Муниципальное бюджетное общеобразовательное учреждение

«Средняя общеобразовательная школа № 20»

муниципального образования города Невинномысска

ПРОЕКТНАЯ РАБОТА

**Машинный перевод нейронными**

**сетями архитектуры «Трансформер»**

Автор работы:

\_\_\_\_\_ Андрей

Учащийся 11 «А» класса

Руководитель работы:

Буракова Наталья Сергеевна

Невинномысск

2023-2024

**ОГЛАВЛЕНИЕ**

ВВЕДЕНИЕ**3**

ГЛАВА 1 Введение в нейронные сети**4**

ГЛАВА 2 Принцип работы моделей «Transformer» в задачах NLP**5**

ГЛАВА 3 Настройка оборудования для дальнейшей работы**7**

ГЛАВА 4 Подготовка датасета**8**

ГЛАВА 5 Кодирование слов в векторы**8**

ГЛАВА 6 Архитектура модели**9**

ГЛАВА 7 Тестирование модели**10**

ЗАКЛЮЧЕНИЕ**11**

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ ИНФОРМАЦИИ**11**

Введение

В последнее десятилетие нейронные сети стремительно развиваются. Сегодня они масштабно применяются во многих сферах экономики и общества, они настолько-же привычны как, например, web-страница в интернете, хотя довольно многие не замечают, что в какой-нибудь задаче использовалась нейронная сеть.

В разработку оборудования для обучения и использования искусственного интеллекта, а также исследования новых методов и архитектур, сегодня вкладываются колоссальные суммы. Например, 7 из 10 самых дорогих компаний на данный момент так или иначе связанны с компьютерной архитектурой или разработкой программного обеспечения, а значит многие из них очень активно занимаются исследованием машинного обучения.

Машинное обучение затрагивает многие задачи. Сегодня основными направлениями глубокого обучения являются: компьютерное зрение (CV), обработка естественного языка (NLP), генеративные модели (GAN, диффузионные модели), а также существует множество других, которые тоже активно развиваются. Мой проект будет связан с сетями для машинного перевода текста (перевод с одного языка на другой), я самостоятельно реализую модель нейронной сети и обучу её на довольно большом наборе данных.

**Цель:**

Ознакомить слушателей с работой нейронных сетей и языковых моделей, а также реализовать собственную сеть архитектуры «Transformer» для машинного перевода, показать пример её работы.

**Задачи:**

* Изучить необходимый для работы материал
* Разобраться в поэтапном создании о обучении модели машинного обучения.
* Создать собственную модель и показать пример её работы.

**Этапы работы над проектом:**

* Очень краткое введение в нейронные сети
* Принцип работы моделей «Transformer» в задачах NLP
* Настройка оборудования для дальнейшей работы
* Загрузка и подготовка датасета.
* Принцип кодирования слов (Word2Vec).
* Конструирование и обучение модели
* Тестирование модели

# Глава 1

## Введение в нейронные сети

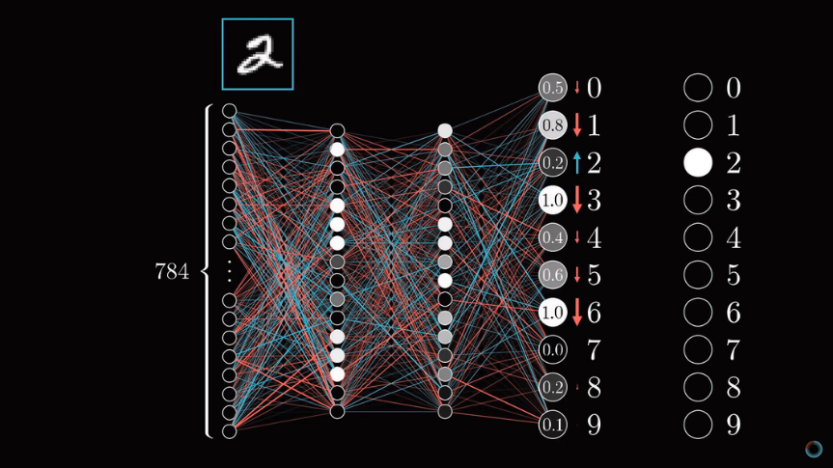
Как ни странно, в «нейронных сетях» нет ни «сетей», ни «нейронов» в прямом смысле этого слова. Их правильнее было бы назвать «обучаемые многоуровневые представления», или «обучаемые иерархические представления», или даже «глубокие дифференцируемые модели». Всё дело в том, что задача нейронных сетей сводится к тому, чтобы с помощью элементарных математических преобразований (умножение, сложение и др.) получить из входных данных, как бы «выделить» выходные данные, которые соответствовали бы ответу на задачу. Именно поэтому модели глубокого обучения так хорошо справляются с задачами компьютерного зрения, распознавания голоса, «понимания» текста, потому что они как бы «выделяют» ту полезную информацию, которую несёт в себе поток входных данных.

