Martin BRICENO
jbriceno@telecom-paris.fr

1 Question 1

Square mask:
In the language model the idea is to predict the next token given the precedents, this works well if we only take into account the latter without considering the future tokens, however when using transformers all the tokens are available. Therefore the idea is to hide, by means of a mask, the tokens that are not necessary so that attention is only focused on the previous tokens.

Thus, we must adjust the parameters of the transformer model in the equation below so that it gives zero attention to the tokens whose mask is of value "-inf" as seen in the method "generate_square_subsequent_mask" of the class "TransformerModel".

$$F = softmax(\frac{Q * K^\top + mask}{\sqrt{d_k}}) * V \tag{1}$$

Positional Encoding:
A sequence of tokens are passed to the transformer model, however since we do not work with LSTM cells, we do not know the order of these. The Positional Encoding module adds this missing information, so in case of shuffle on the sequences we can still determine the order of these.

This module is encoded by means of sine and cosine functions, as seen in the equations below, where the even index tokens are associated to a vector with the sine function while the odd index tokens are associated to a cosine vector. Finally we add these vectors to those obtained after the embeddings layer

$$PE(pos, 2i) = sin(pos/10000^{2i/d_m odel}) \tag{2}$$

$$PE(pos, 2i+1) = cos(pos/10000^{2i/d_m odel}) \tag{3}$$

2 Question 2

We replaced the classification head so that the base model can be used for the language model task and with this we can pretrain the base parameters. Once the base is pretrained, we will use transfer learning to train the classification for better performance and faster convergence.

The difference is in the last layer, in the case of the language model, the task is to predict the next word given an input sequence, on the other hand, in the case of classification the task is to predict a label (0/1 or positive/negative) given an input sequence.

3 Question 3

To calculate the total of trainable parameters in each case, we will analyze our model according to the base model and the classifier.

$$Total\_param = param\_base + param\_second\_layer \tag{4}$$

To calculate the parameters of the base model, first we note that it is composed of 2 parts, the parameters of the layer of embedding and the each of encoder.

$$param\_base = param\_embedding + param\_encoder * 4 \tag{5}$$

For the embedding layer the number of trainable parameters depends on the size of the vocabulary ($n_{token} = 100$) and the size of the hidden layer ($n_{hidden} = 200$).

$$param\_embedding = n_{token} * n_{hidden} = 20000 \tag{6}$$

Now analyzing the encoder layer, PyTorch indicates that it depends on 3 elements:

$$param\_encoder = param\_attention + param\_FC + param\_norm \tag{7}$$

Analyzing the first term:

$$param\_attention = 3 * n_{hidden} * (n_{hidden} + 1) + n_{hidden} * (n_{hidden} + 1) = 160800 \tag{8}$$

Analyzing the second term, it depends of:

$$param\_FC = n_{hidden} * (n_{hidden} + 1) * n_{heads} = 80400 \tag{9}$$

Analyzing the third term, we have one for the attention and one for the norm layer:

$$param\_norm = 2 * n_{hidden} * n_{heads} = 800 \tag{10}$$

So:

$$param\_encoder = 160800 + 80400 + 800 = 242000 \tag{11}$$

So the number of trainable parameters in the base model is:

$$param\_base = 20000 + 242000 * 4 = 988000 \tag{12}$$

Now, for the second layer, we have 2 cases:
The classification depends of a linear layer of size ($h_{hidden}, n_{classes}$) and bias, so:

$$param\_second\_layer = (h_{hidden} + 1) * n_{classes} = 402 \tag{13}$$

$$Total\_param = 98800 + 402 = 988402 \tag{14}$$

The language model depends of a linear layer of size (h_{hidden}, n_{token}) and bias, so:

$$param\_second\_layer = (h_{hidden} + 1) * n_{token} = 20100 \tag{15}$$

$$Total\_param = 98800 + 20100 = 1008100 \tag{16}$$

4 Question 4

In the Fig. 1 we see that from the first epoch the training of the classification with the pretrained weights is superior than with the weights initiated in a random way in the scratch method. This behavior is maintained during the 15 epochs of training.

Although it is possible that with more epochs the scratch method and the pretrained method converge to the same accuracy value, the second method is faster because it needs less training time. If we look after 2 epochs, we notice that the pretrained method has already converged approximately, while for the other method, we still need more epochs.

This is because in the pretrained method our model does not learn from zero as it does in the scratch method. On the contrary, by using the knowledge already learned through its pretrained weights, this model needs less training time since it fine-tunes its learning based on its previous knowledge as indicated by the transfer learning.

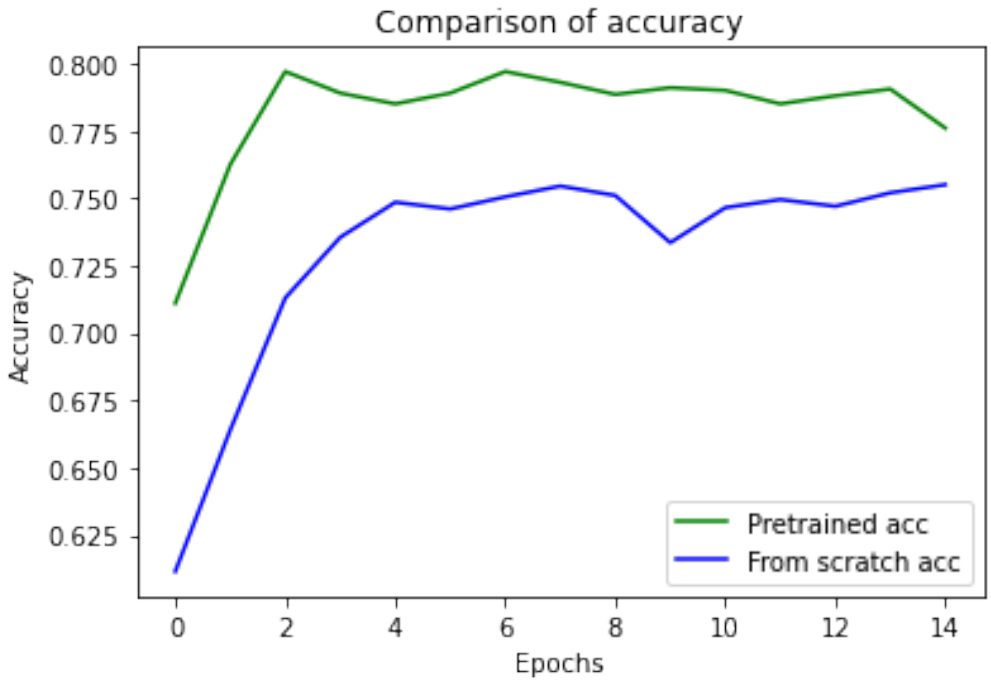

Figure 1: Comparison of accuracy between the pre-trained and scratch method.

5 Question 5

In this laboratory, the objective of our model is to learn general representations of language, with a unidirectional language model. The mask that was implemented in the TransformerModel allows that each token can only attend to the preceding tokens in order to train the model.

The problem with this unidirectionality is that for some contexts and tasks, it could harm the results even with a fine tuning. In the article [1] a bidirectional language model is used, with a masked language model in the pretraining phase, where each token is randomly masked from the input, and the objective is to predict the vocabulary based only on the context.

With this improvement in focus and attention we will be able to expand the range of tasks we can perform with this architecture as it is more generally applicable.

References

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