Автономная некоммерческая организация высшего образования

«Российский новый университет»

Институт Информационных систем и инженерно-компьютерных технологий

Курсовая работа

по дисциплине: Прогамирование

на тему: “Нейронные сети”

студента (ки) 417 группы 1 курса

очной формы обучения

направления подготовки:

Информатика и вычислительная техника

направленности (профиля):

Безопасность информационных систем и вычислительной техники

Маринченко Антон Олегович

Руководитель:

старший преподаватель

Зайцев Сергей Валерьевич

Работа допущена к защите «\_\_\_\_»\_\_\_\_\_\_\_20\_\_\_г.

Оценка\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Регистрационный номер\_\_\_\_\_\_\_\_ от «\_\_\_\_\_»\_\_\_\_\_\_\_\_\_20\_\_\_г.

Москва

2023 г.

Оглавление:

Теория:

1. ВВЕДЕНИЕ
   1. Цель курсовой работы
   2. Что такое нейросеть и нейрон
   3. История
2. Зачем нужны нейросети
   1. Сферы применения
   2. Типы нейросетей
   3. Устройство нейросетей
3. Примеры применения нейросетей в реальных проектах
   1. Автопилот Tesla
   2. Яндекс – “YaGPT ver2.0“
4. Практика
5. Заключение
6. Источники
7. Приложение к работе

Введение.

В данной курсовой работе, будут рассмотрены виды нейросетей и принципы их работы. А так-же будет представлена реализация простейшего нейрона с подбором веса и функциями для его обучения и калибровки функции активации. Цель работы показать зачем и как возможно заменить функции кода нейросетью.

Что такое нейросеть и нейрон

Нейросеть – крайне сложная математическая модель, которая состоит из нейронов, связанных друг с другом синапсами.

Нейрон – основной блок для создания нейросети его задача вычислить взвешенную сумму значений его выходных данных, нормализировать значение функцией активации и подать на выход понятное другим нейронами значение.

Сигмойда – отображает выходные данные на значения от 0 до 1

Нейросети состоят из 3 и более слоёв:

1 входной слой

1 выходной слой

N скрытых слоёв

Нейросеть с одним нейроном умеет предугадывать значения легких уравнений из линейной алгебры при заданных 2 входных значениях x и z:

x + y = z

x – y = z

x // y = z

x \* y = z

# История нейросетей.

История нейросетей насчитывает несколько десятилетий и включает в себя значительные достижения и прорывы. Вот основные моменты в истории развития нейросетей:

1943 год: Уоррен МакКаллок и Уолтер Питтс представили первую концепцию искусственных нейронных сетей, основанную на работе о биологических нейронах. Они предложили модель искусственного нейрона, который может принимать входные сигналы и передавать их через взвешенные связи.

1956 год: Джон Маккарти провел конференцию "Летний проект Дартмута", которая считается рождением искусственного интеллекта (ИИ). Маккарти включил в понятие ИИ искусственные нейронные сети, поощривая исследования в этой области.

1960-е годы: Франк Розенблатт разработал перцептрон, одну из первых форм искусственных нейронных сетей. Перцептрон был способен обучаться на основе входных данных и использовался для решения задач классификации паттернов.

1980-е годы: Обратное распространение ошибки (backpropagation) было предложено в качестве алгоритма обучения нейронных сетей. Этот алгоритм позволил эффективно настраивать веса нейронных сетей и повысил их способность к обучению.

1990-е годы: В это время возникли сверточные нейронные сети (CNN), которые показали большой успех в обработке изображений и распознавании образов. Леон Ботулик разработал метод обучения глубоких нейронных сетей с прямыми связями (Deep Learning by Superposition of Elementary Computations), который позволял обучать более глубокие архитектуры нейронных сетей.

2000-е годы: В это время появились более сложные архитектуры глубоких нейронных сетей, включая рекуррентные нейронные сети (RNN). RNN обладает способностью обрабатывать последовательные данные и работать с временными рядами, что сделало их эффективными для задач обработки естественного языка и прогнозирования.

2012 год: Команда исследователей из Университета Торонто под руководством Джеффри Хинтона представила сверточную нейронную сеть, известную как AlexNet. AlexNet была обучена на огромном наборе изображений и показала выдающуюся производительность в задачах распознавания и классификации изображений.

2014 год: Исследователи из Google представили нейронную сеть под названием Inception, которая победила в соревновании ImageNet Large Scale Visual Recognition Challenge (ILSVRC). Архитектура Inception использовала идею глубоких сверточных слоев и стала примером эффективной архитектуры для классификации и распознавания изображений.

2014 год: В этом же году был представлен DCGAN (Deep Generative Adversarial Network), который способен генерировать убедительные изображения, используя глубокое обучение и генеративные модели. DCGAN открыл новые возможности в области генерации изображений.

2015 год: Рекуррентные нейронные сети с долгой краткосрочной памятью (LSTM) стали популярными в обработке естественного языка и последовательных данных. LSTM обладает способностью учитывать долгосрочные зависимости и стал важным инструментом для машинного перевода, генерации текста и других задач.

2016 год: Исследователи из Google DeepMind разработали нейронную сеть AlphaGo, которая победила в игре Го (одной из сложнейших стратегических настольных игр) чемпиона мира Ли Седоля. AlphaGo применила глубокое обучение для изучения тактик и стратегий игры.

2017 год: Модель Transformer, основанная на механизмах внимания, принесла прорыв в области перевода и обработки естественного языка. Transformer стала широко применяемой архитектурой для машинного перевода и генерации текста.

2018 год: Развитие генеративных моделей, включая генеративно-состязательные сети (GAN), позволило создавать фотореалистичные изображения, а также генерировать музыку и видео.

2020 год: Открытие архитектуры GPT-3 (Generative Pre-trained Transformer 3) от OpenAI, которая стала одной из самых больших и мощных моделей языкового моделирования. GPT-3 показало значительные успехи в генерации текста, ответах на вопросы и других задачах обработки естественного языка.

2023 год: Нейронные сети продолжают развиваться, применяться в широком спектре задач и находить новые применения в различных областях, таких как медицина, финансы, автономная навигация и многое другое. Исследователи и разработчики постоянно стремятся к улучшению производительности и эффективности нейронных сетей.