Рисунок 1 - Графический пример работы нейронной сети

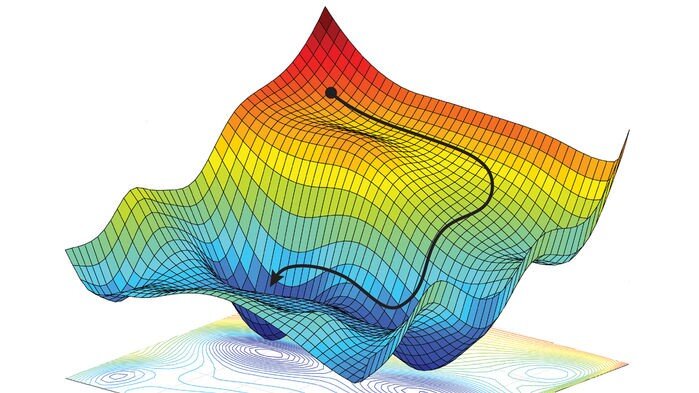
Итак, если работа сводится только к многократному перемножению матриц (рисунок 1), то как они обучаются хорошо справляться с задачами? На самом деле все современные нейронные сети обучаются при помощи *обратного распространения ошибки*, называемого «Градиентный спуск» (рисунок 2). Обучаясь на большом количестве тренировочных образцов (иногда даже больше десятков миллионов) они сравнивают свои ответы с *эталонными* ответами и считают величину своей ошибки, работая в *обратную* сторону, они находят для каждого параметра величину, которую он внёс либо в саму ошибку, либо для её предотвращения (словно находя миллионы производных для заданных параметрами точек). И по найденным величинам корректирует каждый параметр так, чтобы на данном образце он выдавал меньшую ошибку, именно поэтому для обучения нужно так много образцов, чтобы сеть запомнила основные признаки в тренировочных данных, а не сами данные (тогда бы она ни на что не годилась).

Рисунок 2 - Графическое изображение градиентного спуска

Почему же глубокое обучение начало развиваться относительно недавно? Этому поспособствовали многие важные факторы:

* Многократное увеличение вычислительных способностей (современные видеокарты GPU, тензорные процессоры TPU, крупные *датацентры* (рисунок 3) и др.)
* Появление больших датасетов из огромного объёма данных со всего интернета

Рисунок 3 - Современный дата-центр

* Разработка библиотек и фреймворков, которые многократно ускоряют разработку и сильно снижают порог входа в сферу разработки
* Активное вложение денег развития отрасли и быстрое внедрение технологий в экономику

# Глава 2

## Принцип работы моделей «Transformer» в задачах NLP

Обработка текстов на естественном языке (Natural Language Processing, NLP) — общее направление искусственного интеллекта и математической лингвистики. Оно изучает проблемы компьютерного анализа и синтеза текстов на естественных языках. Существует множество видов подзадач для обработки естественного языка, машинный перевод – одно из ключевых направлений. Текст – это последовательность данных, а для обработки последовательности существует 3 вида нейронных сетей:

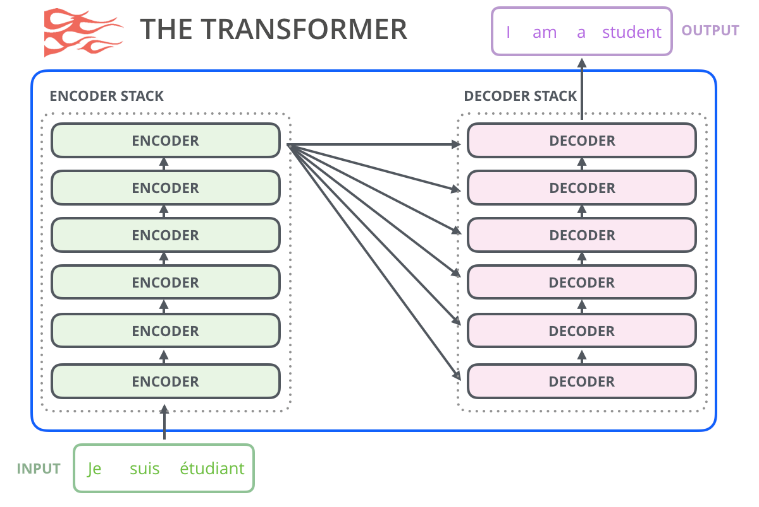
1. Свёрточные (CNN, Conv) (5) – для обобщения текстов подходят плохо, но всё ещё используются для нестандартных задач.
2. Рекурентные (RNN) – обрабатывают последовательности шаг за шагом, обладают краткосрочной памятью, до 2017 года были лучшими в обработке текста.
3. Трансформеры (Transformers) (1,2,3,4) – новая архитектура, которая занимает первое место в обработке текстов, её главное преимущество — это то, что она смотрит на весь текст целиком и может улавливать контекст. На основе трансформеров работают такие популярные сети как Chat-GPT, Google Translate, BERT и др. (рисунок 4)

