Федеральное государственное бюджетное образовательное

учреждение высшего образования

Московский авиационный институт

(национальный исследовательский университет)

Факультет №3

«Системы управления, информатика и электроэнергетика»

Кафедра 304

«Вычислительные машины, системы и сети»

Отчет по научно-исследовательской работе в семестре

на тему «Классификация текстов на основе нейронных сетей»

Преподаватель: Чебатко М.И.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Студент: Долгополов Н.И.

Группа: М3О-107М-17

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Москва, 2017

**Оглавление**

[Введение 3](#_Toc9800938)

[Основные определения 4](#_Toc9800939)

[**Нейронная сеть** 4](#_Toc9800940)

[**Нейрон** 5](#_Toc9800941)

[**Функции активации** 6](#_Toc9800942)

[**Обучение нейронной сети** 15](#_Toc9800943)

[**Инициализация нейронных сетей** 17](#_Toc9800944)

[**Сверточные нейронные сети** 19](#_Toc9800945)

[**Регуляризация** 23](#_Toc9800946)

# Введение

Главная цель исследовательской работы – изучение нейронных сетей, используемой в сфере глубоко обучения терминологии и подходов.

Для достижения поставленной цели, в ходе исследования необходимо решить следующие задачи:

* Изучить теоретические основы нейронных сетей
* Определить круг алгоритмов, которые будут сравниваться в ходе практической работы

Научно-исследовательская работа выполнялась в Московском Авиационном Институте (Национальном Исследовательском Университете) на кафедре 304.

# Основные определения

### **Нейронная сеть**

Как известно, понятие нейронной сети (НС) пришло из биологии и представляет собой несколько упрощенную модель строения человеческого мозга. Проще всего представить нейрон (в том числе, искусственный) как некий черный ящик с множеством входных отверстий и одним выходным.

Математически, искусственный нейрон осуществляет преобразование вектора входных сигналов (воздействий) X в вектор выходных сигналов Y при помощи функции, называемой функцией активации. В рамках соединения (искусственной нейронной сети — ИНС) функционируют три вида нейронов: входные (принимающие информацию из внешнего мира – значения интересующих нас переменных), выходные (возвращающие искомые переменные – к примеру, прогнозы, или управляющие сигналы), а также промежуточные – нейроны, выполняющие некие внутренние («скрытые») функции. Классическая ИНС, таким образом, состоит из трех или более слоев нейронов, причем на втором и последующих слоях («скрытых» и выходном) каждый из элементов соединен со всеми элементами предыдущего слоя.

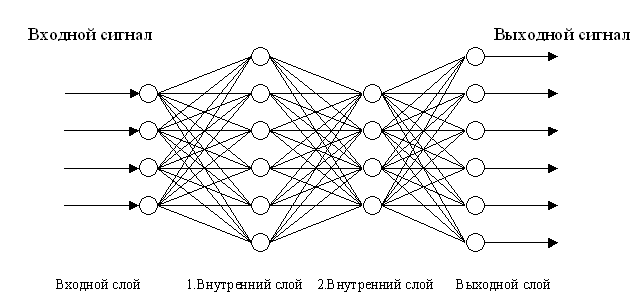


Рисунок 3.1 Пример нейронной сети

Важно помнить о понятии обратной связи, которое определяет вид структуры ИНС: прямой передачи сигнала (сигналы идут последовательно от входного слоя через скрытый и поступают в выходной слой) и рекуррентной структуры, когда сеть содержит связи, идущие назад, от более дальних к более ближним нейронам). Все эти понятия составляют необходимый минимум информации для перехода на следующий уровень понимания ИНС – обучения нейронной сети, классификации его методов и понимания принципов работы каждого из них.

### **Нейрон**

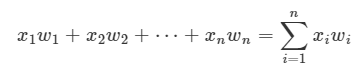


Рисунок 3.2 Схема нейрона

У каждого нейрона, в том числе и у искусственного, должны быть какие-то входы, через которые он принимает сигнал. Мы уже вводили понятие весов, на которые умножаются сигналы, проходящие по связи. На картинке выше веса изображены кружками.

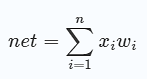
Поступившие на входы сигналы умножаются на свои веса. Сигнал первого входа ​x1​ умножается на соответствующий этому входу вес ​w1​. В итоге получаем ​x1w1​. И так до ​n​-ого входа. В итоге на последнем входе получаем ​xnwn​.

Теперь все произведения передаются в сумматор. Уже исходя из его названия можно понять, что он делает. Он просто суммирует все входные сигналы, умноженные на соответствующие веса:



Результатом работы сумматора является число, называемое взвешенной суммой.

Взвешенная сумма (Weighted sum) — сумма входных сигналов, умноженных на соответствующие им веса:



Роль сумматора очевидна – он агрегирует все входные сигналы (которых может быть много) в какое-то одно число – взвешенную сумму, которая характеризует поступивший на нейрон сигнал в целом. Еще взвешенную сумму можно представить как степень общего возбуждения нейрона.

