Ссылка на репозиторий проекта: https://github.com/maksiam/Transformer-Decoder/

Описание набора данных

Так как целью нашего проекта было создание генератора текста похожего на книги о Гарри Поттере, сначала мы решили использовать все оригинальные книги от Дж. К. Роулинг по Поттериане. Однако в последствии выяснилось, что такого количества данных недостаточно и поэтому было принято решение дополнить наши данные наиболее популярными фанфиками по Гарри Поттеру из данной статьи.
Канонические книги о Гарри Поттере:

- “Гарри Поттер и философский камень” (1997), Дж. К. Роулинг
- “Гарри Поттер и Тайная комната” (1998), Дж. К. Роулинг
- “Гарри Поттер и узник Азкабана” (1999), Дж. К. Роулинг
- “Гарри Поттер и Кубок огня” (2000), Дж. К. Роулинг
- “Гарри Поттер и Орден Феникса” (2003), Дж. К. Роулинг
- “Гарри Поттер и Принц-полукровка” (2005), Дж. К. Роулинг
- “Гарри Поттер и Дары Смерти” (2007), Дж. К. Роулинг

Фанфики по вселенной Гарри Поттера:

- “Однажды двадцать лет спустя” @Alteya
- “Время толерантности” @flamarina
- “Дочь Волдеморта” Алевтина Варава
- “Нормальные герои всегда идут в расход” @katss
- “Несмотря ни на что” Натали Поттер
- “Кукольные человечки” @Nilladell
- “Паутина” Сфинкс
- “Команда” @Tansan
- “Гарри Поттер и методы рационального мышления”, Элиезер Юдковский

Предварительная обработка данных

Для корректной работы модели трансформера не требуется серьезной предварительной обработки данных, так как основная идея проекта заключается в воспроизведении текста без изменений. Однако была проведена проверка текста на наличие возможных артефактов после извлечения из разных форматов. В случае чтения из PDF была исправлена проблема с лишними пробелами.

Препроцессинг текста и токенизация

Основной этап препроцессинга состоит в преобразовании исходного текста в токены. В данном проекте используется токенизатор "sberbank-ai/rugpt3large_based_on_gpt2" из библиотеки transformers, так как он широко используется в подобных задачах и ориентирован на русский язык. Jupyter-ноутбуки с примерами расположены в notebooks/data_explore.ipynb и notebooks/data_preproc.ipynb

Использование системы DVC для работы с данными

В текущем проекте мы используем систему DVC (Data Version Control) для работы с данными. Эта библиотека значительно упрощает работу с большими объемами данных, которые нельзя загрузить на GitHub. С помощью DVC любой человек, желающий воспроизвести наш проект, может установить все сырые и обработанные данные, а также модели из S3 хранилища всего одной командой - dvc pull. Команды dvc add и dvc push используются для добавления новых файлов для отслеживания DVC, добавления их в .gitignore и создания конфигурационного файла, с помощью которого DVC извлекает необходимые данные и отправляет их в S3 хранилище.

Постановка задачи и построение модели

Главная задача нашего проекта - создать модель Transformer Decoder, которая будет генерировать текст, похожий на книги о Гарри Поттере. Для выполнения этой задачи необходимо ясное понимание того, как устроена структура моделей Transformer и почему она работает.

Освоение модели Transformer и сбор информации

Процесс освоения данной модели мы вели в телеграм канале для лучшего усвоения материала. Все полезные ссылки и идеи мы собирали в отдельном документе.

Имплементация модели в код

После освоения идеи трансформеров, мы приступили к ее имплементации в код. Для этого мы использовали фреймворк PyTorch, так как он позволяет гибко строить нейронные сети и поддерживает обучение на GPU.Код модели можно найти по пути src/models/model.py. Также для воспроизводимости модели мы использовали библиотеку Poetry, которая берет на себя работу с зависимостями, чтобы любой человек мог установить те же библиотеки и запустить модель без конфликтов. poetry install - для установки зависимостей, poetry shell - для перехода в виртуальную среду с установленными зависимостями.

Настройка серверов и инструментов для обучения модели

Так как модель содержит большое количество весов, мы арендовали сервера с поддержкой GPU и CUDA, на которых развернули наш проект и проводили обучение трансформера. Для отслеживания артефактов и параметров модели мы арендовали еще один сервер, на котором развернули MLflow с помощью Docker. Мы также связали MLflow с базой данных Postgresql для хранения параметров и с minio для хранения артефактов в S3 бакете. Результаты обучения и параметры модели можно посмотреть по данной ссылке: http://188.225.84.65:5000/#/experiments/1.

