выходного слоя

Формула 3 — Расчёт выходного значения 2 нейрона выходного слоя

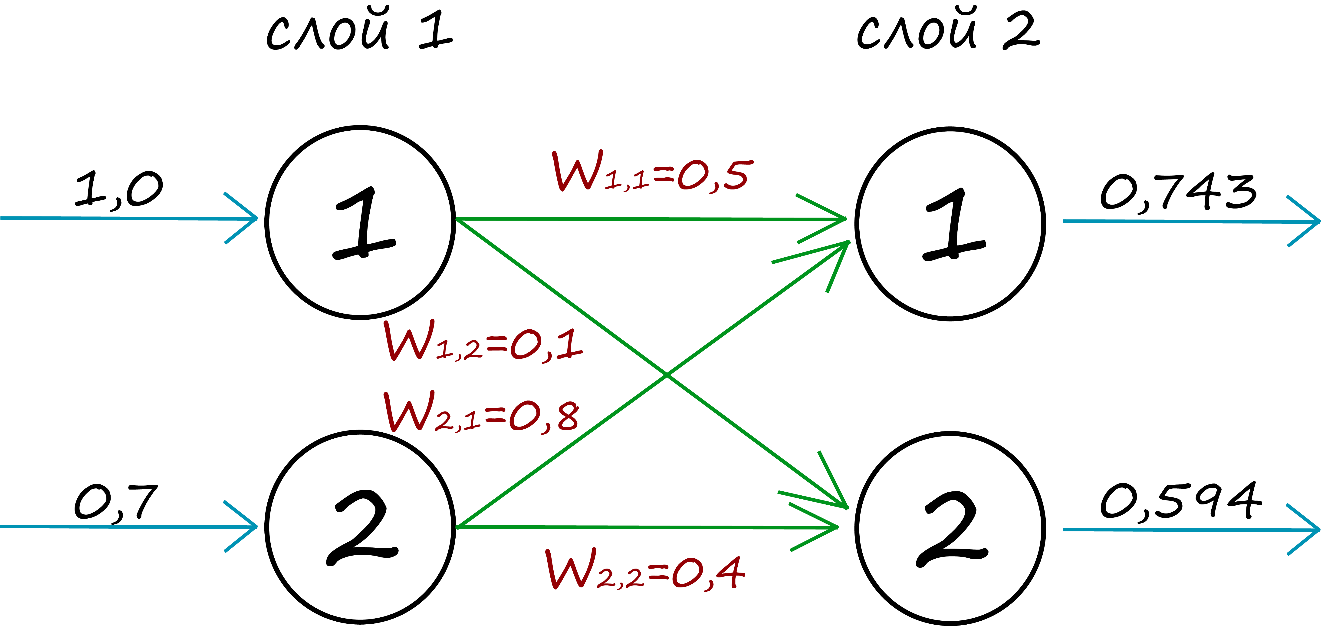
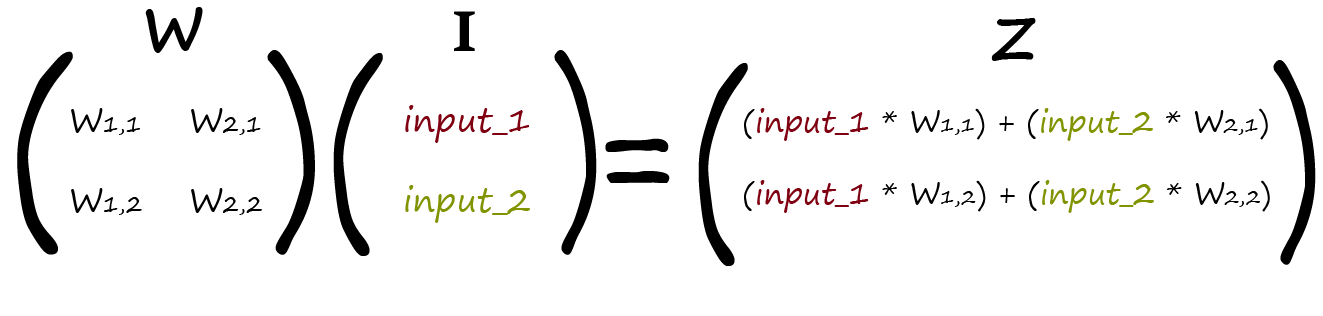
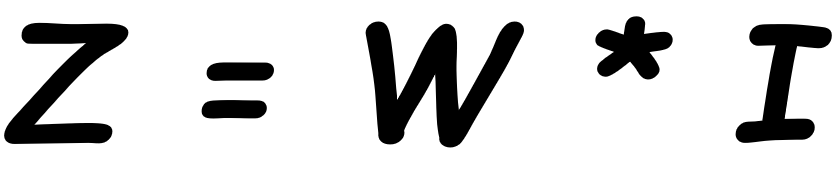
Рассчитанные выходные значения представлены на диаграмме ниже (рис. 7)

Рисунок 7 — Пример результата перцептрона

Однако вычисления выше можно упростить при помощи матриц. Вот так будет выглядеть выражение из примера в матрице (ф-ла 4)

Формула 4 — Умножение матрицы весов на матрицу входных данных

Из примера следует, что при умножении матриц выполняются те же действия, что и при самостоятельном вычислении (**НО!** Матрицы должны заполняться по правилу: кол-во строк соответствует кол-ву нейронов в текущем слое, а кол-во столбцов кол-ву нейронов следующего слоя). Таким образом, для получения выходных значений слоя следует выполнить следующую операцию (ф-ла 5):

Формула 5 — Получение результирующих значений слоя

Где **W** — матрица весов, **I —** матрица входных значений слоя, а **Z** — результирующая матрица.

Остаётся только применить функцию активации к каждому значению матрицы (ф-ла 6):

Формула 6 — Применение логистической функции к каждому значению матрицы

В формуле 6 **O** — конечный результат, **Z** — выходные данные прямого распространения.