# Зачем нужны нейросети?

Нейросети имеют широкий спектр применения в различных областях, таких как:

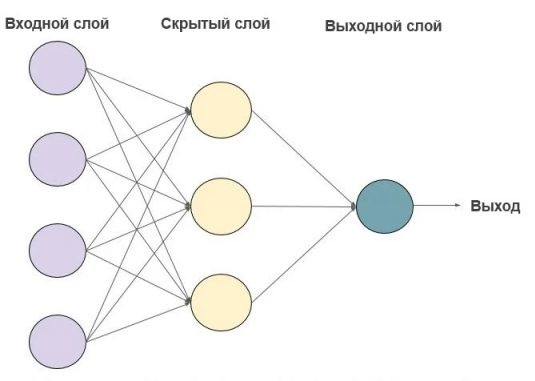
* Распознавание образов: классификация изображений, распознавание лиц, оптическая символика и т.д.
* Обработка естественного языка: машинный перевод, распознавание речи, генерация текста и т.д.
* Прогнозирование и анализ данных: прогнозирование временных рядов, анализ данных, обнаружение аномалий и т.д.
* Управление и оптимизация: автоматическое управление, оптимизация процессов и т.д.
* Медицина: диагностика заболеваний, анализ медицинских изображений, прогнозирование результатов лечения и т.д.

# Типы нейросетей и их различия

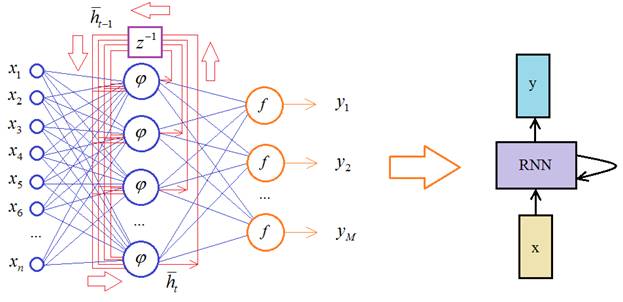
Существует несколько типов нейросетей:

1. Полносвязные [FCN]
2. Рекурентные [RNN]
3. Свёрточные [CNN]
4. Сети долгой краткосрочной памяти [LSMN]
5. Генеративно-состязательные [GAN]
6. Автоэнкодеры [Autoencoders]

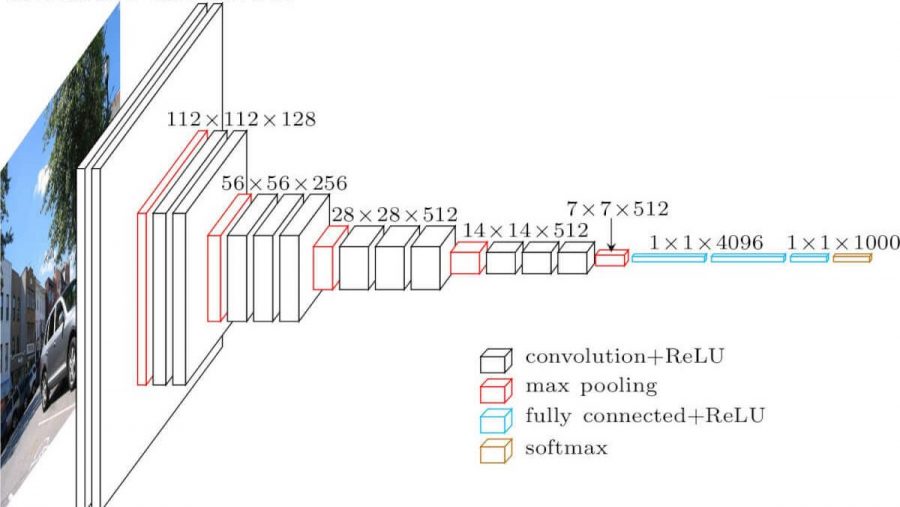
Давайте рассмотрим все виды по порядку.



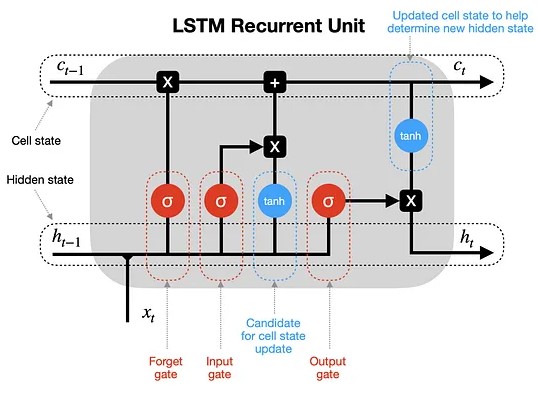
1. Полносвязные – самый простой, он действует строго от входного слоя до выходного, проходя через скрытые слои их применение возможно в таких задачах как классификация объектов или регрессии.



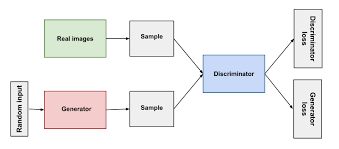
1. Рекуррентные – более сложный алгоритм и в отличие от FNN имеет обратные связи, что даёт им возможность обрабатывать последовательные данные или задействовать временные ряды или естественный язык



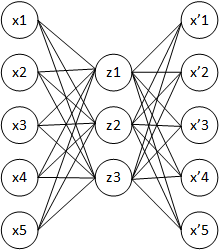
1. Свёрточные - они особенно хорошо подходят для обработки данных с пространственной структурой, таких как изображения. Они используют операцию свертки для извлечения важных признаков из входных данных.



1. Сети долгой краткосрочной памяти - специальный тип рекуррентных нейронных сетей, способных лучше сохранять и использовать информацию о длительных зависимостях в последовательных данных.



1. Генеративно-состязательные сети - особый тип нейронных сетей, состоящий из двух моделей - генератора и дискриминатора. Они применяются для генерации новых данных, таких как изображения, звуки или тексты.



