Advanced Methods in Text Analytics

Exercise 6: Transformers - Part 2

Daniel Ruffinelli

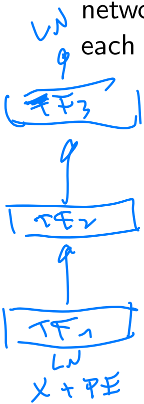
University of Mannheim

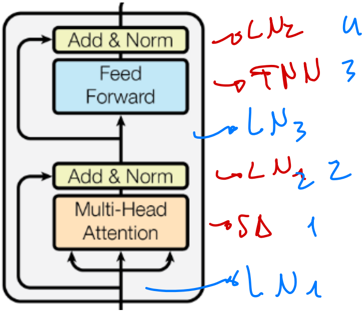
FSS 2024

The Residual Stream

Context

- In this task, we take a closer look at both the forward and backward pass of a transformer layer (see image from original paper below).
- 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.





The Residual Stream

Question 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. (we assume dot-product attention)

The Residual Stream

Answer a)

- We have:

$$\mathrm{SA}(\boldsymbol{X}) = \mathrm{softmax}\left(\frac{\overbrace{\boldsymbol{X}\boldsymbol{W}^Q}^{Q}\overbrace{(\boldsymbol{X}\boldsymbol{W}^K)^T}^{K}}{\sqrt{d}}\right)\overbrace{\boldsymbol{X}\boldsymbol{W}^V}^{V} \tag{1}$$

  where the softmax function is applied row-wise and $\boldsymbol{X}\boldsymbol{W}^Q = \boldsymbol{Q}$ is referred to as the query matrix, $\boldsymbol{X}\boldsymbol{W}^K = \boldsymbol{K}$ is the key matrix, and $\boldsymbol{X}\boldsymbol{W}^V = \boldsymbol{V}$ the value matrix.
- Recall that the factor $\frac{1}{\sqrt{d}}$ is used for numerical stability (scaled dot-product attention).

The Residual Stream

Question 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 both projections, and make sure to specify the size of the parameters in FNN.

The Residual Stream

Answer b)

- We have:

$$\mathrm{FNN}(\boldsymbol{X}) = \mathrm{ReLU}\Big(\overbrace{\overbrace{\underbrace{(\boldsymbol{X}\boldsymbol{W}_1 + \boldsymbol{b}_1)}_{1}}^{W_1}\overbrace{\boldsymbol{W}_2}^{2} + \boldsymbol{b}_2}^{W_2}\Big) \tag{2}$$

where parameters are $\boldsymbol{W}_1 \in \mathbb{R}^{d\times m}$, $\boldsymbol{b}_1 \in \mathbb{R}^{m}$, $\boldsymbol{W}_2 \in \mathbb{R}^{m\times d}$, and $\boldsymbol{b}_2 \in \mathbb{R}^{d}$.

ReLU

The Residual Stream

Question 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.

The Residual Stream

Answer c)

- We do this in stages. First, we focus on SA, its corresponding residual connection and layer normalization LN_1. We have:

$$\boldsymbol{O}_1 = \mathrm{LN}_1(\mathrm{SA}(\boldsymbol{X}) + \boldsymbol{X}) \tag{3}$$

Then, we focus on FNN, its corresponding residual connection and layer normalization area applied. That is:

$$\boldsymbol{O}_2 = \mathrm{LN}_2(\mathrm{FNN}(\boldsymbol{O}_1) + \boldsymbol{O}_1) \tag{4}$$

2nd Residual Connection

All together, we have:

$$\mathrm{TF}(\boldsymbol{X}) = \mathrm{LN}_2(\mathrm{FNN}(\mathrm{LN}_1(\mathrm{SA}(\boldsymbol{X}) + \boldsymbol{X})) + \mathrm{LN}_1(\mathrm{SA}(\boldsymbol{X}) + \boldsymbol{X})). \tag{5}$$

1st Residual Connection

1st Residual Connection

The Residual Stream

Question d)

