МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«КРЫМСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ имени
В. И. ВЕРНАДСКОГО»
(ФГАОУ ВО «КФУ им. В. И. Вернадского»)
Таврическая академия (структурное подразделение)
Факультет математики и информатики
Кафедра информатики

Козенко Валерий Сергеевич

# СРАВНИТЕЛЬНЫЙ АНАЛИЗ БИБЛИОТЕК ГЛУБОКОГО ОБУЧЕНИЯ.

Курсовой проект

обучающегося 3 курса

направления подготовки 01.03.02. Прикладная математика и информатика

форма обучения очная

Научный руководитель
доцент кафедры информатики,
кандидат физико-математических наук А. С. Анафиев.

Оценка руководителя: _____ _____

Симферополь, 2020

## Оглавление

Введение 3

1. История машинного обучения 4

2. Что такое машинное обучение и какие задачи оно решает 7

2. 1. Обучение с учителем 7

2. 2. Обучение без учителя 8

3. Математическая составляющая машинного обучения 10

3. 1. Softmax 10

3. 2. Принцип максимального правдоподобия 11

3. 3. Negative log likelihood 11

3. 4. Регуляризация (regularisation) 12

3. 5. Градиентный спуск (gradient descent) 12

3. 6. Стохастический градиентный спуск (Stohastic gradient descent) 13

3. 7. Улучшения 14

3. 8. Граф вычислений (computational graph) и обратное распространение ошибки (back propogation). 16

4. Сравнение библиотек глубокого обучения TensorFlow, Keras, PyTorch. 18

4.1. Теоретическое сравнение. 18

4.2. Программная реализация. 21

Заключение 22

Список использованных источников 23

Приложения 24

**Введение**

Сейчас у всех на слуху такие выражения, как «искусственный интеллект», «глубокое обучение», «машинное обучение». Возникают вопросы: что стоит за этими словами? Как это работает? На самом ли деле это тот искусственный интеллект, который восхваляют, либо которого бояться множество писателей-фантастов? Как выглядит работа людей, занимающихся этой областью знаний, областью компьютерных наук?

Целью данной работы является анализ библиотек глубокого обучения. В ней мы постараемся кратко осветить историю машинного обучения, основные математические идеи, стоящие за ним, и разберем, какими инструментами пользуются дата саентисты, работающие с нейросетями, сравним эти инструменты.

## 1. История машинного обучения

Считают [1], что родителями науки об искусственном интеллекте являются Джон Маккарти (John McCarthy), Марвин Минский (Marvin Minsky), Натаниэль Рочестер(Nathaniel Rochester) и Клод Шеннон (Claude Shannon), которые в 1956 году организовали Дартмутский семинар. Джон Маккарти в грантозаявке на этот семинар писал: «Мы предлагаем исследование искусственного интеллекта сроком на два месяца с участием десяти человек летом 1956 года в Дартмутском колледже, Гановер, Нью Гемпшир. Исследование основано на предположении, что всякий аспект обучения или любое другое свойство интеллекта может в принципе быть столь точно описано, что машина сможет его имитировать. Мы попытаемся понять, как обучить машины использовать естественные языки, формировать абстракции и концепции, решать задачи, сейчас подвластные только людям, и улучшать самих себя. Мы считаем, что существенное продвижение в одной или более из этих проблем вполне возможно, если специально подобранная группа ученых будет работать над этим в течение лета». Довольно амбициозная затея, которая, к сожалению, не увенчалась успехом. Однако эта неудача не послужила основой для угасания интереса к этой области и прекращению работы в ней.

Следующими шагами в этой области считают попытки построить программы, строящие выводы в заданных исследователями формальных системах. Например, в 1959 году появилась программа под названием «Универсальный решатель задач» (General Problem Solver, G. P. S.). Она могла строить логические выводы в системах, заданных формулами специального вида (формулами из хроновских дизъюнктов). Другие программы того времени решали задачи в более узких областях, например словесные алгебраические задачи и т.п.

Так же в 1960-х был изобретен перцептрон Розенблата, - первая искусственная нейронная сеть.

С изобретением перцептрона, людям стало казаться, что недалеко и изобретение ИИ. Однако эти виденья развеялись, как и надежды людей на нейросети, с провалом крупного проекта по машинному переводу. (этот проект в то время невозможно было осуществить с должным качеством в силу нехватки производительности). К слову, люди только недавно достигли более менее удовлетворительных результатов в этой области [11].

Позже, в 1970-х, люди отошли от работы с нейросетями, в силу предшествующих провалов и крупное развитие получили системы, основанные на знаниях (knowledge-based-system), которые на данные момент являются важной части науки об экспертных системах. Суть работы этих систем: накапливается достаточно большой набор знаний о предметной области, а затем, на их основе, делаются выводы. Ярким примером такой системы является MYCIN, идентифицирующая бактерий, которые вызывают серьезные инфекции, а затем назначающая антибиотики. В ней примерно 600 правил, она выдавала правильный результат в 69 % случаев, а это не хуже, чем у опытных врачей. Такие системы могли объяснить, каким образом они пришли к своему выводу, оценить уверенность в своем решении. Они стали прообразами графических вероятностных моделей.

В 1980-х люди вернулись к нейронным сетям. Был разработан метод обратного распространения ошибки (подробнее о методе в разделе 3. 8.), что поспособствовало скачку в разнообразии архитектур нейросетей. Люди ушли от идеи создания аксиоматичексих формальных систем: их внимание привлекла идея создания больших ансамблей параллельно работающих нейронов, которые, как и человеческий мозг, сами разберутся, что им делать. Впоследствии в то время было создано большинство классических архитектур нейросетей: сверточные сети, автокодировщики, рекуррентные сети. Как видим, в это время был второй взрыв популярности нейросетей, ИИ. Однако и он закончился: появилось много стартапов, раздающих дюжины заоблачных обещаний и, в итоге, не выполнивших их и, в конце концов, лопнувших. В результате чего интерес к нейросетям опять угас и стали развиваться другие методы машинного обучения.

Итак, в 1990-х люди потеряли былой интерес к нейросетям и стали больше увлекаться машинным обучением и поиском закономерностей в данных. Однако ракурс снова поменялся в 2000-х: активно развивался интернет, повышался объем доступной информации, мощности вычислительных машин росли, и в итоге в 2005-2006 годах группы исследователей из университета Торонто (под руководством Джеффри Хинтона (Geoffrey Hinton)) и из университета Монреаля (под руководством Йошуа Бенджи (Yoshua Bengio)) научились обучать глубокие нейронные сети. Первым громким результатом в этой области было значительное улучшение результатов в распознавании речи, полученное группой Хилтона. С того момента началась современная революция в машинном обучении.

Некоторые достижения машинного обучения в классических задачах этой области (рис 1. 1, рис. 1. 2) [12].