Проблемы при разворачивании проекта на сервере с GPU

В процессе разворачивания проекта на сервере с GPU мы столкнулись с несколькими проблемами:

1. Отсутствие допуска к apt (Решение)
2. Установка pip для нужной версии Python (Решение).
3. Установка PyTorch с поддержкой CUDA 11.6 (Решение).

Улучшение модели и добавление данных

В процессе обучения и подбора параметров мы поняли, что чем больше текста мы используем для обучения, тем лучше качество модели. Мы экспериментировали с увеличением числа слоев, голов, размеров блока (block_size) и размера пакета (batch_size), и это действительно улучшало качество модели. Однако, мы столкнулись с ограничением памяти, которое быстро исчерпывалось. Чтобы справиться с этой проблемой, мы решили обратиться к фанфикам по Гарри Поттеру и добавить их в обучающие выборки. Это решение помогло уменьшить потери (loss), однако из-за ограниченного времени, высокой стоимости аренды GPU и длительности обучения модели, мы решили остановить наши исследования. Мы сохранили лучшую модель (models/model.pt), загрузили ее через DVC в S3 бакет и создали скрипт для генерации текста (src/models/predict_model.py).

Оценка модели и результатов

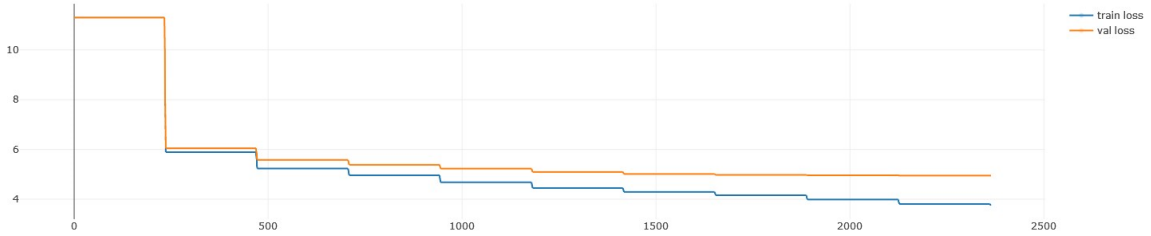
Ниже приведены параметры нашей лучшей модели и ее метрики на обучении и валидации.

Name	Value
BATCH_SIZE	32
BLOCK_SIZE	128
DEVICE	cuda
DROPOUT	0.3
EVAL_INTER	500
LEARNING_RATE	0.0003
MAX_ITER	5000
NUM_EMBED	1024
NUM_HEAD	8
NUM_LAYER	8
No. params	203.854929
decode	<function decode at 0x7fd64f637d30>
encode	<function encode at 0x7fd64f637af0>
estimate_loss	<function estimate_loss at 0x7fd64f637ee0>
get_batch	<function get_batch at 0x7fd64f637dc0>
save_model_to_chekpoint	<function save_model_to_chekpoint at 0x7fd64f63c040>

В качестве метрики мы использовали [Cross-Entropy Loss]

Metrics (2)

Name	Value
train loss	3.751
val loss	4.949



Пример вывода модели:

```
<pad> Диггори зашиф, и вид из камина Министерство едва сдерживала отдельные линии огня. Когда они свернули со уголков, о
ни уже прошли сквозь проём отправились в гостиную, кори до края острова, за ним закрылась заклятием и как только Седрик,
 простирался сквозь смех, что Флёр Делакур. — Выпил кнул Гарри, юноша, Долохов. — Гарри! — Большие круглые её сзади, сна
бжённых пересчитал пять шагов вперёд. — Спасибо, Молли! —
(transformer-py3.9) ubuntu@transformeraddhp:~/Transformer-Decoder$
```

Как можно заметить на картинке выше, в ходе работы над проектом удалось достигнуть неплохого результата. Модель выводит достаточно связный, читаемый текст. Это стало возможным благодаря сложной структуре модели и большого количества данных для обучения. Наличие dropout слоя помогает бороться с переобучением, а оценка модели каждые 500 итераций позволяет контролировать процесс обучения и проверять производительность модели. С учетом ограниченного количества ресурсов и времени обучения модели получилось достичь, как нам кажется, очень неплохих результатов. Однако все еще есть куда стремиться и улучшить качество выводимого текста можно следующими способами: увеличение вычислительных мощностей вместе с параметрами модели, увеличение материала для обучения, fine-tune готовых LM или LLM.