1. Автоэнкодеры - они используются для изучения компактного представления входных данных. Автоэнкодеры состоят из кодировщика, который сжимает данные в скрытое представление, и декодера, который восстанавливает данные из скрытого представления.

# Устройство нейросетей.

1. Полносвязная нейронная сеть - допустим у к нашей нейросети подключены 2 датчика выдающих только значения от 0 до 1, соответственно функция активации:

F(x) = {0, X<=0,5 }, {1, X<0.5}

простым языком 1 – да, 0 – нет.

Зададим конкретную задачку нашей нейросети: “Найти все круги желтого цвета”

Датчики назначаем на их задачи:

1. Датчик 1 – цвет.
2. Датчик 2 – форма.

У нас появляется 4 возможных комбинации (в скобках указан пример обектов): (красный круг) - [0,1]

(желтый треугольник) - [1,0]

(розовый треугольник) - [0,0]

(желтый круг) - [1,1]

Зададим равный вес синапсов по 0,5:

(красный круг) - x=0,5\*0+0,5\*1=0

(желтый треугольник) - x=0,5\*1+0,5\*0=0

(розовый треугольник) - x=0,5\*0+0,5\*0=0

(желтый круг) - x=-0,5\*1+0,5\*1=1

Соответственно желтый круг выдал положительный результат.

1. Рекуррентная нейронная сеть - тип нейросети по большей части использущейся для работы и генерации текста и изображений.

Например, нашумевшая “ChatGPT”, все версии данной нейросети являются рекуррентными, так как оперируют огромной базой/моделью/словарём слов и предложений.

Основное отличие RNN от FCN в том, что RNN использует информацию последовательно, т.е. опирается на предыдущие ответы и запросы.

Такой метод генерации можно назвать “памятью”, и это даёт возможность поддержания здравого смысла при генерации текстов.



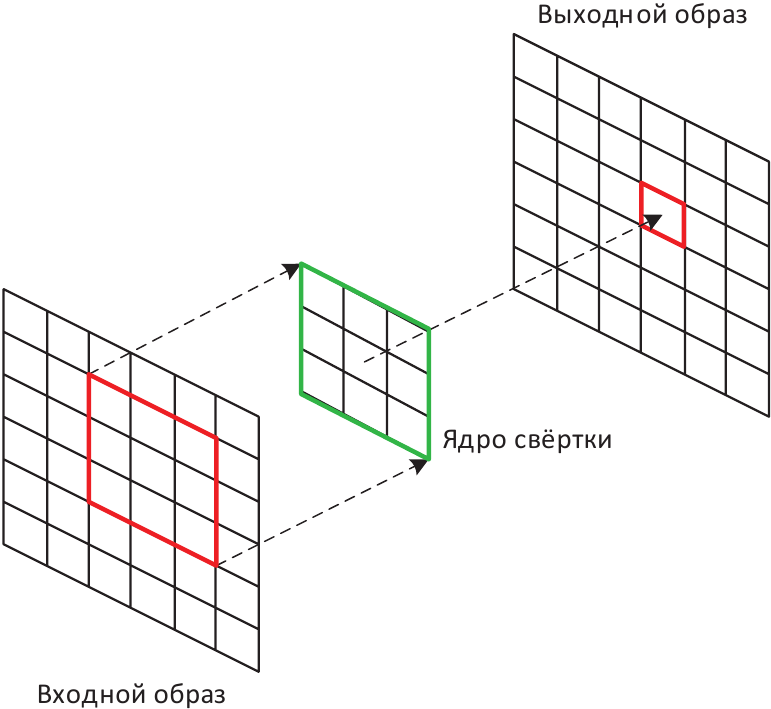
На схеме показано как RNN разворачивается в полную сеть.

x\_t - вход на временном шаге t, например вектор с одним состоянием.

s\_t - это скрытое состояние на шаге t или память.

o\_t - выход на шаге t. Например вектор предсказанных вероятностей.

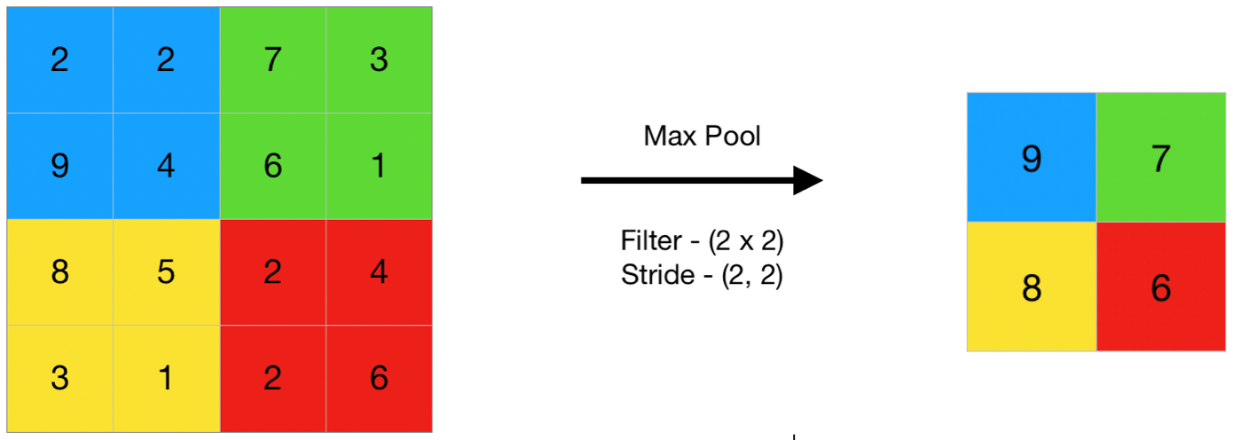
Обучение RNN так-же как и у FCN происходит через алгоритм обратного распространения ошибки, но с поправкой на то, что изменения зависят не только от текущего шага, но и от предыдущих.

1. Свёрточная нейронная сеть – алгоритм, принимающий изображения и присваивая веса аспектам картинки может определять объекты или различать изображения. Для данного типа нейросетей лучше всего подходит плотный поток взаимосвязанных данных, таких как пиксели. В изображениях есть визуальные данный такие как цвет и яркость, соседние пиксели содержат похожие данные с минимальными отклонениями. Основные сферы использования данных нейросетей это классификация и распознавание объектов.

