**Городской округ Краснотурьинск**

**Муниципальное автономное образовательное учреждение**

**«Средняя общеобразовательная школа № 24»**

**Направление:**

**Секция:**

**Проект**

**Искусственный интеллект. Модель, проектирование, разработка.**

**Автор проекта:**

***Самонин Евгений Игоревич****, 10 класс*

**Руководитель проекта:** *Ф.И.О., должность (с кем работаете)*

*телефон, эл. почта*

**2025**

**ОГЛАВЛЕНИЕ**

**Введение…**……………………………………………………………………………3

1. Теоретическая часть………………………………………………………………5
   1. Ключевые понятия………………………………………………………………5
   2. История создания нейронных сетей……………………………………………6
   3. Устройство нейронных сетей…………………………………………………...8

**Введение**

Искусственный интеллект каждый день берет новые вершины во всех сферах человеческой жизни. Он находит применение как в бытовой сфере, так и в производстве, образовании, медицине и многих других областях. Развитие и внедрение ИИ в нашу жизнь позволяет повысить эффективность работы сотрудников и компаний, улучшить качество предоставляемых услуг и оптимизировать многие сферы производства, а также увеличить собственный комфорт и понизить нагрузку на себя в тех вопросах, где вмешательство человека не обязательно.

Проектирование и разработка искусственного интеллекта в современности — одна из наиболее быстро развивающихся и перспективных областей как бизнеса, так и науки. В реалиях скачка компьютерных технологий и глобализации ИИ становится универсальным помощником для достижения преимуществ перед конкурентами и инноваций в различных направлениях экономики.

ИИ включает в себя огромное количество технологий, включая машинное обучение, обработку языка, компьютерное зрение, робототехнику и многое другое. Эти технологии и методы позволяют создавать системы, способные анализировать большие объемы данных и оперировать ими, принимать решения и обучаться на основе полученного опыта.

Для быстрого развития и внедрения ИИ требуются квалифицированные специалисты в области математики, информатики, программирования, статистики и других наук. Но, к сожалению, сейчас мало людей понимают устройство нейронных сетей и искусственного интеллекта.

**Проблема**

Большинство людей знают об искусственном интеллекте, но не знают, как он устроен и как он создается. Так проблема этого проекта состоит в том, что при огромном влиянии ИИ на нашу жизнь, большинство людей все равно им не пользуются и не понимают всей выгоды, что он несет для них.

**Обоснованность темы**

Тема "искусственный интеллект и его интеграция в нашу жизнь" обоснована, обоснована тем, что искусственный интеллект становится все более важным элементом нашего общества. Он применяется в различных сферах и полезен в жизни обычных людей, не связанных с ИИ.

**Актуальность работы**

Эта тема актуальна в современных условиях стремительного технологического прогресса. Разработка ИИ сейчас является большой сферой экономики и науки, в последние годы развитие ИИ стало появляться в заголовках новостей почти каждый месяц. Искусственный интеллект уже сегодня меняет нашу повседневную жизнь. С его помощью решаются сложные задачи, требующие анализа больших объёмов данных и автоматизации процессов. Интеграция ИИ способствует повышению эффективности и снижению человеческих ошибок. В образовательной сфере ИИ открывает новые возможности для личного обучения и повышения качества образовательных программ. Тема проекта даёт возможность исследовать, как работает искусственный интеллект и нейронные сети.

**Задачи проекта**

* Узнать историю искусственного интеллекта.
* Понять, что такое ИИ и нейронные сети.
* Разобраться в принципах работы нейронных сетей.
* Создать свою модель ИИ.

1. **Теоретическая часть**

В этом проекте мы будем рассматривать нейронную сеть — полносвязный перцептрон. Свёрточные нейронные сети сложнее и требуют больших вычислительных мощностей, поэтому рассматриваться не будут.

* 1. **Понятие нейронной сети в современном мире**

Рассмотрим определение термина нейронной сети и ИИ.

