Муниципальное общеобразовательное учреждение

«Средняя школа с углубленным изучением отдельных предметов № 30

имени Медведева С.Р. г. Волжского Волгоградской области»

ИНДИВИДУАЛЬНЫЙ УЧЕБНЫЙ ПРОЕКТ

«Свёрточные нейронные сети»

Выполнил Смольяков Всеволод Романович,

учащийся 10 А класса

Руководитель:

Крашенинникова Татьяна Анатольевна

г. Волжский, 2023

Оглавление

1. Введение
   1. Актуальность
   2. Цели
   3. Задачи
2. Теория
   1. Обозначения
   2. Прямой проход
      1. Полносвязная нейронная сеть или нейросеть прямого распространения
      2. Функции активации
      3. Смещение нейрона
      4. Свёрточные нейронные сети
      5. Свёрточный слой
      6. Слой подвыборки
   3. Обратный проход
      1. Функция потерь
      2. Алгоритм обратного распространения ошибки через полносвязные слои
      3. Алгоритм обратного распространения ошибки через слой подвыборки
      4. Алгоритм распространения ошибки через свёрточный слой
      5. Гиперпараметры
      6. Adam gradient – улучшенный метод градиентного спуска
      7. Проблемы при обучении
3. Практика
   1. Идея и реализация
   2. Библиотеки
   3. Структура нейросети
4. Заключение
5. Источники

1. Введение

1.1. Актуальность

В современном мире мы всё чаще встречаемся с нейронными сетями: в камере телефона она улучшает фотографию, искусственно повышает разрешение, предлагает нам видео, музыку и новости, основываясь на наших интересах и вкусах, управляет машинами и приборами, облегчая жизнь людям. Сейчас эта тема особенно интересна, так как недавно вышел ChatGPT4 и MidJourney, которые просто поражают своими возможностями, поэтому я решил написать свою, конечно, более простую нейросеть. Я считаю, что в будущем нейросети смогут почти полностью заменить людей на рабочих местах, благодаря этому у людей появится возможность не беспокоиться о деньгах и просто заниматься своим любимым делом.

1.2. Цели

В своём проекте я преследовал две главные цели. Во-первых, написать полностью рабочую нейронную сеть, обучить и протестировать её в действии. Во-вторых, не использовать готовые решения или вспомогательные модули. Я хотел написать всё с нуля, чтобы понять, как оно работает изнутри и чтобы в случае чего я мог бы легко исправить ошибку или улучшить часть кода.

1.3. Задачи

Чтобы выполнить цели, мне нужно было выбрать язык программирования, спланировать структуру проекта, написать нейросеть, создать обучающие данные, сделать функции сохранения и загрузки нейросети из файла, обучить её, проверить результаты, написать приложение для демонстрации и провести окончательную проверку продукта.

2. Теория

2.1. Обозначения

К этим обозначениям необходимо возвращаться по мере прочтения, так как сейчас они могут быть непонятны.

Снс – это свёрточная нейронная сеть

f – функция активации

L – функция потерь

n – скорость обучения

w – изменения весов

GRAD – градиент или ошибка, которая считается во время обратного распространения

x – входные данные, дополнительное название в снс изображение

w – веса, дополнительное название в снс ядро свёртки, фильтр

b – смещение или сдвиг

output или o – взвешенная сумма, дополнительное название в снс карта признаков без применённой функции активации

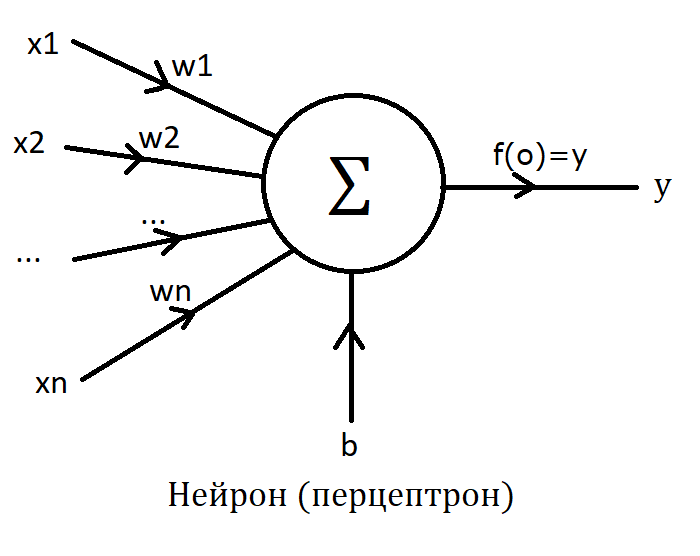
y, y = f(o) – выход нейрона, дополнительное название в снс карта принаков, выходные данные

ideal – требуемый результат, то есть выходные данные, которые должна выдавать нейросеть

2.2. Прямой проход

2.2.1. Полносвязная нейронная сеть или нейросеть прямого распространения

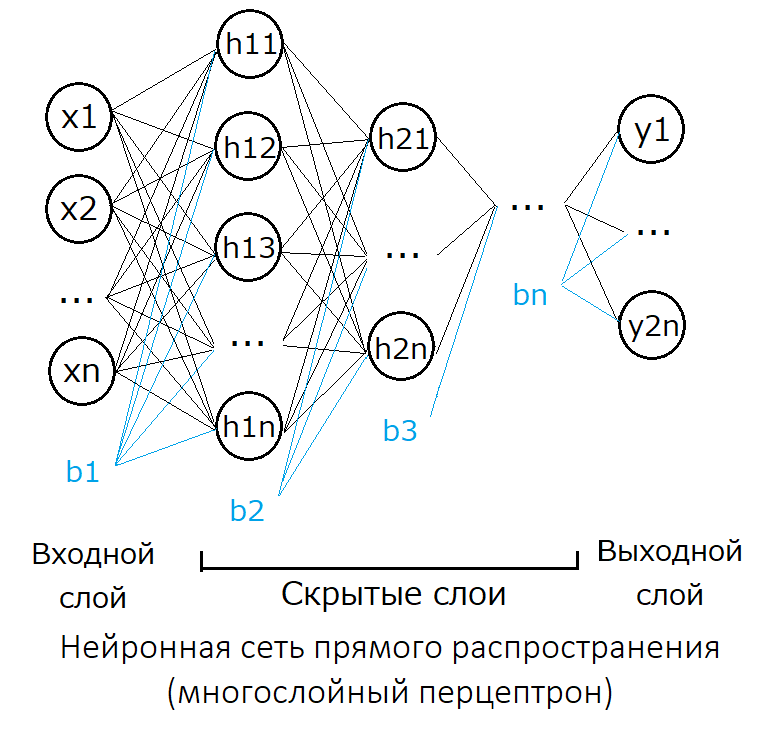
Нейронные сети сделаны наподобие человеческого мозга, где единичные элементы – нейроны объединяются в большую сложную конструкцию. На вход нейрона поступают какие-то данные, которые обрабатываются и передаются дальше. Самой простой нейросетью является перцептрон – одиночный нейрон. Но из-за простоты он не в силах делать даже простую классификацию.



Перцептрон работает следующим образом:

Входные данные перемножаются с весами и суммируются, также прибавляется смещение. После к полученной сумме (её ещё называют взвешенная сумма, так как происходит домножение на веса) применяется функция активации, это и есть выход нейрона. С тем, что такое функция активации и смещение разберёмся чуть позже.

Обычно несколько нейронов соединяют в одну полносвязную нейронную сеть (сеть прямого распространения) или многослойный перцептрон. Такие нейронные сети уже могут выполнять достаточно сложные задачи классификации и регрессии (регрессия – нахождение предположительного ответа по входным данным, например цену на квартиру по площади, району, этажу и т.д.).



Первый (входной) слой принимает данные в целочисленном или дробном формате. Следующие несколько скрытых слоёв проводят обработку данных. Последний (выходной) слой является результатами нейронной сети. Работает она также как и перцептрон, только выход нейронов первого слоя идёт на вход нейронов второго слоя, выход второго на вход третьего и так далее. Также важно отметить, как ведут себя смещения. Смещение первого слоя (b1) соединено со всеми нейронами второго слоя (h11, h12, h13, …, h1n), кроме смещения второго слоя (b2). В полносвязной нейронной сети смещения не принимают никаких значений, так как они не являются нейронами.