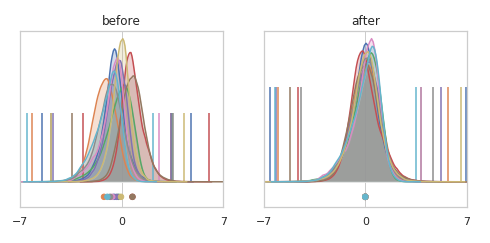
Данный тип нейросетей работает с Датасетами (Datasets) в которых содержится информация о классификации объектов. Сравнивая входной образ с образами из датасета, нейросеть может последовательно определить несколько схожих частей тела/объекта и выдать правильный ответ.

Архитектура свёрточной нейросети состоит из 4 пунктов:

1. Свёрточный слой (convolutional layer) - основной блок свёрточной нейронной сети. Он отвечает за удаление лишних деталей и оптимизацию данных для последующего анализа. Свёртку возможно применять для каждого признака, нейросеть будет подбирать недостающие данные сама, сравнивая из с даннами из датасета.
2. Пулинг (pooling) - нелинейное уплотнение карты признаков, которое следует после свёрточного слоя. Из оптимизированных данных данный слой выбирает самые важные и удаляет несущественные признаки. К результату пулинга можно применить свёрточный слой и получить более точные данные. Это требуется для выстраивания иерархии признаков. Начиная от контура и мелких деталей, заканчивая формой глаз, ушей, лица\морды и ушей.



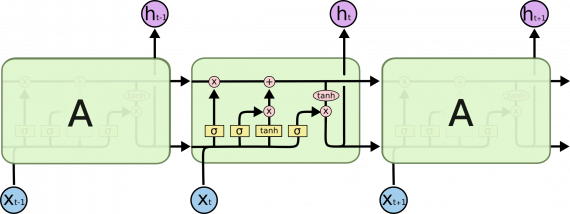
1. Нормализация по батчу (bath-normalization) - Пакетная нормализация, она выполняется путём масштабирования данных, подаваемых в функцию активации. Обычно это принято делать в скрытых слоях нейронной сети. Данный метод помогает ускорить обучение.



1. Полносвязный слой – выходные нейроны связанные со всеми нейронами нейросети.

У свёрточной нейросети есть свои камни преткновения. Ведь она не может работать с табличными данными и им крайне сложно работать с данными разного масштаба такими, как картинки разного разрешения и документы разного формата. Если задать цель сравнить документы А5 и А3 то надо свести их в один формат А4 иначе нейросеть не поймёт что это одинаковый текст.

1. Сети долгой краткосрочной памяти (LSTM) - Особая разновидность архитектуры рекуррентных нейронных сетей, способная к обучению долговременным зависимостям. Данный тип нейросети разработан специально для исключения проблем долгосрочной зависимости. Такие нейросети изначально запоминают информацию на долгие промежутки времени, не обучаясь этому.



Структура LSTM очень похожа на RNN, но модули немного отличаются. Вместо одного слоя, они содержат четыре слоя, и эти слои взаимодействуют особым способом.

Основа LSTM – cell state – ячейка, проходящая через всю цепочку и участвующая лишь в нескольких линейных преобразованиях. LSTM имеет возможность удаления информации из ячейки, данный процесс регулируется фильтрами.

Шаги работы алгоритма LSTM:

1. Лишняя информация фильтруется и удаляется из дальнейшей обработки. Фильтрация осуществляется сигмовидным слоем (гейт утраты).
2. Решаем какую информацию надо сохранить в состоянии ячейки.
3. Гейт входа решает какие значения требуется обновить
4. Слой tanh создаёт вектор новых значений, которые добавляются в состояние.
5. Объединяем значения со второго щага.
6. Обновляем предыдущее состояние ячейки для получения нового состояния (C). Затем добавляем новые значений кандидатов (i\*C).
7. Пропускаем состояние ячейки через tanh и умножаем на его на входной сигнал сигмовидного гейта.

Данная схема является традиционной.

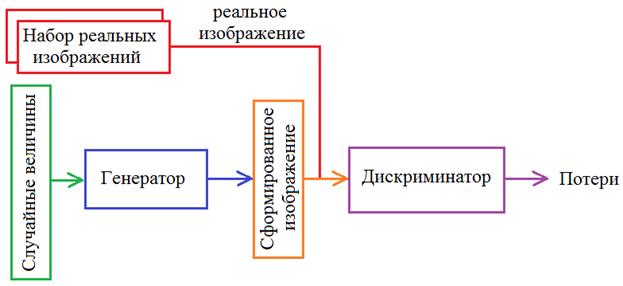
1. Генеративно-состязательные сети (GAN) – данный тип нейросетей является для меня самым интересным экземпляром. В ней задействована архитектура, включающая генератор и дискриминатор которые работают друг против друга, поэтому GAN и называется генеративно-состязательной.

GAN нейросеть одна из самых потенциальных, она может работать с широким спектром данных.

Почему именно генеративно-состязательная?

Такое название было дано так как она генерирует изображения и сравнивает из с реальными, то есть цель нейросети сделать настолько реалистичное изображение, чтобы вторая нейросеть – дискриминатор, приняла его за реальное фото.

При взаимодействии двух данных нейросетей генератор подаёт изображения, а в ответ получает значение от 0 до 1 которое обозначает уверенность дискриминатора в реальности полученного изображения. Генератор в свою очередь пытается обмануть дискриминатор и получить значение близкое к 1, сгенерировав фото не отличимое от реального.



1. Автоэнкодеры - специальная архитектура искусственных нейронных сетей позволяющая применять обучение без учителя при использовании метода с обратного распространения ошибки.

Особенность архитектуры автоэнкодера в том, что он выходной слой должен иметь столько же нейронов сколько и входной, а в основе лежит сеть прямого распространения. Данная нейросеть состоит из энкодера (g) и декодера (f). Энкодер переводит входной сигнал в его представление (код):

*h*=*g*(*x*)

а декодер восстанавливает сигнал по его коду:

*x*=*f*(*h*)

Автокодировщик, изменяя *f* и *g*, стремится выучить тождественную функцию

*x*=*f*(*g*(*x*)),

минимизируя какой-то функционал ошибки.