### **Функции активации**

Функция активации - функция, вычисляющая выходной сигнал искусственного нейрона. В качестве аргумента принимает сигнал Y, получаемый на выходе входного сумматора.

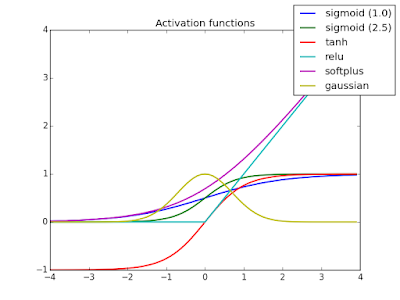


Рисунок 3.3 Графики основных функций активации

На рисунке 3.3 представлены графики основных функций активации:

* + 1. Сигмоидальная с a=1
    2. Сигмоидальная с a=2.5
    3. Гиперболический тангенс
    4. ReLU (выпрямитель)
    5. Softplus
    6. Гауссова

#### **Логистическая (сигмоидальная) функция**

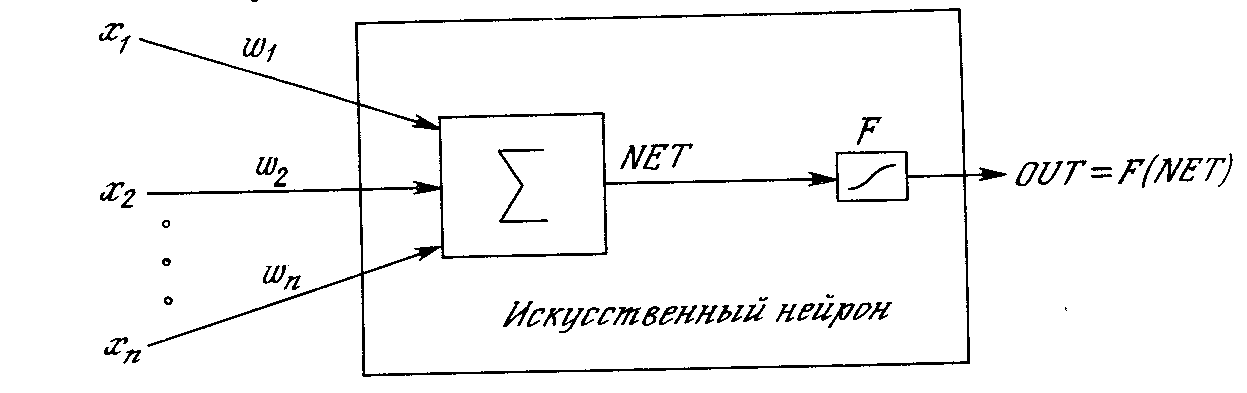


Рисунок 3.4 Искусственный нейрон с активационной функцией

Сигмоида (sigmoid) выражается следующей формулой: σ(x) = 1 / (1 + e-x). Эта функция принимает на входе произвольное вещественное число, а на выходе дает вещественное число в интервале от 0 до 1. В частности, большие (по модулю) отрицательные числа превращаются в ноль, а большие положительные – в единицу. Исторически сигмоида находила широкое применение, поскольку ее выход хорошо интерпретируется, как уровень активации нейрона: от отсутствия активации (0) до полностью насыщенной активации (1).

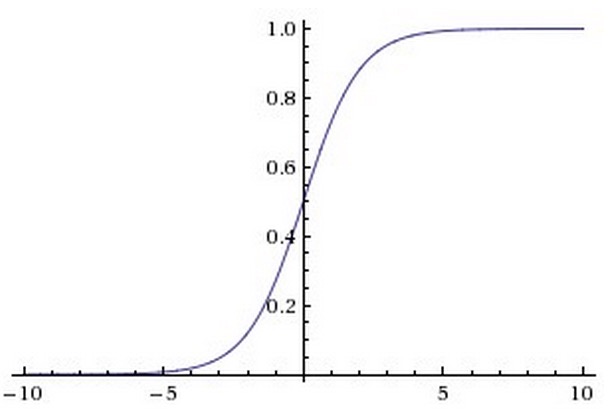


Рисунок 3.5 График сигмоиды

На текущий момент сигмоида утратила свою былую популярность и используется очень редко. Данная функция имеет два серьезных недостатка:

Насыщение сигмоиды приводит к затуханию градиентов. Крайне нежелательное свойство сигмоиды заключается в том, что при насыщении функции с той или иной стороны (0 или 1), градиент на этих участках становится близок к нулю. Напомним, что в процессе обратного распространения ошибки данный (локальный) градиент умножается на общий градиент. Следовательно, если локальный градиент очень мал, он фактически обнуляет общий градиент. В результате, сигнал почти не будет проходить через нейрон к его весам и рекурсивно к его данным. Кроме того, следует быть очень осторожным при инициализации весов сигмоидных нейронов, чтобы предотвратить насыщение. Например, если исходные веса имеют слишком большие значения, большинство нейронов перейдет в состояние насыщения, в результате чего сеть будет плохо обучаться.