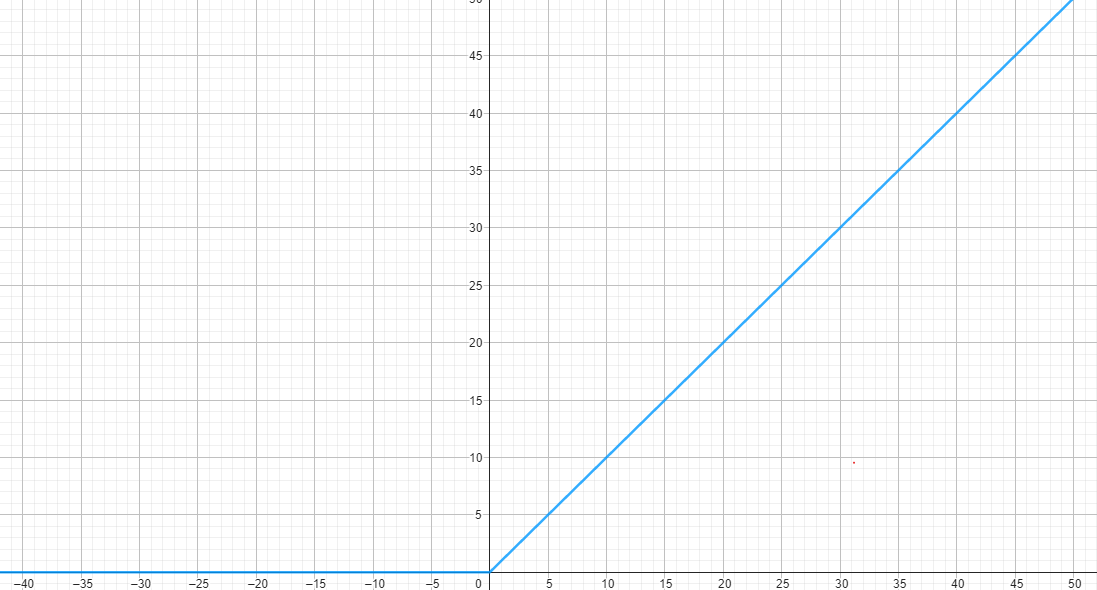
Для большей ясности приведу пример формул для нейрона скрытого слоя:

2.2.2. Функция активации

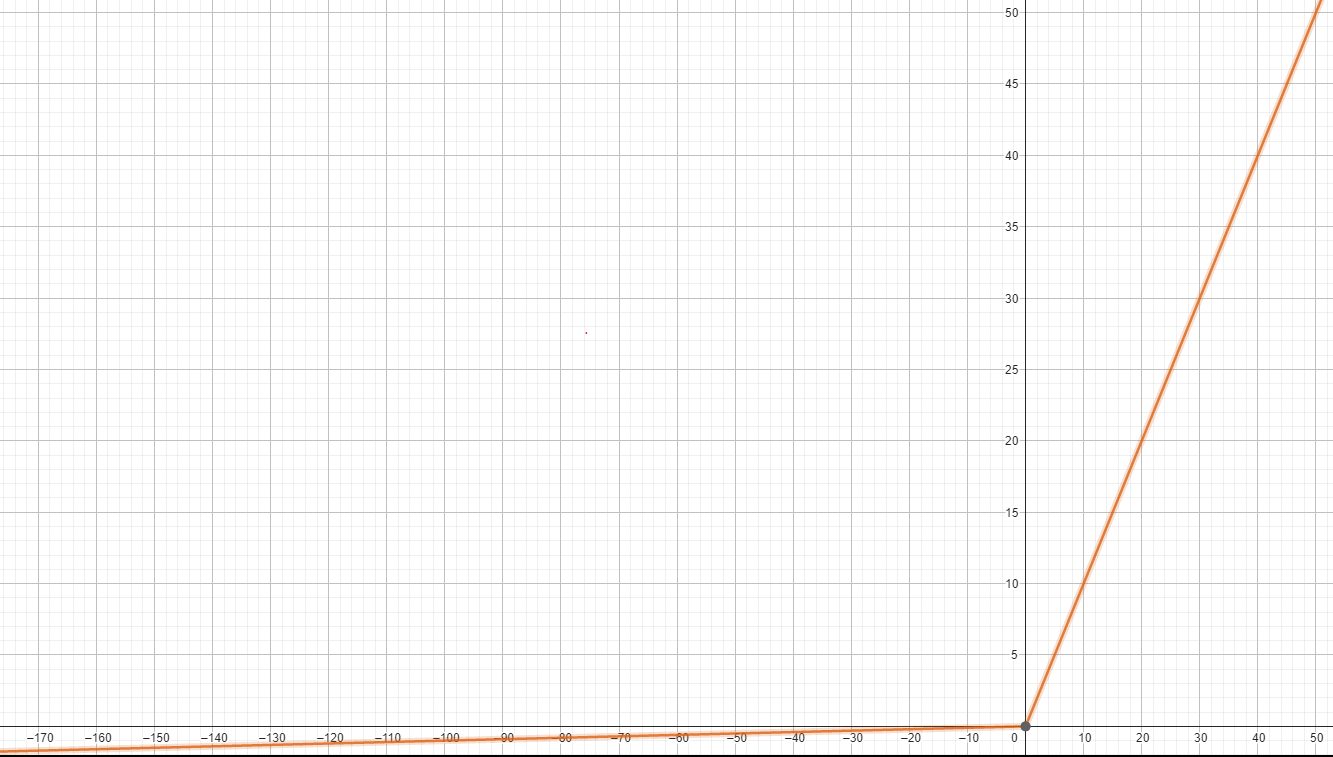
Функция активация – это специальная функция, которая показывает на сколько активирован нейрон. Может появится вопрос, а зачем она нужна, нельзя ли просто использовать взвешенную сумму. Ответ - нет. Во-первых, во время обучения нейронной сети, мы будем использовать алгоритм обратного распространения ошибки, который требует от нас, чтобы функция была не линейна и дифференцируема в любой точке. Во-вторых, если просто передавать взвешенную сумму дальше, то у нас может случиться переполнение и в ответе мы получим 10^300 или и того хуже -9713198014, если мы пишем на языках, где переполнение типа приводит к отрицательным числам (C/C++). В-третьих, эта функция схожа с работой настоящих нейронов, которые передают сигнал дальше только по достижению некоторого порогового значения.

Теперь нужно разобраться с тем какие и когда необходимо использовать. Всего существует огромное количество различных функций активации, при необходимости можно даже сделать свою. Из самых популярных это:

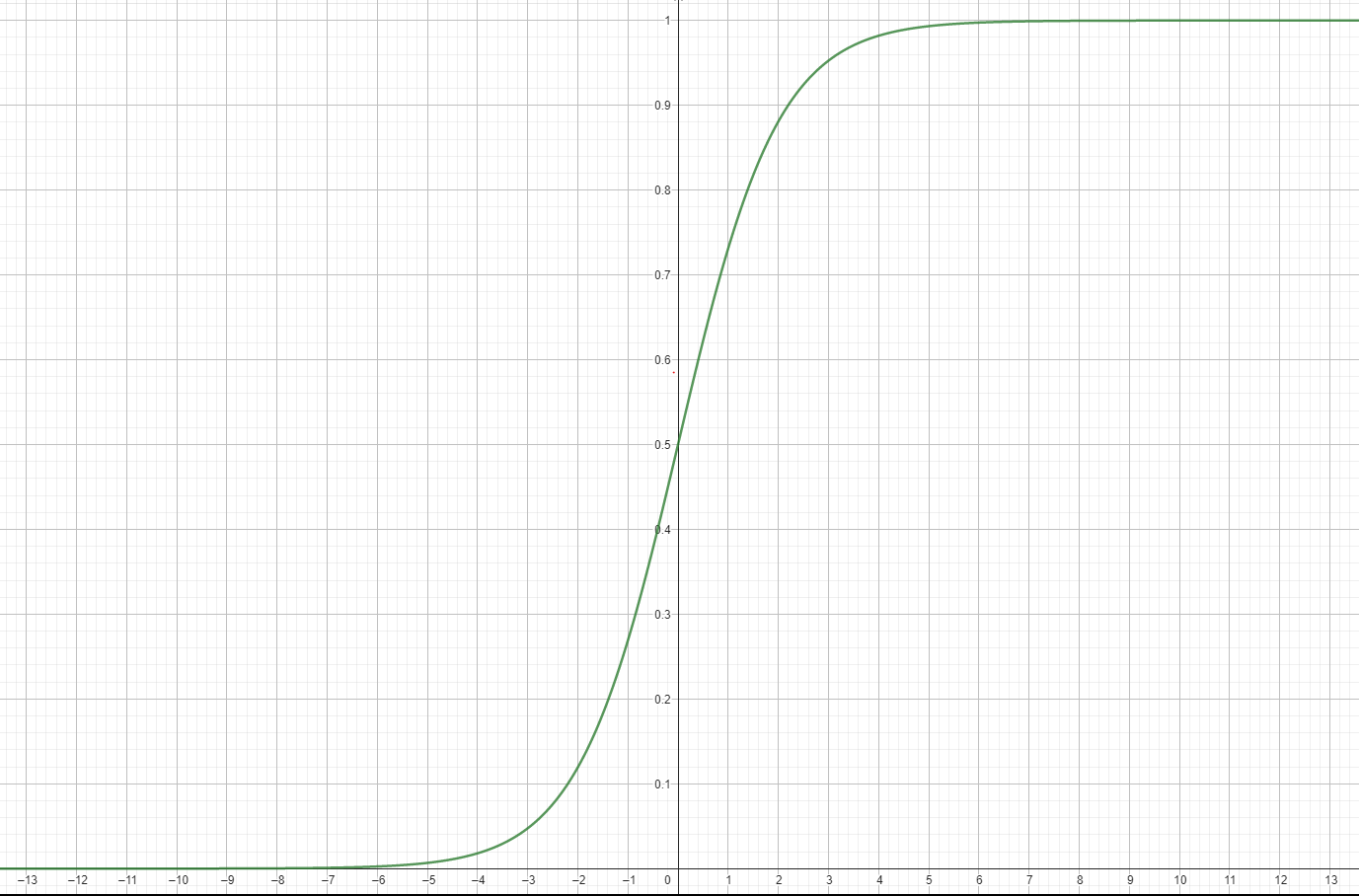
ReLU (rectified linear unit) - линейный выпрямитель. Это самая популярная функция активации. Из достоинств можно отметить, что при очень больших x, f(x) не стремится к какому-то значению, то есть нет ограничения, как у Sigmoid или Tanh, не является линейной, так как есть перегиб в x = 0. Недостатки: нет ограничения на положительные значения, из-за чего на выходе могут получаться огромные значения, особенно в многослойных сетях. Также при обучении возникает проблема с обратным распространением ошибки, так как происходит домножение на производную, а при x <= 0 она равна 0. Из-за этого ошибка может не доходить до первых слоёв, следовательно обновление весов не произойдёт.