*L*(*x*,*f*(*g*(*x*)))

При этом семейства функций энкодера *g* и декодера *f* как-то ограничены, чтобы автоэнкодер был вынужден отбирать наиболее важные свойства сигнала.

# Примеры использования нейросетей в реальных проектах.

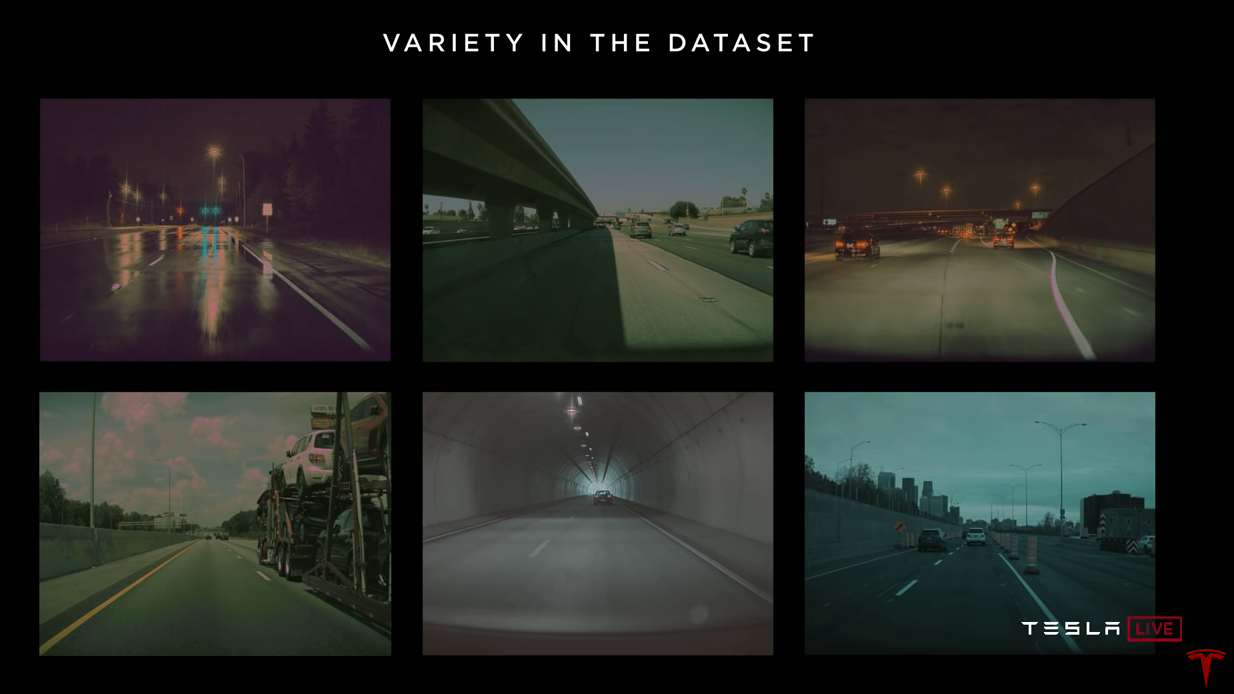
1. Распознавание и классификация изображений:
   1. Компания Google использует нейронные сети для разработки своей системы распознавания изображений. Это позволяет им создавать приложения, которые могут распознавать и классифицировать объекты на фотографиях с высокой точностью.
   2. Facebook также применяет нейросети для автоматического распознавания лиц на фотографиях и предложения тегов с именами людей. Это значительно облегчает процесс разметки фотографий и их организации.
2. Обработка естественного языка и машинный перевод:
   1. Google Translate использует нейросети для улучшения качества машинного перевода. Это позволяет достичь более точного и естественного перевода между различными языками.
   2. Siri от Apple и Google Assistant используют нейронные сети для распознавания и обработки речи, что позволяет пользователям задавать вопросы голосом и получать соответствующие ответы.
3. Автономные транспортные средства:
   1. Компания Tesla разрабатывает автопилот с использованием нейронных сетей. Нейросети обрабатывают данные с камер, радаров и других датчиков для распознавания объектов на дороге, принятия решений и управления автомобилем без участия водителя.
   2. Google разработала проект Waymo, использующий нейросети для создания автономных автомобилей. Нейросети помогают распознавать объекты, определять маршруты и принимать решения на основе данных с датчиков.
4. Медицинская диагностика:
   1. Нейронные сети применяются для обработки медицинских изображений, таких как снимки рентгена и МРТ. Они могут автоматически распознавать признаки различных заболеваний и помогать врачам в диагностике.
   2. Исследователи также разрабатывают нейронные сети для анализа генетических данных и предсказания риска развития различных заболеваний, таких как рак и диабет.

Остановимся подробней на самых интересных нейросетях.

# Tesla.

Нейросеть компании Tesla разработанной для автопилота одноимённых автомобилей – самая продвинутая нейросеть среди ИИ автопилотов, компаний конкурентов.

Обучение ИИ.



Обучение нейронных сетей Tesla производится на необработанных изображениях, извлеченных из видеозаписей, снятых с сетевых камер с видом с высоты птичьего полета, которые выводят план дороги, статическую инфраструктуру и 3D-объекты непосредственно в виде сверху вниз. Изображения данных не имеют маркировки и охватывают множество разнообразных сценариев по всему миру и включают один миллион

транспортных средств в режиме реального времени

Компоненты автомобиля.

