Homework 1

11-967

In this homework, you will implement a basic Transformer archi-tecture

Note: Only report.pdf files submitted to the "written" Gradscope submission page will be graded.

Problem 1: Implementing a Transformer

In this problem, you will learn to implement components of the Transformer architecture. You will first implement a decoder-only transformer model. An outline of the code is provided for you in model.py. This outline contains all the class and function declarations that are expected for the submission. **Do not modify the classes, functions, or their arguments. Do not import new Python dependencies. This may break the automatic code test pipeline and result in failed unit tests.** You should aim to have an efficient implementation for the model to train in a reasonable amount of time. This means calling PyTorch functions whenever possible, and also means that you should not write matrix operations using a for loop. That said, you may not use layers or functions (e.g., torch.nn.TransformerDecoder) that make implementation trivial. If you are unsure whether using something is acceptable, ask course staff. All relevant code for this question is in src/lm.

Recall the Transformer Decoder from Lecture 2, shown in Figure 1. There are four classes within the transformer that you are expected to implement:

1. MultiHeadAttention - the "Masked Multi-head Attention" module.
2. FeedForward -the "Feed Forward" module.
3. DecoderBlock -a single decoder block, as described in "The Decoder Step-by-Step" in Lecture 2. Note that since we are implementing the decoder only, you do not need to implement the Encoder-Decoder Multi-Head Attention or the second "Add & Norm" operation.
4. DecoderLM -the full decoder model: the embedding step, multiple decoder blocks, and the final output logits.

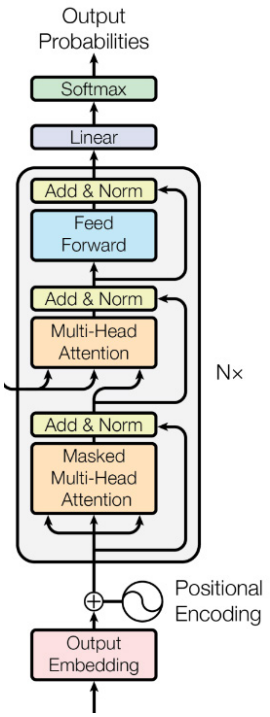

Figure 1: Transformer decoder

[Question 1.1] (Written, 5 points) Press and Wolf (2017) propose a weight tying technique for projecting hidden states of a language model to token logits.

DELIVERABLES FOR Q1.1

Read this paper, and in at most three sentences, explain what weight tying does.

[Question 1.2] (Written, 5 points) Let d be the hidden size of the model, v be the vocab size, b be the batch size, and s be the sequence length. Suppose you have hidden states $h \in \mathbb{R}^{b \times s \times d}$ and token embeddings $E \in \mathbb{R}^{v \times d}$ stored in PyTorch tensors.

DELIVERABLES FOR Q1.2

Write one line of Python code here (potentially with functions in PyTorch) that computes the token logits using weight tying.

[Question 1.3] (Coding, 25 points) Complete model.py, implementing all of the classes above. All the unit tests are provided in the test script test_model.py. Your implementation will be awarded points for each of the five unit tests that pass.

DELIVERABLES FOR Q1.3

Upload your code to Gradescope as instructed in the starter code README and make sure all test cases pass.