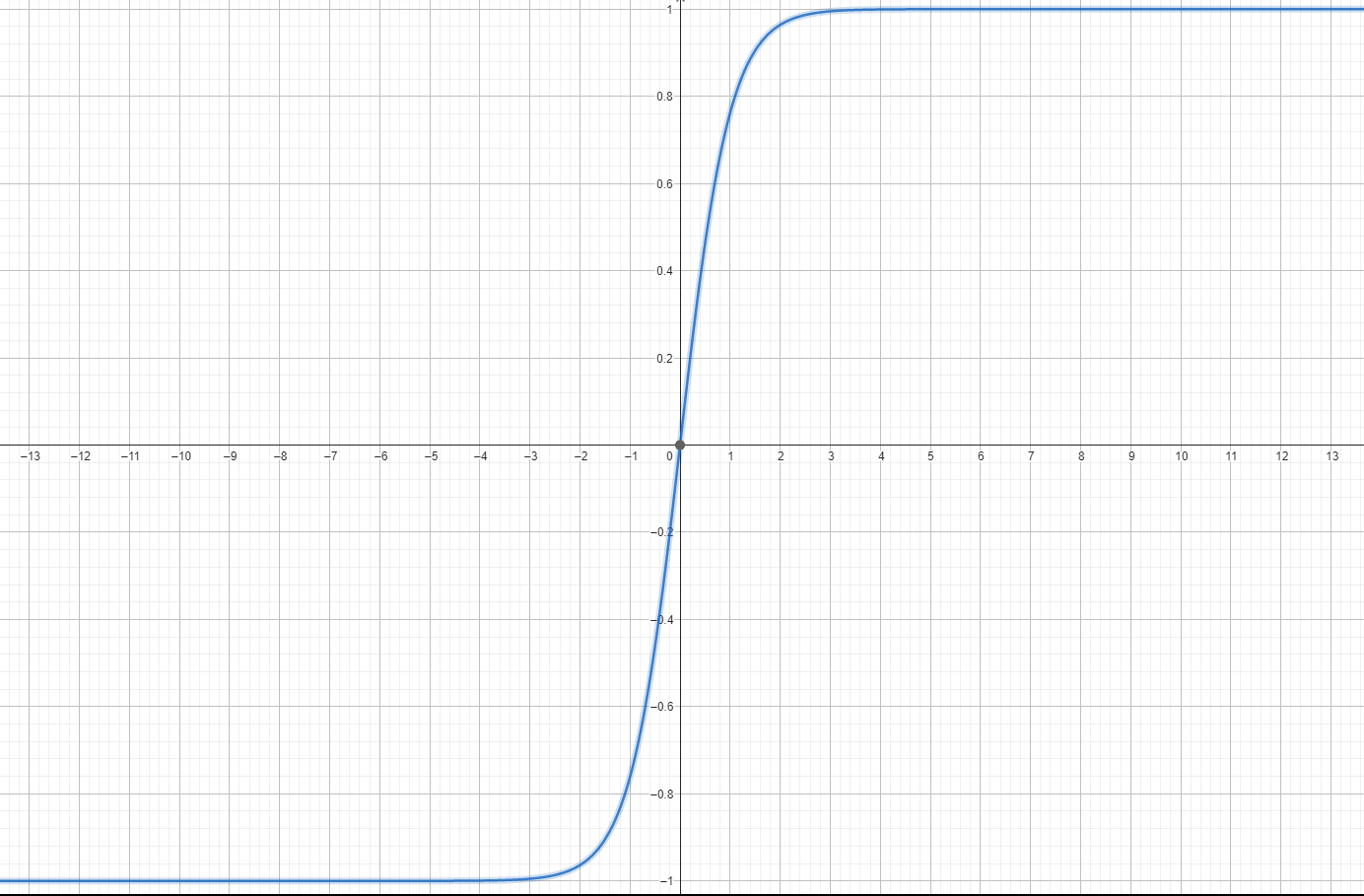
Проблему с обратным распространением ошибки решает усовершенствованная функция. Leaky ReLU – линейный выпрямитель с утечкой. Но несмотря на решение этой проблемы, данная функция достаточно редко применяется в каких-либо проектах.



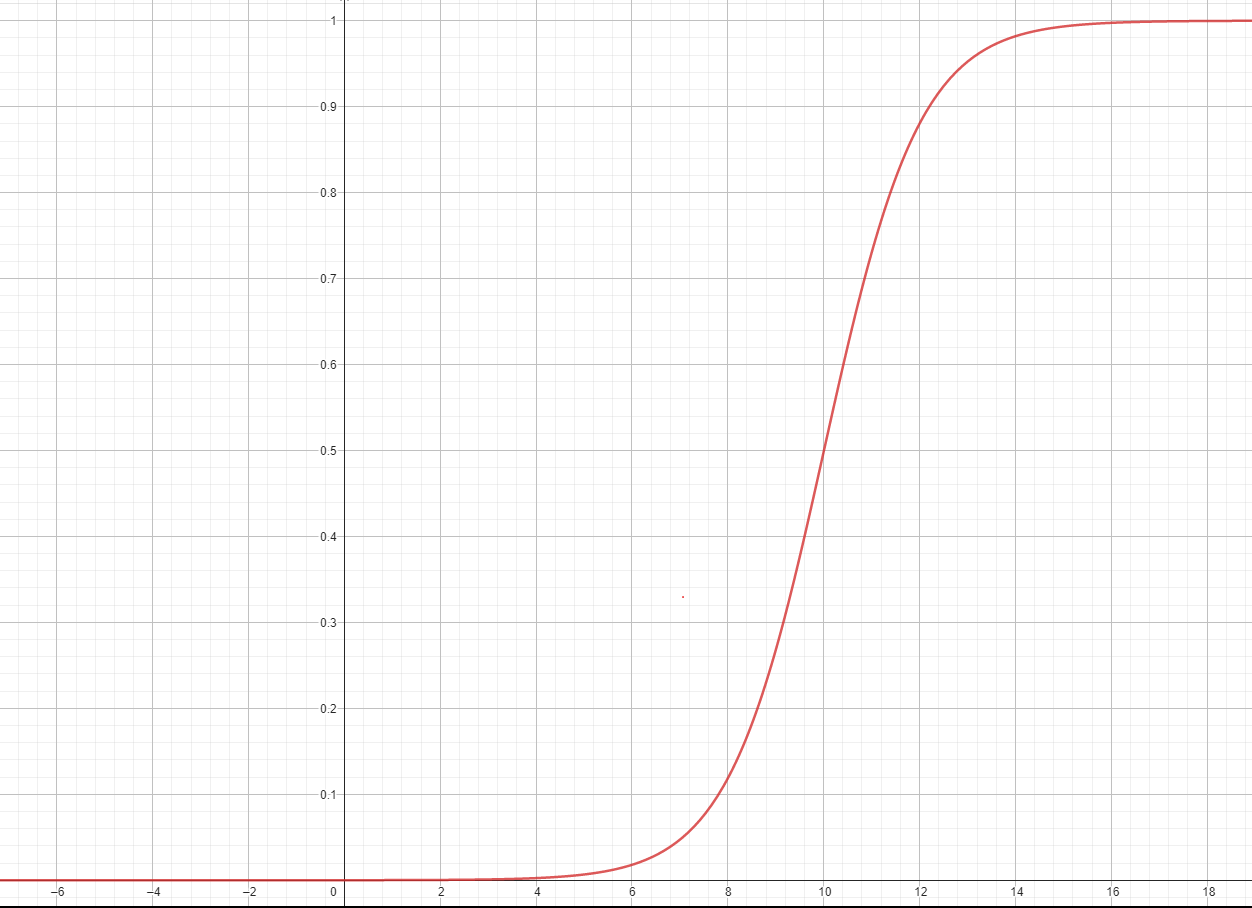
Sigmoid – сигмоида или логистическая. Это вторая по популярности функция активации. В отличие от ReLU Sigmoid ограничена сверху y = 1, что убирает возможность огромных значений, но при этом появляется другая проблема. При больших x f(x) очень быстро стремится к 1, из-за чего при x > 40, f(x) становится неразличимым. К плюсам можно отнести, что её можно использовать на выходном слое в задачах классификации, но с этой ролью лучше справляется SoftMax.