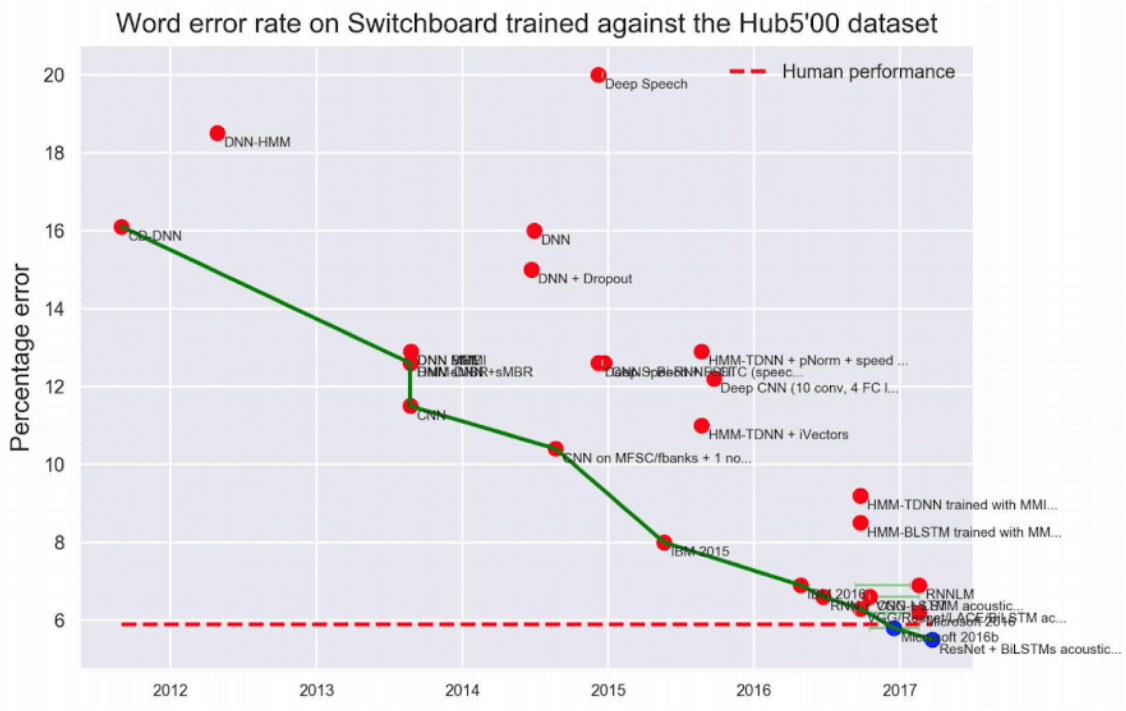


Рис. 1. 1. График зависимости процента ошибки от времени в задаче распознавания речи.

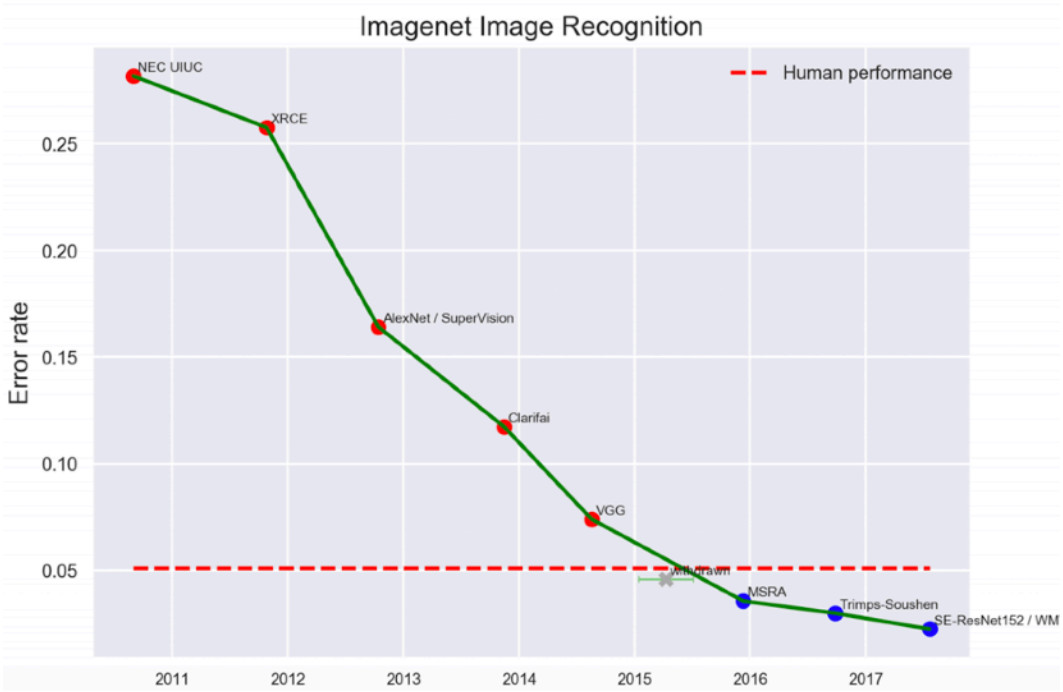


Рис. 1. 1. График зависимости процента ошибки от времени в задаче распознавания изображений.

## 2. Что такое машинное обучение и какие задачи оно решает

Выясним, что же такое машинное обучение, глубокое обучение и ИИ. Искусственным интеллектом называют ветвь компьютерных наук, которая занимается симуляцией интеллектуального поведения [7].

Машинное обучение - это ветвь компьютерных наук, которая дает компьютерам возможность обучаться без прямого программирования [7].

Глубокое обучение - совокупность методов машинного обучения, основанных на обучении представлениям, а не специализированным алгоритмам под конкретные задачи [7].

Компьютерная программа обучается по мере накопления опыта относительно некоторого класса задач T и целевой функции P, если качество решения этих задач (относительно P) улучшается с получением нового опыта [1].

Итак, задачи машинного обучения делят на два класса обучение с учителем (supervised learning) и обучение без учителя (unsupervised learning) (рис. 2. 3).

### 2. 1. Обучение с учителем

На вход подается размеченный набор тренировочных данных (training set) и суть задачи в том, чтобы продолжить уже известные ответы на новый опыт, представляемый обычно в виде тестового набора данных (training set). Важным предположением является то, что тренировочные данные чем-то похожи на те, на которых будет применяться уже обученная модель.

Задачи обучения с учителем обычно представляют собой задачи классификации и регрессии.