Иску́сственный интелле́кт (ИИ) (англ. artificial intelligence; AI) в самом широком смысле — это интеллект, демонстрируемый машинами, в частности компьютерными системами. Это область исследований в области компьютерных наук, которая разрабатывает и изучает методы и программное обеспечение, позволяющие машинам воспринимать окружающую среду и использовать обучение и интеллект для выполнения действий, которые максимально увеличивают их шансы на достижение поставленных целей. Такие машины можно назвать искусственным интеллектом. ИНС представляет собой систему соединённых и взаимодействующих между собой простых процессоров (искусственных нейронов). Такие процессоры довольно просты (особенно в сравнении с процессорами, используемыми в персональных компьютерах). Каждый процессор подобной сети имеет дело только с сигналами, которые он периодически получает, и сигналами, которые он периодически посылает другим процессорам. И, тем не менее, будучи соединёнными в достаточно большую сеть с управляемым взаимодействием, такие простые по отдельности процессоры вместе способны выполнять довольно сложные задачи [1].

Нейронная сеть (также искусственная нейронная сеть, ИНС, или просто нейросеть) — математическая модель, а также её программное или аппаратное воплощение, построенная по принципу организации биологических нейронных сетей — сетей нервных клеток живого организма. Это понятие возникло при изучении процессов, протекающих в мозге, и при попытке смоделировать эти процессы. Первой такой попыткой были нейронные сети У. Маккалока и У. Питтса. После разработки алгоритмов обучения получаемые модели стали использовать в практических целях: в задачах прогнозирования, для распознавания образов, в задачах управления и др. [2].

* 1. **Ключевые для понимания определения**

СПР — сеть прямого распространения.

Искусственный нейрон или просто нейрон — это составляющая часть слоя нейронной сети, обладающая связями (синапсами) с нейронами другого слоя. В нейрон входит сумматор и числовое значение нейрона (смещение).

Синапс — связь между нейронами разных слоев ИНС, имеющая вес.

Сумматор — это часть нейрона, суммирующая сигналы, поступающие от нейронов прошлого слоя через синапсы.

Полносвязный перцептрон — искусственная нейронная сеть, состоящая из полносвязных слоев.

Гиперпараметры нейросети — это параметры, которые не меняются в процессе обучения нейросети и подбираются вручную разработчиком исходя из вычислительных мощностей и потребностей. В нейросети это количество скрытых слоев, количество нейронов в скрытых слоях.

Весы ИНС — все весы синапсов и смещения (все обучаемые характеристики)

Батч (batch) — неполная подвыборка из всего массива данных.

Стохастический градиентный спуск ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) Stochastic gradient descent, SGD) — [итерационный](https://ru.wikipedia.org/wiki/%D0%98%D1%82%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D1%8F_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) метод для [оптимизации](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D1%82%D0%B8%D0%BC%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) [целевой функции](https://ru.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F_%D0%BF%D0%BE%D1%82%D0%B5%D1%80%D1%8C) с подходящими свойствами [гладкости](https://ru.wikipedia.org/wiki/%D0%93%D0%BB%D0%B0%D0%B4%D0%BA%D0%B0%D1%8F_%D1%84%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F) (например, [дифференцируемость](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D1%84%D1%84%D0%B5%D1%80%D0%B5%D0%BD%D1%86%D0%B8%D1%80%D1%83%D0%B5%D0%BC%D0%B0%D1%8F_%D1%84%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F) или [субдифференцируемость](https://ru.wikipedia.org/wiki/%D0%A1%D1%83%D0%B1%D0%B4%D0%B8%D1%84%D1%84%D0%B5%D1%80%D0%B5%D0%BD%D1%86%D0%B8%D0%B0%D0%BB)). Его можно расценивать как [стохастическую аппроксимацию](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D0%BE%D1%85%D0%B0%D1%81%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D0%B0%D0%BF%D0%BF%D1%80%D0%BE%D0%BA%D1%81%D0%B8%D0%BC%D0%B0%D1%86%D0%B8%D1%8F) оптимизации методом [градиентного спуска](https://ru.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D0%B4%D0%B8%D0%B5%D0%BD%D1%82%D0%BD%D1%8B%D0%B9_%D1%81%D0%BF%D1%83%D1%81%D0%BA), поскольку он заменяет реальный градиент, вычисленный из полного набора данных, оценкой, вычисленной из случайно выбранного подмножества данных. Это сокращает задействованные [вычислительные ресурсы](https://ru.wikipedia.org/wiki/%D0%92%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D1%81%D0%BB%D0%BE%D0%B6%D0%BD%D0%BE%D1%81%D1%82%D1%8C) и помогает достичь более высокой скорости итераций в обмен на более низкую [скорость сходимости](https://ru.wikipedia.org/wiki/%D0%A1%D0%BA%D0%BE%D1%80%D0%BE%D1%81%D1%82%D1%8C_%D1%81%D1%85%D0%BE%D0%B4%D0%B8%D0%BC%D0%BE%D1%81%D1%82%D0%B8). Особенно большой эффект достигается в приложениях, связанных с обработкой [больших данных](https://ru.wikipedia.org/wiki/%D0%91%D0%BE%D0%BB%D1%8C%D1%88%D0%B8%D0%B5_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D0%B5) [3].