Tanh (hyperbolic tangent)- гиперболический тангенс. Эта функция очень похожа на Sigmoid, главное отличие в том, что область значения (-1, 1).

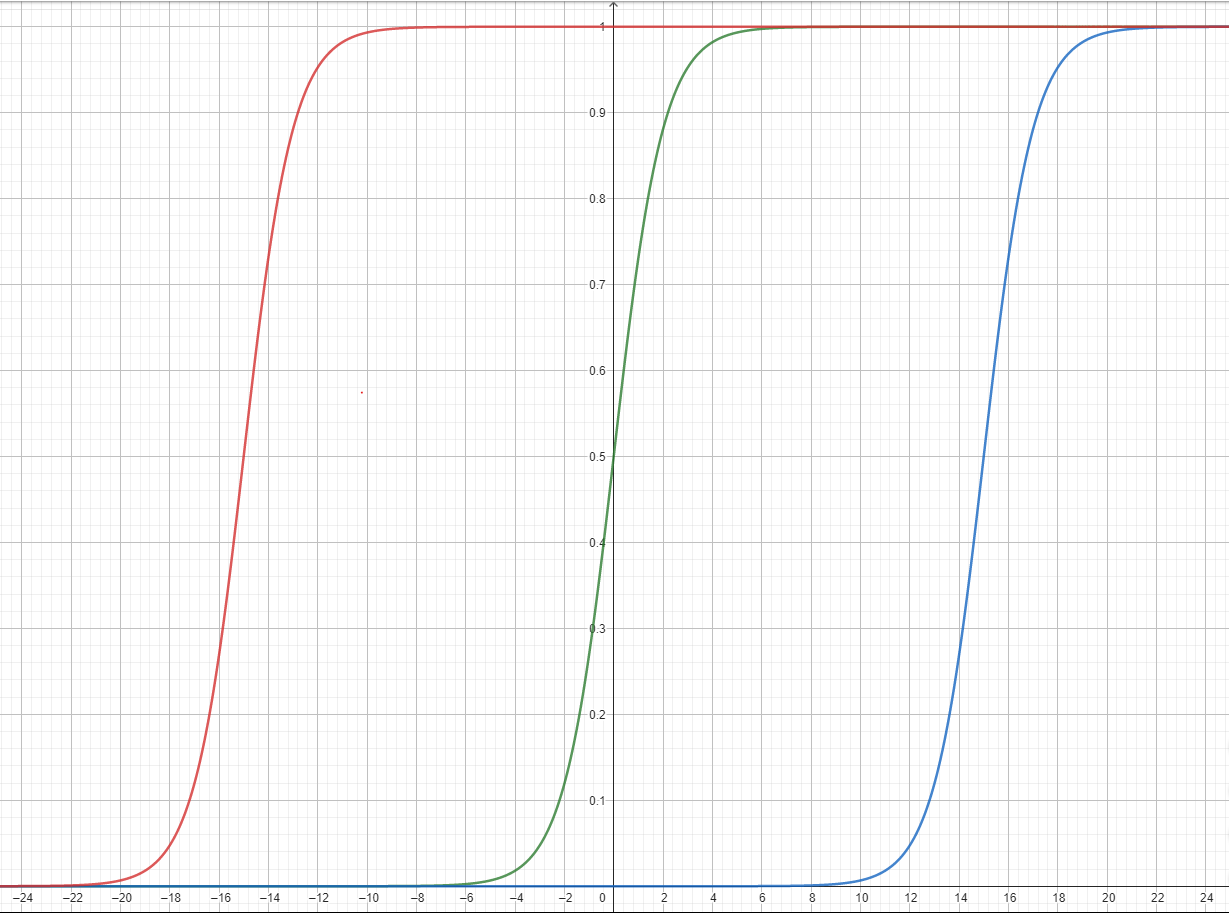


SoftMax - софт макс. Эта функция применяется только на выходном слое нейросети. Её используют для классификации данных, так как область значений (0, 1]. Главным свойством этой функции является то, что сумма f(xi) выходного слоя равна 1. На графике представлена f(x5), где x1 = 12, x2 = -1, x3 = 40, x4 = 13.



2.2.3. Смещение нейрона

Смещение нейрона или сдвиг (bias) – это свободный коэффициент в уравнение, он нужен для достижения требуемых ответов, это можно увидеть на графике.



– сигмоидная функция активации

Зелёная b = 0

Красная b = 15

Синяя b = -15

Зелёным обозначена функция активация без смещения, как видно по графику при больших или малых x функция стремится к 1 или 0 соответственно. Из-за этого нам необходимо, чтобы входные данные были равномерно расположены относительно x = 0, то есть среднее значение входных данных стремилось к 0, иначе разница в ответах нейросети при разных данных будет несущественна. Пример без смещения:

Как видно разница очень маленькая, но теперь добавим смещение. На графике это синяя функция.

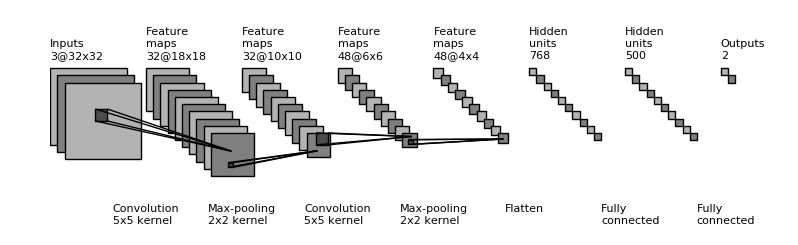
Разница изменилась и при том очень сильно, что мы и пытались добиться. То есть, чтобы нормализовать функцию активации около какого-то значения, необходимо использовать смещение.

2.2.4. Свёрточные нейронные сети

Свёрточная нейронная сеть (convolutional neural network) – это модель нейросети, которая обрабатывает изображения. Она состоит из свёрточных слоёв, слоёв подвыборки и полносвязных слоёв.

В самом начале необходимо получить изображение. На компьютере оно хранится в jpg, jpeg, png или других форматах, нам нужно считать его и обработать. В разных язык программирования это происходит по-разному, но в итоге мы должны получить двумерный массив, в каждой ячейке которого находится массив из трёх чисел, который представляет цвет данного пикселя в формате RGB. После необходимо “поменять оси”, то есть сделать из изображения форматом [height][width][3] изображение [3][height][width]. Мы разделили цветное изображение на 3 канала R, G, B (красный, зелёный, синий). Это необходимо, так как свёрточная нейронная сеть получает на вход многоканальное изображение именно в таком формате [channels][height][width].

Эти входные данные поступают в первый слой, который является свёрточным, далее идёт чередование слоёв подвыборки и свёрточных. В конце этой части сети мы получаем трёхмерный массив [channels][height][width], который необходимо перевести в одномерный массив [channels \* height \* width], чтобы подать его на вход полно связанной сети.

