Alexandre Zajac
zajac.alexandre@gmail.com

1 Question 1

In our implementation, we are using a transformer-based mechanisms with self-attention. For doing so, we use the nn.TransformerEncoder abstraction that doesn't allow "peaking ahead" when deriving self-attention: if we are at position t, and we want to find the self attention for the current token, we are not allowed to consider tokens at times $t+1, ..., Tx$. If we retake the standard notation for transformers, here is the Z matrix that is passed to fully connected layers in each encoder:

$$Z = softmax(\frac{Q * K^T}{\sqrt{d_k}}) * V \tag{1}$$

where d_k is the dimension of the encoding. We can see that by applying a mask whose values are $-inf$ for future tokens, the softmax function considers their values really close to zero and ignores their contribution for self-attention at time t.

We need positional encoding because we are not using any sequential-type architecure for our network (LSTM, RNN, GRU...) and the position of each word in a sentence has an influence on the attention and context it should get. To do so, we must use a positional embedding vector, and adding it to each embedding vector for each word. These values added to the embedding provides meaningful distances between the embedding vectors once they're projected into Q/K/V vectors and during dot-product attention.

2 Question 2

We need to replace the classification head because when using transfer learning, the network was trained for a specific classification task that could be very different from the one that we will train on. The dataset used, the size of the vocabulary and relations between words is preserved for our case, but the "fine-tuning" phase is precisely designed to fit a different task.

The main objective of language modelling is to deal with challenging language understanding problems such as translation, question answering or sentiment analysis for example. A language model attempts to learn the structure of natural language through hierarchical representations, both at the level of word representation, and meaning/semantics of a sequence in a particular context. Almost always, language modelling is "generative", meaning that it aims to predict the next word given a previous sequence of words (many-to-many models).

On the other hand, classification maps the input sequence into a finite set of classes, like for example sentiment analysis (positive or negative). The set of classes for language modelling is also fixed, but much larger. In this sense, the model types are more of type many-to-one, and is not generative, the prediction is made on the sentence as a whole, without intermediate predictions.

3 Question 3

The number of trainable parameters for both cases is equal to:

$$num\_params = num\_params\_base + num\_params\_classifier \tag{2}$$

Let's calculate the number of trainable parameters for the base model. Here is the complete representation for an encoder layer of a transformer (since we are only using an encoder step for the base):

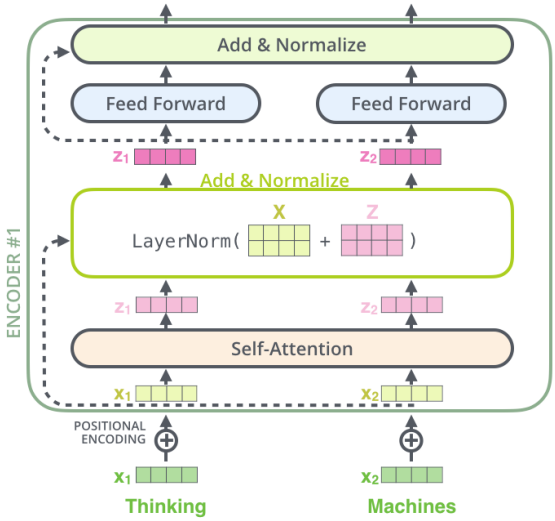

Figure 1: A high level representation of an encoder layer

Let's define first our parameters for the model: $n_{token} = 100$ (vocab size); $n_{hid} = 200$ (hidden dimension of the transformer layers; $n_{layers} = 4$ (number of transformer layers in the encoder part); $n_{head} = 2$ (attention heads); $n_{classes} = 2$ (classes for classification in first example on sentiment analysis). First, the embedding size (nothing is learned in positional encoding):

$$embedding\_weights = n\_token * n\_hid = 200\ 000 \tag{3}$$

By looking at the internals of the Pytorch implementation of the TransformerEncoderLayer, we can see that since we do not specify values for the dimension of K and V, the weights for the input part of the attention mechanism take the shape of $(3 * nhid, nhid)$ (with $3 * nhid$ for the bias) and $(nhid, nhid)$ for the output part of the attention mechanism (with $nhid$ for the bias, which yields:

$$num\_weights\_attention = 3 * nhid * (nhid + 1) + nhid * (nhid + 1) = 160\ 800 \tag{4}$$

We then have the FC layer with dimension $(nhid, nhid)$ and bias $nhid$ in our case. This is the the same for the 2 attention heads:

$$num\_weights\_FC = (nhid * (nhid + 1)) * 2 = 80\ 400 \tag{5}$$

We finally have the LayerNorm with $nhid = 200$ parameters for both the self-attention mechanism and the feed forward part, and for both attention heads. In total, for 1 encoder layer, we have:

$$\begin{aligned} num\_weights\_encoder\_layer &= num\_weights\_attention + num\_weights\_FC + num\_weights\_layernorm \\ &= 160\ 800 + 80\ 400 + (200 * 2 * 2) \\ &= 242\ 000 \end{aligned} \tag{6}$$