Более подробно об обучении с учителем на задаче классификации. У нас есть некоторый набор данных, например картинки или звуки. Для начала нам надо представить их в виде, удобном для обработки компьютером (feature extraction) - в виде некоторого числового вектора признаков (features), который будет характеризовать каждый элемент (sample) нашего набора данных (рис. 2. 1). Есть люди специализирующиеся именно на подборе этих признаков для определённого набора данных. После этого каждому элементу входных данных мы сопоставляем некоторые метку (label) - класс к которому относится этот элемент. Далее мы эти данные (признаки и метки) подаем на вход некоторому алгоритму машинного обучения, который, в свою очередь, дает нам на выход некоторую модель. И уже этой модели на вход мы подаем какие-то новые данные, которых не было в тренировочных, а она делает предсказание метки: к какому классу она относит эти новые данные.

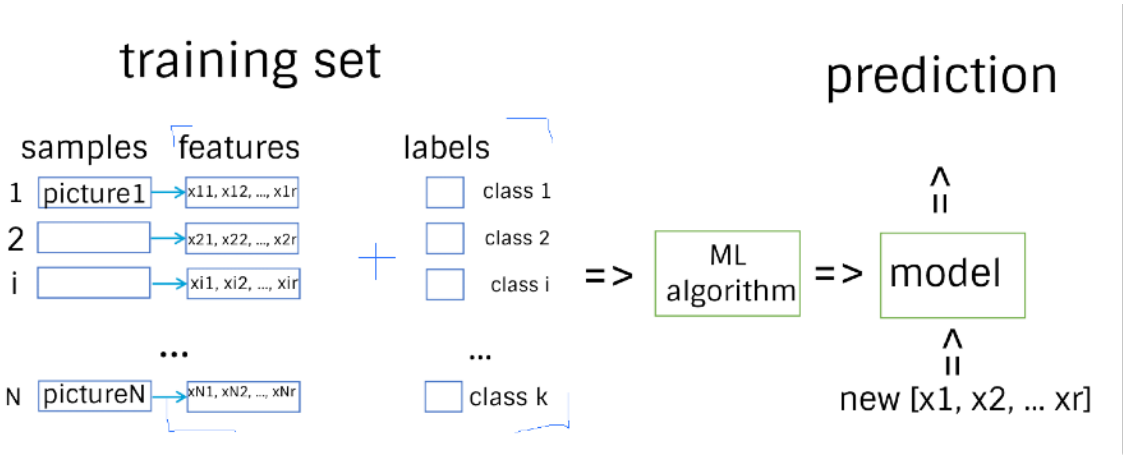


Рис. 2. 1. Обучение с учителем.

В задаче регрессии обученная модель должна предсказать некоторое значение функции.

Также есть иные виды задач, например, задача обучения ранжирования (learning torank): по имеющимся данным (в поисковой системе это будут тексты документов и прошлое поведение пользователей) отранжировать, расставить имеющиеся объекты в порядке убывания целевой функции (в поисковой системе она называется релевантностью). Эта задача чем-то похожа на задачу регрессии - нам так или иначе нужно предсказывать непрерывную целевую функцию, ту самую релевантность. Но при этом значений самой функции в данных совсем нет, да они нам и не важны. Имеют значение только результаты сравнения этой функции на разных объектах (какой документ будет выше другого в поисковой выдаче).

### 2. 2. Обучение без учителя

В задачах обучения без учителя на вход алгоритму машинного обучения подается тот же преобразованный набор данных только без меток и алгоритм должен «найти какой-либо смысл» (рис. 2. 2). Задачами обучения без учителя являются: задача класстеризации - когда в данных по некоторой мере похожести нужно выделить классы, которые заранее не известны, и точки, относящиеся к одному классу, должны располагаться как можно ближе к друг другу и быть как можно дальше от точек других классов; задача снижения размерности - когда входные данные имеют очень большую размерность и нам надо ее снизить без особой потери характерности; задача поиск аномалий - поиск в наборе данных

явно не вписывающихся в него элементов; самый общий класс задач обучения без учителя - задача оценки плотности.

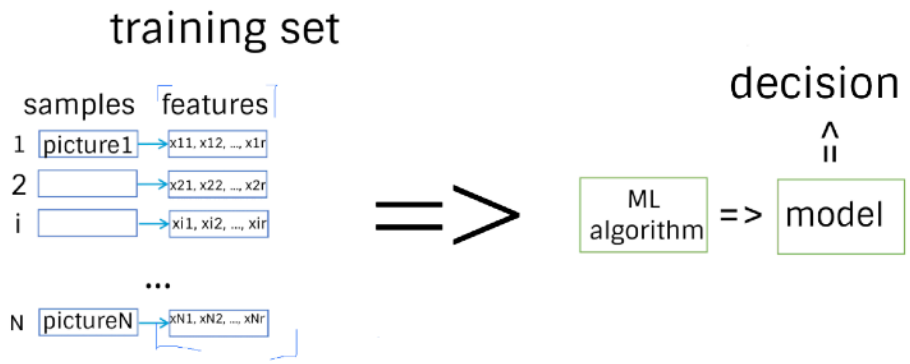


Рис. 2. 2. Обучение без учителя

Бывает, что возникает нечто среднее между обучением без учителя и обучением с учителем. Причиной этого является обилие неразмеченный данных и скупость размеченных. Такую ситуацию называют полуконтролируемым обучением (semi-supervised learning).

Еще одним способом машинного обучения является обучение с подкреплением (reinforcement learning), - когда агент, находясь в некоей среде, производит те или иные действия и получает за это награды. Цель агента - получить как можно большую награду с течением времени, а для этого неплохо бы понять, какие действия в конечном счете ведут к успеху. Так можно обучиться играть в разные игры или, например, сделать отличный протокол для A/B-тестирования.


Рис. 2.3. Общая классификация постановок задач машинного обучения.

### 3. Математическая составляющая машинного обучения

Рассмотрим некоторые базовые математические понятия на основе модели классификации картинок.

Допустим у нас есть такой алгоритм классификации, на вход которому подаются пиксели, мы этот вектор пикселей как-то линейно преобразуем матрицей весов W, и добавляем некоторое смещение b, а выходом модели является n-мерный вектор, где n - количество возможных классов (линейный классификатор) (рис. 3. 1).