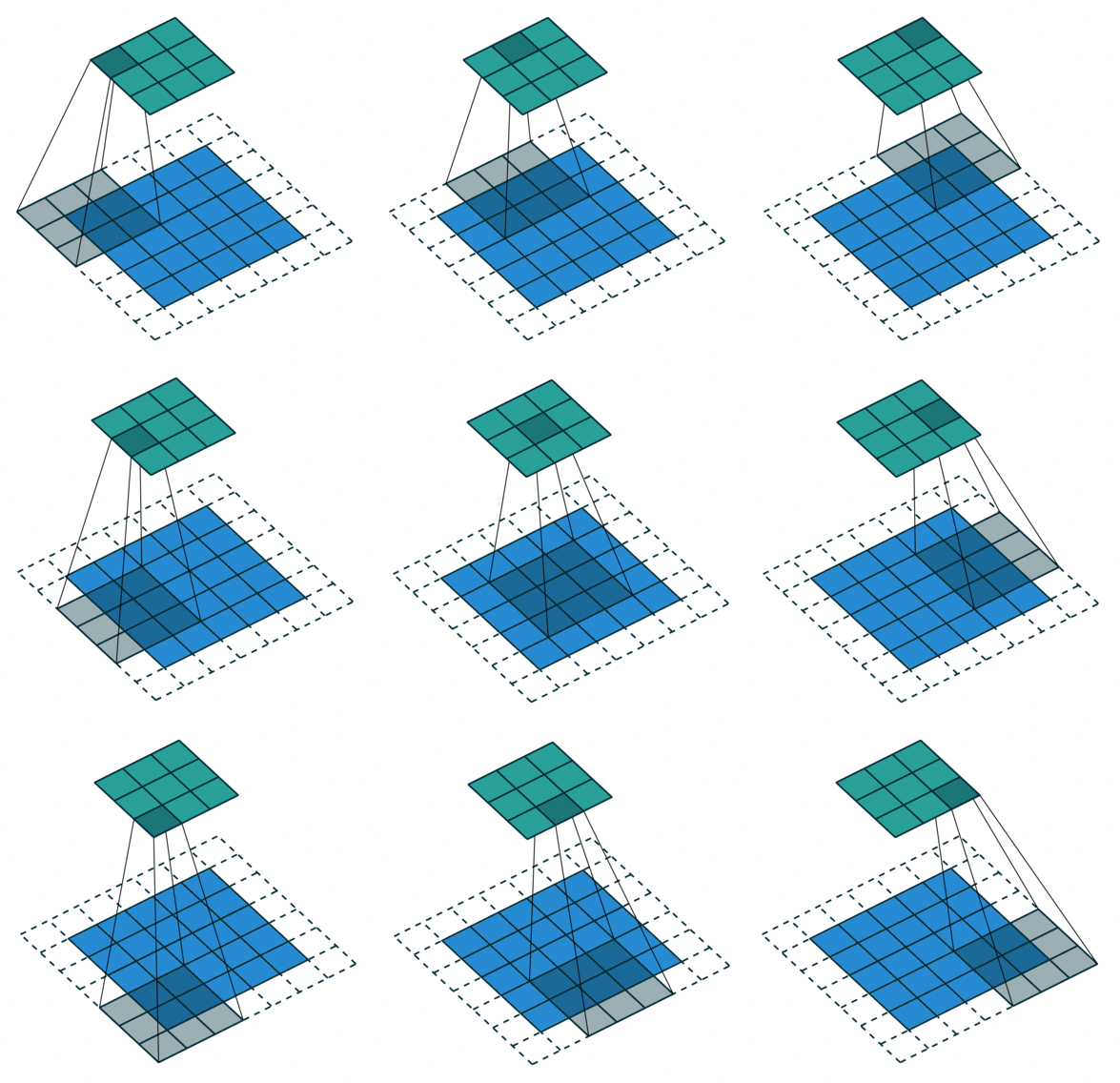


Источник https://jefkine.com/general/2016/09/05/backpropagation-in-convolutional-neural-networks/

2.2.5. Свёрточный слой

Свёрточный слой – слой, на котором происходит выделение признаков при помощи матриц, на выходе мы получаем карты признаков.

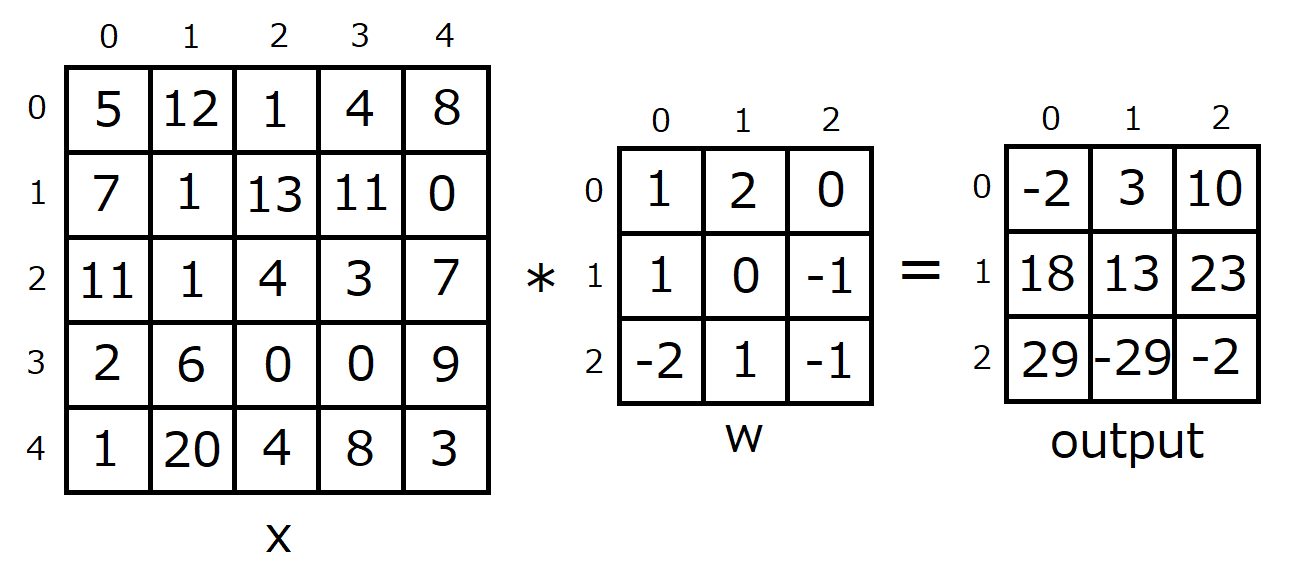
Для начала нужно разобраться, как происходит свёртка между изображением с 1 каналом и матрицей.



Источник https://habr.com/ru/companies/ods/articles/344008/

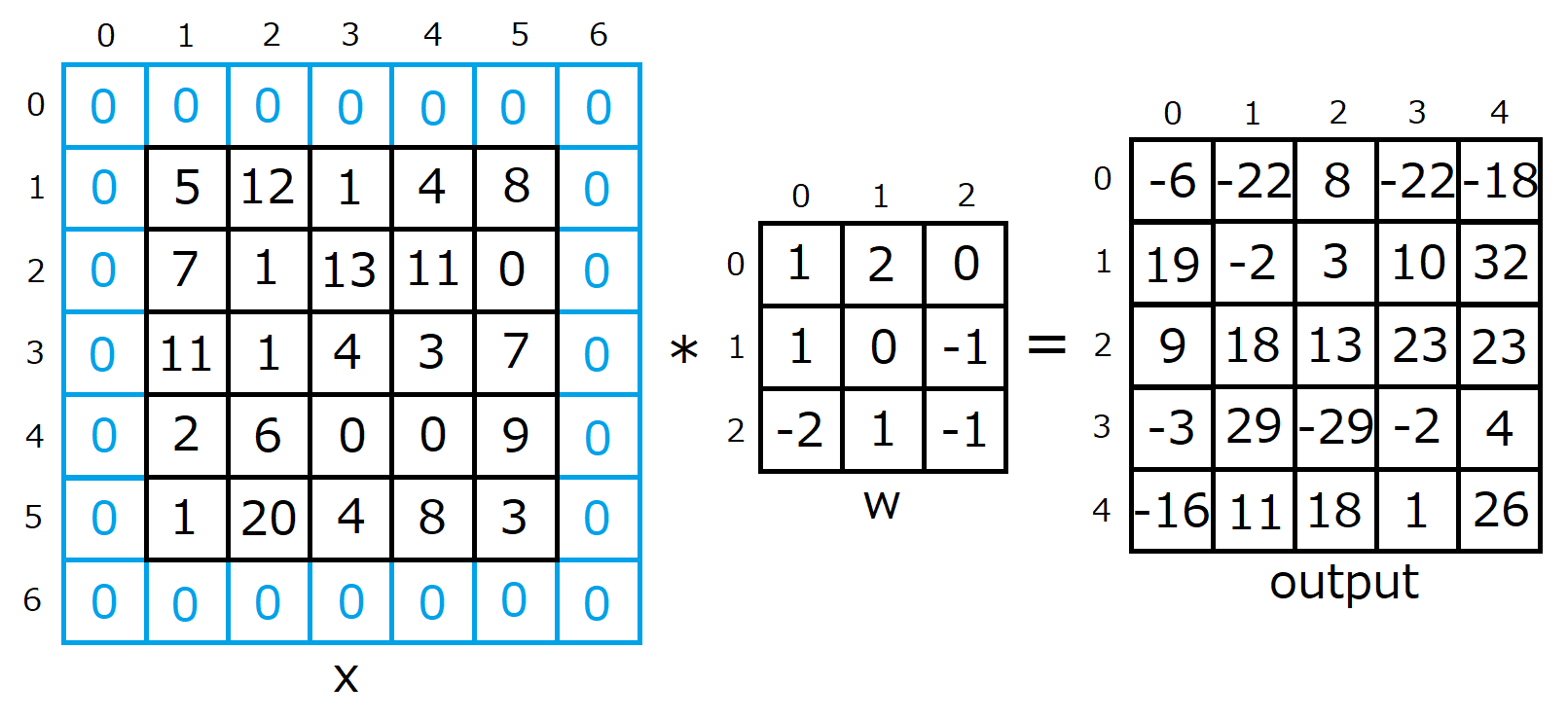
Синее — это изображение, серый квадрат, перемещающийся по нему, — это ядро свёртки, матрица, фильтр или весовые коэффициенты. Зелёный квадрат сверху – это выход свёрточного слоя или карта признаков. По краям чёрточками обозначены ячейки дополнения изображения или падинга (padding), о нём мы поговорим чуть позже. Мы проходим по изображению матрицей, перемножаем значения с весами, суммируем их на каждом шаге отдельно и соответственно записываем в ячейки (output).

Где h – координата элемента в карте признаков по y, w по x, mh – высота матрицы, mw – ширина матрицы.