Рисунок 4 - Схема модели трансформера

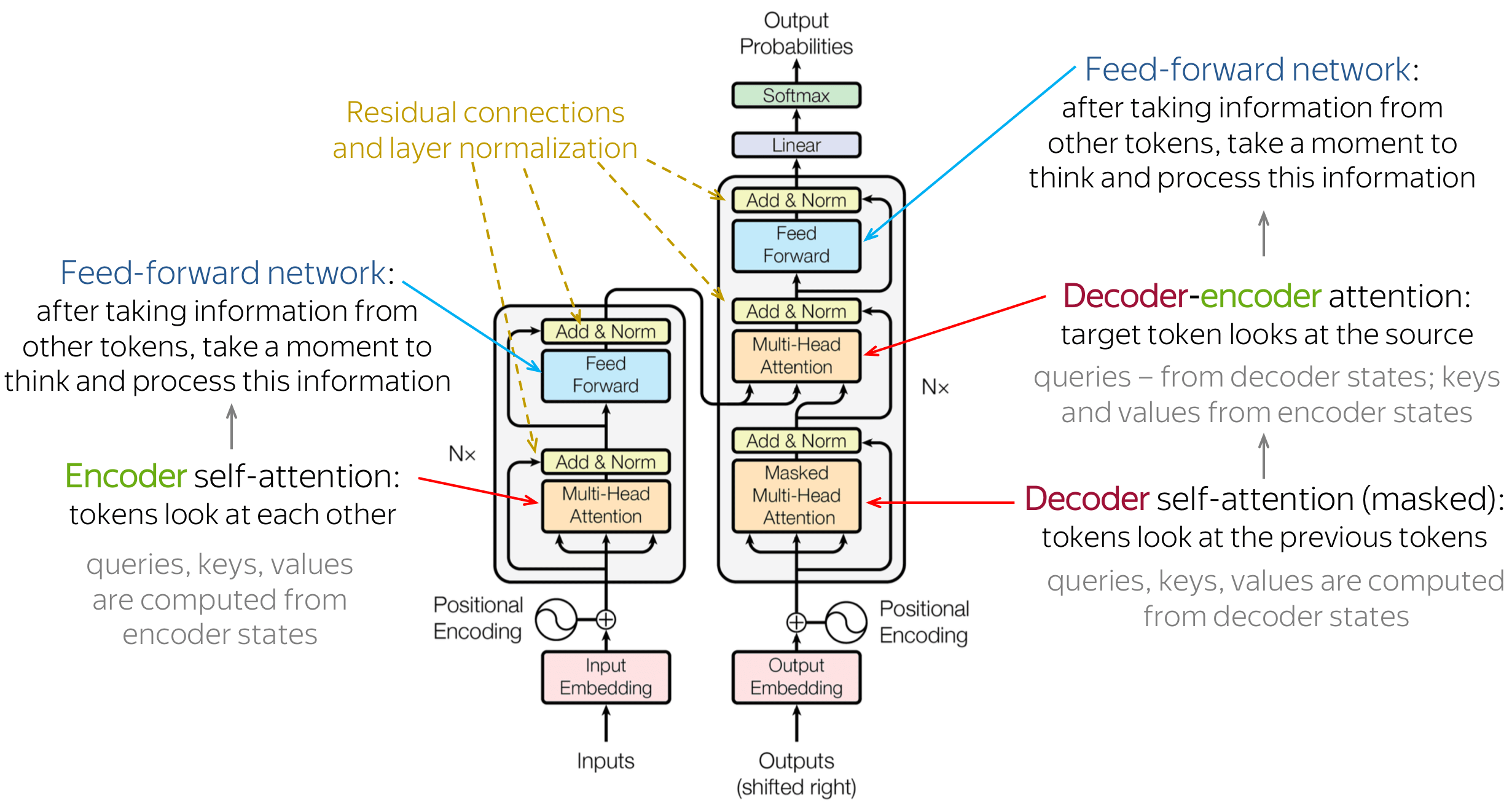
Трансформер (рисунок 5) содержит в себе 2 основных блока – это *энкодер* и *декодер*. Они довольно схоже работают, однако декодер берёт дополнительно закодированную информацию, а также имеет в составе фильтр, который не даёт ему обрабатывать следующие по ходу написания слова. В блоках применяются такие слои как: остаточные связи (для более *плавного* градиента), нормализация вывода каждого слоя (для большего обобщения) и самый классический полносвязный слой сразу за слоем внимания (для большей *неровности* в контексте). Основа всей этой архитектуры – это довольно современный «*Механизм внимания*», когда нейронная сеть, имея кокой-либо контекст, может отдавать *предпочтение* более значимым признакам (в нашем случае словам). Именно поэтому трансформеры так хорошо улавливают контекст в цепочке слов, они многократно связывают между собой разные слова.

Рисунок 5 – Схематическое строение двух блоков трансформера (по слоям нейронной сети)

В нашем случае архитектура будет следующей:

1. Первый блок энкодера будет принимать закодированные вектора слов исходного предложения (о векторах слов сказано в 5 главе), и искать связи между разными словами.
2. Далее он предаст результат другому блоку энкодера, и так по цепочке, пока все блоки не выделят *смысл* этого предложения.
3. Дальше блоки декодера будут по цепочке принимать закодированные вектора слов того, что он уже успел перевести, и, пользуясь полезной информацией от энкодера, будет предсказывать следующее слово в переводимом предложении.

# Глава 3

## Настройка оборудования для дальнейшей работы

Для работы с нейронными сетями нужен достаточно мощный персональный компьютер, либо аренда вычислительных ресурсов в облаке. Вот рекомендуемая конфигурация:

* GPU Nvidia RTX – очень сильно ускоряет работу нейросетей в сравнении с процессором (~50-100 раз)
* Современный CPU
* 16-32 GB оперативной памяти DDR4 – датасет частично хранится в памяти во время обучения
* 50 GB свободного места – для экспериментов с датасетами
* Операционная система любая, у меня Windows (хотя для машинного обучения часто используют Linux)

Для начала работы также необходимо установить определенное совместимое программное обеспечение, иначе не получится начать работу над созданием собственной сети (в скобках версии, установленные у меня):