Выход сигмоиды не центрирован относительно нуля. Это свойство является нежелательным, поскольку нейроны в последующих слоях будут получать значения, которые не центрированы относительно нуля, что оказывает влияние на динамику градиентного спуска (gradient descent). Если значения, поступающие в нейрон, всегда положительны (например, x > 0 поэлементно в f = ωTx + b), тогда в процессе обратного распространения ошибки все градиенты весов ω будут либо положительны, либо отрицательны (в зависимости от градиента всего выражения f). Это может привести к нежелательной зигзагообразной динамике обновлений весов. Однако следует отметить, что когда эти градиенты суммируются по пакету, итоговое обновление весов может иметь различные знаки, что отчасти нивелирует описанный недостаток. Таким образом, отсутствие центрирования является неудобством, но имеет менее серьезные последствия, по сравнению с проблемой насыщения.

#### **Приближенно-сигмоидальная функция (hard sigmoid)**

Сигмоидальная функция обладает многими преимуществами, но относительно долго вычисляется, так как использует операцию возведения в степень. Для случаев, когда гладкостью и непрерывной дифференцироемостью можно пренебречь, используется приближенно-сигмоидальная функция – кусочно-линейная функция, напоминающая своей формой сигмоиду. Ее форма близка к сигмоиде, но вычисление значительно быстрее, так как при вычислении используются только операции сложения и умножения.

В использованной в работе библиотеке Keras используется следующая реализация hard\_sigmoid:

* 0 if x < -2.5
* 1 if x > 2.5
* 0.2 \* x + 0.5 if -2.5 <= x <= 2.5.

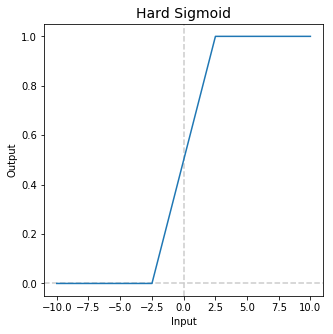


Рисунок 3.6 Приближенно-сигмоидальная фунцкия

#### **Гиперболический тангенс**

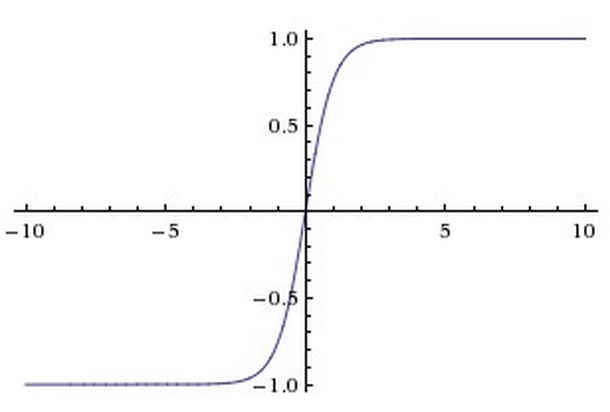
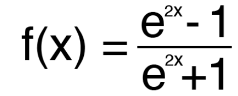


Рисунок 3.7 Гиперболический тангенс

Гиперболический тангенс (hyperbolic tangent, tanh) принимает на входе произвольное вещественное число, а на выходе дает вещественное число в интервале от –1 до 1. Гиперболический тангенс вычисляется следующим образом:



Подобно сигмоиде, гиперболический тангенс может насыщаться. Однако, в отличие от сигмоиды, выход данной функции центрирован относительно нуля.

#### **ReLU (выпрямитель)**

Известно, что нейронные сети способны приблизить сколь угодно сложную функцию, если в них достаточно слоев и функция активации является нелинейной. Функции активации вроде сигмоидной или тангенциальной являются нелинейными, но приводят к проблемам с затуханием или увеличением градиентов. Однако можно использовать и гораздо более простой вариант — выпрямленную линейную функцию активации (rectified linear unit, ReLU), которая выражается формулой:

https://habrastorage.org/webt/tq/by/ht/tqbyhtxpyotkoeqrn6aygpmne2m.png

График функции ReLU в соответствии с рисунком ниже:

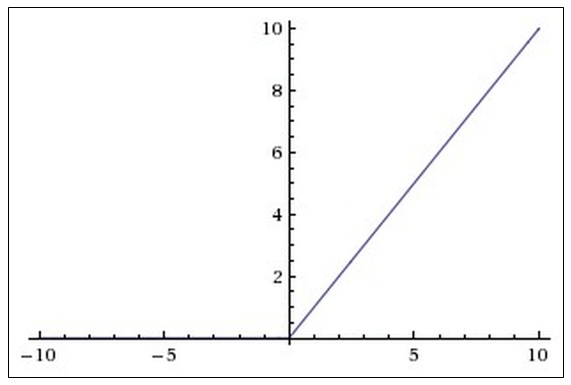


Рисунок 3.8 График функции ReLU

Преимущества использования ReLU:

Во-первых, ее производная равна либо единице, либо нулю, и поэтому не может произойти разрастания или затухания градиентов, т.к. умножив единицу на дельту ошибки мы получим дельту ошибки, если же мы бы использовали другую функцию, например, гиперболический тангенс, то дельта ошибки могла, либо уменьшиться, либо возрасти, либо остаться такой же, то есть, производная гиперболического тангенса возвращает число с разным знаком и величиной, что можно сильно повлиять на затухание или разрастание градиента. Более того, использование данной функции приводит к прореживанию весов;

Во-вторых, вычисление сигмоиды и гиперболического тангенса требует выполнения ресурсоемких операций, таких как возведение в степень, в то время как ReLU может быть реализован с помощью простого порогового преобразования матрицы активаций в нуле;

В-третьих, отсекает ненужные детали в канале при отрицательном выходе.

Из недостатков можно отметить, что ReLU не всегда достаточно надежна и в процессе обучения может выходить из строя («умирать»). Например, большой градиент, проходящий через ReLU, может привести к такому обновлению весов, что данный нейрон никогда больше не активируется. Если это произойдет, то, начиная с данного момента, градиент, проходящий через этот нейрон, всегда будет равен нулю. Соответственно, данный нейрон будет необратимо выведен из строя. Например, при слишком большой скорости обучения (learning rate), может оказаться, что до 40% ReLU «мертвы» (то есть, никогда не активируются). Эта проблема решается посредством выбора надлежащей скорости обучения.

#### **Модификации ReLU**

**Leaky ReLU:**

ReLU с «утечкой» (leaky ReLU, LReLU) представляет собой одну из попыток решить описанную выше проблему выхода из строя обычных ReLU. Обычный ReLU на интервале x < 0 дает на выходе ноль, в то время как LReLU имеет на этом интервале небольшое отрицательное значение (угловой коэффициент около 0,01). То есть функция для LReLU имеет вид f(x) = αx при x < 0 и f(x) = x при x ≥ 0, где α – малая константа. Некоторые исследователи сообщают об успешном применении данной функции активации, но результаты не всегда стабильны.

**Parametric ReLU:**

Для параметрического ReLU (parametric ReLU, PReLU] угловой коэффициент на отрицательном интервале не задается предварительно, а определяется на основе данных. Авторы публикации утверждают, что применение данной функции активации является ключевым фактором, позволившим превзойти уровень человека в задаче распознавания изображений ImageNet. Процесс обратного распространения ошибки и обновления для PReLU достаточно прост и подобен соответствующему процессу для традиционных ReLU.

**Randomized ReLU:**

Для рандомизированного ReLU (randomized ReLU, RReLU) угловой коэффициент на отрицательном интервале во время обучения генерируется случайным образом из заданного интервала, а во время тестирования остается постоянным.

В работе [<https://arxiv.org/abs/1505.00853>] авторы сравнили точность классификации двух сверточных сетей с различными функциями активации на наборах данных CIFAR-10, CIFAR-100 и NDSB. Результаты приведены в рисунках ниже.

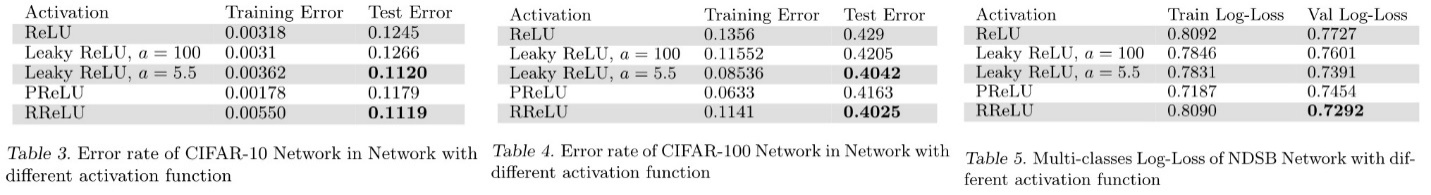


Рисунок 3.9 Сравнение модификаций ReLU

Результаты говорят о том, что для всех трех наборов данных модифицированные ReLU превзошли традиционные. В случае LReLU большее значение углового коэффициента α обеспечивает более высокую точность. PReLU склонны к переобучению на малых наборах данных (ошибка на обучающем наборе наименьшая из всех, в то время как ошибка на тестовом наборе больше, чем у конкурирующих модификаций ReLU). При этом PReLU все же превосходит традиционный ReLU. Следует отметить, что RReLU существенно превосходит другие функции активации на наборе данных NDSB. Это говорит о том, что RReLU позволяет избежать переобучения, поскольку этот набор содержит меньше обучающих данных, чем еньше обучающих данных, чем набор CIFAR-10 и CIFAR-100.