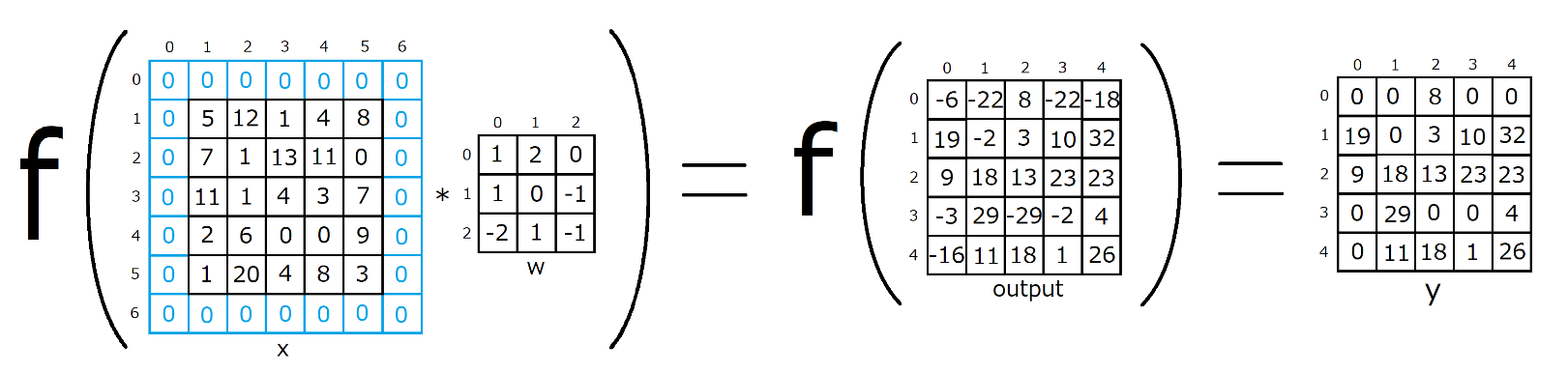


Для большей ясности приведу примеры:

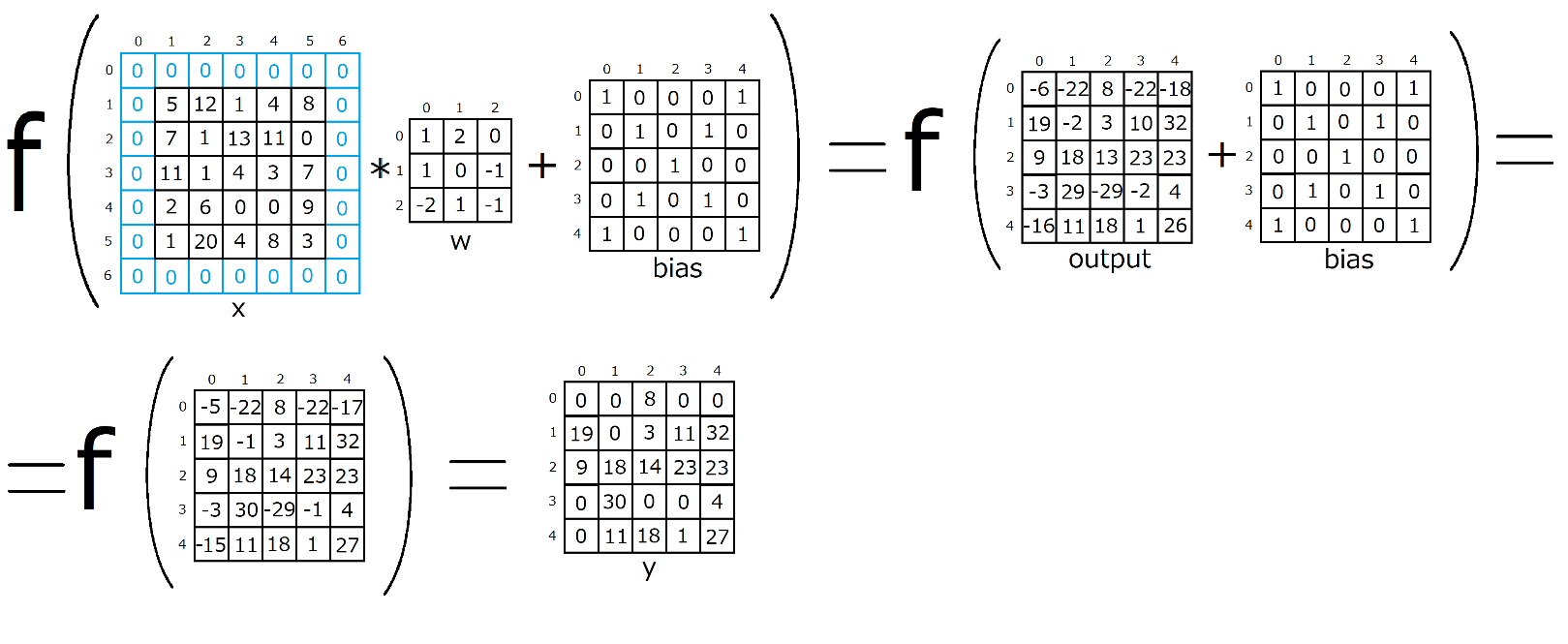
Теперь поговорим о падинге (padding). Как вы видите выходное изображение уменьшилось в размере, но на этом слое мы хотим лишь выделить признаки, а не уменьшать размерность изображения. Эту проблему решает дополнение изображения. До операции свёртки мы добавляем вокруг изображения пустые значения, то есть окружаем его 0. При этом у нас может быть не одно нулевое значение, мы можем сделать падинг двумя или более ячейками, правда это очень редко используется на практике.



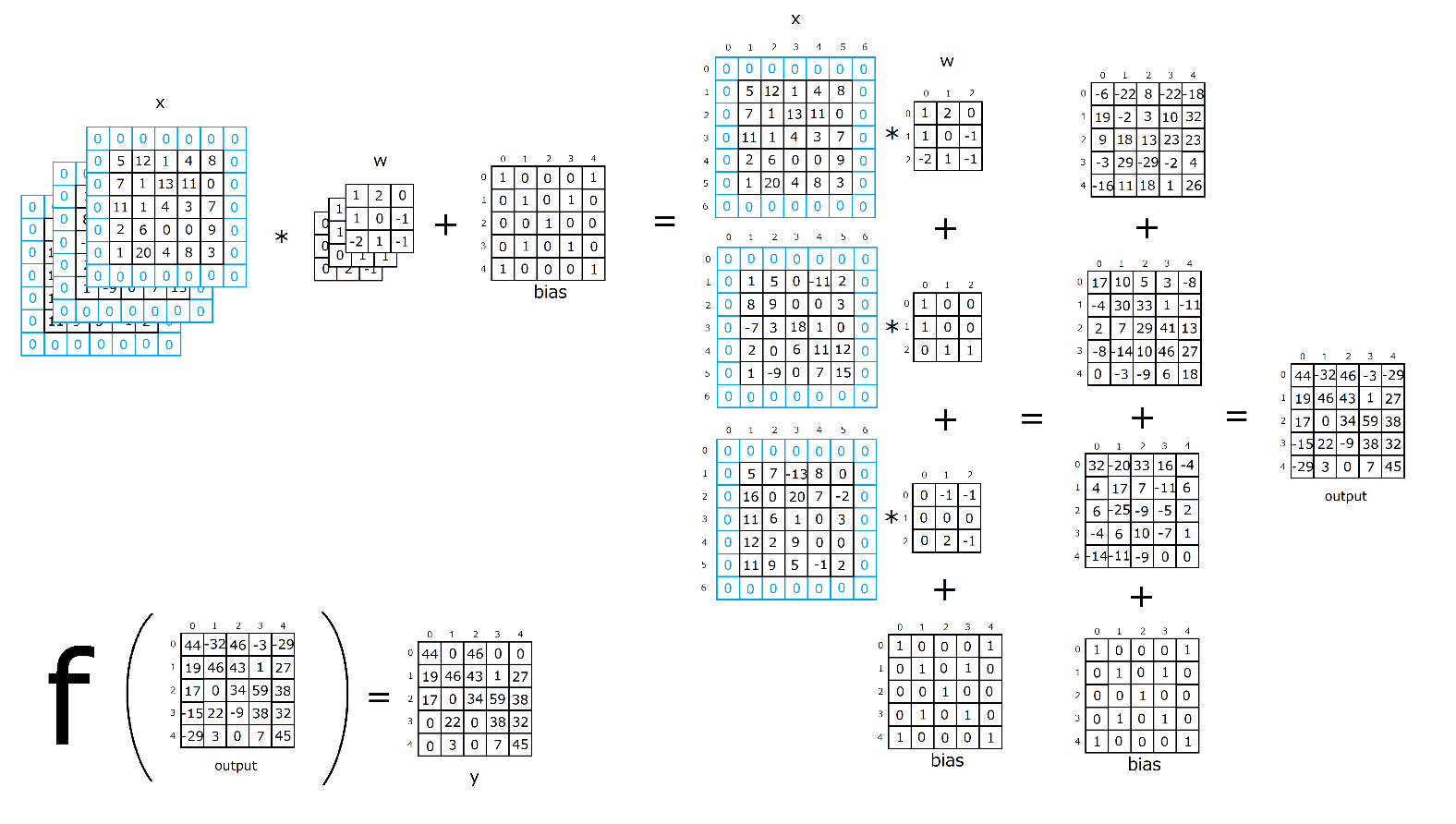
И теперь финальный этап, также как в полносвязной нейронной сети, в свёрточной есть функция активации и смещение. Функция активации применяется к выходному слою (output) поэлементно, то есть к каждому по отдельности. Для примера воспользуемся функцией активации ReLU.