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

The Residual Stream

Answer d)

- Layer normalization is used to normalize the vectors it is given as input.
- Concretely, given a vector $\boldsymbol{x} \in \mathbb{R}^d$, the layer normalization operation is given by:

$$\mathrm{LN}(\boldsymbol{x}) = \frac{\boldsymbol{x} - \mu}{\sigma}\gamma + \beta \tag{6}$$

  where γ and β are learned parameters.
- This operation can be seen as a form of data centering and is commonly used so keep training stable, sometimes at the cost of performance on some downtream tasks. (depends on positions)
- It can be applied anywhere in a network/transformer.
- In fact, a common variant is the *pre-norm* transformer, which applies layer normalization *before* each of SA and FNN.
- Typically, this variant also includes a single final layer normalization operation applied only to the output of the last transformer layer.

The Residual Stream

Question 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 $\mathrm{TF_i}$, these operators are SA, FNN, $\mathrm{LN_1}$ and $\mathrm{LN_2}$.
- 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.

The Residual Stream

Answer e) (1)

$y = f(x)$

$L = g(y)$

$$\frac{\delta L}{\delta x} = \underbrace{\frac{\partial L}{\partial y}}_{A} \underbrace{\frac{\delta y}{\delta x}}_{B}$$

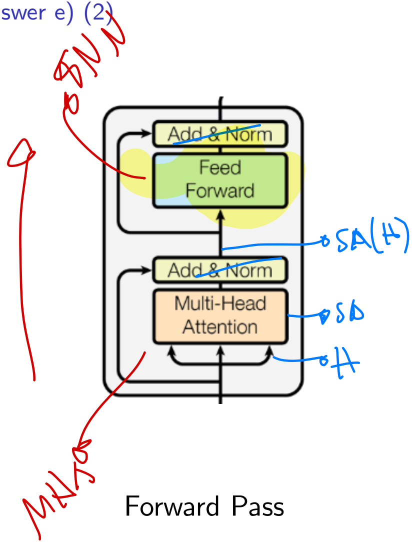
- To update parameters θ_{SA} of SA w.r.t. some loss function L during training, we need:

$$\frac{\partial L}{\partial \theta_{SA}} = \overbrace{\frac{\partial L}{\partial \, \mathrm{SA}(\boldsymbol{H})}}^{A} \overbrace{\frac{\partial \, \mathrm{SA}(\boldsymbol{H})}{\partial \theta_{SA}}}^{B}, \tag{7}$$

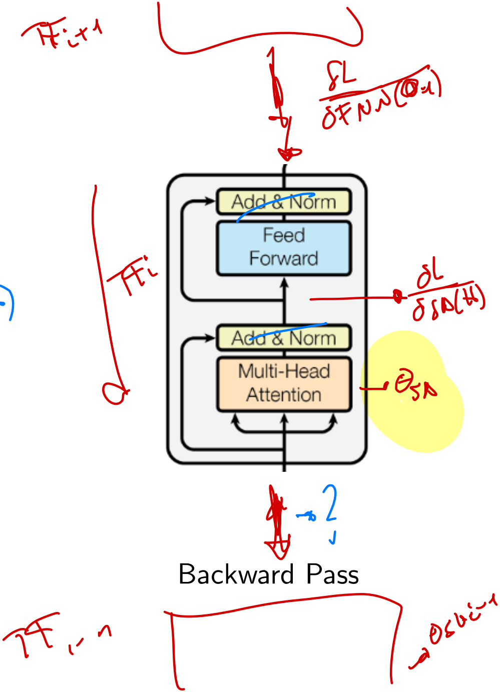
  where $\boldsymbol{H}$ is the input to SA, which in turn is the output of the previous transformer layer $\mathrm{TF}_{\text{i-1}}$.
- The term $\frac{\partial L}{\partial \, \mathrm{SA}(\boldsymbol{H})}$ is the partial gradient received during backpropagation from the upper layer, the FNN operator in the context of a transformer layer.
- Let's try to visualize this in the next slide.