#### **Softplus**

Еще одной разновидностью функции активации является функция softplus (f(x) = ln(1+ex))

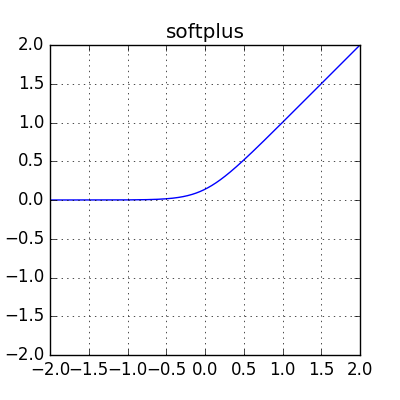


Рисунок 3.10 Функция softplus

В отличие от сигмоидальной функции и гиперболического тангенса, функция не ограничена сверху: область ее значений составляет (0, +∞). Данная функция формой напоминает ReLU, но, в отличие от него, является дифференцируемой в точке 0, что упрощает построение математических моделей нейронных сетей.

### **Обучение нейронной сети**

Под искусственными нейронными сетями понимают класс методов для решения определенных практических задач, среди которых главными являются задачи распознавания образов, принятия решений, аппроксимации и сжатия данных, а также наиболее интересные для нас задачи кластерного анализа и прогнозирования.

При любых обстоятельствах именно способность нейронной сети к обучению (с учителем или «самостоятельно») и является ключевым моментом использования ее для решения практических задач.

В общем случае, обучение ИНС заключается в следующем:

1. входные нейроны принимают переменные («стимулы») из внешней среды;
2. в соответствии с полученной информацией изменяются свободные параметры НС (работают промежуточные слои нейронов);
3. в результате изменений в структуре НС сеть «реагирует» на информацию уже иным образом.

Очевидно, что универсального алгоритма обучения не существует и, скорее всего, существовать не может; концептуально подходы к обучению делятся на обучение с учителем и обучение без учителя. Первый алгоритм предполагает, что для каждого входного («обучающегося») вектора существует требуемое значение выходного («целевого») вектора – таким образом, два этих значения образуют обучающую пару, а вся совокупность таких пар – обучающее множество. В случае варианта обучения без учителя обучающее множество состоит лишь из входных векторов – и такая ситуация является более правдоподобной с точки зрения реальной жизни.

#### **Глубокое обучение**

Понятие глубокого обучения (deep learning) относится к другой классификации и обозначает подход к обучению так называемых глубоких структур, к которым можно отнести многоуровневые нейронные сети. Простой пример из области распознавания образов: необходимо научить машину выделять все более абстрактные признаки в терминах других абстрактных признаков, то есть определить зависимость между выражением всего лица, глаз и рта и, в конечном итоге, скопления цветных пикселов математически. Таким образом, в глубокой нейронной сети за каждый уровень признаков отвечает свой слой; понятно, что для обучения такой «махины» необходим соответствующий опыт исследователей и уровень аппаратного обеспечения. Условия сложились в пользу глубокого обучения НС только к 2006 году – и спустя восемь лет можно говорить о революции, которую произвел этот подход в машинном обучении.

Итак, прежде всего стоит заметить следующее: глубокое обучение в большинстве случае не контролируется человеком. То есть этот подход подразумевает обучение нейронной сети без учителя. Это и есть главное преимущество «глубокого» подхода: машинное обучение с учителем, особенно в случае глубоких структур, требует колоссальных временных – и трудовых – затрат. Глубокое же обучение – подход, моделирующий человеческое абстрактное мышление (или, по крайней мере, представляет собой попытку приблизиться к нему), а не использующий его.

### **Инициализация нейронных сетей**

#### **Инициализация нулями**

В идеальном случае при правильной нормализации данных логично предположить, что примерно половина весов будет иметь положительные значения, а другая половина – отрицательные. Далее нам может показаться, что рационально будет задать все исходные веса равными нулю. Однако это суждение ошибочно. В этом случае все нейроны вычислят одинаковые выходы, следовательно, далее они также вычислят одинаковые градиенты в процессе обратного распространения ошибки и, соответственно, обновление параметров также будет одинаковым. Другими словами, если веса всех нейронов исходно будут одинаковыми, у нас не будет необходимого источника асимметрии между нейронами.

#### **Инициализация малыми случайными значениями**

Таким образом, нам нужно, чтобы веса были близки к нулю, но не равны ему. Для этого мы можем инициализировать их малыми случайными значениями очень близкими к нулю, что позволит нарушить симметрию. В результате, все исходные веса будут случайными и уникальными, следовательно, обновляться они будут по-разному, что нам и нужно. Вычислить веса можно следующим образом:

weights ~ 0,001 × N(0, 1)

где N(0, 1) – нормальное распределение с математическим ожиданием, равным 0, и среднеквадратическим отклонением, равным 1. Кроме того, можно использовать малые случайные значения из равномерного распределения, но на практике этот подход не оказывает существенного влияния на результат.