Теперь добавим смещение. Биас это матрица с такой же размерностью, как и карта признаков (y). Он прибавляется к output, до функции активации.



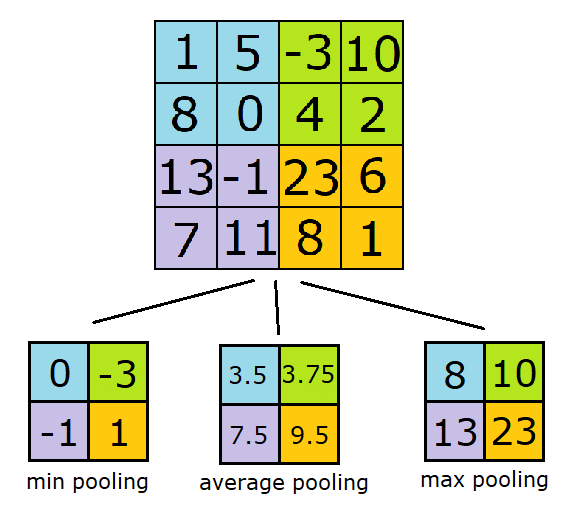
Но, как было сказано ранее, это для одноканального изображения. В реальности используется трёхканальное изображение на входном слое с последующим увеличением количества каналов. Для свёртки многоканального изображения, нужно использовать матрицу с таким же количеством каналов. Мы проделываем свёртку для каждого канала по отдельности, после чего суммируем результат (output) и прибавляем смещение. Дальше прогоняем всё это через функцию активации.



В результате мы получаем одноканальную карту признаков, но это только один признак. Если мы хотим отразить несколько признаков, то мы должны использовать не одну, а несколько пар весов и смещений. То есть мы просто делаем такую же свёртку многоканального изображение, но при этом повторяем её несколько раз с разными весами и смещениями. В итоге мы получаем многоканальную карту признаков.

2.2.6. Слой подвыборки

Слой подвыборки (пулинг слой, pooling layer) – слой, который понижает размерность изображения, то есть он сжимает изображение, выделяя только главные признаки. Существуют три вида такого слой: max pooling (максимум), min pooling (минимум), average pooling (среднее значение). Раньше активно использовался average pooling, но сейчас используют max pooling. Min pooling очень редко используется, лично я ни разу не видел его в проектах. Теперь нужно понять, как этот слой работает. Мы проходим непересекающимися окнами по изображению и делаем обработку внутри них. Различия max, min, average pooling заключается в том, что происходит в окне, какая обработка. Max pooling выбирает максимальное значение из окна и переносит его в выходной слой. Min pooling выбирает минимальное, а average pooling считает среднее значение среди всех значений в окне.



2.3. Обратный проход

2.3.1. Функция потерь

Функция потерь — это функция, которая используется на выходном слое нейронной сети, она вычисляет ошибку сети, а также участвует в обучении. Она на вход получает два значения: предсказанный нейросетью ответ и правильный ответ. Функция сравнивает их и выдаёт ошибку сети. Производная функции потерь используется при обучении нейросети с помощью алгоритма обратного распространения ошибки, об этом позже. Существуют различные функции потерь:

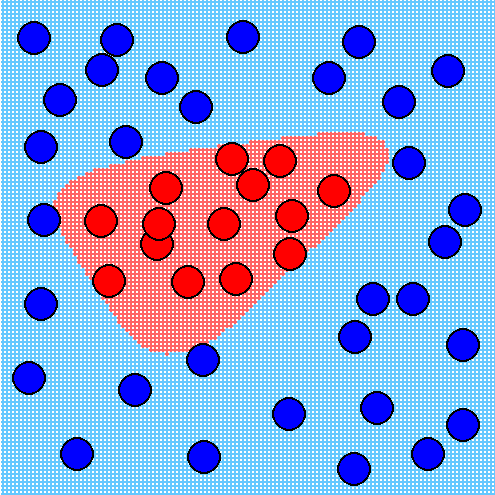
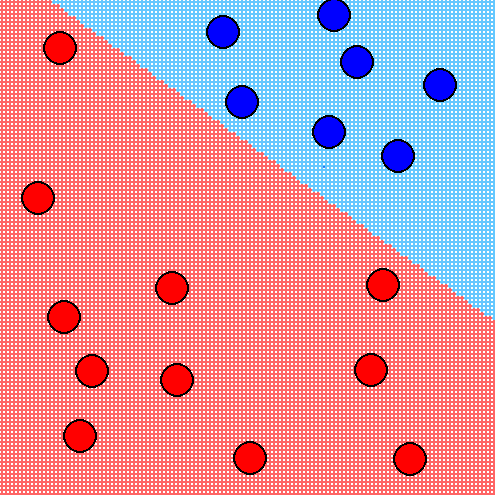
MSE (mean squared error) - среднеквадратичная ошибка. Не устойчива к выбросам, потому что мы возводим разницу в квадрат. Output – выходные данные сети, ideal – результат, к которому стремимся, n – количество выходных нейронов. Заметим, что в функции output и ideal – это матрицы, а в производных просто числа, в следующих функциях будет также.

MAE (mean absolute error) - средняя абсолютная ошибка.

Cross Entropy - перекрёстная энтропия. Функция, которая работает только, когда в выходных данных (в требуемом от нейронной сети ответе, не в её выходе) есть только 0 или 1. Из-за этого её можно применять только в задачах классификации, также есть ограничение, связанное с формулой. Так как мы используем логарифм, то на выходном слое необходимо использовать или Sigmoid или SoftMax, чтобы значения выходного слоя не были отрицательными или слишком большими.

Huber loss - функция потерь Хьюбера. Схожа с MSE, но при этом более устойчива к выбросам, поэтому её можно успешно применять в задачах регрессии. Коэффициент выбирается вручную, подбирается опытным путём.

Hinge loss - потеря шарнира. Используется в задачах классификации. Часто применяется при обучении машин опорных векторов (решение задач классификации, разделение плоскости гиперплоскостью).



2.3.2. Алгоритм обратного распространения ошибки через полносвязные слои

Алгоритм обратного распространения ошибки - алгоритм обучения нейронной сети с учителем.

Обучение с учителем – это один из видов обучения, когда для входных данных уже известны выходные данные. Помимо его есть обучение без учителя, когда нет необходимых выходных данных, нейросеть решает задачу кластеризации или ассоциации, и обучение с подкреплением, когда нейросеть учиться на своих ошибках, то есть она не знает нужный ответ, но неправильный выход нейросети приводит к наказанию.

По его названию можно полностью понять, как он работает. На выходном слое мы рассчитываем ошибку сети (градиент), после чего распространяем её на предыдущие слои. Для этого мы на выходном слое перемножаем производную функции потерь от значения нейрона с производной функции активации от него же, так делаем для каждого выхода.

После идёт распространение ошибки на предыдущие слои, то есть с конца в начало. Мы умножаем производную функцию активации от нейрона предыдущего слоя на сумму произведений ошибки нейрона из следующего слоя на веса, которые соединяют нейрон из предыдущего и следующего слоёв. k – количество нейронов на i – 1 слое.

В итоге мы получаем градиент на каждом нейроне. Теперь мы будем обновлять весовые коэффициенты, для этого нужно подсчитать изменение весов delta\_w. Мы перемножаем коэффициент обучения (это один из гиперпараметров, о нём мы поговорим позже), взятый с минусом, градиент нейрона, к которому идут веса, и выход нейрона, от которого идут веса. После мы прибавим к весам их изменения.

2.3.3. Алгоритм обратного распространения ошибки через свёрточный слой

Задача такая же, как и в полносвязных слоях, найти изменение весов и передать ошибку на предыдущий слой. На вход поступает матрица ошибки с такой же размерностью как у выходной.

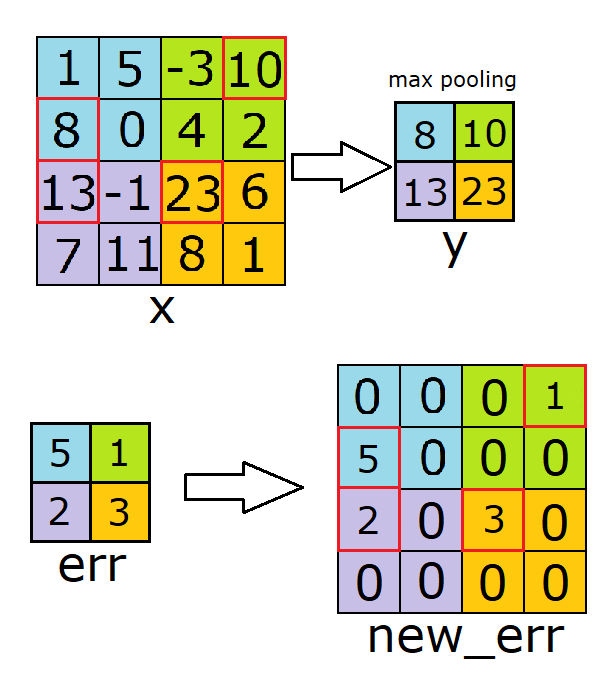
Для начала найдём изменения весов. Так как ядро свёртки проходит по изображению, то, чтобы посчитать изменение ошибки нам также необходимо пройти по изображению. Проходим по входной матрице и на каждом шаге считаем изменение для каждого весового коэффициента таким образом: перемножаем коэффициент обучения, значение ячейки изображения, из которой выходит связь, и ошибка ячейки, к которой она присоединяется. Суммируем эти произведения со всех шагов, для каждого веса по отдельности, это и будет изменение. Из каждого весового коэффициента вычитаем соответственное изменение. h, w – соответственно высота и ширина матрицы ошибок или выходных данных, размерности у них одинаковые.

