***Доклад***

**Архитектура**

Имплементирал съм модел от семейството на трансформърите.

Той съдържа само декодъри понеже:

* Имаме задача, за догенериране текста на английски и после преводът му на български. Това предвещава да третираме входните данни като 1 редица, а не 2 отделни редици, както е описано в “Attention is all you need”.
* Щом ще имаме 1 входна редица, то можем да използваме GPT архитектура.

Моделът приема бачове от tokenized редици, и ги падва за са изгради нормален тензор (ако такъв не е подаден).

**Tokenization**

Използвам BPE (Byte Pair Encoding) за избирането на кои subwords да използвам за тоукени.

Пред избрал съм размера на речника да е **vocab\_size=8\_000**. Това е базирано на статиите, посочени в условието, и на мощността на компютърът ми да изкара една тренировка с този размер. С този размер пускаме алгоритъма и накрая получаваме token-ите и съответно merge таблицата, която после се използва в последствие за tokenization на изреченията.

Имплементацията ми се различава от главния алгоритъм с това, че не позволявам нито на скритите тоукъни (за начало, край, …), нито на white-space символите, нито на пунктуационните символи да се обединяват с други символи в по-дълги тоукъни. Този избор е базиран, на факта примерно, че ако някой тоукен представлява често срещана представка, ако пред нея слеем с празно място, то ще ограничим модела думите винаги да започвам с тази представка, докато може и да има по-сложни думи, които да започва с още една представка пред първата. По-този начин ограничаваме алгоритъма една идея по-семантично да разделя теста на морфеми. От друга страна, имаме отделен тоукен за празно място, който се среща много често; това увеличава дължината на редиците в тоукъни спрямо базовия алгоритъм.

**Embedding Layer**

Всеки token от редиците се замества с вектор отговарящ на token-a, взет от Embedding матрицата. Това е прост look-up по индекси.

**Positional Encoding**

След това към всеки вектор се добавя отместване базирано на позицията на съответстващия token. Използваме схемата, представена в “Attention is all you need”.

**Decoder blocks**

Имаме 5 такива блока. Данните минават линейно пред тях. Всеки декодър блок съдържа:

* МultiheadAttention layer
  + 8 Attention глави.
  + Подават му се тензора от предходния блок или от Embedding layer-a за Query, Key, Value.
  + Подават се key\_padding\_mask и causal attention mask, за пропускане на изчисленията на корелацията със Padding tokens и future tokens.
  + Не се взимат изчислените тегла за ключовете, защото не се използват в останалата част на архитектурата.
* Residual connection – с входа на блока
* Layer Normalization
* Feed Forward Block:
  + 2 Linear Blocks с Rely activation между тях, и с Dropout слой накрая и на двата.
  + Разширяват размерността на векторите от 512 до 1536. Тук реално модела “заучава” информация по време на тренировката.
* Процента за Dropout-a е съставен от аргумента на декодър блока и сумарно от двата Dropout-а е аргумента.
  + Layer Normalization

**Linear layer**

Накрая данните се подават на линеен слой който преоразмерява векторите до vocab\_size. Резултатът третираме като logits за Softmax layer-a

**Inference mode**

Logits за последния тоукен в редицата се подават на Softmax, който ни връща вероятностното разпределение за следващия тоукен. Преди да се подадат, те се разделят на temperature=0.2 на модела, който променя вероятността на модела да избере най-вероятния следващ token. Избора е през multinomial дистрибуция.

**Training mode**

Всяка редица с дължина L реално представлява L – 1 екземпляра за трениране. Затова, ние като input приемаме редицатите без последния им елемент и като target приемаме редиците без първия им алгумент. Logits за всички тоукени в input-a се подават заедно с target, предвалително flattened, на Cost функцията.

**Трениране**

**Tokenizer**

Отне около половин-един час

**Model**

Моделът е трениран за 14 епохи върху training dataset-a, разделен на бачове по 16 екземпляра, като dataset-a бива shuffle-нат преди всяка епоха.

На всеки 10 итерации се логва loss-a на модела върху текущия бач (от training set-a).

На всеки 2000 итерации се логва перплексията на модела върху dev-сета.

**Loss/Cost function**

Използвам CrossEntropyLoss както е описано в “Attention is all you need”.

Отделните оценки на екземплярите се осредняват за numerical stability.

Овен това прилагаме label\_smoothing=0.1 къв loss-a, както е описано в “Attention is all you need”.

**Optimizer**

Използвам Adam optimizer с β1=0.9 и β2=0.98 и ε=10-9, както е описано в “Attention is all you need”.

Основния Learning rate е 0.001, като в началото се клипва отдолу с 0.0005, понеже той се варира по начинът описан в “Attention is all you need” за **warmup\_steps=4000**. В началото, рейта расте линейно (преди клипването), а след това намаля експоненциално към 0.

**Всички параметри**

vocab\_size = 8\_000  
max\_seq\_len = 1532  
num\_blocks = 5  
d\_model = 512  
num\_heads = 8   
dropout = 0.1  
d\_ff = 1536  
label\_smoothing = 0.1  
  
  
learning\_rate = 0.001  
batch\_size = 16  
clip\_grad = 5.0  
max\_epochs = 10  
warmup\_steps = 4000  
  
log\_every = 10  
test\_every = 2000  
  
temperature = 0.2

**Експерименти**

Пробвал съм около 4-5 тренировки. Тренирал съм както в Google Colab, Kaggle Colab, така и локално.

В тях пробвах:

* num\_blocks да е 5 или 6
* d\_ff да е 2048 или 1536
* batch\_size да е 32 или 16. Това указваше влияние при трениране в облака заради ограниченията по GPU памет в безплатния вариант на услугите.

Направих O(n2) вариант на generation-a, но на практика е по-бавно от наивния O(n3) вариант, затова на края реших да ползвам наивния.

**Крайни Резултати**

И двете метрики са изчислени върху тест сета.

Перплексия: 8.406167109942377

BLEU: 37.17

**Как да ръннем проекта**

Почти всичко е както в условието, само в началото вместо prepare, имаме:

* python run.py train\_tokenizer
  + Изисква да има поддиректория model/tokenizer
  + Тренира тоукънайреза.
* python run.py tokenize\_corpus
  + Изисква да има поддиректория data
  + Тоукенизира корпусите, които са ни дадени, и запазва резултатите в pickle файлове.

**Източници**

<https://huggingface.co/learn/nlp-course/en/chapter6/5>

<https://medium.com/nearist-ai/word2vec-tutorial-the-skip-gram-model-c7926e1fdc09>

<https://medium.com/@hunter-j-phillips/positional-encoding-7a93db4109e6>

<https://marinafuster.medium.com/cross-entropy-loss-for-next-token-prediction-83c684fa26d5>

<https://pytorch.org/get-started/previous-versions/#linux-and-windows>