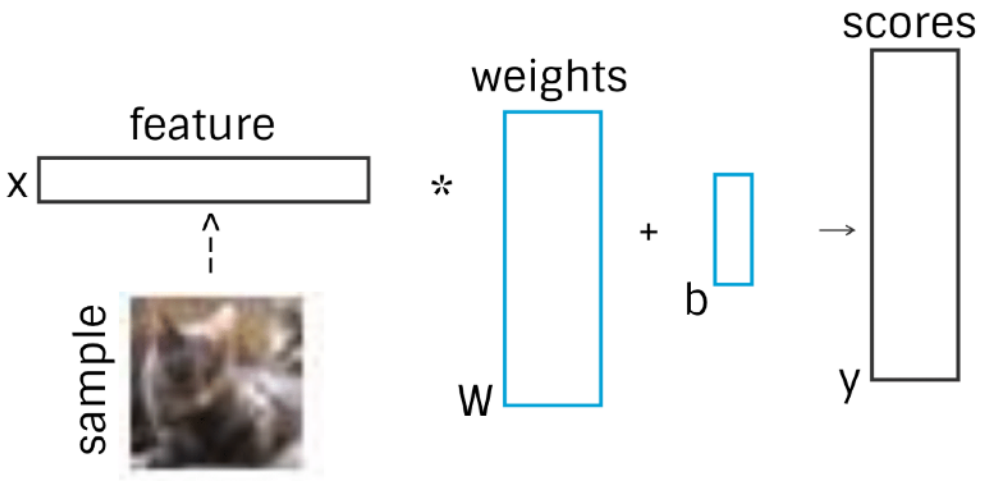


Рис. 3.1. Линейный классификатор.

$$y_i = \bar{w}_i \bar{x} + b_i \qquad (3.\ 1)$$

Каждая из компонент выходного вектора - это число $\in [-\infty, +\infty]$, максимальное из этих чисел указывает тот класс, к которому наша модель относит входящую картинку. Однако не совсем понятно, как интерпретировать эти выходящие числа. Решает этот вопрос функция Softmax.

### 3. 1. Softmax

Softmax основывается он на таком допущении:

$$\begin{aligned} P(x \mid C = 0) &\sim N(\mu_0, \sigma) \\ P(x \mid C = 1) &\sim N(\mu_1, \sigma) \\ &\dots \\ P(x \mid C = K) &\sim N(\mu_K, \sigma) \end{aligned} \qquad (3.\ 2)$$

где $K$ - количество классов. То есть распределние sample-ов по каждому классу - нормальное, при чем дисперсия у каждого распределения одинакова.

Это сильное допущение, однако на практике оно показывает хороший результат. А из формул (3. 2) следует:

$$\sigma_i = \frac{e^{y_i}}{\sum_{k=1}^{K} e^{y_k}} \quad (3.\ 3)$$

где $\sigma_i$ - i-тая компонента преобразованного вектора, $y_i$ - i-тая компонента выходящего вектора модели, $e$ - экспонента, $K$ - количество классов. Softmax располагает все компоненты выходящего вектора в сегменте [0, 1], при чем максимальный элемент выходящего вектора остается максимальным после преобразования, а сумма $\sum_{i=1}^{K} \sigma_i = 1$, таким образом, он преобразует выходящий вектор в вектор вероятностей принадлежности картинки к каждому классу. А уже с выходящим вектором в таком виде удобно работать, можно придумывать универсальные методы сравнения моделей.

### 3. 2. Принцип максимального правдоподобия

Далее встает вопрос о том, как оценить точность работы модели. В этом нам помогает так называемое правдоподобие (likelihood). Правдоподобием называется такая функция:

$$P(data) = \cap_j P(c = y_j | x_j) \quad (3.\ 4)$$

где $P(c = y_j | x_j)$ - вероятность правильного класса $y_j$, которую нам дает модель с заданными весами (W) и смещением (b) при $x_j$ на входе. Теперь встает вопрос максимизации функции правдоподобия.

### 3. 3. Negative log likelihood

Однако, так исторически сложилось, что решают не задачу максимизации правдоподобия $P(data)$, а задачу минимизации функции $-P(data)$ (negative likelihood). Также можно сделать следующее упрощение: если рассматривать не функцию правдоподобия, а ее логарифм, то правая часть в выражении (1.4) преобразуется в сумму (поскольку логарифм произведения - это сумма логарифмов). Поэтому от задачи минимизации negative likelihood переходят к задаче минимизации negative log likelihood:

$$-lnP(data) = -\sum_{j=0}^{N} lnP(c = y_j | x_j) = L \quad (3.\ 5)$$

Функцию $L$ называют cross-entropy loss. Однако с задачей минимизации функции $L$ есть некоторые проблемы.

### 3. 4. Регуляризация (regularisation)

В задаче минимизации функции $L$ не существует единственного решения: если ее полностью расписать получим:

$$L = -\sum_{j=0}^{N} ln \frac{e^{\bar{w}_j \bar{x} + b_j}}{\sum_{k=1}^{K} e^{\bar{w}_k \bar{x} + b_k}} \quad (3.\ 6)$$

заметим, что, если к $b$ в числителе и знаменателе прибавить по константе $C$, то ее можно вынести за экспоненту, как $e^C$, и мы сверху и снизу получим одинаковый множитель, который сократиться. То есть, если все $b$ сдвинуть на константу, то $L$ не изменится. Поэтому часто к $L$ добавляют некий дополнительный компонент, который обеспечивает единственность решения. Обычно этим компонентом является $\lambda R(W, b)$, где $\lambda$ - некоторый вес, а $R(W, b)$ - некоторая функция, чаще всего $R(W, b) = \|w\|_2^2 + \|b\|_2^2$. Теперь возникает вопрос: как решают такую задачу минимизации?

### 3. 5. Градиентный спуск (gradient descent)

Для минимизации функции считают ее антиградиент в данной точке (W, b), а потом пересчитывают коэффициенты по формулам:

$$W_{k+1} = W_k - \eta \nabla_W L \quad (3.\ 7)$$

$$b_{k+1} = b_k - \eta \nabla_b L \quad (3.\ 8)$$

где $\nabla_i$ - градиент по переменной i, $\eta$ - шаг итерационного процесса.

Геометрически этот процесс можно представить так - рис. 3. 2.

Градиенты в такой задаче считают либо вручную, либо методом конечных разностей.

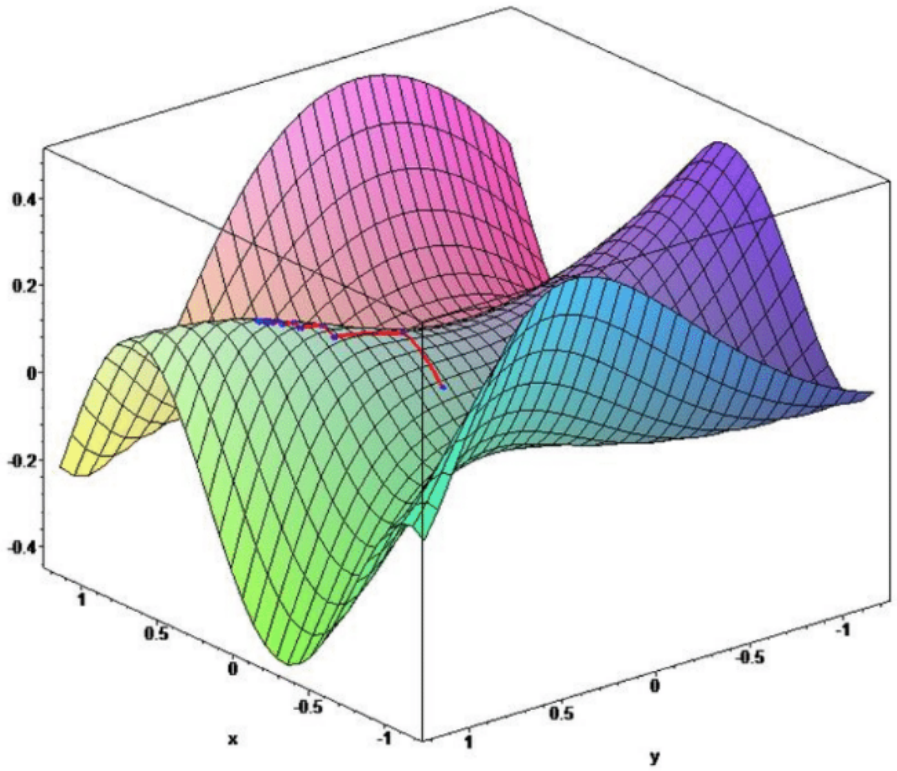


Рис. 3. 2. Градиентный спуск.