Теперь необходимо распространить ошибку на предыдущий слой. Новая матрица ошибки имеет такую же размерность, как и входное изображение, если было дополнение изображения (падинг), то он тоже не учитывается. Начинаем с прохода по изображению, на каждом шаге мы к каждой ячёки прибавляем ошибку нейрона, с которым эта группа соединена, помноженную на вес. Это и будем итоговая новая ошибка.

2.3.4. Алгоритм обратного распространения ошибки через слой подвыборки

Так как в этом слое нет никаких весов и коэффициентов, то и обучать нам нечего, поэтому единственное, что нужно сделать это передать ошибку на предыдущий слой. На самом деле этот алгоритм отличается от того, какой вид подвыборки выбран. Разберём только максимальную подвыборку.

На вход нам подаётся матрица с ошибками с такой же размерностью, как и выходной слой. Мы проходим по входному изображению и ищем значение, которое передалось в выходной слой (так как мы используем максимальную подвыборку, нужно искать максимум из окна), в эту ячейку мы вставляем ошибку из входных данных.



2.3.5. Гиперпараметры

Во время обучения нейронной сети мы используем коэффициент обучения – это гиперпараметр, существуют и другие: количество слоёв в нейронной сети, количество нейронов в них, моментум – это специальный коэффициент, который применяется в улучшенном алгоритме обратного распространения ошибки. Гиперпараметры – это параметры сети, которые нельзя обучить, они настраиваются вручную, чаще всего просто подбором, но можно сделать некоторые зависимости. Например, от количества эпох обучения или от ошибки сети. Именно от них зависит результат обучения нейросети.

3. Практика

3.1. Идея и реализация

Я создал приложение, которое распознаёт рукописные цифры. Для этого я использовал язык программирования Python, встроенные библиотеки os, sys, json, math, random, time, deepcopy, pprint, а также дополнительные модули pygame, numpy и Pillow. Структура программы такова:

Есть папка core, где хранится нейросеть, в этой папке есть несколько файлов. full\_connected.py – класс полносвязной нейросети, convolution – свёрточный слой, max\_pooling – слой максимальной подвыборки, library – здесь подключаются все дополнительные библиотеки, прописаны функции активации, функции потерь и дополнительные переменные, \_\_init\_\_.py – файл, в котором хранится основной класс, который собирает всё в одну нейросеть, добавляет функционал.

В другой папке app хранится приложение, main.py – приложение, data\_loader.py – скрпит, который превращает все изображения в один датасет, train.py – подпрограмма, которая обучает нейросеть.

3.2. Библиотеки

os, sys – работа с файловой системой компьютера, выход из программы

json – сохранение данных в файлы и загрузка из них

math – логарифмы, экспонента

random – случайная инициализация весов и смещений нейросети

numpy + Pillow – загрузка и сохранение изображений

pygame – основное приложение

3.3. Структура нейросети

В каждом отдельном файле содержится отдельный класс, представляющий собой слой. Всего есть три класса: FullConnected – полносвязная нейронная сеть, Convolution – свёрточный слой, MaxPooling – слой максимальной подвыборки. У каждого слоя есть две основные функции помимо \_\_init\_\_. Это forward – прямой проход и backward – обратный проход. Чтобы всё работало, как одно целое, я написал класс nn, у него тоже есть функции forward и backward, но ещё есть save и load, которые отвечают за сохранение и загрузку весов и смещений из отдельного файла. Также есть info – печатает в консоль данные о структуре нейросети, result – функция, выдающая ответ нейросети, train\_alpha – обучает функцию пока ошибка не будет меньше одно из параметров.

4. Заключение

По итогу у меня получилось выполнить все цели и задачи, но, пока я занимался созданием проекта, у меня появилось ещё несколько идей, которые я реализую в будущем. Во-первых, обучение нейросети происходит долго и результат недостаточно точен, поэтому я хочу реализовать улучшенный метод обратного распространения ошибки – Adam. Во-вторых, нужно сделать распознавание всех цифр от 0 до 9 и увеличить размер поля. Проект я считаю завершённым, но будут производиться доработки и изменения, с целью его улучшения.

5. Источники

1. Нейронные сети для начинающих. Часть 1 - <https://habr.com/ru/post/312450/>
2. Нейронные сети для начинающих. Часть 2 - <https://habr.com/ru/post/313216/>
3. Common Loss functions in machine learning - <https://towardsdatascience.com/common-loss-functions-in-machine-learning-46af0ffc4d23>
4. Университете ИТМО Обратное распространение ошибки - <https://neerc.ifmo.ru/wiki/index.php?title=Обратное_распространение_ошибки>
5. Understanding RMSprop - faster neural network learning - <https://towardsdatascience.com/understanding-rmsprop-faster-neural-network-learning-62e116fcf29a>
6. A Step-by-Step Backpropagation Example - <https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/>
7. English Wikipedia Backpropagation - <https://en.wikipedia.org/wiki/Backpropagation#Derivation>
8. What is momentum in neural network? - <https://datascience.stackexchange.com/questions/84167/what-is-momentum-in-neural-network>
9. Нейронная сеть с SoftMax слоем на c# - <https://habr.com/ru/post/155235/>
10. Алгоритм обратного распространения ошибки с регуляризацией на c# - <https://habr.com/ru/articles/154369/>
11. Сверточная сеть на python. Часть 1. Определение основных параметров модели - <https://habr.com/ru/company/ods/blog/344008/>
12. Сверточная сеть на python. Часть 2. Вывод формул для обучения модели - <https://habr.com/ru/company/ods/blog/344116/>
13. Stochastic Gradient Descent with momentum - <https://towardsdatascience.com/stochastic-gradient-descent-with-momentum-a84097641a5d>
14. Генетический алгоритм. Просто о сложном - <https://habr.com/ru/post/128704/>
15. Loss Functions in Neural Networks - <https://www.theaidream.com/post/loss-functions-in-neural-networks>
16. Cross entropy - <https://en.wikipedia.org/wiki/Cross_entropy>
17. Xgboost-How to use "mae" as objective function? - <https://stackoverflow.com/questions/45006341/xgboost-how-to-use-mae-as-objective-function>
18. Обзор 7 основных функций потерь в машинном обучении (с кодом Python) - <https://russianblogs.com/article/34271570686/>
19. A Gentle Introduction to Dropout for Regularizing Deep Neural Networks - <https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/>
20. Convolutions and Backpropagations - <https://pavisj.medium.com/convolutions-and-backpropagations-46026a8f5d2c>
21. What Are Channels in Convolutional Networks? - <https://www.baeldung.com/cs/cnn-channels>
22. Backpropagation In Convolutional Neural Networks - <https://www.jefkine.com/general/2016/09/05/backpropagation-in-convolutional-neural-networks/>
23. The Softmax function and its derivative - <https://eli.thegreenplace.net/2016/the-softmax-function-and-its-derivative/>
24. Why do we rotate weights when computing the gradients in a convolution layer of a convolution network? - <http://soumith.ch/ex/pages/2014/08/07/why-rotate-weights-convolution-gradient/>
25. Convolutional Neural Networks - <https://andrew.gibiansky.com/blog/machine-learning/convolutional-neural-networks/>
26. Convolutional Neural Networks backpropagation: from intuition to derivation - <https://grzegorzgwardys.wordpress.com/2016/04/22/8/#unique-identifier2>
27. Свёрточная нейронная сеть - <https://ru.wikipedia.org/wiki/Свёрточная_нейронная_сеть#Архитектура_и_принцип_работы>
28. Backpropagation in a convolutional layer - <https://towardsdatascience.com/backpropagation-in-a-convolutional-layer-24c8d64d8509>
29. Does bias in the convolutional layer really make a difference to the test accuracy? - <https://stackoverflow.com/questions/51959507/does-bias-in-the-convolutional-layer-really-make-a-difference-to-the-test-accura>
30. Adam - <https://optimization.cbe.cornell.edu/index.php?title=Adam>
31. Question about bias in Convolutional Networks - <https://datascience.stackexchange.com/questions/11853/question-about-bias-in-convolutional-networks/>
32. Стохастический градиентный спуск - <https://ru.wikipedia.org/wiki/Стохастический_градиентный_спуск>