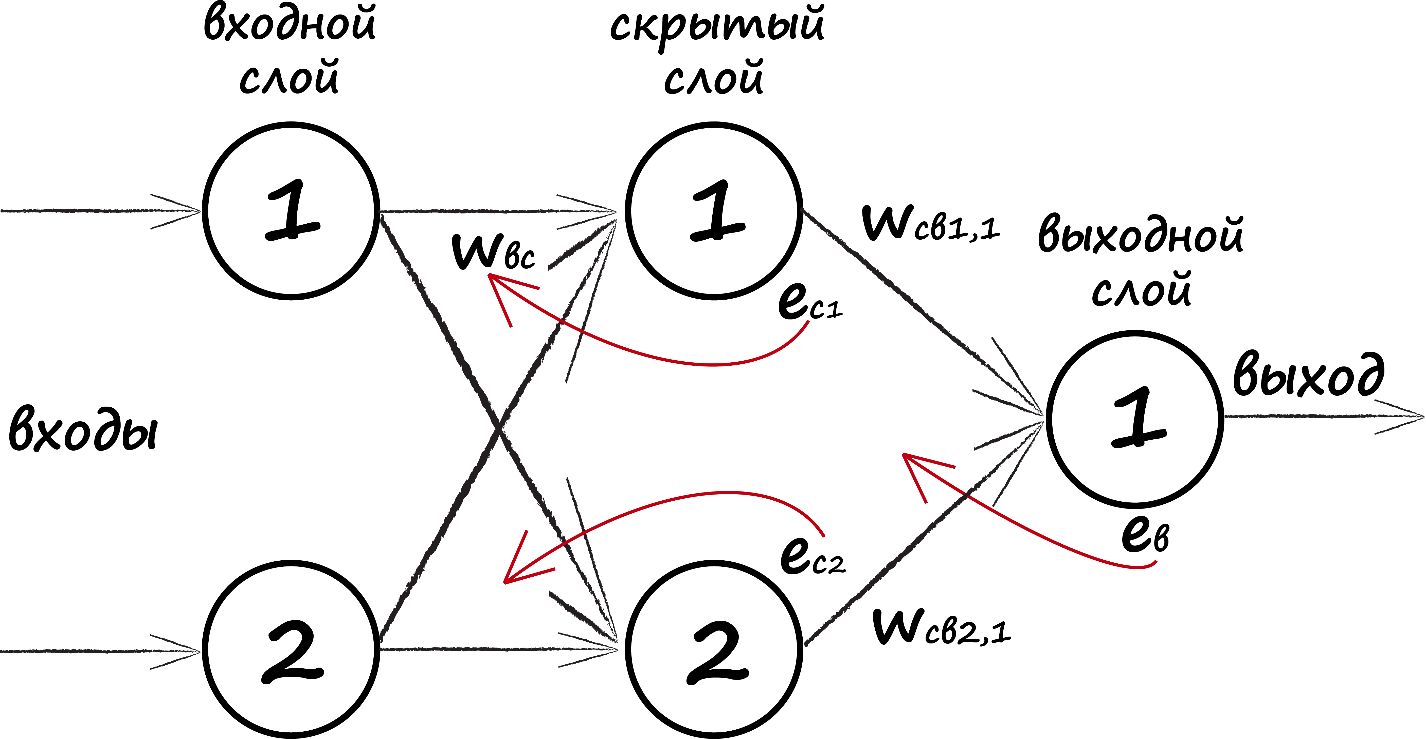
Благодаря матрицам и языку программированию, способному выполнять скалярные произведения, кол-во нейронов в слоях может быть как 5, 10, так и 10000 без представления отдельных инструкций для умножения матриц.

# 3.5. Обратное распространение ошибок

Раннее уже описывалось, что нейронная сеть имеет возможность обучаться благодаря корректировке весовых коэффициентов. И для того, чтобы узнать значения, на которые нужно изменить вес нейронов необходимо определить то, насколько сеть ошиблась во время прямого распространения.

Ошибка для каждого узла представляет собой сумму ошибок, распределенных по всем связям, исходящим из этого узла в прямом направлении.

Чтобы легче понять принцип высчитывания ошибок ниже представлен пример нейросети с 3 слоями: входным, скрытым, выходным (рис. 8), на котором **e**в — ошибка выходного слоя, **W**св1,1 и **W**св2,1 — весовые коэффициенты между скрытым и выходным слоями, **e**с1 и **e**с2 — ошибки скрытого слоя, **W**вс — матрица весовых коэффициентов между входным и скрытым слоями.

Рисунок 8 — 3-х слойная нейросеть с одним выходом

Выходная ошибка равна разнице полученного значения при прямом распространении и целевого значения — **e**в = **Out - Target** (**Out** — результат прямого распространения, **Target** — целевое значение).

Далее необходимо распределить ошибку среди всех весовых коэффициентов, связывающих с предыдущем слоем, пропорционально их вкладу в результат. Таким образом находится ошибка предыдущего слоя. Соответственно в примере **e**с1 = **e**в \* **W**св1,1, а **e**с2 = **e**в \* **W**св2,1.

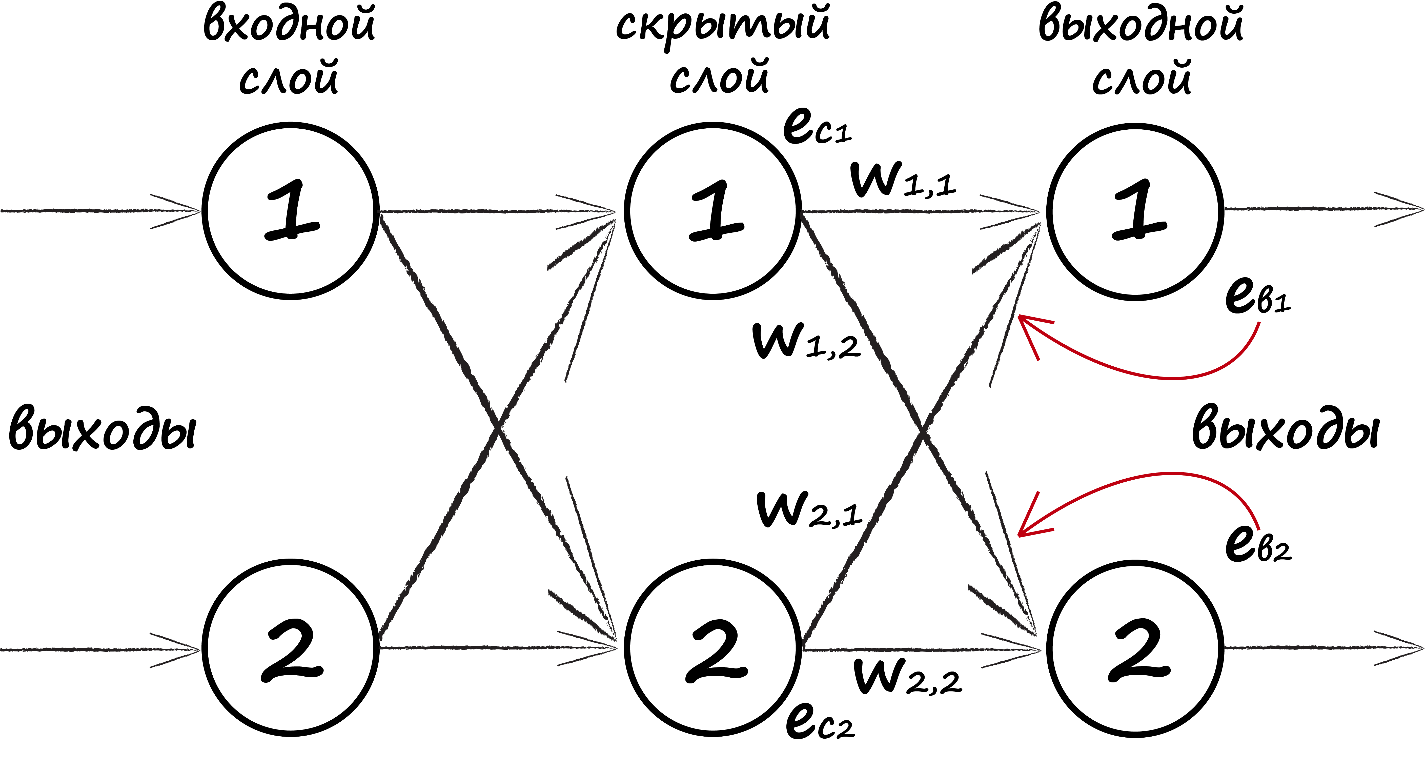
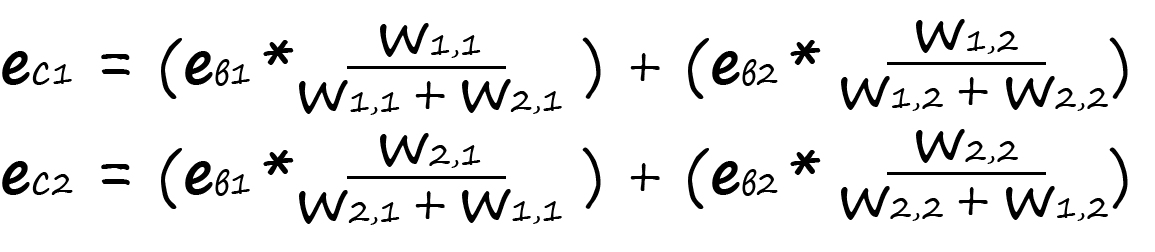
Однако не всегда ошибка будет одним числом, например на рисунке ниже выходной слой состоит из двух значений (рис. 9).