The Residual Stream

Answer e) (2)



Forward Pass



Backward Pass

The Residual Stream

Question f)

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

The Residual Stream

Answer f)

- Just as we received a gradient from the upper layer during training, we pass down the following gradient to the lower transformer layer $\text{TF}_{\text{i-1}}$ during backpropagation:

$$\frac{\partial L}{\partial \boldsymbol{H}} = \frac{\partial L}{\partial \, \text{SA}(\boldsymbol{H})} \frac{\partial \, \text{SA}(\boldsymbol{H})}{\partial \boldsymbol{H}}. \tag{8}$$

- This is the gradient of the output of $\text{TF}_{\text{i-1}}$ w.r.t. L, and is needed to compute the weight updates of the operators in lower layers that contribute to the computation of the input to TF_{i}, i.e. $\boldsymbol{H}$.

The Residual Stream

Question 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.

The Residual Stream

Answer g) (1)

- Let $\boldsymbol{O}_1$ be the output of SA with a residual connection around it, i.e. $\boldsymbol{O}_1 = \text{SA}(\boldsymbol{H}) + \boldsymbol{H}$.
- Then, the gradient "received" from the upper layer is $\frac{\partial L}{\partial \boldsymbol{O}_1}$.
- Eq. 8 can then be rewritten as follows:

$$
\begin{aligned}
\frac{\partial L}{\partial \boldsymbol{H}} &= \frac{\partial L}{\partial \boldsymbol{O}_1} \frac{\partial \boldsymbol{O}_1}{\partial \boldsymbol{H}} \\
&= \frac{\partial L}{\partial \boldsymbol{O}_1} \frac{\partial (\text{SA}(\boldsymbol{H}) + \boldsymbol{H})}{\partial \boldsymbol{H}} \\
&= \frac{\partial L}{\partial \boldsymbol{O}_1} \left(\frac{\partial\, \text{SA}(\boldsymbol{H})}{\partial \boldsymbol{H}} + \frac{\partial \boldsymbol{H}}{\partial \boldsymbol{H}} \right) \\
&= \frac{\partial L}{\partial \boldsymbol{O}_1} \frac{\partial\, \text{SA}(\boldsymbol{H})}{\partial \boldsymbol{H}} + \frac{\partial L}{\partial \boldsymbol{O}_1}.
\end{aligned}
$$

- In other words, when using residual connections, the gradient $\frac{\partial L}{\partial \boldsymbol{O}_1}$ we receive from higher layers during backpropagation is passed down to lower layers by making a modification via the addition operation.

The Residual Stream

Answer g) (2)

- Note that without the residual connection, the gradient we pass down would be modified by the multiplying factor $\frac{\partial \mathrm{SA}(\boldsymbol{H})}{\partial \boldsymbol{H}}$, and the additive term would not exist.
- Since such a multiplying factor can quickly reduce the size of the gradient that is passed backward during training, the residual connection was designed by He et al. 2015 so that gradients can be more safely passed through an operator without it having a severe impact on those gradients.
- The name "residual" connection comes from the fact that, due to this change to prevent gradients from vanishing during training, the function $f(\boldsymbol{X}) = \mathrm{OP}(\boldsymbol{X})$ computed by some operator becomes $f(\boldsymbol{X}) = \mathrm{OP}(\boldsymbol{X}) + \boldsymbol{X}$.

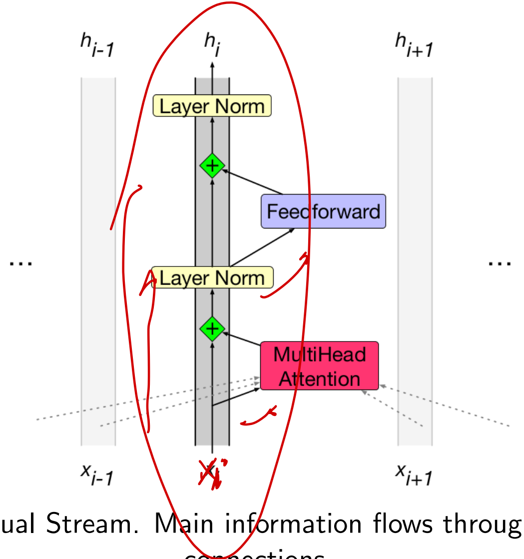
The Residual Stream

Answer g) (3)