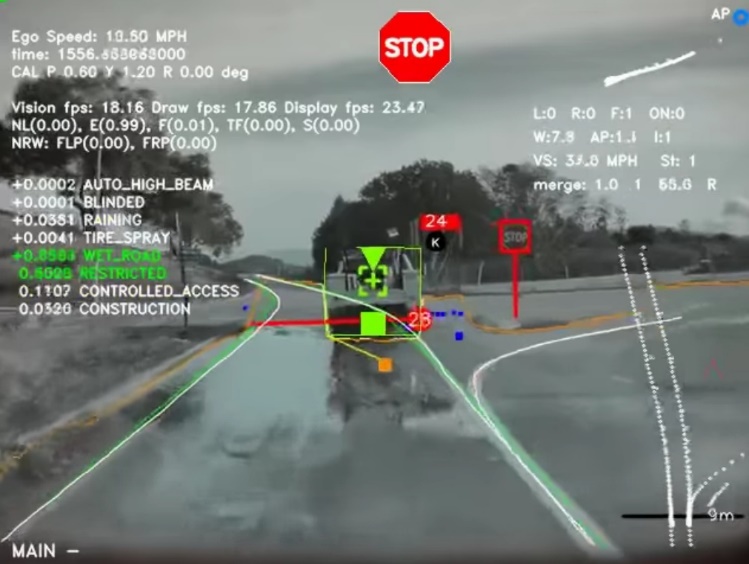
Сеть состоит из 70,000 48 графических процессоров (GPU), которые обучают XNUMX глубокое обучение модели. Аппаратные компоненты автомобиля, включая камеры и датчики, предоставляют неконтролируемые данные, которые передаются через сеть этих моделей.

Автомобиль узнает о возможных объектах в окружающей среде, таких как пешеход, дерево и т.д., из предоставленных данных. Архитектура также состоит из двух чипов ИИ, которые используют принципы глубокое обучение. Эти чипы помогают принимать решения для автомобиля в режиме реального времени, например, когда и как поворачивать во время вождения.

FSD-Чип.

FSD (Full Self Drive - Полное Cамостоятельное Вождение) - это чипы вывода ИИ, на которых работает программное обеспечение автопилота Tesla. Эти чипы были разработаны с микроархитектурными улучшениями, которые обеспечивают максимальную производительность кремния на ватт.

FSD реализуют планирование этажа, анализ времени и мощности, а также пишут надежные тесты и таблицы результатов для проверки функциональности и производительности ИИ.



Алгоритм Автономии.

Основные алгоритмы, которые управляют автомобилем, создавая высокоточное представление мира и планируя траектории в заданном пространстве.

Чтобы обучить нейронные сети прогнозировать такие представления, Тесла алгоритмически создает точные и крупномасштабные достоверные данные, комбинируя информацию от датчиков автомобиля в пространстве и времени.

Эти алгоритмы используют передовые методы для создания надежной системы планирования и принятия решений, которая работает в сложных реальных ситуациях в условиях неопределенности.

Инфраструктура Оценки.

Инфраструктура оценки Tesla включает в себя инструменты оценки с разомкнутым циклом, замкнутым циклом и аппаратные средства оценки, а также инфраструктуру в масштабе.

Эта инфраструктура позволяет ИИ отслеживать улучшения производительности и предотвращать регрессию.

Характеристики Tesla.

* Камеры, ультразвуковые датчики и радары воспринимают окружающую среду
* Радар измеряет расстояние вокруг автомобиля
* Ультрафиолетовые методы измеряют расстояние, а пассивное видео распознает

объекты вокруг автомобиля.

* Использует две микросхемы ИИ, построенные на принципах глубоких нейронных сетей.
* Чипы ИИ, состоящие из 6 миллиардов транзисторов
* В 21 раз быстрее, чем чипы Nvidia
* Чипы AI имеют 32 мегабайта высокоскоростной памяти SRAM.
* Состоит из 48 моделей глубокого обучения.
* Содержит 70,000 XNUMX графических процессоров (GPU)
* Выводит 1000 различных тензоров (прогнозов) на каждом временном шаге

# Яндекс – YandexGPT.

Нейросеть Яндекс – самая молодая нейросеть в наше время, которая находиться в тестовом режиме. Её добавили, как функцию к голосовому ассистенту “Алиса” и позволяет генерировать разный текст на естественном языке.

Возможности.

YandexGPT обладает несколькими возможностями, которые делают его мощным инструментом для работы с текстами:

1. Генерация текста: YandexGPT способен генерировать связные и смысловые тексты на основе заданного контекста или вопроса. Он способен продолжать начатые предложения, отвечать на вопросы, создавать описания и многое другое. Модель обучена на большом объеме разнообразных текстов, что позволяет ей генерировать тексты различной тематики и стиля.
2. Дополнение предложений: YandexGPT может помочь вам сформулировать или дополнить предложения. Вы можете предоставить начало предложения, и модель продолжит его, учитывая контекст и смысл. Это может быть полезным при написании статей, сочинений, или просто для получения вдохновения и идей.
3. Ответы на вопросы: YandexGPT способен отвечать на вопросы, предоставляя информацию на основе своего обучения. Вы можете задать конкретный вопрос, и модель постарается предоставить соответствующий ответ на основе своего знания и обучения.
4. Перевод текста: YandexGPT может использоваться для перевода текстов с одного языка на другой. Вы можете предоставить текст на одном языке, и модель постарается сгенерировать его эквивалент на другом языке. Важно отметить, что переводы модели могут быть полезными, но они могут не всегда быть безошибочными и требуют дополнительной проверки.
5. Генерация кода: YandexGPT также может помочь в генерации программного кода на основе предоставленных шаблонов или задач. Модель может помочь с написанием функций, классов, условий и других элементов кода, учитывая контекст и требования.

Архитектура.

Модель YandexGPT, основанная на архитектуре Transformer, была разработана для генерации текста и общения с пользователями. Она обучается на огромных объемах текстовых данных, чтобы научиться понимать естественный язык и генерировать связные и информативные ответы. Для достижения лучшего качества генерации текста, YandexGPT может содержать модификации и улучшения, которые оптимизированы для конкретной задачи и данных.

Transformer - это архитектура глубокого обучения, которая значительно повлияла на область обработки естественного языка. Она была представлена исследователями Google Brain в 2017 году и представляет собой альтернативу классическим рекуррентным нейронным сетям, таким как LSTM (Long Short-Term Memory) и GRU (Gated Recurrent Unit).