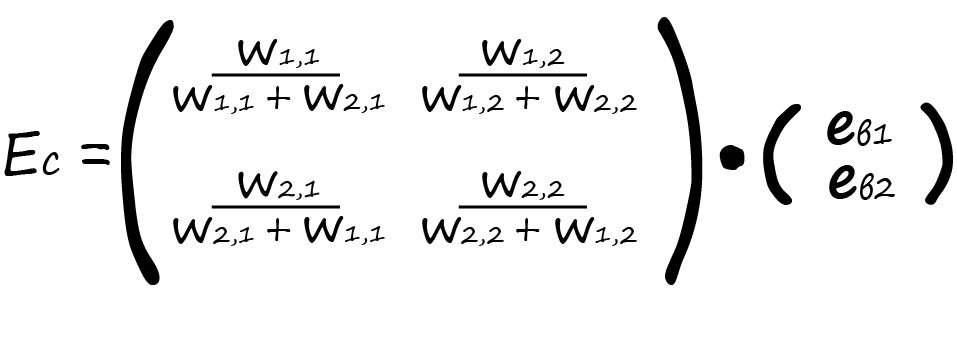
Рисунок 9 — 3-х слойная нейросеть с двумя выходами

В этом примере **e**в1 и **e**в2 также находятся путём вычитания из результата прямого распространения целевых значений. А вот ошибки **e**с1 и **e**с2 будут находиться с учётом дополнительных веса и ошибки, т.к. оба выходных значений будут влиять на распределение ошибок. Таким образом получим следующие уравнения (ф-ла 7):

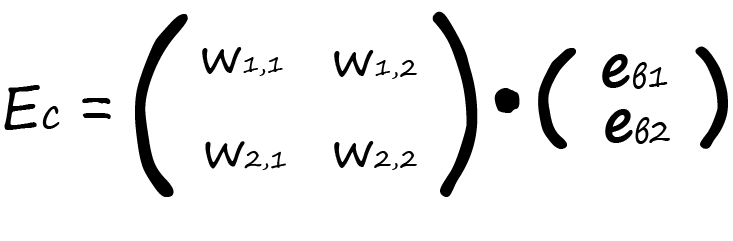
Формула 7 — Формулы расчёта ошибок

Но такие расчёты ошибок для каждого нейрона в реальной задаче необходимо оптимизировать, т.к. в ином случае все инструкции придётся описывать вручную, что трудоёмко и неэффективно. В данном случае решить проблему помогут матрицы.

Для начала следует представить выходные ошибки как матрицу с двумя строками: **e**в1 и **e**в2 (ошибки первого и второго узлов выходного слоя).

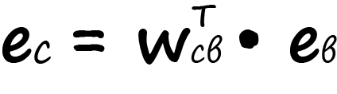
Далее построим матрицу для ошибок скрытого слоя и отобразим полученное выражение в формуле 8 (значок точки в данном случае — скалярное произведение матриц, т.е. точечное перемножение всех элементов матриц):

Формула 8 — Векторное вычисление ошибок скрытого слоя

Однако данная матрица не облегчает задачу, ведь в идеале нужно получить перемножение двух известный матриц. И такая возможность существует. Для того, чтобы облегчить данное выражение следует понимать, что данное действие нужно для того, чтобы распределить ошибку пропорционально вкладу коэффициента. То есть чем больше значение коэффициента, тем большее влияние оказывает он на результат в прямом распространении и тем б*о*льшая доля ошибки будет передана в скрытый слой. В формуле дробь играет нормирующую роль, т.е. если её убрать, то будет потеряно только масштабирование ошибок, что не так критично по сравнению с получаемой выгодой. В итоге получится уравнение (ф-ла 9):

Формула 9 — Упрощённое векторное вычисление ошибок скрытого слоя

Матрица, полученная при расчёте прямого распространения очень похожа на получившуюся выше, только она повёрнута вокруг диагонали. Такая матрица называется транспонированной и для обозначения действия транспонирования принято ставить надстрочную букву **T** (**WT**).

Подводя итог, получим следующее уравнение, описывающее обратное распространение ошибок (ф-ла 10):

Формула 10 — Конечная формула вычисления ошибок скрытого слоя

Полученная модель обратного распространения является упрощением от выведенной изначально формулы, однако пренебрежение масштабированием практически не скажется на модели во время её обучения, ведь даже если в одной из итераций ошибка будет принижена или увеличена, последующие итерации исправят данную погрешность, т.к. распределение до сих пор зависит от весового коэффициента.

# 3.6. Обновление коэффициентов весов

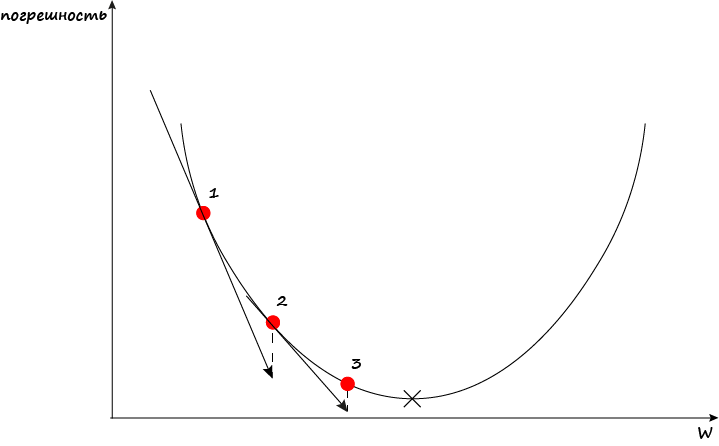
После вычисления ошибки каждого узла, необходимо как-то применить полученные значения для улучшения будущих предсказаний сети. Основной целью является нахождение таких весов, при которых погрешность предсказаний минимальна. Для лучшего понимания рассмотрим график ниже (рис. 10).

