HW3 – The One with the Transformer

For this assignment you'll have to train a "GPT" like Transformer for yourself, and generate Shakespeare plays. The end-goal of this homework is to understand step by step how a training and inference pipeline looks when working for transformers. End-value model performance is not the key metric that we're looking for; given a slew of optimisation and design choices go into building transformer models.

Dataset

The dataset you'll be using is a collection of Shakespeare plays. You will be required to do a minimal amount of preprocessing in order to build your entire corpus. You have several options when doing so but here is our general recommendation:

- Create a long list of play-lines which you can treat as an isolated text sample, or as a sequence of dialogues.
- Keep track of which person says a particular line

Additionally, if you find the time to do so, experiment with keeping track of the title of the play as well.

Tokenization

During this homework, you'll have to compare your results with **two tokenization approaches**:

- **Character level tokenizer** - You will have to implement your own tokenizer that treats each individual character (including punctuation and spaces) as a singular token. – i.e. "Anne," would be tokenised to ["A", "n", "n", "e", "," ," "]
- **Subword tokenizer** - You can use any pre-computed tokenizer you can find online. For this step we recommend either the Hugging Face GPT2 one or the Llama-2 tokenizers.

Remember to make sure that when you are performing tokenization you should include *Start of Sequence* and *End of Sequence* tokens.

- For example: [Anne, has, apples] should be tokenized to [<SOS>, Anne, has, apples, <EOS>], if "Anne has apples" is an individual sample. This is solely meant to highligt the insertion of these two special tokens, not how the precise tokenization should look like.

Model

The task is to implement a classic **decoder-only transformer network** using the following building blocks that you have to implement yourself:

- **Positional Embeddings** - Use sinusoidal embeddings
- **Multi-Head Attention** - Implement the classic multi-head attention operation with the following constraints:
  - 1. Auto-regressive masking on the attention logits, i.e. you should be applying a lower triangular mask to your attention scores
  - 2. Use only matrix-multiplies when doing the Query, Key, Value and Output projections. You should declare these as standalone `nn.Parameter` parameters instead of `nn.Linear` layers and do the operations by hand. Do not use einstein summation tricks or operators such as `torch.einsum`.
- **Feed Forward Network** - Use a simple feedforward network with 2 layers and the non-linear activation in-between them. The intermediate size should be larger than the input size
- **Layer Normalization** - Reimplement the layer norm operation. Apply Layer Normalization before the Attention and Feed Forward components.

We provide the following paper as an operation stacking reference. **You should be implementing the `PreLN` flow `(Figure.1 b)`.**

The transformer model will be trained with a causal language model objective. This looks like following:

- INPUT - [“<SOS>”, “A”, “n”, “n”, “e”]
- OUTPUT - [“A”, “n”, “n”, “e”, “<EOS>”]

In order to optimize this, you should be using the `CrossEntropyLoss` which has as a target the corresponding token in the vocabulary for the output sequence.

Minimal example:

```
sample_toks_ids = np.random.randint(0, 100, (100,)).to_list()
sample_toks_ids = [vocab.get("SOS")] + input_seq + [vocab.get("EOS")]
input_toks_ids = sample_toks[:-1] # (BATCH_SIZE, SEQ_SIZE)
output_toks_ids = sample_toks[1:] # (BATCH_SIZE, SEQ_SIZE)

hidden_states = MagicModel.transformer(input_toks) # (BATCH_SIZE,
SEQ_SIZE, HIDDEN_DIM)
output_logits = MagicModel.linear_lm_head(hidden_states) # (BATCH_SIZE,
SEQ_SIZE, VOCAB_SIZE)
loss = CrossEntropyLoss(output_logits, output_toks_ids,
num_classes=VOCAB_SIZE)
```

We recommend you first find a working model configuration that works for you. After that you have to train two variants:

- **a small one** - should be similarly sized to first working model
- **a large one** - where you increase the number of layers / hidden dimension / attention heads/ etc.

Evaluation. What does “model works” mean ?

The qualitative evaluation of the model follows the ability of your model to (1) generate correct English syntax, (2) Have an interplay of characters that have lines (i.e. the text should not be the monologue of a single character).
For reference, you can find below examples of two degrees of model performance:

- You're getting there.
  As censtharyoty, myof n, moo doy, Bofoye
- It works.
  GLEORKEN VINGHARD III: Whell’s the couse, the came light gacks, And the for mought you in Aut fries the not high shee

Quantitative evaluation

We recommend you monitor **perplexity**, on a **hold-out set**. Perplexity can vary based on the amount of tokens in your vocabulary, therefore we generally advise to look-out for values in the ballpark range 1.6 – 3.6. Lower is generally better, and at around 4.0 the generated text might spell correct words, but not make sufficient sense.