Одной из основных особенностей Transformer является механизм само-внимания (self-attention). Само-внимание позволяет модели вычислять взаимодействия и зависимости между различными словами или элементами в последовательности данных. В отличие от рекуррентных моделей, которые обрабатывают последовательности поэлементно, само-внимание позволяет модели "обращать внимание" на все элементы последовательности одновременно.

Архитектура Transformer состоит из нескольких слоев, называемых "трансформерными блоками". Каждый трансформерный блок состоит из двух подслоев: слоя само-внимания и полносвязного слоя с применением пакетной нормализации. Входной сигнал последовательности проходит через каждый трансформерный блок в нейросети.

Слои само-внимания внутри каждого блока Transformer позволяют модели узнавать взаимосвязи между словами во входном тексте. Они вычисляют веса важности для каждого слова в контексте остальных слов в предложении. Затем эти взвешенные значения используются для создания контекстно-зависимых представлений слов.

Кроме слоев само-внимания, каждый блок Transformer также содержит полносвязные слои. Полносвязные слои выполняют преобразования над представлениями, полученными из слоев само-внимания. Они могут включать в себя операции, такие как линейные преобразования и функции активации, чтобы модель могла выявить более сложные зависимости и взаимосвязи в тексте.

# Практическая работа.

Данная программа представляет собой реализацию простейшего нейрона в нейронной сети на языке программирования C++. Нейрон обучается на входных данных и ожидаемых результатах, чтобы предсказывать значения на основе заданных входных данных.

# Разбор кода нейрона.

#include <iostream>

#include <cmath>

using namespace std;

class Neuron

{

public:

Neuron()

{

weight = 1;

last\_error = 1.1;

smoothing = 0.01;

}

В этой части кода определяется класс Neuron. Конструктор класса инициализирует три приватных переменных: weight (вес), last\_error (последняя ошибка) и smoothing (коэффициент сглаживания) значениями по умолчанию.

float get\_last\_error()

{

return last\_error;

}

void set\_smoothing(float new\_smoothing)

{

smoothing = new\_smoothing;

}

float get\_smoothing()

{

return smoothing;

}

float get\_weight()

{

return weight;

}

float proccess\_data(float input\_data)

{

return input\_data \* weight;

}

void train(float input\_data, float ExpectedResult)

{

float result\_now = input\_data \* weight;

last\_error = float(ExpectedResult) - float(result\_now);

float correction = (last\_error / result\_now) \* smoothing;

weight += correction;

}

В этой части определены различные методы класса Neuron. get\_last\_error() возвращает значение переменной last\_error, set\_smoothing() устанавливает новое значение переменной smoothing, get\_smoothing() возвращает текущее значение smoothing, get\_weight() возвращает текущее значение переменной weight, proccess\_data() выполняет обработку входных данных, умножая их на вес нейрона, а train() проводит обучение нейрона на основе входных данных и ожидаемого результата. Ошибки вычисляются, и вес нейрона корректируется в соответствии с этой ошибкой.

void training(float task, float ans)

{

int iteractions = 0;

while (check\_training())

{

iteractions++;

train(task, ans);

cout << "Epoh: " << iteractions << " | Weight: " << get\_weight() << endl;

}

}

Метод training() выполняет обучение нейрона. Он использует цикл while, который продолжается до тех пор, пока метод check\_training() возвращает true. В каждой итерации цикла происходит вызов метода train(), а также выводятся на экран номер эпохи и текущее значение веса нейрона.

bool check\_training()

{

if (weight == last\_error)

{

return true;

}

else

{

if (last\_error > smoothing || last\_error < -smoothing)

{

return true;

}

else

{

return false;

}

}

}

Метод check\_training() проверяет условие, при котором обучение нейрона должно продолжаться. Если вес нейрона равен последней ошибке или если последняя ошибка превышает значение smoothing или меньше его отрицательного значения, метод возвращает true, в противном случае - false.

private:

float weight;

float last\_error;

float smoothing;

};

int main()

{

Neuron neuron;

float task = 1;

float ans = 1;

int iteraction = 0;

cout << "Traning task >>";

cin >> task;

cout << "Traning ans >>";

cin >> ans;

cout << "Start traning." << endl;

neuron.training(task, ans);

neuron.set\_smoothing(0.001);

int right\_ans = 00;

while (true)

{

cout << "task> ";

cin >> task;

ans = neuron.proccess\_data(task);

cout << "------------" << endl;

cout << ans << " | " << round(ans) << endl;

cout << "------------" << endl;

cout << "Right ans (0 for skip): ";

cin >> right\_ans;

if (right\_ans != 0)

{

neuron.training(task, right\_ans);

}

cout << "Weight: " << neuron.get\_weight() << endl;

}

}

В функции main() создается объект класса Neuron под названием neuron. Затем задается переменные task и ans и запрашиваются их значения у пользователя. После этого выводится сообщение о начале тренировки нейрона, и метод neuron.training() вызывается для обучения нейрона с использованием введенных задач и ответов. Затем значение smoothing устанавливается на более низкий уровень (0.001).

Затем следует бесконечный цикл while (true), который запрашивает у пользователя ввод задачи (task). Значение task подвергается обработке нейроном с помощью метода neuron.proccess\_data(), и результат выводится на экран. Затем запрашивается правильный ответ (right\_ans), и если он не равен 0, то метод neuron.training() вызывается снова для дальнейшего обучения нейрона с использованием введенных данных. В конце каждой итерации цикла выводится текущее значение веса нейрона.

Это простейший пример кода, демонстрирующий работу нейрона в нейросети. Он имеет свои ограничения и не представляет полноценную нейронную сеть, но может быть использован в различных задачах, включая классификацию, регрессию и аппроксимацию функций. Вот несколько примеров использования одиночного нейрона:

1. Бинарная классификация: Одиночный нейрон может использоваться для решения задачи бинарной классификации, где требуется разделить данные на два класса. Например, предсказание, является ли электронное письмо спамом или не спамом. Нейрон может быть обучен на основе входных данных (например, признаков электронных писем) и ожидаемого результата (спам или не спам), и затем использоваться для классификации новых данных.
2. Линейная регрессия: Одиночный нейрон также может использоваться для решения задачи линейной регрессии, где требуется предсказать непрерывный выходной результат на основе входных данных. Например, предсказание цены дома на основе его площади и количества комнат. Нейрон может быть обучен на основе пар входных данных (площадь, количество комнат) и соответствующих выходных значений (цена), и затем использоваться для предсказания цены дома для новых данных.
3. Аппроксимация функций: Одиночный нейрон может использоваться для аппроксимации сложных функций. Например, задача приближения синусоиды или кривой. Нейрон может быть обучен на основе пар входных данных (например, значения x) и соответствующих выходных значений (например, значения sin(x)), и затем использоваться для аппроксимации функции sin(x) для новых значений x.