**1.2** **История создания нейронных сетей [4]**

**1.2.1 Первые разговоры**

В середине XX века начала формироваться первая концепция нейросети: исследователи пытались создать машину, имитирующую функции человеческого мозга. В 1943-м Уорреном Мак-Каллоком и Уолтером Питтсом была предложена математическая модель нейрона, а уже в конце 1950-х Фрэнк Розенблатт представил персептрон – простую модель машинного обучения, созданную для помощи компьютерам в обучении на разных объёмах данных. Именно его можно назвать первой практической реализацией нейросети.

В период 1980-2000 стали разрабатываться и появляться первые алгоритмы обучения, сравнения и анализа данных, в XXI направление стало стремительно развиваться. В 2000-х появились мощные графические процессоры, и стали доступны большие объёмы данных, что побудило сообщество начать разработку алгоритмов Deep Learning – совокупность методов машинного обучения, основанных на обучении представлениям, а не на специализированных алгоритмах под конкретные задачи.

Интерес воплощения человеческого мышления «на плате» появился почти сразу же, как вычислительная техника стала доступна большинству людей. В XX веке люди смогли довольно быстро достичь тех или иных результатов за счёт алгоритмов, которые были придуманы человеком, но не смогли продвинуть это дальше ввиду недостаточного развития технологий.

Развитие современный нейронных сетей

**1.2.2 OpenAI**

В декабре 2015 года была основана некоммерческая организация OpenAI – американская научно-исследовательская организация, занимающаяся разработками в области искусственного интеллекта.

1. 27 апреля 2016 компания выпустила публичную бета-версию OpenAI Gym, платформы для разработки и сравнения алгоритмов обучения с подкреплением

Это была своего рода разновидность задачки машинного обучения. Суть в том, что пользователю предлагалась среда с настроенными правилами и тело, способное действовать в пределах этой среды. Пользователи разрабатывали и сравнивали свои алгоритмы на этой платформе.

1. В 2017 году были представлены ИИ-боты для игры в Dota 2

Неожиданный, но проявивший себя продукт. Действительно, OpenAI смогли создать бота для популярной соревновательной игры, которые действуют не по описанному алгоритму, а принимают решение самостоятельно, основываясь на ранее загруженном дата-сете (набор данных для обучения нейросетей)

1. 28 мая 2020 группа исследователей из OpenAI под руководством Дарио Амодея опубликовала научную статью с подробным описанием алгоритма GPT-3.

На этом этапе миру был представлен принцип работы генеративной нейросети GPT-3. Презентация не навела шуму, ведь никакого готового продукта не было, была показана только общая идея. Однако, это не помешало привлечь в проект ключевые инвестиции, которые помогли выпустить инновационный продукт на рынок!

1. 30 ноября 2022 года был запущен ChatGPT - чат-бот с искусственным интеллектом, разработанный компанией OpenAI и способный работать в диалоговом режиме, поддерживающий запросы на естественных языках

Это уже всем известный ChatGPT. Лишь написав одно предложение в чат, пользователь мог получить: простой программный алгоритм, рецепт, рекомендацию, идею, совет и многое другое.

Исторический день, после которого все, кто позиционируют себя как IT гиганты, не могли не принять вызов рынка и не начать работу над своим аналогом.

1. В марте 2023 года была интегрирована более продвинутая модель – GPT- 4.

В данном релизе модель обучилась на существенно бОльшем объёме денных, научилась распознавать и работать с изображением. Качество ответов существенно улучшилось.

**1.3 Устройство нейронных сетей**

**1.3.1 Основные выполняемые нейросетями функции**

Классификация — распределение данных по классам на основе определённых параметров. Например, на вход дается набор характеристик человека и от нейросети требуется понять: давать человеку кредит или нет.