Видим, что подсчет градиента и функции ошибки для большого объема входных данных достаточно затратная операция поэтому применяют стохастический градиентный спуск.

### 3. 6. Стохастический градиентный спуск (Stohastic gradient descent)

При стохастическом градиентном спуске вместо того, чтобы считать функцию ошибки и градиент на всех данных, мы считаем их на небольшой случайной выборке (minibatch) из всех данных.

Итак процесс обучения выглядит так:

1) Инициализируем веса W, задаем точность $\varepsilon$;
2) Создаем minibatch;
3) Считаем cross-entropy loss;
4) Если ошибка меньше заданной точности, то останавливаем алгоритм, иначе переходим к шагу 4;
5) Считаем градиенты и пересчитываем веса;
6) Возвращаемся к шагу 2.

### 3. 7. Улучшения

Мы получили некоторый алгоритм машинного обучения. Возникает вопрос: можно ли как-то улучшить его? Ответ - можно:

1) в векторе features вместо пикселей придумать некоторые более характерные признаки;
2) Увеличить количество слоев.

Однако увеличивая просто количество слоев линейной трансформации мы не добьемся улучшения, поскольку все слои можно будет заменить слоем, который равен произведению матриц линейных трансформаций. Но если между слоями линейной трансформации вставить некоторое нелинейное преобразование, то наша система сможет выражать более сложные нелинейные трансформации, чем просто одно умножение на матрицу (рис. 3. 3).

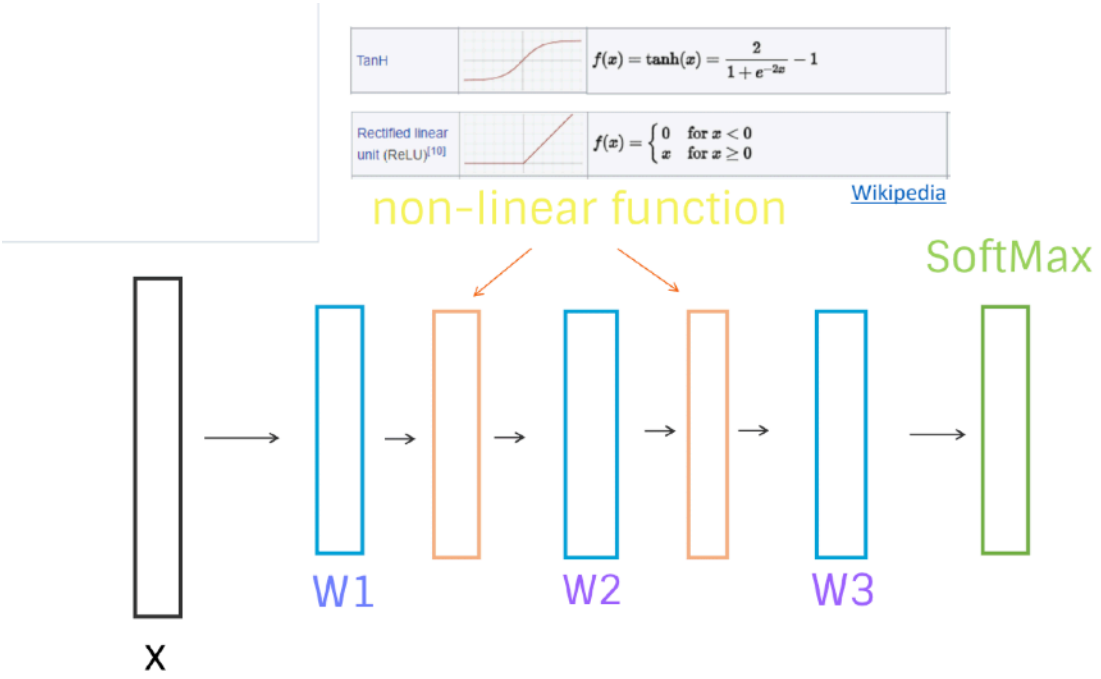


Рис. 3. 3. Добавление слоев нелинейности.

Такую систему уже можно называть нейросетью. Эти системы так называют потому, что они модулируют работу нейрона, правда очень отдельно, очень абстрактно. Можно сказать, что сама биология человека и строение нейрона послужили вдохновением для информатиков при создании этих моделей. На рис. 3. 4. приведена модель человеческого нейрона.

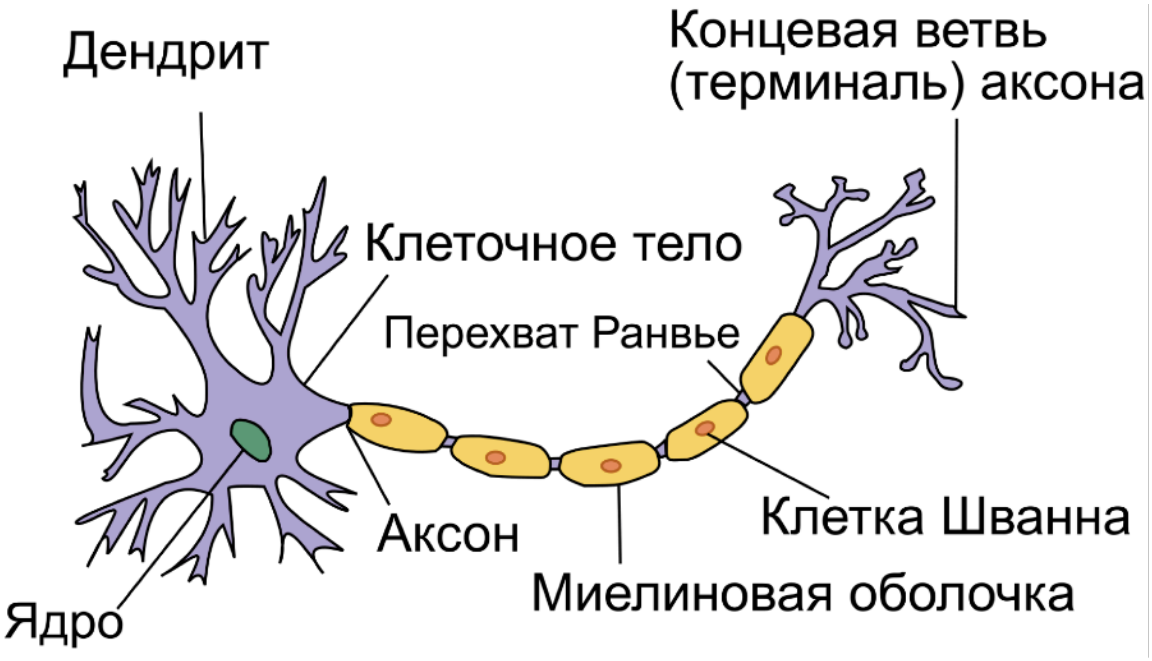


Рис. 3. 4. Типичная структура нейрона.