#### **Калибровка дисперсии**

Проблема описанного выше подхода заключается в том, что дисперсия распределения выхода нейрона, инициализированного случайным образом, возрастает с увеличением количества входов. Мы можем нормализовать дисперсию выхода каждого нейрона, разделив вектор весов на квадратный корень из количества входов:

*>>> w = np.random.randn(n) / sqrt(n)*

где функция randn() генерирует случайные числа из упомянутого выше нормального распределения, а n является количеством входов. Благодаря этому подходу, все нейроны сети исходно имеют приблизительно одинаковое выходное распределение, что позволяет повысить скорость сходимости. Подробный вывод этой формулы можно найти на страницах 18 – 23 слайдов. Обратите внимание, в выводе формулы не учитывается влияние ReLU-нейронов.

#### **ReLU - нейроны**

Как уже было сказано, предыдущий метод инициализации с применением калибровки дисперсии не учитывает влияние ReLU-нейронов. В одной из недавних работ [<https://arxiv.org/abs/1502.01852>.] была выведена формула инициализации, предназначенная специально для ReLU-нейронов:

*>>> w = np.random.randn(n) \* sqrt(2.0/n) # current recommendation*

### **Сверточные нейронные сети**

#### **Свертка**

Свертка — операция над парой матриц A (размера nx×ny) и B (размера mx×my), результатом которой является матрица C=A∗B размера (nx−mx+1)×(ny−my+1). Каждый элемент результата вычисляется как скалярное произведение матрицы B и некоторой подматрицы A такого же размера (подматрица определяется положением элемента в результате). То есть, .

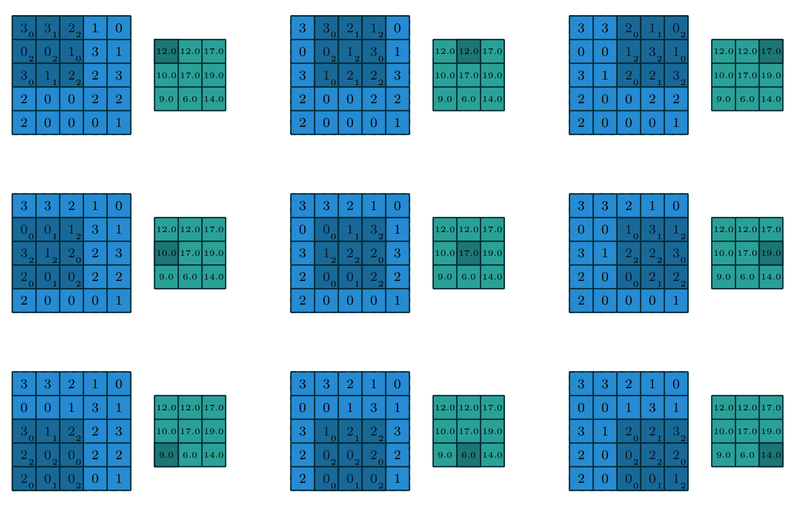


Рисунок 3.11 Пример свертки двух матриц

На рис 3.13 можно видеть, как матрица B «двигается» по матрице A, и в каждом положении считается скалярное произведение матрицы B и той части матрицы A, на которую она сейчас наложена. Получившееся число записывается в соответствующий элемент результата.

Логический смысл свертки такой — чем больше величина элемента свертки, тем больше эта часть матрицы A была похожа на матрицу B (похожа в смысле скалярного произведения). Поэтому матрицу A называют изображением, а матрицу B — фильтром или образцом.

В сверточной нейронной сети выходы промежуточных слоев образуют матрицу (изображение) или набор матриц (несколько слоёв изображения). Так, например, на вход сверточной нейронной сети можно подавать три слоя изображения (R-, G-, B-каналы изображения). Основными видами слоев в сверточной нейронной сети являются сверточные слои (англ. convolutional layer), пулинговые слои (англ. pooling layer) и полносвязные слои (англ. fully-connected layer/ dense layer).

#### **Сверточные слои**

Сверточный слой нейронной сети представляет из себя применение операции свертки к выходам с предыдущего слоя, где веса ядра свертки являются обучаемыми параметрами. Еще один обучаемый вес используется в качестве константного сдвига. При этом есть несколько важных деталей.

В одном сверточном слое может быть несколько сверток. В этом случае для каждой свертки на выходе получится своё изображение. Например, если вход имел размерность w×h, а в слое было n сверток с ядром размерности kx×ky, то выход будет иметь размерность n×(w−kx+1)×(h−ky+1).

Ядра свертки могут быть трёхмерными. Свертка трехмерного входа с трехмерным ядром происходит аналогично, просто скалярное произведение считается еще и по всем слоям изображения. Например, для усреднения информации о цветах исходного изображения, на первом слое можно использовать свертку размерности 3×w×h. На выходе такого слоя будет уже одно изображение (вместо трёх).