Рисунок 10 — Визуализация градиентного спуска

На графике изображена погрешность, зависящая от скалярного произведения весов (W). Существует такое значение W, при котором погрешность минимальна (обозначено X), однако оно не известно. Начало подсчёта ведётся с рандомно сгенерированным значением W, которое даёт погрешность, обозначенную на графике красной точкой 1. Далее необходимо изменить значение W таким образом, чтобы максимально приблизиться к минимальному значению погрешности. Для решения этой задачи было принято решение использовать градиентный спуск.

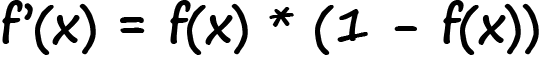
Сначала находится градиент погрешности по отношению к W, который является уровнем наклона кривой, проведённой в соответствующей точке (на рис. 10 чёрные стрелки). Градиент сообщает о том, в каком направлении (если отрицательное значение, то уменьшается, если положительное, то увеличивается), и на какую величину необходимо изменить W.

Также в формуле обновления весов есть переменная alpha. Она отвечает за сглаживание величины изменения весов во избежание проскоков через минимальную погрешность. Чаще всего ей присваивают значения ниже 0, например: 0.1, 0.01, 0.001 и т.д.

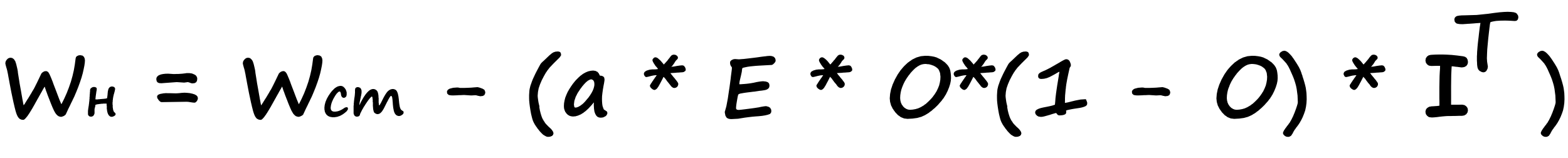
Таким образом метод градиентного спуска для достижения минимальной погрешности использует градиент, который обновляет веса в каждой итерации при помощи следующей формулы (ф-ла 11):

Формула 11 — Общая формула обновления весов

Где **W**н — новые веса, **W**ст — старые веса, **alpha** — шаг обучения (также называют коэффициентом обучения), а **gradient** — полученный градиент. Градиент находится путём умножения полученных ошибок слоя на входные данные слоя. Однако в проекте используется функция активации сигмоида, которая изменяет выходные значения, поэтому над ними необходимо произвести действие, обратное этой функции, т.е. вычислить её производную.

Для сигмоиды производная рассчитывается по формуле 12 (полное выведение формулы [2]):

Формула 12 — Производная логистической функции

В результате, получим для расчёта обновления весов формулу 13 (полный вывод формулы и подробное описание метода градиентного спуска [3]):

Формула 13 — Конечная формула обновления весов

Где **W**н — матрица новых весов, **W**ст — матрица старых весов, **a** —

коэффициент обучения, **E** — матрица ошибок слоя, **O** — матрица выходных данных слоя, к которым применена сигмоидальная функция активации, и **I** — транспонированная матрица входных данных слоя.

4. Создание нейронной сети на Python

В этой главе будет применена теория и выведенные формулы из прошлой для создания нейронной сети, способной обнаружить на рентгеновском снимке признаки инфекционных болезней. В качестве инструмента для создания будет использоваться язык программирования Python.

# 4.1. Подготовка тренировочных данных

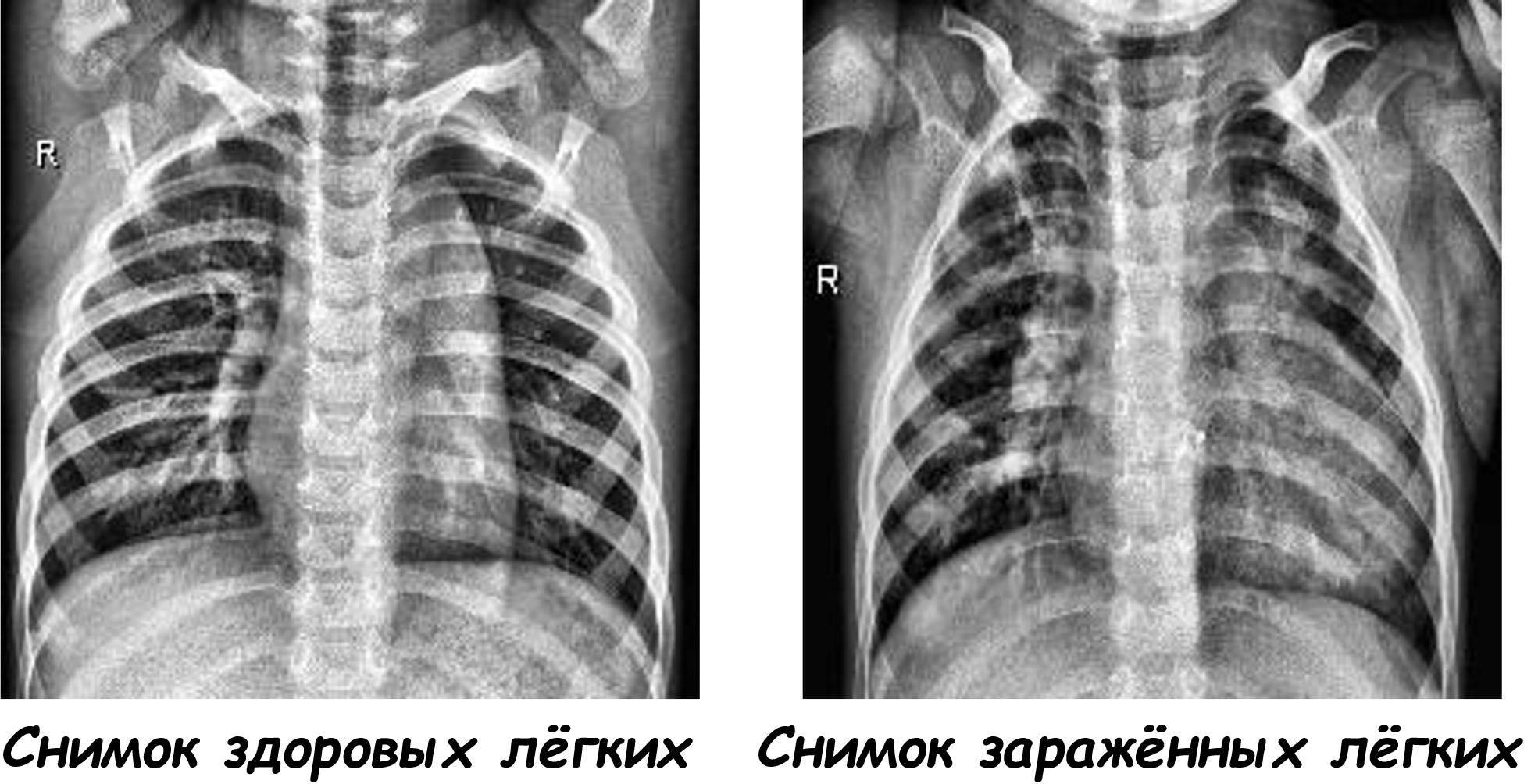
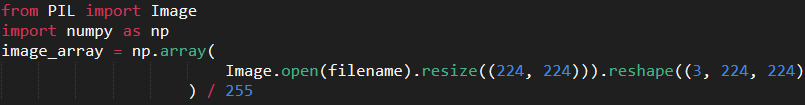
Чтобы нейронная сеть смогла отличить поражённые лёгкие от здоровых нужно предоставить ей для обучения оба вида снимков. Для этого необходимо найти готовый датасет, в котором снимки уже собраны и разделены по категориям. В данном проекте будет использована база данных: Chest X-Ray Images (Pneumonia) [4]. Ниже приведены примеры снимков здоровых и заражённых лёгких (рис 11).