Очень упрощенно его работу можно описать так: нейрон по дендритам получает электрические импульсы от других нейронов, каждая из веточек дендритов имеет свою пропускную способность, т. е. пропускает сигнал не выше некоторого значения, далее сам нейрон начинает испускать электрические импульсы только если сумма всех импульсов пришедших по дендритам не ниже некоторого условленной именно для этого нейрона. Можно заметить, что система изображенная на рис. 3. 3. реализует именно эти аспекты работы нейрона: вес, на который умножаются все признаки - это модель пропускной способности дендритов, сумма всех признаков, помноженных на веса аналогична сумме импульсов на дендритах, функция нелинейности - пороговая функция самого нейрона. Существует теорема, что такая система с одним скрытым слоем может модулировать любую гладкую функцию.

До недавнего времени в виде преобразования нелинейности пользовались гиперболическим тангенсом:

$$f(x) = tanh(x) = \frac{2}{1 + e^{-2x}} - 1 \qquad (3.\ 9)$$

однако это не очень удачный выбор и сейчас самая популярная функция нелинейность - это ReLU (Rectified linear unit):

$$f(x) = \begin{cases} 0, & \text{if } x < 0 \\ x, & \text{otherwise} \end{cases} \qquad (3.\ 10)$$

Процесс тренировки нейросети выглядит точно так же, как и раньше, только теперь намного больше весов и добавилась функция нелинейности, а это накладывает трудности на вычисление градиента: теми методами, которыми мы вычисляли его до этого, это сделать либо невозможно, либо крайне затруднительно. Решить эту проблему помогает такая абстракция, как граф вычислений и метод обратного распространения ошибки.

### 3. 8. Граф вычислений (computational graph) и обратное распространение ошибки (back propogation).

Вычисление функции $L$ у системы, изображенной на рис. 3. 3. можно представить в виде графа (рис 3. 5).

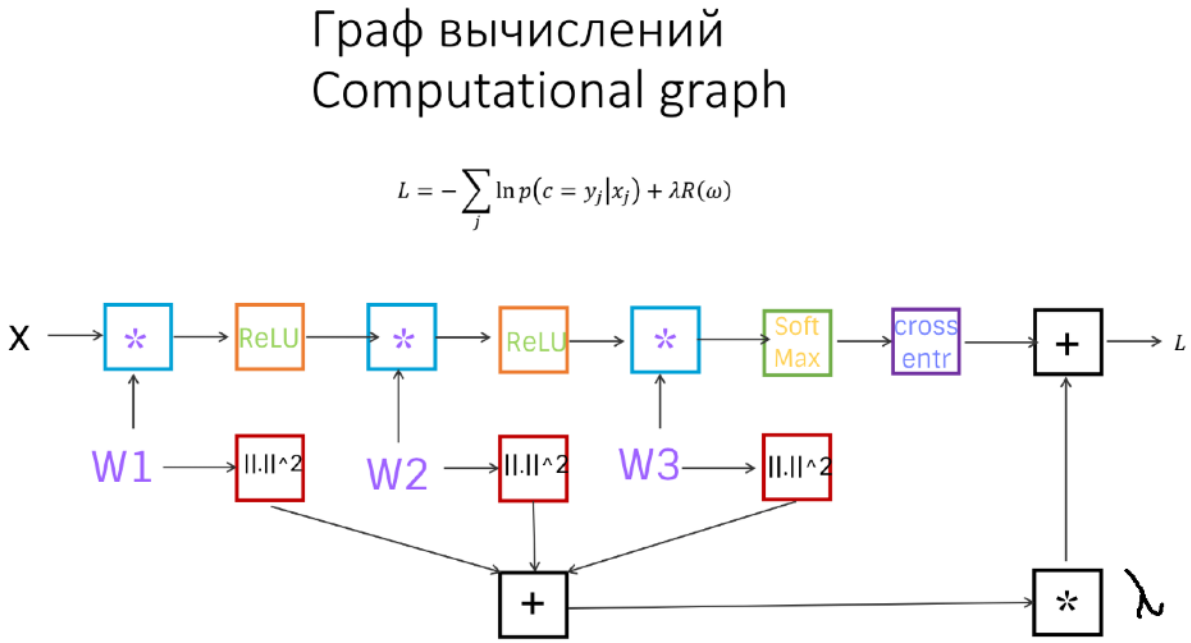


Рис. 3. 5. Граф вычислений.

Для вычисления градиентов эффективно использовать метод обратного распространения ошибки. Заключается он в том, что кроме прямого прохода по графу вычислений, осуществляется и обратный, который последовательно считает градиенты по каждому ноду, используя при этом формулу:

$$\frac{df}{dx} = \frac{df}{dz} \cdot \frac{dz}{dx} \qquad (3.\ 11)$$

Функция ошибки может быть записана в виде:

$$L = f_1(f_2(\ldots f_n(\bar{x} \cdot W_1) \cdot W_2)\ldots) \cdot W_n) \qquad (3.\ 12)$$

Первым делом вычисляется $\frac{dL}{dL}$, затем $\frac{dL}{df_1} = \frac{dL}{dL} \cdot \frac{dL}{df_1}$, затем $\frac{dL}{df_2} = \frac{dL}{df_1} \cdot \frac{df_1}{df_2}$, и так далее. Общая схема вычисления на каждой вершине графа на рис. 3. 5.