В каждом из этих примеров одиночный нейрон может быть использован как базовый строительный блок для более сложных нейронных сетей. Однако в реальных приложениях часто требуется использование более сложных архитектур нейронных сетей с несколькими слоями нейронов для достижения более высокой производительности и точности.

# Заключение.

В заключение курсовой работы по нейросетям можно отметить, что нейронные сети являются мощным инструментом в области глубокого обучения и обработки данных. Они обладают способностью извлекать сложные зависимости и выявлять скрытые паттерны в больших объемах информации, что делает их очень полезными в различных приложениях.

В ходе исследования и написания данной работы были изучены основы нейронных сетей, включая их архитектуру, принципы работы и методы обучения. Было обращено внимание на различные типы нейронных сетей, такие как перцептрон, сверточные нейронные сети (CNN), рекуррентные нейронные сети (RNN) и трансформеры. Каждый тип нейронных сетей имеет свои уникальные свойства и применения, что позволяет решать разнообразные задачи, включая классификацию, сегментацию, распознавание образов, обработку естественного языка и генерацию текста.

Основное внимание в работе уделено архитектуре Transformer, которая показала впечатляющие результаты в задачах обработки естественного языка. Transformer представляет собой инновационный подход к моделированию последовательностей данных, основанный на слоях само-внимания и полносвязных слоях. Благодаря своей способности обрабатывать контекстную информацию на разных уровнях, модель Transformer может эффективно моделировать длинные зависимости и генерировать связные и смысловые тексты.

Однако следует отметить, что нейросети не являются универсальным решением для всех задач. Они требуют больших объемов данных для обучения, правильной настройки гиперпараметров и оптимизации моделей, а также вычислительных ресурсов для обучения и вывода результатов. Также важно учитывать этические аспекты, связанные с использованием нейросетей, включая прозрачность, справедливость и безопасность моделей.

В целом, нейронные сети представляют собой захватывающее и активно развивающееся направление в области искусственного интеллекта. Они продолжают привлекать большой интерес и исследования в этой области, и их применение ожидается во многих сферах нашей жизни. Эта курсовая работа позволила глубже понять основы нейронных сетей и их применение, и надеюсь, что она будет полезна для дальнейшего изучения и работы с этой захватывающей технологией.

# Источники

1. Глубокое обучение (Ян Гудфеллоу, Аарон Курвилль, Иошуа Бенджио)
2. Глубокое обучение с использованием Python (Шолле Франсуа)
3. Neural Networks and Deep Learning (Charu C. Aggarwal)
4. GANs in Action: Deep learning with Generative Adversarial Networks (by Jakub Langr, Vladimir Bok)
5. <https://paperswithcode.com/paper/weight-uncertainty-in-neural-networks>
6. <https://paperswithcode.com/paper/neural-architecture-design-for-gpu-efficient>
7. <https://paperswithcode.com/paper/selective-brain-damage-measuring-the>
8. <https://paperswithcode.com/paper/integer-quantization-for-deep-learning>
9. https://habr.com/ru/companies/yandex/articles/672396/
10. <https://paperswithcode.com/paper/pytorch-an-imperative-style-high-performance-1>
11. https://hashdork.com/tesla-neural-network-architecture-explained/
12. https://paperswithcode.com/paper/deep-high-resolution-representation-learning

# Приложение к курсовой работе.

# Исходный код проекта.

#include <iostream>

#include <cmath>

using namespace std;

class Neuron

{

public:

    Neuron()

    {

        weight = 1;

        last\_error = 1.1;

        smoothing = 0.01;

    }

    float get\_last\_error()

    {

        return last\_error;

    }

    void set\_smoothing(float new\_smoothing)

    {

        smoothing = new\_smoothing;

    }

    float get\_smoothing()

    {

        return smoothing;

    }

    float get\_weight()

    {

        return weight;

    }

    float proccess\_data(float input\_data)

    {

        return input\_data \* weight;

    }

    void train(float input\_data, float ExpectedResult)

    {

        float result\_now = input\_data \* weight;

        last\_error = float(ExpectedResult) - float(result\_now);

        float correction = (last\_error / result\_now) \* smoothing;

        weight += correction;

    }

    void training(float task, float ans)

    {

        int iteractions = 0;

        while (check\_training())

        {

            iteractions++;

            train(task, ans);

            cout << "Epoh: " << iteractions << " | Weight: " << get\_weight() << endl;

        }

    }

    bool check\_training()

    {

        if (weight == last\_error)

        {

            return true;

        }

        else

        {

            if (last\_error > smoothing || last\_error < -smoothing)

            {

                return true;

            }

            else

            {

                return false;

            }

        }

    }

private:

    float weight;

    float last\_error;

    float smoothing;

};

int main()

{

    Neuron neuron;

    float task = 1;

    float ans = 1;

    int iteraction = 0;

    cout << "Traning task >>";

    cin >> task;

    cout << "Traning ans >>";

    cin >> ans;

    cout << "Start traning." << endl;

    neuron.training(task, ans);

    neuron.set\_smoothing(0.001);

    int right\_ans = 00;

    while (true)

    {

        cout << "task> ";

        cin >> task;

        ans = neuron.proccess\_data(task);

        cout << "------------" << endl;

        cout << ans << " | " << round(ans) << endl;

        cout << "------------" << endl;

        cout << "Right ans ( 0 for skip ): ";

        cin >> right\_ans;

        if (right\_ans != 0)

        {

            neuron.training(task, right\_ans);

        }

        cout << "Weight: " << neuron.get\_weight() << endl;

    }

}