Предсказание — возможность анализировать закономерности и на их основе давать предсказания. Например, рост или падение акций, основываясь на ситуации на фондовом рынке.

Распознавание — Идентификация и детектирование объектов на изображении.

В этом проекте мы будем рассматривать устройство классифицирующей нейросети для дальнейшего ее создания.

**1.3.2 типы слоев ИНС**

Входной — нейроны, на которые изначально подается входная информация, будь это слова текста или пиксели фотографии.

Скрытые — нейроны, находящиеся между входными и выходными, слоев этих нейронов может быть сколько угодно. Пользователь никак не контактирует с этими нейронами, поэтому они и называются скрытыми. Количество скрытых нейронов во внутренних слоях может меняться.

Выходные нейроны — последний слой нейронов, дающий окончательный результат.

Рассмотрим каждый тип подробнее:

Входной слой всегда один. Входных нейронов в слое должно быть столько же, сколько у нас есть входных данных. Например, если на вход подается изображение разрешением 28 на 28 пикселей и каждый пиксель закодирован одним числом, то входных нейронов должно быть 28 \* 28 = 784. Значение входного нейрона никак не обрабатывается и передается следующим слоям неизменным.

Скрытых слоев может быть сколько угодно (гиперпараметр нейросети), количество нейронов в каждом слое тоже гиперпараметр. В классифицирующих нейросетях в идеальном случае нейроны каждого последующего слоя должны отвечать за более общую картину. Например, в нейросети, классифицирующей изображение цифры, в первом слое (входном) каждый нейрон отвечает за один пиксель, нейроны второго слоя — за отдельные черты, третьего — за отдельные элементы цифры, и после в выходном слое всё это сводится до одного ответа, какая это цифра. Нейроны скрытых слоев берут свое значение исходя из значений всех нейронов предыдущего слоя и передают его на следующий слой.

Выходной слой тоже только один. Нейронов в этом слое столько же, сколько возможных вариантов ответа. Ответом является нейрон с наибольшим значением.

**1.3.3 Как вычислить результат работы нейронной сети зная весы**

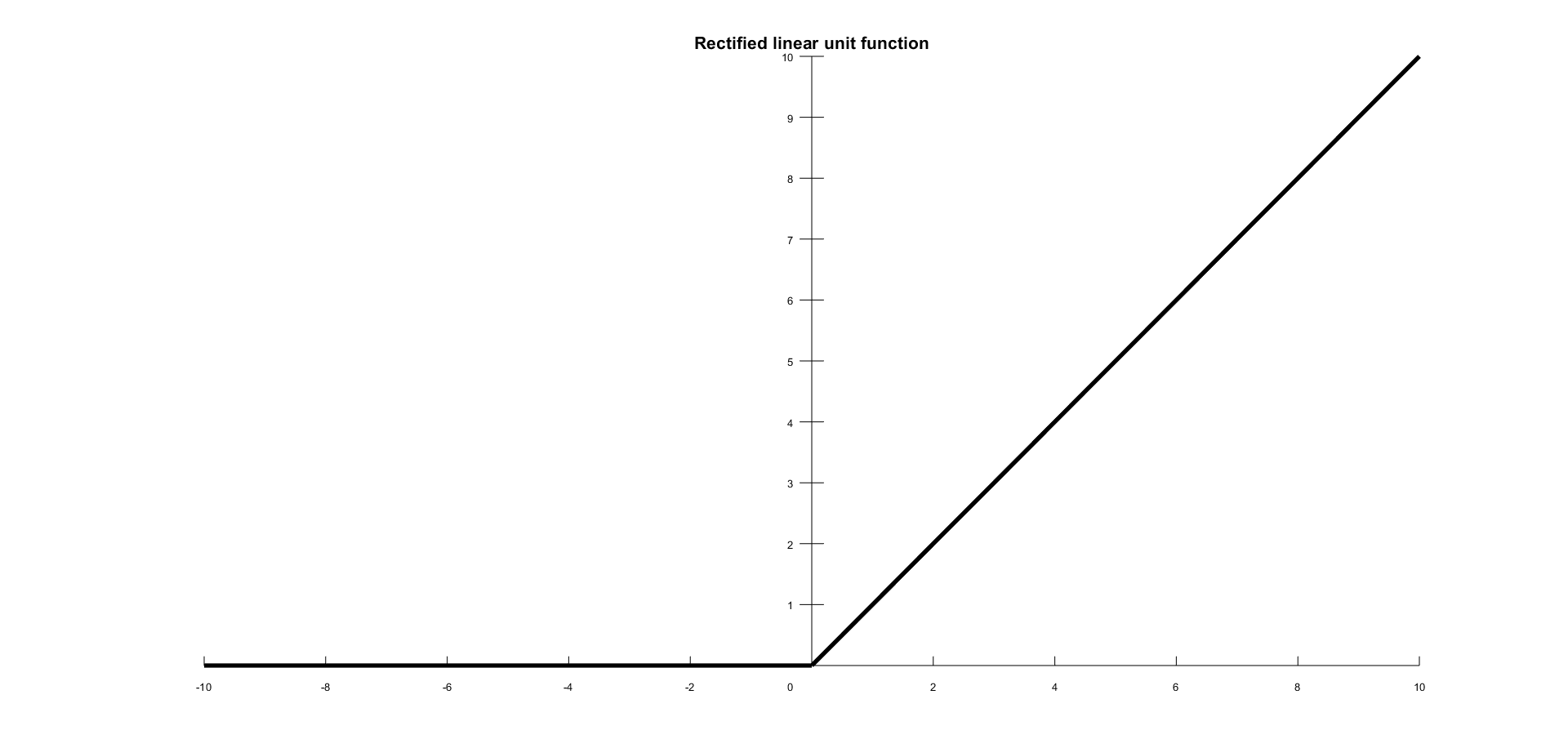
Мы поняли, какие есть слои и что примерно делают нейроны в этих слоях. Пришло время разобраться, как они это делают.