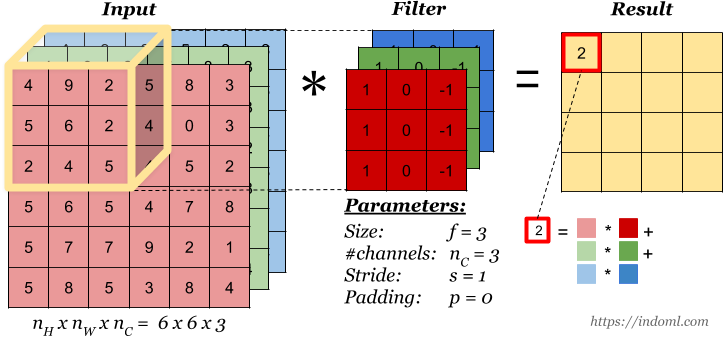


Рисунок 3.12 Пример свертки с трехмерным ядром

Можно заметить, что применение операции свертки уменьшает изображение. Также пиксели, которые находятся на границе изображения участвуют в меньшем количестве сверток, чем внутренние. В связи с этим в сверточных слоях используется дополнение изображени. Выходы с предыдущего слоя дополняются пикселями так, чтобы после свертки сохранился размер изображения. Такие свертки называют одинаковыми, а свертки без дополнения изображения называются правильными. Среди способов, которыми можно заполнить новые пиксели, можно выделить следующие:

zero shift: 00[ABC]00;

border extension: AA[ABC]CC;

mirror shift: BA[ABC]CB;

cyclic shift: BC[ABC]AB.

Еще одним параметром сверточного слоя является сдвиг. Хоть обычно свертка применяется подряд для каждого пикселя, иногда используется сдвиг, отличный от единицы — скалярное произведение считается не со всеми возможными положениями ядра, а только с положениями, кратными некоторому сдвигу s. Тогда, если если вход имел размерность w×h, а ядро свертки имело размерность kx×ky и использовался сдвиг s, то выход будет иметь размерность  .

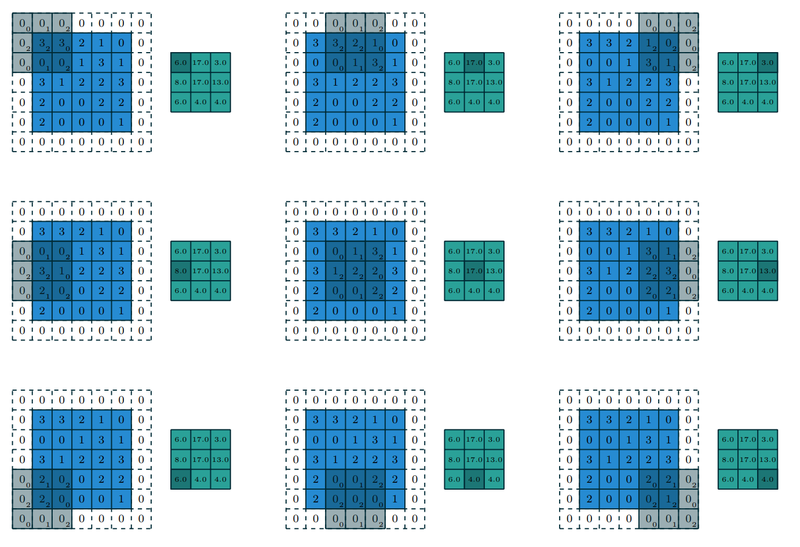


Рисунок 3.13 Пример свертки двух матриц с дополнением нулями и сдвигом 2

#### **Пулинговые слои**

Пулинговый слой призван снижать размерность изображения. Исходное изображение делится на блоки размером w×h и для каждого блока вычисляется некоторая функция. Чаще всего используется функция максимума или взвешенного среднего. Обучаемых параметров у этого слоя нет. Основные цели пулингового слоя:

* уменьшение изображения, чтобы последующие свертки оперировали над большей областью исходного изображения;
* увеличение инвариантности выхода сети по отношению к малому переносу входа;
* ускорение вычислений



Рисунок 3.14 Пример операции пулинга с функцией максимума

### **Регуляризация**

Существует несколько методов контроля емкости нейронной сети, позволяющих предотвратить переобучение.

#### **L1-регуляризация**

L1-регуляризация является еще одним распространенным методом регуляризации. В рамках этого метода для каждого веса ω мы прибавляем к целевой функции слагаемое λ|ω|.

Применяется также комбинация L1- и L2-регуляризации: λ1|ω| + λ2ω2 Этот метод имеет название эластичная сеть (elastic net).