Рисунок 11 — пример снимков здоровых и заражённых лёгких

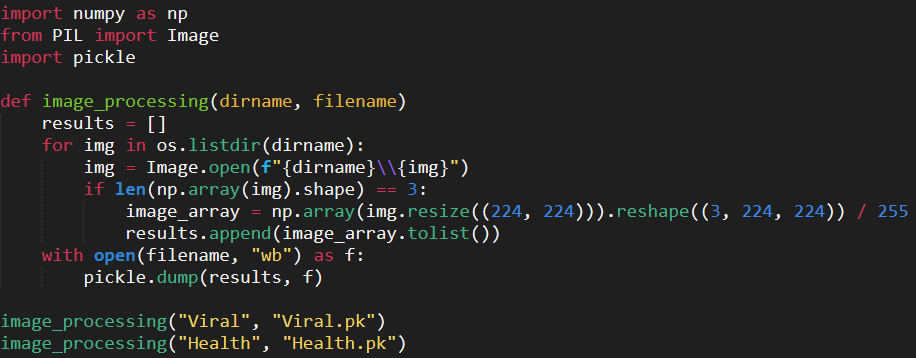
Однако сеть не имеет глаз и не может увидеть изображение, поэтому следует предоставить фото в понятном ей виде — матрице, со значениями от 0 до 1 (для лучшей совместимости с логистической функцией активации). Очевидным способом представить фотографию в виде матрицы — это сохранить её пиксели построчно. Также следует каждый пиксель привести к диапазону от 0 до 1 поделив его на 255 (максимальная яркость каждого цвета 256, но отсчёт идёт от 0, поэтому 255).

Также все снимки должны иметь одинаковое разрешение, т.к. используя параметрическую модель нужно позаботиться о том, чтобы количество входных данных всегда было одинаковое. В качестве константы для разрешения снимков стало число 224, то есть все картинки будут сохранены в 3-х цветах и разрешением 224x224 пикселя в трёхмерной матрице.

На языке Python данные действия выполняет следующий код (сниппет 1).

Сниппет 1 — Перевод изображения в матрицу 3x224x224

К каждому снимку будет сопоставлено значение 0, если изображены здоровые лёгкие и 1, если больные соответственно.

Для более эффективного использования данных было принято решение обработать все снимки одного типа, сохранить в один массив и сохранить его в файл при помощи библиотеки pickle. Таким образом получается два файла с обработанными снимками: здоровыми и заражёнными, готовыми к применению в обучении модели. Ниже код, выполняющий эти действия (сниппет 2).

Сниппет 2 — Код обработки всех изображений в папке и сохранения их в файл в виде массивов

Помимо обучающих файлов, в каждом из которых по 800 снимков, также были созданы два файла (в каждом по 500 снимков) для последующей проверки сети.

Далее потребуется получить доступ к данным, а также смешать два набора снимков случайным образом так. Для этого был написан загрузчик, код которого доступен по ссылке [6].

# 4.2. Архитектура создаваемой модели

Исходя из поставленной задачи и знаний об устройстве нейронных сетей была составлена следующая архитектура создаваемой нейронной сети (рис 12):

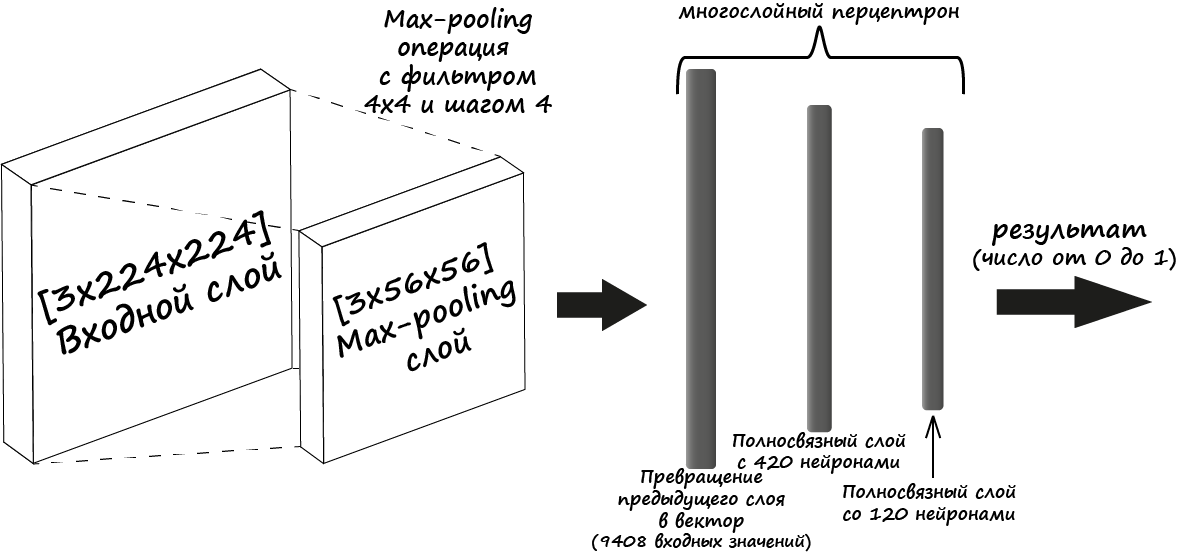
1. Один слой максимального объединения (max pooling layer) для сжатия входных данных и выделения важных характеристик исходного изображения.
2. Многослойный перцептрон (Multi-Layer Perceptron, MLP), созданный для выявления закономерностей и бинарной классификации полученных характеристик изображения.