Начнем с того, как нейрон преобразует переданную ему информацию. Как мы разобрались, нейроны входного слоя ничего не делают с этой информацией, тогда нужно понять, как это делают нейроны скрытых и выходного слоев.

Синапс — это связь между двумя нейронами, которая имеет вес. У каждого нейрона одного слоя есть синапсы с каждым нейроном предыдущего и последующего слоя. И так каждый нейрон передает свое значение каждому следующему нейрону, умножая его на вес синапса конкретного нейрона. Эта информация поступает на нейрон и суммируется с также полученными значениями других нейронов, после чего к этой сумме прибавляется смещение нейрона. Дальше эту сумму нужно нормализовать, для этого и используется функция нормализации, в нашем случае мы будем использовать функцию ReLU.

ReLU (rectified linear unit) — это функция, превращающая отрицательные значения в 0, а положительные оставляет как есть.

Формула ReLU:

И ее график:

Для этого и нужно смещение, чтобы задать больше, чего должна быть сумма значений предыдущих нейронов, например, чтобы нейрон активировался, только если сумма больше 10 или больше -100.

После нормализации значения сигнал передается дальше по синапсам в следующие слои.

Рассмотрим всю эту схему на примере с двумя входными нейронами и одним скрытым нейроном. Обозначим входные нейроны I1, I2, скрытый как N, веса синапсов как W и смещение B. Тогда входными и выходными данными на нейрон будут:

*Ninput = I1\*W1 + I2\*W2*

*Noutput = ReLU(I1\*W1 + I2\*W2 + B)*

Запишем всё это в алгебраическом виде и сделаем формулы общими. Представим n-ный (отсчет идет с нуля) нейрон L-ого слоя (кроме входного и выходного) нейронной сети, обозначим ALn, нейроны предыдущего слоя AL-1k, весы синапсов как Wk, n, а смещение как Bn, а, где k— это номер нейрона предыдущего слоя, а n— это номер нейрона, которого мы взяли. Значение этого нейрона: A = ReLU (W0,0AL-10 + W1,0AL-11+ ... + Wn, 0\*AL-1k + Bn). Если рассмотреть эту формулу без смещения и ReLU поподробнее, можно понять, что это первый элемент вектора, получаумого при умножении вектора значений нейронов предыдущего слоя и матрицы весов (каждая строка — это вектор весов синапсов нейрона предыдущего слоя со всеми нейронами текущего) = W0,0\*A0 + W0,1\*A1 + ... + W0,n\*An. Тогда, чтобы получить наше значение, нужно к этому произведению прибавить вектор смещений и взять сигмоиду. И все это можно красиво записать для первого слоя: A1 = ReLU(WA0 + B) или в общем виде:

L – номер слоя, n – количество нейронов в новом слое, k – количество нейронов в предыдущем слое.

