15-780 HW5

Due 3/25

In this problem you will build a Transformer-based language model from scratch (using the code we have built in class as a baseline). The starter code you need is all in the `hw5.tar.gz` file, which contains the notebook, code written in the lecture, and the Shakespeare data.

"Post LayerNorm" Transformer Blocks

In class we discussed the original form of the Tranformer block

$$Z = \text{LayerNorm}(X + \text{SelfAttention}(X)) \tag{1}$$
$$Y = \text{LayerNorm}(Z + \sigma(ZW_1)W_2) \tag{2}$$

(where we could optionally also add bias terms to the linear layers in the two-layer feedforward network, and to the layer in the self-attention layer). This layer is sometimes called a "post-norm" attention block, since the layer normalization happens after the residual connections.

However, for the optimizer we're going to use in this lecture, it's much better to use an alternative "pre-norm" attention block, which has become the dominant form to use anyway (any modern LLM will do this). The block is given by

$$Z = X + \text{SelfAttention}(\text{LayerNorm}(X)) \tag{3}$$
$$Y = Z + \sigma(\text{LayerNorm}(Z)W_1)W_2 \tag{4}$$

(plus an additional normalization layer right before the final linear output layer).

Implement this form of Transformer block in the `hw5.ipynb` file, in the `TransformerBlockPreNorm` class.

A full Transformer-based language model

Implement a full language model using an embedding layer, positional encodings, the pre-norm Tranformer block, and a linear output layer[1]. Crucially (you will get "good" performance if you don't do this, but the actual predictions will be meaningless), you should properly apply the causal mask to self-attention layers.

Implement this form of Transformer block in the `hw5.ipynb` file, in the `LanguageModel` class. Here are some important things to note:

1. You can use the function or class defined in `code_15780.py`, e.g., `PositionalEncoding`.
2. This code defines the Transformer language model `LanguageModel(128, seq_len, 8, 1024, 5, tokenizer.vocab_size)`. In this line, 128 is the number of hidden dimensions; 8 is the number of attention heads per layer; 1024 is the number of hidden dimensions in FFN; 5 is the number of Transformer blocks.
3. Since it is an autoregressive Transformer, you should define the attention mask correctly, i.e., the parameter `mask` in the forward function of `SelfAttention`.
4. The parameter count of the Transformer should match 9892554 as shown in the notebook.

[1] As described in the previous question, you should apply a layer normalization layer before this final linear output layer. This is common practice in modern large language models.

Training the language model

Implement the `epoch_transformer_lm` function. This runs a single epoch of the language model over the sequence of tokens.

To implement this function correctly, you will need to set up the batches and losses properly. First, you'll want to reshape each minibatch into a `batch_size`×`seq_len` tensor. The loss should then be the output of the language model on the first $1, \ldots,$ `seq_len` $- 1$ tokens in each batch, and the targets are the next $2, \ldots,$ `seq_len` tokens.

You can try out different parameters for the network, but in our implementation we used a transformer with an embedding dimension of 128, a feedforward inner dimension of 1024, 5 layers and 8 attention heads per layer. Training with SGD for 1 epoch (learning rate = 0.5) results in an evaluation loss of around 5.9 (training for longer can bring the eval loss down to around 5.1, which is substantially lower than what we could attain with our previous linear or two-layer NN language models).

You can try sampling from this model by applying it in a sliding window fashion, but we won't implement this for now (we'll be waiting for the next assignment, when we implement key-value caches and other optimizers).