We know that a configuration of 6 layers: a feature dimension of 384, and 6 transformer layers with 6 attention heads can produce decent results. You can explore similar configurations.

Testing

We generally recommend you keep any where in between 10 – 20% of your corpus as a testing set. Be careful with data leakage (i.e. training data or patterns ending up in the test set).

For evaluation process we suggest you use an online and an offline metric:

- Online - **Perplexity** for the predicted next token on your hold-out set. You use teacher forcing for input in the model, and just evaluate the next tokens individually (you don’t roll-out the sequence with the model)
- Offline - **BLEU / ROUGE** for unrolling a set of sequences. Ideally you should do this on the entire dataset, but for time and compute purposes you can just subsample a predefined set. Then, you select a set amount of starting words (say the first 5-10) from each sample and unroll the model with them. You evaluate the **BLEU / ROUGE** scores on the remaining “completition ground truth”
- You should unroll either until you reach the <EOS> token, or you reached your **maximum generation context**. – You can limit your ground-truth length to your maximum generation context. (i.e. you only test on chunks of text of a maximum size)

What is the Generation Context?

Although the attention operation can scale infinitely, for computational reasons we recommend you use a maximum sequence size of anything in between 256 - 1024 tokens.

- This means that the maximum input tokens in your model will be your chosen *CONTEXT* value (e.g. 256)
- When you generate a sequence you can attend to at most *CONTEXT* tokens.
  - If you don't reach <EOS> in *CONTEXT* tokens, start taking tokens out 1-by-1 from the context.
    - Example: You have a context of 4 and you've generated so far ["<SOS>", "Anne", "has", "many"]. Your context is now filled, therefor you eliminate the oldest token. Your new trimmed context should be ["Anne", "has", "many"]
- Experiment with various sampling strategies:
  - **argmax** - you choose the most likely token
  - **top k** - you select the tokens with the highest *K* probabilities. You do a weighted sample from those *K* samples. To get the weights you renormalize the *K* probabilities.
- In case your model exhibits a repetition pattern (they sometimes do), make sure to guard rail yourself with a maximum generation amount (e.g. 1024 tokens at most)

Reporting

Four setups are requested for development:

- Small model with a character-level tokenization scheme
- Small model with a subword-level tokenization scheme
- Large model with a character-level tokenization scheme
- Large model with a subword-level tokenization scheme

Your written homework report must include the following:

- How you organised and processed your data
- **Small** and **Large** architecture parameters (i.e. hidden dimension, number of layers, number of attention heads, any other improvement you consider adding)
- **Training loop parameters** (i.e, how many tokens, epochs, batches, etc. you trained the model for, what optimizer, what learning rate schedule, gradient schedule, etc.)
- **Graphs for your train/test loss** and **comparison across model size** for each tokenization scheme (prefferably perplexity, not loss). Group plots by Tokenization scheme, since the vocabulary size directly impacts the numeric value of the perplexity metric.

- **BLEU / ROUGE comparison** across all 4 final experiments (you can include more) – 2 model sizes & 2 tokenization methods
- Sampling parameters that you used for **top-k**
- **Qualitative Samples** from your hold-out set – i.e. select some sentences, and put side by side what each model generated and what was the expected target.

Homework grading:

- Implementing and processing the data **[1p]**
- Implementing, training character-level tokenization + small model **[3p]**
- Reporting small character level model quantitatively and qualitatively (see above) - **[1p]**
- Implementing all experiments - **[4p]**
- Well organised & full report – **[1p]**

- **BONUS**: In a play (e.g. Hamlet) add an unrelated character (e.g. Romeo) and have them interact with the original characters in that passage.
- **Top 5 students**: get a bonus of 5-4-3-2-1% to their final NN grade.
  - How do we score this? – A mix of factors: generation length and cohesiveness, (how well the conversation evolves), generation quality (although it’s ok to generate broken language, better grammar will be exponetially better scored here), historical accurateness (Romeo should be a rather emotional person, not a motivational hustle culture speaker)
  - Anti ChatGPT measures: Those that submit the bonus will have to run a live generation during office hours to showcase that the model is actually capable of generating similar passages as the ones submitted (since sampling from a Transformer can be a stochastic process). To avoid randomness you can generate these using the **argmax** strategy.

TL;DR

Train a language model with next-token prediction objective on Shakespeare text.