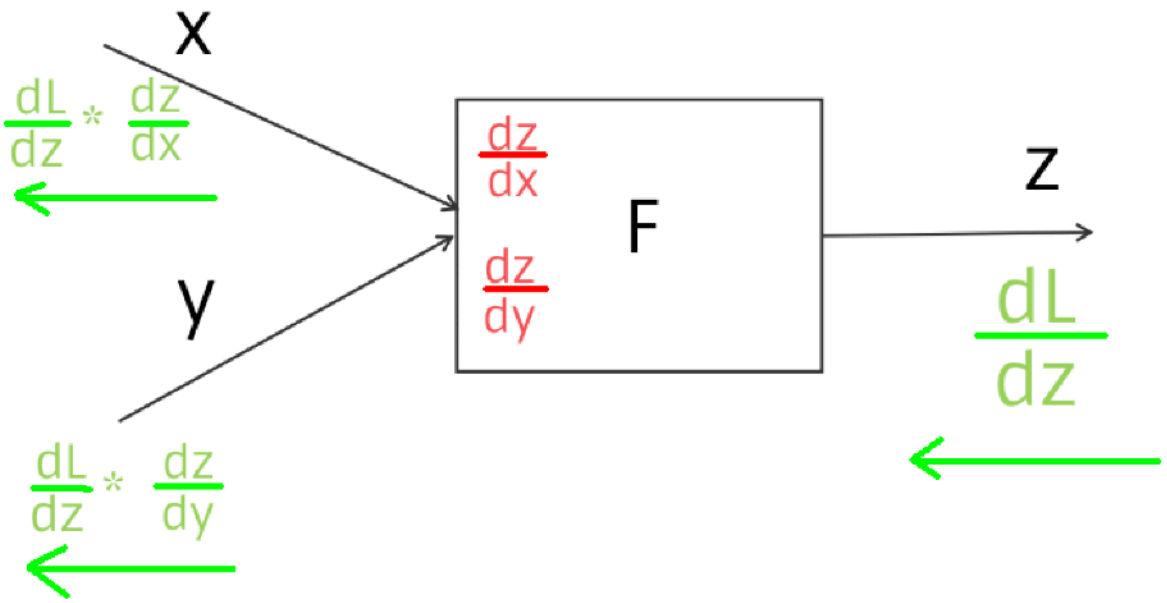


Рис. 3. 6. Общая схема вычисления градиента.

## 4. Сравнение библиотек глубокого обучения TensorFlow, Keras, PyTorch.

### 4.1. Теоретическое сравнение.

TensorFlow - открытая библиотека для машинного обучения, разработанная компанией Google, для решения задач построения и тренировки нейронных сетей. Основной API для работы с библиотекой реализован для Python также существуют реализации для C Sharp, C++, Haskell, Java, Go и Swift. Является продолжением закрытого проекта DistBelief. Изначально TensorFlow была разработана командой Google Brain для внутреннего использования в Google, в 2015 году система была переведена в свободный доступ [7].

PyTorch - библиотека машинного обучения для языка Python с открытым исходным кодом, созданная на базе Torch. Разрабатывается преимущественно группой искусственного интеллекта Facebook. Также вокруг этого фреймворка выстроена экосистема, состоящая из различных библиотек, разрабатываемых сторонними командами: Fast.ai, упрощающая процесс обучения моделей, Pyro, модуль для вероятностного программирования, от Uber, Flair, для обработки естественного языка и Catalyst, для обучения DL и RL моделей [7].

Keras - открытая нейросетевая библиотека, написанная на языке Python. Она представляет собой надстройку над фреймворками Deeplearning4j, TensorFlow и Theano. Нацелена на оперативную работу с сетями глубинного обучения, при этом спроектирована так, чтобы быть компактной, модульной и расширяемой. Она была создана как часть исследовательских усилий проекта ONEIROS (Open-ended Neuro-Electronic Intelligent Robot Operating System), а ее основным автором и поддерживающим является Франсуа Шолле (François Chollet), инженер Google. Планировалось что Google будет поддерживать Keras в основной библиотеке TensorFlow, однако Шолле выделил Keras в отдельную надстройку, так как согласно концепции Keras является скорее интерфейсом, чем сквозной системой машинного обучения. Keras предоставляет высокоуровневый, более интуитивный набор абстракций, который делает простым формирование нейронных сетей, независимо от используемой в качестве вычислительного бэкенда библиотеки научных вычислений. Microsoft работает над добавлением к Keras и низкоуровневых библиотек CNTK [7].

Вообще все современные фреймворки можно разделить на три крупных класса:

1) Фреймворки с фиксированные модулями.

В таких фреймворках пользователь комбинирует уже готовые блоки в граф вычислений и запускает его. Прямой и обратный проход уже предусмотрен в каждом блоке. Расширяемость близка к нулю: определять новые блоки гораздо сложнее, чем использовать уже готовые. Однако, если готовых блоков достаточно для вашей задачи, то скорость разработки максимальна, как и скорость работы программы (т.к. заранее написанный код высоко оптимизирован). Такие библиотеки используются для прототипирования. К ним относится Keras, а так же Caffe, Caffe2, CNTK, Kaldi, DL4J.

2) Фреймворки со статическим графом вычислений.

С помощью таких фреймворков уже можно создавать вычислительные графы произвольной сложности и размера, однако, после компиляции, в нем ничего изменить нельзя будет: он будет доступен только для прямого и обратного проходов. Из-за этого возрастает сложность отладки программы. Все такие фреймворки используют декларативный стиль программирования и напоминают функциональный язык или математическую нотацию. Представителями этого класса являются TensorFlow, а так же Theano, MXNet.

3) Фреймворки с динамическим графом вычислений.

В фреймворках данного класса граф вычислений строится при каждом запуске программы, при каждом проходе по нему, в отличии от предыдущего класса. Подобный подход даёт максимальную гибкость и расширяемость, позволяет использовать в вычислениях все возможности используемого языка программирования и не ограничивает пользователя вообще ничем. К этому классу фреймворков относятся Torch, PyTorch, Chainer, DyNet.

Сравним ярких представителей этих классов.

Видим, что у PyTorch граф вычислений динамический, у TensorFlow - статический, а у Keras - зависит от того, как интерфейс какой библиотеки мы его используем.

Из ограничений, накладываемых видом графа вычислений вытекает следующее отличие: логичность построенной программы. Сравним два кода с использованием цикла `while`:

- на TensorFlow:

```
import tensorflow as tf
```

```
x = tf.constant(2, shape=[2, 2])
w = tf.while_loop(
  lambda x: tf.reduce_sum(x) < 100,
  lambda x: tf.nn.relu(tf.square(x)),
  [x])
```

- на PyTorch:

```
import torch.nn
from torch.autograd import Variable

x = Variable(torch.ones([2, 2]) * 2)
while x.sum() < 100:
    x = torch.nn.ReLU()(x**2)
```

Циклы в графах TensorFlow следует представлять как операции `tf.while()`, принимающей на вход `condition` и подграф `body`. Так же и с операцией ветвления: она принимает в качестве ввода три параметра: условный подграф и два подграфа для двух веток развития условия: `if` и `else`. А для реализации этих операций на PyTorch нам хватает просто языка Python, и эта реализация выглядит логичнее.