Так применяя эту формулу для каждого слоя, мы доходим до последнего, а в нем нужно использовать другую функцию. Функция Softmax преобразует данный ей вектор в вектор вероятностей, так чтобы сумма этих вероятностей была равна единице. Её формула для каждого члена вектора:

Так ответом нейросети является класс, у которого наибольшая вероятность.

**1.3.4 Обучение нейронной сети**

Нейросеть учится на данных. Эти данные должны быть заранее заготовлены и сгруппированы. Дальше эти данные делятся на два типа: тренировочные и тестовые. Тренировочные используются напрямую при тренировке. Тестовые используются для отслеживания динамики обучения и в конце для оценки эффективности работы ИНС.

Обучение нейронной сети происходит по этапам, каждый этап называется эпохой. Данные для обучения для каждой эпохи одинаковые, и эпохой и называется одна итерация обучения на базе данных. Для каждой эпохи желательно каждый раз перемешивать данные. В каждой эпохе происходят следующие действия:

1. Деление данных на батчи (размер батча — это гиперпараметр).
2. Прямое распространение, т. е. получение результатов предсказания нейросети для этих данных.
3. Обратное распространение ошибки — это способ получения вектора градиента.
4. Корректировка значений весов на основе суммарной ошибки за весь батч.

И далее те же действия по итерациям батчей и эпох.

Рассмотрим обучение способом стохастического градиентного спуска. Для осуществления градиентного спуска мы должны получить градиент функции ошибки. Функция ошибки — это функция, показывающая насколько ошиблась нейросеть. А градиент — это вектор, каждая координата, которого показывает на сколько зависима функция ошибки от значения этой переменной и, следовательно, на сколько нужно изменить соответствующий вес (следовательно, координат у него столько же сколько весов и смещений у нейросети), и направление которого указывает, как функция будет возрастать с наибольшей скоростью. Так как это градиент ошибки, а ее нам необходимо минимизировать, то мы берем этот вектор и вычитаем из наших весов его значение, соответствующий этому весу. Получить этот вектор можно с помощью обратного распространения ошибки. Рассмотрим формулы ниже.

Пусть L это номер слоя, T это транспонированная версия вектора или матрицы, WL это матрица весов между слоями L и L – 1, BL это вектор смещений данного слоя, Z то, что передается в функцию активации, F это функция активации, ⊙это поэлементное умножение, AL это вектор значений слоя L. То на сколько нужно изменить вес обозначается производной ошибки по этому весу, но мы будем рассматривать формулы относительно матриц и векторов целых слоев.

После чего становится для предыдущего слоя и все повторяется по итерациям. Но для первой итерации нужно знать последнего слоя. Для этого есть отдельная формула. Для нее введём еще несколько обозначений. Y это истинный результат, SoftMax(Zmax) = P (наш результат работы нейросети в вероятностях):

Но так как мы работаем не с единичными прогнозами, а с батчами, то формулы изменятся. Из-за свойств матричного умножения почти все формулы останутся теми же, кроме одной (сумма производной ошибки по Z за весь батч):

A так же вычислим саму ошибку для отслеживания прогресса обучения нейросети. Для этого воспользуемся функцией CrossEntropy(P, Y) = . Но так как в истинном распределении вероятности только одна единица и остальные нули и Y это номер элемента правильного ответа, от суммы останется только

После того как мы узнали, как натренировать нашу нейросеть, нужно научиться делать это в меру, так как ИНС может переобучиться. Это значит, что она слишком хорошо запомнит тренировочные данные и станет хуже работать на тестовых данных. Оптимальные гиперпараметры для конкретной нейросети подбираются в ходе ее обучения по валидационным данным.

1. **Практическая часть**

**Задача**

Реализовать полносвязный перцептрон на языке Python без использования заготовленных библиотек вроде TensorFlow, дающих возможность создавать нейронные сети без написания их логики работы. И сделать это парадигме ООП, то есть создать класс нейросети для дальнейшей возможности импортирования этого класса в другие программы и большей универсальности.