* Python (3.10) – лидирующий язык программирования для машинного обучения
* CUDA (11.4) и cuDNN (8.9.3.28) – очень важные отдельные библиотеки для работы с видеокартой и ускорения обучения путём оптимизации
* NumPy (1.23.5), Matplotlib, SciPy, Pandas – стандартный набор библиотек (на Python) для машинного обучения.
* TensorFlow (2.10.1) и Keras (2.10) – сами фреймворки (на Python) для создания, обучения и использования нейронных сетей.
* Keras-nlp, Keras-tuner, TensorFlowText – дополнение к фреймворкам, необходимые для конкретно нашей задачи.
* Jupyter Notebook – web IDE и эффективная среда разработки для машинного обучения
* IDE JetBrains PyCharm (Professional or Community) – лучшая среда разработки для Python

Также в процессе запуска моделей могут понадобиться и другие библиотеки, совместимые с ключевыми для работы фреймворками.

# Глава 4

## Подготовка датасета

Датасет – один из самых важных компонентов для обучения нейросети. Это большой набор из обучающих образцов: картинок, текста или звука - в зависимости от задачи. От его качества, обширности и количества образцов – напрямую зависит качество итоговой модели.

Для своего исследования я использовал два разных датасета:

* [WMT15 English-French](https://www.kaggle.com/datasets/dhruvildave/en-fr-translation-dataset) (~9 GB) – большой англо-французский датасет переводов, содержащий примерно 21 миллион обучающих образцов
* [YandexCorpus En-Ru](https://disk.yandex.ru/d/48MZIaUoGzMOkA) (~300 MB) - средний англо-русский датасет переводов, содержащий 1 миллион обучающих образцов

Также в коде проекта идет процесс частичной загрузки данных в память, предобработка текстов, обработка будущих пакетов и распараллеливание нагрузки между ядрами процессора – весь этот функционал предоставляет класс tf.data.Dataset из фреймворка TensorFlow.

# Глава 5

## Кодирование слов в векторы

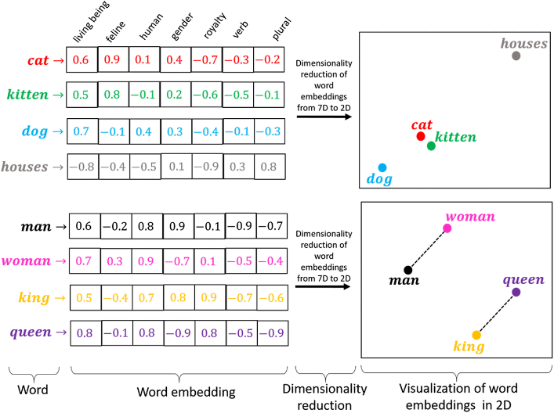
Как так текст не является числовыми данными, нейронные сети не могут его обработать при помощи математических преобразований. Поэтому текст необходимо представить в виде чисел или векторов. На самом деле возможно передать смысл текста при помощи векторов, так как существует множество таких комбинаций векторов, которые могут передать полезную информацию, такую как пол, количество, синонимичность, группу и некоторые другие (рисунок 6). Этот непростой процесс проходит в несколько этапов и может иметь разные реализации:

Рисунок 6 - Кодирование слов

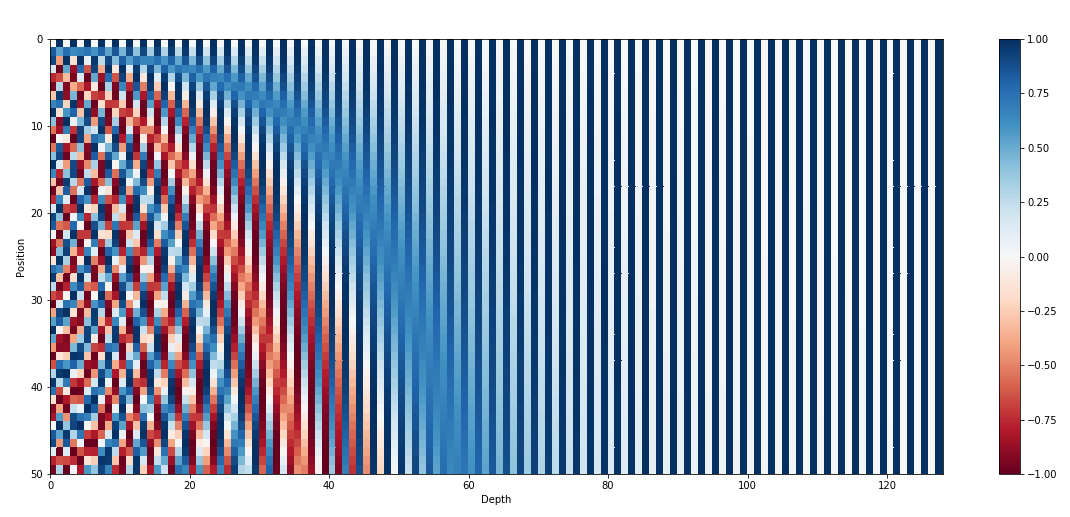
1. Приведение текста в нижний регистр, удаление ненужных символов, *лемматизация* (приведение слов в начальную форму, т.к. «яблоко» и «яблока» имеют очень схожий смысл), разбиение текста на токены (обычно отдельные слова, но могут быть и символы, и предлоги, и группы слов).
2. Токенизация – выборка самых частых токенов и присвоение им уникального индекса.
3. Создание модели векторных представлений слов (таких как Word2Vec или GloVe), либо добавление к нашей модели Embedding слоя (является полносвязным слоем), который сам при обучении создаст для каждого токена (слова) совой вектор.
4. (Только для архитектуры Трансформер) Необходимо также закодировать позицию каждого слова в предложении, потому что сеть не будет различать его положение самостоятельно. Поэтому нужен либо дополнительный Embedding слой, либо прибавлять к векторам метку, например *SinePositionEncoding* (рисунок 7), которую использовали авторы оригинальной статьи (4).

Рисунок 7 - Позиционное кодирование

# Глава 6

## Конструирование и обучение модели

После всех проделанных шагов осталось сконструировать нашу модель. Готовые блоки энкодера, декодера и позиционного кодирования есть в библиотеке keras-nlp, поэтому в их реализации нет необходимости.

Фреймворк Keras имеет удобный функционал для создания и обучения моделей, поэтому процесс обучения и контроль метрик проходит очень просто. Чтобы модель достигла лучших результатов нужно подобрать правильные гиперпараметры, делается это простым перебором разных значений для достижения лучших результатов. Гиперпараметры данной модели: шаг обучения, размер пакета, оптимизатор, количество блоков и головок внимания в них, размер вектора кодирования, коэффициент отсева, размер полносвязного слоя в блоках.

Это самый затратный по времени этап работы, так как обучение должно занимать в нашем случае минимум 30 минут, а лучше всего будет продолжительно обучать ее, вплоть до нескольких суток. Весь код с комментариями будет прикреплен к проекту.

Наиболее удачная конфигурация модели обучалась 70 эпох (по 1000 пакетов в каждой эпохе), прошло времени: 7000 сек или ~2 часа, вот ее характеристики:

* Размер пакета (батча) 64 образца для одного прохода
* Шаг обучения 0,00025, само-понижающийся
* 2 блока энкодера и декодера
* 8 независимых потоков внимания в каждом блоке
* Размерность вектора для токена 256
* Всего токенов 20000
* Длина предложений обрезалась или дополнялась до 40 токенов

# Глава 7

## Тестирование модели

Качество перевода текста ожидаемо не самое лучшее, из-за множества причин: модель относительно не большая, *словарный запас* модели очень маленький, время обучения и мощность оборудования недостаточная, обработка текста довольно грубая. Поэтому сравнивать с переводчиками от Google и Яндекс нет смысла.