Каждый из фреймворков поддерживает реализацию новых операций. Для PyTorch их можно написать на Python и на C++ [4], для TensorFlow на C++ [5], для Keras на Python [6]. Разработчики предоставляют инструкции по данным вопросам.

Специалисты придерживаются мнения, что у TensorFlow плохое качество документации, в отличии от Keras и PyTorch [9].

Связав все предыдущие параметры воедино, а так же заметив, что API TensorFlow предоставляет лишь простейшие инструкции по сборке, необходимые для создания расчетных графов, он лишен «стандартной библиотеки» для наиболее распространенных фрагментов программ. Поэтому более высокоуровневые API поверх TensorFlow реализованы сообществом, из-за чего их появилось множество и они совершенно различны (наиболее популярные из этих API: Keras, TFLearn, PrettyTensor, TF-Slim). В случае с

PyTorch все сложилось иначе: он уже оснащен самыми ходовыми элементами, нужными для ежедневных исследований в области глубокого обучения. В PyTorch есть нативный Keras-подобный API в пакете torch.nn.

Соберем результаты сравнения в таблицу.

Таблица 4. 1.

Итоги сравнения

| | **Тип графа** | **Логичность** | **Поддержка новых операций** | **Качество документации** | **Удобность API** |
|---|---|---|---|---|---|
| **PyTorch** | Динамический | Высокая | Python, C++ | Высокое | Высокая |
| **TensorFlow** | Статический | Средняя | C++ | Среднее | Средняя |
| **Keras** | Зависит от backend | Высокая | Python | Высокое | Средняя |

### 4.2. Программная реализация.

В приложениях приведены листинги программ классификации набора изображений рукописных цифр MNIST с использованием библиотеки Keras и библиотеки PyTorch.

**Заключение**

Несмотря на то, что понятия «глубокое обучение», «искусственный интеллект», «машинное обучение» только недавно стали очень популярны и широко обсуждаемы, в науке они начали разрабатываться около полувека назад. Основой этой области являются программирование и такие разделы математики: теория вероятности, методы оптимизации, математический анализ, линейная алгебра. Учеными за время существования глубокого обучения было разработано множество библиотек. Мы разделил их на три основных класса и сравнили самых ярких представителей из них. Выбирая между библиотеками, использующими статический и динамический граф стоит брать во внимание: динамизм может как оптимизировать программируемость, так и ухудшать производительность - то есть, оптимизировать такие графы сложнее. Поэтому, отличия и компромиссы между PyTorch и TensorFlow во многом такие же, как и между динамическим интерпретируемым языком, например, Python, и статическим компилируемым языком, например, C или C++. Первый проще и работать с ним быстрее, а из второго и третьего удобнее собирать сущности, хорошо поддающиеся оптимизации. Это и есть компромисс между гибкостью и производительностью. Так же, что имеет значение для разработчиков из Крыма, можно отметить, что TensorFlow ограничивает им доступ к своему сайту, в отличии от PyTorch и Keras. Итоги работы можно наблюдать в таблице 4. 1.

**Список использованных источников**


1. Николенко, С. Глубокое обучение / С. Николенко, А. Кудрин, Е. Архангельская. - СПб.: Питер, 2018. - 480 с. - (Серия «Библиотека программиста»).
2. Козлов, С. Курс Deep Learning [Электронный ресурс] / Режим доступа: https://www.youtube.com/playlist?list=PL5FkQ0AF9O_pTeRf6UjyfnsRbMyema6I3
3. Курс о Deep Learning на пальцах [Электронный ресурс] / Режим доступа: https://habr.com/ru/post/414165/
4. PyTorch [Электронный ресурс] / Режим доступа: https://pytorch.org
5. TensorFlow [Электронный ресурс] / Режим доступа: https://www.tensorflow.org
6. Keras [Электронный ресурс] / Режим доступа: https://keras.io
7. Энциклопедия Wikipedia [Электронный ресурс] / Режим доступа: ru.wikipedia.org/wiki/
8. Экскурсия по PyTorch [Электронный ресурс] / Режим доступа: https://habr.com/ru/company/piter/blog/354912/
9. Сравнение фреймворков нейронных сетей [Электронный ресурс] / Режим доступа: https://www.youtube.com/watch?v=3NoyksurUFI&t=312s
10. PyTorch - ваш новый фреймворк глубокого обучения [Электронный ресурс] / Режим доступа: https://habr.com/ru/post/334380/
11. The great A. I. Awakening [Электронный ресурс] / Режим доступа: https://www.nytimes.com/2016/12/14/magazine/the-great-ai-awakening.html
12. AI Progress Measurement [Электронный ресурс] / Режим доступа: https://www.eff.org/ai/metrics

**Приложения**

Листинг программы на Keras:

```
import tensorflow as  tf
from tensorflow import keras
import numpy as np
import matplotlib.pyplot as plt
mnist = keras.datasets.mnist
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
train_images = train_images / 255.0
test_images = test_images / 255.0
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(128, activation=tf.nn.relu),
    keras.layers.Dense(10, activation=tf.nn.softmax)
])
model.compile(optimizer=tf.compat.v1.train.AdamOptimizer(),
              loss='sparse_categorical_crossentropy',
              metrics=[‘accuracy'])
model.fit(train_images, train_labels, epochs=5)
test_loss, test_acc = model.evaluate(test_images, test_labels)
print('Test accuracy:', test_acc)
```

Листинг программы на PyTorch:

```
import torch
import torchvision
from torchvision import transforms, datasets
import torch.nn as nn
import torch.nn.functional as F
```

```
import torch.optim as optim


train = datasets.MNIST('', train=True, download=True,
                       transform=transforms.Compose([
                           transforms.ToTensor()
                       ]))
test = datasets.MNIST('', train=False, download=True,
                       transform=transforms.Compose([
                           transforms.ToTensor()
                       ]))
trainset = torch.utils.data.DataLoader(train, batch_size=10, shuffle=True)
testset = torch.utils.data.DataLoader(test, batch_size=10, shuffle=False)
class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc1 = nn.Linear(28*28, 128)
        self.fc2 = nn.Linear(128, 10)


    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return F.log_softmax(x, dim=1)


loss_function = nn.CrossEntropyLoss()
optimizer = optim.Adam(net.parameters(), lr=0.001)
for epoch in range(5):
    for data in trainset:
        X, y = data
```

```
        net.zero_grad()
        output = net(X.view(-1,784))
        loss = F.nll_loss(output, y)
        loss.backward()
        optimizer.step()
    print(loss)
correct = 0
total = 0

with torch.no_grad():
    for data in testset:
        X, y = data
        output = net(X.view(-1,784))
        #print(output)
        for idx, i in enumerate(output):
            #print(torch.argmax(i), y[idx])
            if torch.argmax(i) == y[idx]:
                correct += 1
            total += 1

print("Accuracy: ", round(correct/total, 3))
```