Рисунок 12 — Архитектура создаваемой нейросети

# 4.3. Программирование нейронной сети

Чтобы код не был привязан к конкретным числам, он будет написан в виде класса, при инициализации которого можно будет указать основные настройки для нейросети: альфа-коэффициент, размер фильтра для max-pooling, количество нейронов для скрытых и выходного слоёв.

## 4.3.1. Скелет кода

Для начала кратко опишем основные методы, которые должны быть в классе: инициализация; создание рандомных весов; функция max-pooling; сигмоидальная функция; производная сигмоидальной функции; функция для опроса нейронной сети (воспроизведение прямого распространения по сети); функция для тренировки сети. Таким образом создадим скелет класса, прописав все необходимые для работы методы и получим код, изображённый на 3 сниппете.

Сниппет 3 — Скелет класса нейронной сети

## 4.3.2. Биасы

При инициализации класса, помимо создания матриц, связывающих слои между собой, создаются 3 матрицы: bl\_1, bl\_2 и bl\_3. Эти матрицы являются так называемыми биасами. Они созданы для того, чтобы улучшить корректировку весов. Делают они это при помощи смещения выходных данных на графике погрешности, описанного в 3 главе, т.е. выступают своего рода свободным коэффициентом в формуле прямого распространения. Соответствующую матрицу биасов нужно просто добавить к полученным значениям прямого распространения перед применением функции активации.

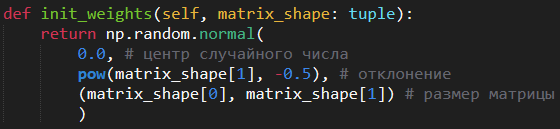
## 4.3.3. Инициализация весовых коэффициентов

Следующим важным шагом создания нейронной сети является случайная генерация исходных весовых коэффициентов. Эти веса используются для расчёта выходных значений всей нейросети, а также по мере обучения корректируются, для достижения минимальной погрешности сети.

Размер матрицы будет задаваться по правилу: кол-во нейронов в следующем слое, а потом кол-во нейронов в текущем слое.

Генерируемые числа должны располагаться в диапазоне от 0 до 1 для предотвращения перенасыщения сети большими числами и лучшей их совместимости с логистической функцией активации.

В итоге, в качестве функции генератора случайных чисел была выбрана функция, генерирующая весовые коэффициенты, выбранные из нормального распределения с центром в нуле и со стандартным отклонением, обратно пропорциональным квадратному корню из количества входных связей на нейрон.

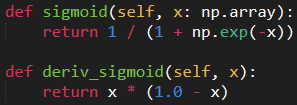
Создание функции, использующей такие правила для генерации случайных чисел, не составит труда благодаря библиотеки numpy. Код функции изображён на сниппете 4.

Сниппет 4 — Скелет класса нейронной сети

## 4.3.4. Функция активации и её производная

Перед тем, как писать код для обучения нейронной сети необходимо написать код функции, возвращающий матрицу, к каждому значению которой была применена выбранная функция активации: сигмоидальная. В реализации этого кода опять же поможет библиотека numpy, к классу матриц которой, при применении функции, она будет применятся ко всем её значениям.

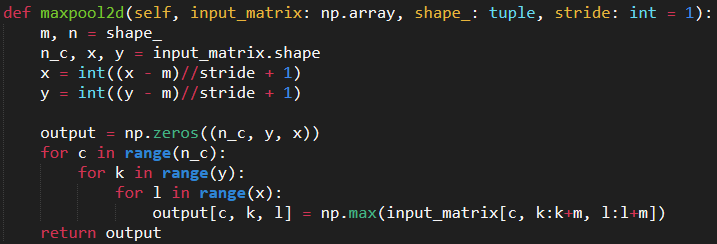
Для написания кода функций будут использованы: формула, изображённая на рисунке 2 в **3.2** (сигмоидальная функция) и формула 12 из **3.6** (производная сигмоидальной функции).

Таким образом получим следующий код обеих функций, изображённый на сниппете 5.

Сниппет 5 — Функция для расчёта сигмоиды и её производной

## 4.3.5. Метод максимального объединения

В разрабатываемой нейронной сети было принято решение использовать метод, позволяющий уменьшить количество входных данных, не потеряв при этом важных характеристик. Для этого будет использоваться метод максимального объединения (max-pooling).

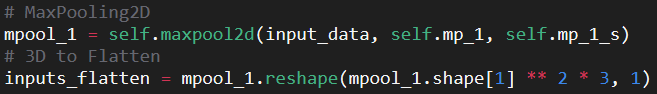
Исходя из принципа его работы, описанного в **3.3**, был написан следующий код (сниппет 6), принимающий в качестве аргументов данные, размер фильтра и шаг сдвига фильтра:

Сниппет 6 — Функция применения метода max-pooling к данным

## 4.3.6. Прямое распространение по сети

Перед тем, как создать функцию для тренировки сети, создадим функцию, выполняющую прямое распространение, т.е. получающую предсказание сети. Она будет нужна и для прямых опросов сети после обучения, и в самом процессе обучения, для получения выходных данных каждого слоя в прямом распространении.

Функция будет называться query и будет принимать только входные значения (т.е. изображение, переведённое в матрицу).

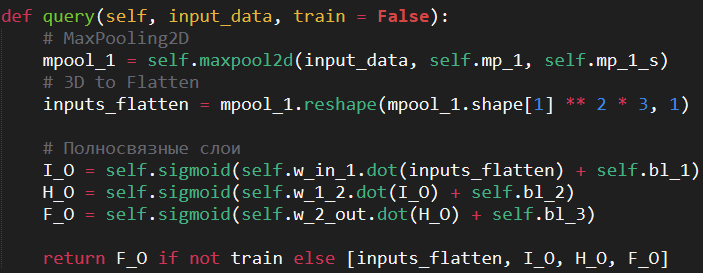
Перед тем, как перейти к полносвязным слоям к данным будет применён метод max-pooling при помощи функции maxpooling2d, с последующим переводом полученных данных в одномерную матрицу для передачи их в полносвязный слой. Код для выполнения данных действий представлен на сниппете 7, где **mp\_1** — размер фильтра, **mp\_1\_s** — шаг сдвига фильтра, **input\_data** — входные данные.

Сниппет 7 — Применение метода max-pooling и превращение 3-х мерной матрицы в одномерную

Для получения выходного значения для полносвязного слоя переведём формулы 5 и 6, выведенные в **3.4**, в код и получим (сниппет 8):

Сниппет 8 — Прямое распространение для каждого слоя

В данном коде буква n на конце каждой переменной обозначает номер слоя и соответствующих ему значений: весов, входных данных и биаса.

Применим данный шаблон к 3 весовым матрицам. В результате, код функции прямого распространения для выбранной архитектуры нейронной сети будет следующим (сниппет 9):

Сниппет 9 — Код функции прямого распространения

Возвращать функция будет разные данные, в зависимости от того, откуда к ней обратиться. Если обращение будет идти при обучении, то она вернёт все промежуточные данные прямого распространения и результат для их последующего применения, если же обращение поступит извне, то функция вернёт только конченый результат предсказания.