Since we have 4 encoder layers, and taking into account the embedding, here is the total number of trainable paramters for the base model:

$$\begin{aligned} num\_weights\_base\_model &= num\_weights\_encoder\_layer * 4 + embedding\_weights \\ &= 968\ 000 + 20\ 000 \\ &= 988\ 000 \end{aligned} \tag{7}$$

- For the classification, we have a linear layer of size $(nhid, n_{classes})$ (and a bias) which here would represent:

$$\begin{aligned} num\_weights\_with\_classifier &= num\_weights\_base\_model + 200 * 2 + 2 \\ &= 988\ 402 \end{aligned} \tag{8}$$

- For the language modelling, we have a linear layer of size $(nhid, n_{token})$ (and a bias) which here would represent:

$$\begin{aligned} num\_weights\_with\_language\_model &= num\_weights\_base\_model + 200 * 100 + 100 \\ &= 1\ 008\ 100 \end{aligned} \tag{9}$$

4 Question 4

Here is the accuracy plot for the pre-trained model, and the model trained from scratch:

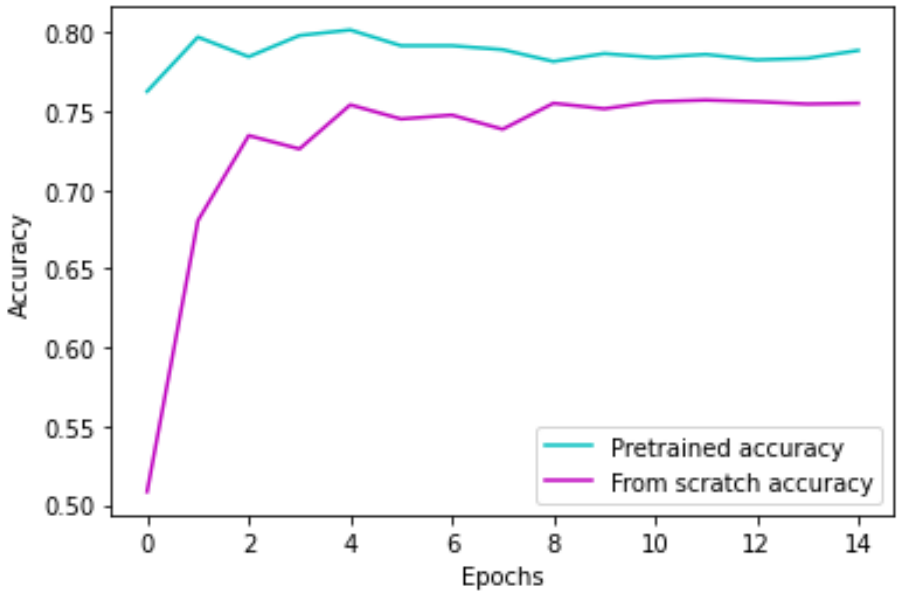

Figure 2: A high level representation of an encoder layer

We can observe a few things:

- The training accuracy is starting a lot higher (25%) for the pre-trained model than for the model trained from scratch. This is expected since the weights are initially close to zero for the model trained from scratch, so one epoch step on an unknown classification problem should yield a lower accuracy than one epoch concentrated on the last classification head (for the pre-trained model).
- After approximately 10 epochs, both accuracies plateau at a certain threshold. For the pre-trained model, this is the case because after improving a bit the accuracy for the first few epochs, the network doesn't find any improvement for the given classification task. For the model trained from scratch, the network has learned quite enough for the given classification task, and is becoming limited.
- The pre-trained model achieves a higher accuracy (80%) compared to the model trained from scratch (75%). One possible explanation is that given that the pre-trained model was trained on a much larger dataset, and for longer, it was able to pick up word semantics/relationships that are deeper than the ones learned from scratch.

5 Question 5

The approach we have used here has the objective function of learning general language representations, with a unidirectional language model. we already saw that the architecture chosen with the nn.TransformerEncoderLayer uses a mask so that every token can only attend to previous tokens in the self-attention layers of the Transformer.

The major drawback of this choice of unidirectionality is that for some context and tasks, it could harm the results even with fine tuning. the approach proposed in [1] uses a bi-directional language model, with a masked language model in the pre-training phase, where each token is randomly masked from the input, and the objective is to predict the vocabulary_id based only on the context. Having this richness of attention and modified training objective allows the model to generalize better for a broader variety of tasks.

References

[1] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.