- This means that we are learning to apply a suitable *difference* OP($\boldsymbol{X}$) that is added to given input $\boldsymbol{X}$.
- In the context of a transformer layer, $\boldsymbol{X}$ is a set of contextualized representations, so what each transformer layer learns is to add something to it *if needed*.
- If $\boldsymbol{X}$ is already suitable for the task, then a layer may make small modifications to it.
- This suggests that the main flow of information is coming from $\boldsymbol{X}$, and not from $\boldsymbol{X}$, or in other words, from the residual connections and not from the outputs of transformer layers.
- This perspective is known as the "residual stream" view of transformers, illustrated in the following image from Jurasfky and Martin (2024), Section 10.4.

The Residual Stream

Answer g) (4)



The Residual Stream. Main information flows through residual connections.

Language Models With Transformers

Context

- 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.

Language Models With Transformers

Question 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?

Language Models With Transformers

Answer a)

- Storing models during of after training is important to later be able to use them, e.g. for inference, fine-tuning or to continue training.
- PyTorch models have a state dictionary (state_dict) that is usually stored on disk in so-called *checkpoint* files.
- It's important that these files contain all the necessary information so that we may later be able to use the model.
- *register_buffer* is used to define “parameters” of a model that need to be stored in its state_dict, but that aren't actually parameters in the learning sense.
- That is, they are not changed during training.
- Since we are implementing a non-parameterized positional encoding, we register these embeddings as buffers, because we later need them for using the model.

Language Models With Transformers

Question b)

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

Language Models With Transformers

Answer b)

- It's an encoder-only transformer. We only use the TransformerEncoder class.
- The code does have a component called a decoder, but what is that exactly?
- It's a linear layer that projects down to a vector space the size of our vocabulary.
- This, in combination with a softmax, will be used to produce probabilities over our vocabulary in order to predict the next word (if we construct the training examples correctly).
- But why isn't the softmax function there then?
- Well, as in our previous coding exercise, we will use CrossEntropyLoss, which includes the softmax function, so we need our model to output logits.

Language Models With Transformers

Question c)

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

Language Models With Transformers

Answer c)

- This function is used to construct an attention mask. Masking is the process by which we “block” some entries in a tensor.
- In this context, the forward pass of the TransformerEncoder takes a mask as input along side the input sequence.
- This mask is added to the matrix of attention *scores* in the self-attention layer.
- If scores are *-Inf*, their attention weight (via softmax) is zero.
- Entry ij in this matrix can be seen as how much attention input token i pays to j.
- Thus, the mask is a matrix with zeros in the lower triangular, and the rest *-Inf*.

Language Models With Transformers

Question d)

- What kind of language model is it? Causal or masked? Why?
  Hint: What does the function triu do?

Language Models With Transformers

Answer d)

- Given the attention mask we use, as explained in the previous answer, this is a causal language model.

Language Models With Transformers

Question e)

- Complete the *forward* function of your transformer.

Language Models With Transformers

Answer e)

- See Jupyter notebook.

Language Models With Transformers

Question 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.

Language Models With Transformers

Answer f)

- The perplexity is slightly higher than with the RNN, which was about 400.
- Recall that this number can be interpreted as proportional to the vocabulary size, which is about 30K.
- The result is comparable to the RNN setting, as both models are evaluated on the same task using the same validation data.
- As for resources, they are also comparable.
- The embedding size is the same as the one used by the RNN, and both use dropout with default values as the only form of regularization.
- It's therefore likely can this slighly lower performance is due to the stronger inductive bias in the model, i.e. it's overfitting.
- It would be nice to try to use a transformer with lower number of attention heads, or lower number of layers.