## 4.3.7. Высчитывание ошибок полносвязных слоёв и биасов

Прежде чем перейти к обновлению весов сети и непосредственному влиянию на её дальнейшие предсказания необходимо сначала рассчитать обратное распространение ошибок. В первую очередь высчитывается ошибка выходного слоя вычитанием из результата прямого распространения целевого значения. Код для данной операции представлен ниже на сниппете 10, на котором **out\_layer** — результат прямого распространения, **target** — целевое значение.

Сниппет 10 — Ошибка выходного слоя

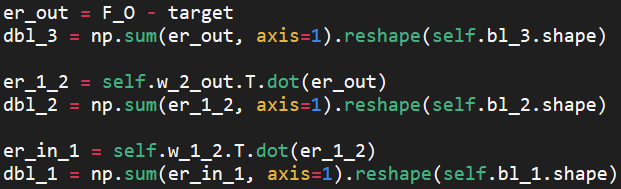
Далее необходимо найти ошибки для каждого слоя, распространив выходную на все предыдущие слои при помощи выведенной формулы 10 в **3.5**. На сниппете 11 **error\_n** —ошибка искомого слоя, **w\_n\_n+1** — веса между искомым и следующим слоем, а **error\_n+1** — ошибка следующего слоя.

Сниппет 11 — Общая формула для вычисления ошибки не выходного слоя

Помимо ошибок полносвязных слоёв, также находятся ошибки биасов. Расчёт ошибок биаса будет производиться при помощи кода, отображённого на 12 сниппете, где **n** — номер слоя, к которому принадлежит матрица биасов.

Сниппет 12 — Расчёт градиента биаса

Применив полученные шаблоны к значениям выбранной нейронной сети, получим следующий код для расчёта ошибок (сниппет 13):

Сниппет 13 — Полный код расчёта ошибок прямого распространения

## 4.3.8. Обновление весовых коэффициентов

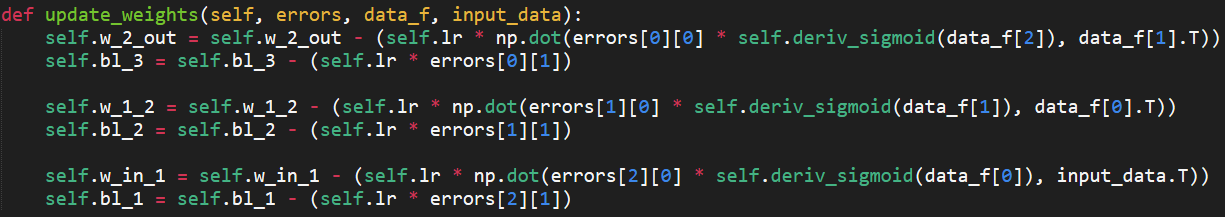
После того, как ошибки всех слоёв были рассчитаны, можно приступать к обновлению весовых коэффициентов в каждом слое. Для этого будет применена полученная формула 13 в **3.6**. При переводе формулы в код получим (сниппет 14):

Сниппет 14 — Обновление весов между слоями

На сниппете 14 **error\_n\_deriv** — ошибка слоя n, к которой применена производная сигмоидальной функции, **error\_n** — ошибка слоя n, **out\_layer\_n** — выходные значения слоя n, **w\_n\_n+1** — веса между слоем n и n+1, **lr** — коэффициент обучения (альфа), **inputs\_layer\_n.T** — транспонированная матрица входных значений слоя n.

Помимо обновления матрицы весов между слоями также необходимо обновлять веса матриц биасов. Для этого используем код, написанный в сниппете 15, в котором **bl\_n** — биас, принадлежащий слою n, а **error\_bl\_n** — ошибка биаса слоя n.

Сниппет 15 — Обновление весов биасов

Конечный код функции *update\_weights,* для обновления весовых коэффициентов, отображён на сниппете 16 ниже.

Сниппет 16 — Полный код обновления весовых коэффициентов сети

Данная функция принимает в себя три пары ошибок слоёв (пара состоит из ошибки весов и ошибки биаса), результаты прямого распространения и исходные данные.

На данном этапе написание основного алгоритма нейросети завершено, полный код класса доступен по ссылке [7].

# 4.4. Обучение нейронной сети

Для визуализации процесса обучения было принято решение о создании обновляющегося прогресс-бара, в котором будет отображаться следующая информация: сколько % выборки обработано, точность предсказаний (accuracy) тренировочной выборки (вычисляется при помощи деления количества правильных предсказаний на обработанное), также будет считаться ошибка предсказаний (loss), для чего была выбрана бинарная кросс энтропийная функция потерь, способная в полной мере отобразить то, какую по величине ошибку за всё время одной эпохи допустила сеть. Код функции, позволяющей вычислить бинарную кросс энтропию отображён на сниппете 17, где t — целевое значение, а y — полученное значение.

Сниппет 17 — Вычисление ошибки при помощи бинарной кросс-энтропийной функции потерь

Также, 20% от всех полученных снимков в эпохе будут служить проверочными данными и не будут участвовать в обучении. При помощи них в конце эпохи будет производится валидация, т.е. находиться точность предсказаний и ошибка.

В конечном итоге была выбрана библиотека tqdm для отображения прогресс-бара, с обновляющимися данными об ошибке и точности предсказаний.

Пример вывода прогресс-бара одной из эпох представлен на рисунке 15.

Рисунок 15 — пример отображения прогресс бара

Данные об общей ошибке эпохи и точности предсказаний будут сохраняться в глобальную переменную для последующей визуализации прогресса обучения по эпохам при помощи построения графика.

# 4.5. Анализ лучшей модели

После анализа нескольких обученных моделей на одном датасете, но с разными параметрами, лучшей оказалась модель со следующими: альфой равной 0.001, max-pooling состоящим из фильтра 4x4 и шагом 4, первым скрытым слоем, содержащим 420 нейронов, вторым 120 и выходным с одним значением.