**Инструментарий**

Язык программирования Python. библиотеки: numpy (np) — для реализации матричных математических операций, Данные: база данных MNIST (60 тыс. тренировочных и 10 тыс. тестовых изображений цифр в формате 28 \* 28 пикселей).

**2.2 Код программы**

Начнем с инициализации класса. Для этого нам нужно знать путь до нейросети для ее сохранения или загрузки и, если нужно ее обучать: количество нейронов в каждом слое, путь до тренировочных и тестовых баз данных, количество эпох, размер батча и скорость тренировки.

class Network: # Инициализация нейросети

    def \_\_init\_\_(self, network\_path, number\_neurons=None, train\_data\_path=None, test\_data\_path=None, epochs=None, batch\_size=None, train\_rate=None): # Все нужные переменные

        self.network\_path = network\_path + '\\'

        self.number\_neurons = number\_neurons

        self.train\_data\_path = train\_data\_path

        self.test\_data\_path = test\_data\_path

        self.epochs = epochs

        self.batch\_size = batch\_size

        self.train\_rate = train\_rate

Далее, при обучении, необходимо инициализировать веса и смещения случайными значениями, например, центрированными около нуля и со средним отклонением, равным квадратному корню из количества нейронов в слое.

После инициализации весов нейросети нужно их сохранять и обратно загружать.

def save(self):

    makedirs(f'{self.network\_path}weights') # Создание пустых папок весов и смещений

    makedirs(f'{self.network\_path}bias')

    for i in range(len(self.layers\_weights)): # Сохранение весов

        np.savetxt(f'{self.network\_path}weights\\weight{i}.csv', self.layers\_weights[i], delimiter=',')

        np.savetxt(f'{self.network\_path}bias\\bias{i}.csv', self.layers\_bias[i])

def load(self):

    self.layers\_weights = [] # Инициализация пустых весов

    self.layers\_bias = []

    i = 0 # проход по файлам, пока они не закончатся

    while path.isfile(f'{self.network\_path}weights\\weight{i}.csv'):

        self.layers\_weights.append(np.loadtxt(f'{self.network\_path}weights\\weight{i}.csv', dtype='float64', delimiter=','))

        self.layers\_bias.append(np.loadtxt(f'{self.network\_path}bias\\bias{i}.csv', dtype='float64'))

        i += 1

def weights\_bias\_init(self):

        self.layers\_weights = [] # Инициализация пустых весов

        self.layers\_bias = []

        for num\_neurons in range(len(self.number\_neurons) - 1): # Присваивание случайных значений для всех слоев

            self.layers\_weights.append(np.random.normal(0.0, pow(self.number\_neurons[num\_neurons], -0.5), (self.number\_neurons[num\_neurons], self.number\_neurons[num\_neurons + 1])))

            self.layers\_bias.append(np.random.normal(0.0, pow(self.number\_neurons[num\_neurons + 1], -0.5), (self.number\_neurons[num\_neurons + 1])))

И прежде, чем реализовывать обучение нейросети, нужно создать функции нужные для этого.

def ReLU(x): # Функция активации

    return max(x, 0.0)

def ReLU\_deriv(x): # Производная ReLU

    return float(x >= 0)

def SoftMax\_batch(vect): # Преобразование батча к вероятностям

    res = np.exp(vect)

    return res / np.sum(res, axis=1, keepdims=True)

def cross\_entropy\_batch(p, y): # Вычисление отличия между вероятностями

    return -np.log(np.array([p[j][y[j]] for j in range(len(y))]))

После каждой эпохи обучения будем выводить cross entropy по всей тестовой выборке.

def learn(self): # Загрузка базы тренировочных данных

    train\_data = np.loadtxt(self.train\_data\_path, dtype="int32", delimiter=',')

    train\_data = np.loadtxt(self.train\_data\_path, dtype="uint8", delimiter=',')

    y\_test = np.concatenate(test\_data[:, :1], axis=0)

    pre = test\_data[:, 1:]

    print('Начало обучения')

    t = time() # Начало отчета времени

    for ep in range(self.epochs):

        np.random.shuffle(train\_data) # Перемешивание данных

        for i in range(train\_data.shape[0] // self.batch\_size):