L1-регуляризация имеет интересное свойство, заключающееся в том, что в ее результате векторы весов становятся разреженными (т.е. очень близкими к нулю). Другими словами, нейроны с L1-регуляризацией в итоге используют только небольшое подмножество наиболее важных входов и, соответственно, почти не подвержены влиянию «шумных» входов.

На практике, если нет необходимости в непосредственном отборе признаков, L2-регуляризация обеспечит лучший результат по сравнению с L1-регуляризацией.

#### **L2-регуляризация**

L2-регуляризация, вероятно, является наиболее распространенным методом регуляризации. Данный метод штрафует модель с помощью квадратов весов. То есть, для каждого веса ω мы прибавляем к целевой функции слагаемое ½λω2 , где λ – коэффициент регуляризации. Множитель ½ используется для того, чтобы градиент этого слагаемого по параметру ω равнялся λω, а не 2λω. Интуитивная интерпретация L2-регуляризации заключается в том, что она сильно штрафует векторы весов с большими значениями, и слабо затрагивает векторы с умеренными значениями.

#### **Ограничение нормы вектора весов**

Еще одним методом регуляризации является метод ограничения нормы вектора весов (max norm constraint). В рамках данного метода мы задаем абсолютный верхний предел для нормы вектора весов каждого нейрона. Соблюдение ограничения обеспечивается с помощью проецируемого градиентного спуска (projected gradient descent). На практике это реализуется следующим образом: обновление весов выполняется как обычно, а затем вектор весов ω каждого нейрона ограничивается так, чтобы выполнялось условие ||ω||2 < c. Обычно значение c составляет порядка 3 или 4. Некоторые исследователи сообщают о положительном эффекте при использовании данного метода регуляризации. Одно из полезных свойств этого метода заключается в том, что он позволяет предотвратить «взрывной» рост весов даже при слишком большой скорости обучения, потому что обновления весов всегда ограничены.

#### **Дропаут**

Дропаут (dropout) – простой и очень эффективный метод регуляризации, дополняющий вышеназванные методы. Суть метода состоит в том, что в процессе обучения из общей сети случайным образом многократно выделяется подсеть, и обновление весов выполняется только в рамках этой подсети. Нейроны попадают в подсеть с вероятностью p, которая называется коэффициентом дропаута. Во время тестирования дропаут не применяется, вместо этого веса умножаются на коэффициент дропаута, в результате чего можно получить усредненную оценку для ансамбля всех подсетей. На практике коэффициент дропаута p обычно выбирают равным 0,5, но его можно подобрать с помощью валидационного набора данных.

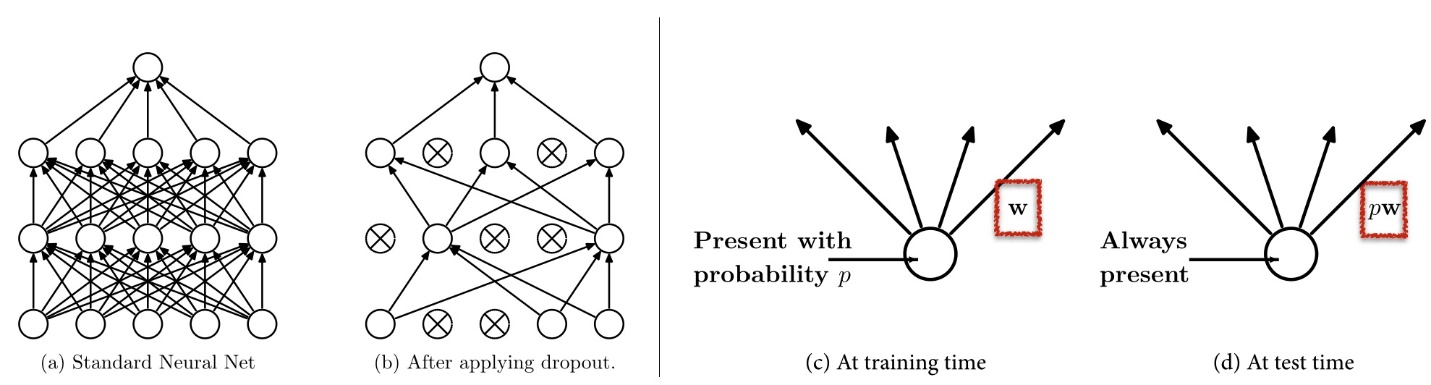


Рисунок 3.15 Dropout

### Выбранные параметры для сравнения

1. Максимальная высота фильтров
   1. 3
   2. 4
   3. 5
2. Количество сверточных слоев
   1. 3
   2. 6
   3. 10
3. Высота плотного слоя
   1. 15
   2. 30
   3. 60
4. Функция активации
   1. Tanh (гиперболический тангенс)
   2. Sigmoid
   3. Hard\_sigmoid (упрощенная сигмоида)
   4. ReLU (rectified linear unit)
5. Dropout-регуляризация
   1. 0 (без регуляризации)
   2. 0.1
   3. 0.2