Advanced Methods in Text Analytics
Exercise 6: Transformers - Part 2

Daniel Ruffinelli

FSS 2025

1 The Residual Stream

In this task, we take a closer look at both the forward and backward pass of a transformer layer. By *transformer layer*, we mean the combination of a multi-head self-attention followed by a feed-forward neural network, each with layer normalization applied after it, and each with a residual connection around them. For example, the encoder block in Figure 1 on the transformers paper linked above.

(a) Let SA be a self-attention layer parameterized by $\boldsymbol{W}^Q, \boldsymbol{W}^K, \boldsymbol{W}^V$, and let $\boldsymbol{X} \in \mathbb{R}^{n \times d}$ be the d-dimensional tokens from an input sequence of length n. Give a formal expression for the output of SA given $\boldsymbol{X}$ as input.

(b) Let FNN be the feed-forward neural network applied after SA in a transformer layer, and assume it projects the vectors in input $\boldsymbol{X}$ to a m-dimensional space and then back down to a d-dimensional space. Give a formal expression for the output of FNN given $\boldsymbol{X}$ as input. Assume a $ReLU$ activation function is used after the first projection, and make sure to specify the size of the parameters in FNN.

(c) Using SA and FNN, give an expression for the output of a transformer layer TF given $\boldsymbol{X}$ as input. Use LN_1 and LN_2 for layer normalization after SA and FNN, respectively.

(d) What do the layer normalization operators LN_i do? And could we use them in a different part of the transformer layer?

(e) Let us now focus on the backward pass. Recall that during backpropagation, we apply the chain rule of calculus to compute the gradients of all parameterized operators used in the forward pass, usually w.r.t. some loss function L. In the context of some transformer layer TF_i, these operators are $\text{SA}, \text{FNN}, \text{LN}_1$ and LN_2. Give a formal expression for the gradient needed to update parameters θ_{SA} of operator SA w.r.t. some loss function L. Omit the use of layer normalization and residual connections for simplicity. Indicate which part of the expression is "received" from the FNN layer above during backpropagation.

(f) Following your previous answer, what is the gradient that is passed down to lower layers, e.g. the lower transformer layer $\text{TF}_{\text{i-1}}$? Give a formal expression for this gradient and briefly describe why this gradient is needed during training.

(g) Now rewrite your previous answer while considering the residual connection around SA. Simplify the resulting expression as much as possible with the goal of answering the question: what happens during training to the gradient that we received from higher layers as it's passed through an SA operator that uses residual connection? **Hint:** No need to compute any gradients, but do use the fact that the gradient is a linear operator.

2 Attention Heads are Independent and Additive

In this task, we take a closer look at the multi-head attention mechanism.

(a) Let MHA be a multi-head attention layer, SA_i its i-th self-attention layer and $\boldsymbol{X} \in \mathbb{R}^{n \times d}$ the d-dimensional tokens from an input sequence of length n. Give a formal expression for the output of MHA as a function of its k attention heads SA_i, i.e. $1 <= i <= k$, given $\boldsymbol{X}$ as input. As usual, make sure to formally define all components in your expression. How is MHA parameterized?

(b) Assume MHA takes as input the output $\boldsymbol{O}^i$ of each self-attention head $\text{SA}_i(\boldsymbol{X})$, i.e. $\boldsymbol{O}^i = \text{SA}_i(\boldsymbol{X})$. Give a formal expression for the computations needed to compute a single component of the output matrix of MHA as a function of each $\boldsymbol{O}^i$.

(c) Using the intuition built in the previous subtask, show that the output of the multi-head attention layer MHA can be expressed as the following linear combination:

$$\text{MHA}(\boldsymbol{X}) = \sum_{i=1}^{k} \boldsymbol{O}^i \boldsymbol{M}^i + \boldsymbol{X}, \tag{1}$$

and make sure to define exactly what each $\boldsymbol{M}^i$ is. What does this rewriting of the multi-head attention layer imply about the operations computed by each attention head?

3 Language Models with Transformers

In this exercise, we have a quick look at how to implement a language model with transformers using PyTorch. As before with our FNN-based and RNN-based language models, we will need to define a class for the model, a training loop and an evaluation loop. In addition, we also need to implement positional encodings. We use the ones used by the original transformer architecture.

(a) Read the code for the class *PositionalEncoding* to make sure it's correct w.r.t. how these positional embeddings are defined. Then check documentation for PyTorch's Module. What does the method *register_buffer* do?

(b) Read the code for the TransformerModel class. What kind of architecture does it use? Encoder-decoder? Encoder-only?

(c) What is the role of the function *_generate_square_subsequent_mask*? Read the documentation for the Transformer Class to find out more.

(d) What kind of language model is it? Causal or masked? Why? **Hint:** What does the function triu do?

(e) Complete the *forward* function of your transformer.

(f) We can train this transformer in the same way that we trained our RNN in our last coding exercise: with *teacher forcing*. So, we will use the same methods for preprocessing the data, creating the batches and training the model. Note that the training method was modified slightly to accomodate this model's forward function (no need to pass the hidden state as with RNNs). Use the given code to train the transformer on the *Shakespeare* data. What perplexity do we get on validation data? Is it higher or lower than when using an RNN as before? Is the task comparable w.r.t. that case? Discuss.

(g) Can you modify the model so it performs better? Consider using different embedding sizes, different number of layers, etc.