            # Прямое распространение нейросети

            layer = [train\_data[i \* self.batch\_size: (i + 1) \* self.batch\_size, 1:]]

            y = np.concatenate(train\_data[i \* self.batch\_size: (i + 1) \* self.batch\_size, :1], axis=0)

            Z = []

            for L in range(len(self.number\_neurons) - 2):

                Z.append(layer[L] @ self.layers\_weights[L] + self.layers\_bias[L])

                layer.append(np.vectorize(ReLU)(Z[L]))

            layer.append(SoftMax\_batch(layer[-1] @ self.layers\_weights[-1] + self.layers\_bias[-1]))

            # Обратное распространение

            dE\_dZ = layer[-1] - y\_vector\_batch(y, self.number\_neurons[-1])

            for L in range(len(self.number\_neurons) - 2, -1, -1):

                dE\_dW = np.transpose(layer[L]) @ dE\_dZ

                dE\_dB = np.sum(dE\_dZ, axis=0, keepdims=True)

                dE\_dB.resize((self.number\_neurons[L + 1], ))

                if L > 0:

                    dE\_dA = dE\_dZ @ np.transpose(self.layers\_weights[L])

                    dE\_dZ = dE\_dA \* np.vectorize(ReLU\_deriv)(Z[L - 1])

                # Корректировка весов

                self.layers\_weights[L] -= self.train\_rate \* dE\_dW

                self.layers\_bias[L] -= self.train\_rate \* dE\_dB

        # Вывод номера эпохи и точности нейросети

        p\_test = self.predict\_batch(pre)

        print(ep, round(np.sum(cross\_entropy\_batch(p\_test, y\_test) / self.batch\_size \* 100),5))

    self.save() # Сохранение нейросети

print(time() - t)

    print("Нейросеть обучена")

# Преобразование вектора правильных ответов к матрице истинных вероятностей

def y\_vector\_batch(y, num\_classes):

    y\_full = np.zeros((len(y), num\_classes))

    for j, yj in enumerate(y):

        y\_full[j, yj] = 1

    return y\_full

После обучения нужно проверить точность нейросети на тестовых дынных, проверять будем в процентах отгаданных цифр.

def predict(self, image):  # Предсказание нейросети

    layer = image

    for i in range(len(self.layers\_bias) - 1):

        layer = np.vectorize(ReLU)(layer @ self.layers\_weights[i] + self.layers\_bias[i])

    layer = SoftMax\_batch(layer @ self.layers\_weights[-1] + self.layers\_bias[-1])

    return layer

def num\_correct(p, y):  # Подсчет верных ответов

    res = 0

    for i in range(len(p) - 1):

        m\_i = p[i].argmax()

        if y[i] == m\_i:

            res += 1

    return res

def test(self):

    test\_data = np.loadtxt(self.test\_data\_path, dtype="uint8", delimiter=',')

    y = np.concatenate(test\_data[:, :1], axis=0)

    p = self.predict(test\_data[:, 1:])

    return round(num\_correct(p, y) / test\_data.shape[0] \* 100, 3)

Пришло время тренировать нейросеть. Количество слоев выберем 3, количество нейронов в слоях: 784, 256, 10, размер батча 500, скорость обучения 0.01 и 25 эпох.

Начало обучения

0 1360.035

1 996.717

28 411.725

29 405.084

59 383.69067

Нейросеть обучена

95.57

digit = Network('Digit', [784, 128, 10], 'mnist\\mnist\_train.csv', 'mnist\\mnist\_test.csv', 60, 500, 0.01)

digit.weights\_bias\_init()

digit.learn()

print(digit.test())

По выводу в терминале видим, что погрешность уменьшается:

Финальная точность вышла 95.57%. Если загрузить нейросеть и пытаться дальше ее обучать, то видно, что погрешность не уменьшается или даже растет, это значит, что нейросеть переобучилась.

Продуктом можно сделать сам класс нейросети так как он подходит для обучения по любому набору данных??

**Библиография**

1. Определение искусственного интеллекта. <https://ru.wikipedia.org/wiki/ИИ>
2. Определение нейронной сети. <https://ru.wikipedia.org/wiki/Нейронная_сеть>
3. Определение стохастического спуска. <https://ru.wikipedia.org/wiki/Стохастический_градиентный_спуск>
4. История создания нейронных сетей. <https://habr.com/ru/articles/861888/>