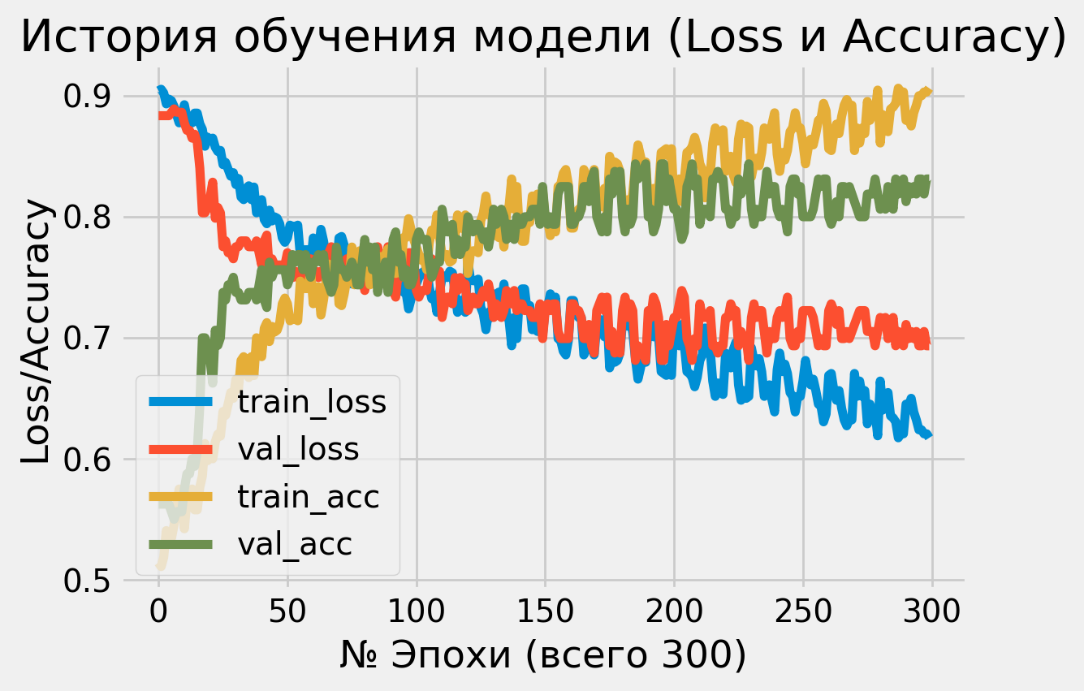
После вызова функции *fit* (исходный код функции доступен по ссылке [8]) для обучения модели, и окончания 300 эпох (подготовленный датасет из 1600 снимков был разбит на два по 800 для ускорения процесса обучения, из этих 800 20%, т.е. 160 уходило на проверку точности предсказаний эпохи), построим график из сохранённых в каждой эпохе данных при помощи функции *build\_graph* (исходный код функции доступен по ссылке [9]) (рис. 16):

Рисунок 16 — график обучения модели

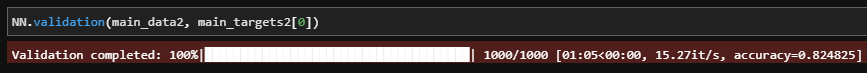
На графике отчётливо видно то, как ошибка тренировочной части датасета (train\_loss) и проверочной (val\_loss) по мере обучения падает, в то время как линии, отображающие точность предсказаний (val\_acc и train\_acc), стремяться вверх. Линия точности проверочных данных (val\_acc) под конец, как-будто колеблется на месте, но это не так. Амплитуда отклонений с каждой эпохой становится меньше, а это сигнализирует о том, что веса максимально приблизились к минимальной величине погрешности. Таким образом данная модель достигла совего максимума. Проведём валидацию полученной модели на проверочном датасете из 1000 снимков при помощи метода *validation* (код функции доступен по ссылке [10]). Результат валидации изображён на рисунке 17.

Рисунок 17 — прогресс-бар валидации модели

Точность предсказаний составила 82.5%, что является отличным результатом для простой нейронной сети, не использующей сложные технологии и свёрточные слои. Для наглядности, загрузим в модель два снимка (здоровых и поражённых лёгких), которые не принимали участие в обучении.

Для этого создадим функцию predict, принимающую в себя экземпляр модели и путь к картинке. Данная функция будет переводить изображение в rgb (если она не была сохранена в этом формате), также изменять размер снимка до 224x224 и переводить её в матрицу, которую отправит обученной модели в метод query (отвечающий за прямое распространение по сети). Ответ будет представлять из себя шанс того, что лёгкие поражены, соответственно если шанс ниже 60%, то лёгкие будут считаться здоровыми, а если больше или равно, то заражёнными. Для удобства, если шанс меньше 60%, то будет выводится не шанс того, что лёгкие заражены, а шанс того, что они здоровые, т.е. из 100 будет вычитаться полученный процент. Результат предсказаний двух случайных картинок привидён на рисунке 18.

Рисунок 18 — пример предсказаний сети

Полученные результаты предсказаний подтверждают то, что сеть хорошо научилась выявлять признаки пневмонии, т.к. предсказания по обоим снимкам весьма уверенные.

5. Заключение

В ходе выполнения проектной работы поставленная цель была достигнута. Созданная нейронная сеть способна выявлять признаки пневмонии по рентгеновскому снимку лёгких, что позволяет ей классифицировать их на здоровые и поражённые. Точность предсказаний нейросети составила 82.5%.

В рамках данного проекта были достигнуты следующие результаты:

* Изучен принцип работы нейронных сетей
* Произведён обзор существующих методов машинного обучения.
* Были разобраны и объяснены все использующиеся при создании нейронной сети технологии.
* Выбран и подготовлен подходящий набор данных для обучения сети.
* Составлена архитектура нейронной сети.
* Написан код нейронной сети.
* Произведён анализ лучшей модели.

Я считаю, что использование подобных нейронных сетей, но с большей точностью, в ближайшем будущем станет неотъемлемой частью всех областей медицины. Прямо сейчас ведётся работа над созданием совершенной нейронной сети COVIDNet, лучшая модель которой: COVIDNet-CXR-3, смогла достичь невероятной точности в 98.3% [11]. Её целью также является обнаружение признаков поражения лёгких по рентгеновским снимкам.

Ссылки на источники

https://medium.com/@balovbohdan/сверточные-нейронные-сети-с-нуля-4d5a1f0f87ec— доп. литература

Книга “Создаём нейронную сеть” Тарика Рашида — доп. литература

Книга “Грокаем глубокое обучение” Эндрю Траска — доп. литература

https://github.com/Scorpiss/school-project-tikhonov — репозиторий проекта

1 — https://towardsdatascience.com/gpt-3-a-complete-overview-190232eb25fd

2 — https://towardsdatascience.com/derivative-of-the-sigmoid-function-536880cf918e

3 — Книга “Создаём нейронную сеть” Тарика Рашида, Глава 1, “Как мы фактически обновляем весовые коэффициенты”

4 — https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia

5 — https://github.com/ieee8023/covid-chestxray-dataset

6 — https://github.com/Scorpiss/school-project-tikhonov/blob/main/data\_loader.py

7 — https://github.com/Scorpiss/school-project-tikhonov/blob/193d92560962be0740266e6dae4a835a8fab419b/NeuralNetwork.py

8 — https://github.com/Scorpiss/school-project-tikhonov/blob/193d92560962be0740266e6dae4a835a8fab419b/NeuralNetwork.py#L149

9 — https://github.com/Scorpiss/school-project-tikhonov/blob/193d92560962be0740266e6dae4a835a8fab419b/NeuralNetwork.py#L200

10 — https://github.com/Scorpiss/school-project-tikhonov/blob/193d92560962be0740266e6dae4a835a8fab419b/NeuralNetwork.py#L220

11 — https://github.com/lindawangg/COVID-Net/blob/master/docs/models.md