Вот три примера для тестового набора данных, того на котором модель никогда не обучалась:

Пример 1:

Input: [START] lake [UNK] is only one mile away . [END]

Predict: [START] [UNK] [UNK] находится находится всего в в минутах минутах ходьбы ходьбы . от [END]

Target: [START] этот семеиныи отель расположен в самом центре города [UNK] . [END]

Пример 2:

Input: [START] the services which must be included in this new declaration are solely the services which will be taxed for vat purposes in the other ms . services which are supplied to persons established outside the eu must not be included . [END]

Predict: [START] услуги [UNK] , , которые которые должны должны быть быть игр природных только услуг по уровне отношению [UNK] к к [UNK] другим китаиского . мира службы . , [END]

Target: [START] услуги , которые должны быть включены в эту новую декларацию , являются исключительно услугами , которые будут [UNK] налогом для целеи [UNK] , в другои [UNK] . [END]

Пример 3:

Input: [START] output files from the [UNK] proxy application are also stored as a proprietary format , and can only be opened using [UNK] dynamics [UNK] toolkit . [END]

Predict: [START] [UNK] [UNK] [UNK] [UNK] также также [UNK] [UNK] в формат формате [UNK] , и а могут также быть быть [UNK] [UNK] с [UNK] помощью [UNK] [UNK] [UNK] [UNK] . . [END]

Target: [START] выходные фаилы от [UNK] прокси приложение также хранятся в [UNK] формате и может быть открыт только с помощью [UNK] dynamics [UNK] [UNK] . [END]

Из вывода можно понять, что Input - входные данные, Predict - предсказание модели, а Target – предполагаемый перевод. Также здесь присутствуют токены начала и конца вывода - [START] и [END], и неизвестных слов (не вошедших в словарь токенов) - [UNK].

Заключение

Подводя общие итоги проекта, можно сделать вывод, что нейронные сети — это перспективная технология, которая активно развивается. И её правильное изучение не является слишком сложным.

Хоть у нас и не получилось создать модель с качественным переводом по определенным причинам, целью проекта было ознакомить аудиторию с работой нейронных сетей, и реализовать собственную модель с современной архитектурой «Transformer», используемой в больших языковых моделях.

Модель реализованную в ходе работы легко применять благодаря развитым фреймворкам. Существует несколько форматов, чтобы сохранить веса модели в файл, который в дальнейшем легко парой строк кода можно загрузить и использовать модель на другом устройстве, например на сервере вебсайта.

Также нужно отметить, что эту модель можно оптимизировать для работы на любых устройствах, особенно маломощных например - смартфонах, почти без потери точности. Вот примеры этих методов:

1. Квантование весов (Quantization) – ускоряет вычисления путем уменьшения точности обрабатываемых типов данных.
2. Отсечение весов (Pruning) - устранение части параметров нейронной сети, почти не влияющих на точность работы.
3. Обучение легкой модели на основе более тяжелой (Knowledge distillation) –модели быстрее обучают другие модели по сравнению с обучением с нуля.
4. Фреймворки оптимизации (например TensorRT или OpenVINO) – ускоряют вычисления путем уменьшения гибкости в работе с моделью и оптимизации под конкретное оборудование.

Ссылка на работу в Colab: (<https://drive.google.com/file/d/1dLxvRoxdZdc20hSxB-DBY9L9W2-zgNX1/view?usp=drive_link>)

# Список использованных источников информации

1. Подробнее об архитектуре трансформер: <https://habr.com/ru/articles/486358/>
2. Пример с сайта tensorflow: <https://www.tensorflow.org/text/tutorials/transformer>
3. Пример с сайта kaggle: <https://www.kaggle.com/code/mateuszk013/neural-machine-translation-with-attention/notebook>
4. Статья с применением архитектуры: <https://arxiv.org/pdf/1706.03762.pdf>
5. Подробнее о свёрточных нейронных сетях: <https://habr.com/ru/articles/456186/>
6. Официальный сайт tensorflow: <https://www.tensorflow.org/overview>
7. Официальный сайт keras раздел keras-nlp: <https://keras.io/api/keras_nlp/>
8. Официальный сайт keras: <https://keras.io>