Megi Dervishi
megi.dervishi@ens.fr

Question 1

- **Square mask.** Square mask is important in order to prevent the attention mechanism of the transformer from knowing the words that the model has not predicted/"seen" yet. This way the model actually learns how to predict the new words and not just "look it up".
- **Positional Encoding.** Unlike RNNs, Transformers process input as a whole using multi-head attention; therefore order becomes obselete. As order and position of words are still important to convey context, then we introduce positional encoding to the output of the embedding layer so that the word encoding is 'aware' of the words positions in the sentence.

Question 2

In a language modeling we want to predict the next word of the sentence. Hence the classification head needs to have as many outputs as the size of vocabulary. Whereas in a classification task we want to predict a class for the sentence. Hence the output of the classification head will have a size equal to the number of classes ($nclasses = 2$ in our code). Since the two tasks are different, and as a result their classification head needs to be different then it's important to replace it accordingly.

Question 3

The following calculations are only valid for the parameters in the code i.e. $d\_model = nhid = 200$, $nlayers = 4$, $nclasses = 2$, $ntokens = 50001$. We can observe the structure of the model in Figure 2. From there we can see that we have the following trainable parameters i.e $require\_grad = True$:

- Embedding layer which has 50001×200 parameters.
- 4 Transformer Encoder Layers each composed by:
  - Multihead attention which has $3 \times (200 \times 200 + 200) + (200 \times 200 + 200)$ trainable parameters corresponding to the weights of Q, K, V each of size $[200, 200]$ and their biases of vector size $(200,)$ and the weights of the out-projection linear layer with size $[200, 200]$ and bias $(200,)$.
  - 2 Linear Layers of size $[200, 200]$ and bias $(200,)$.
  - 2 Normalization layers which have 2 vectors of size $(200,)$
- Classifier head which in the case of a language modeling task will be of size $[200, 50001]$ and bias $(50001,)$, but in the case of classification task has size $[200, 2]$ and bias $(2,)$.

All in all for the language modeling task we have 21,018,401 trainable parameters and for the classification task 10,968,602. These results are consistent with the output of the code in the notebook.

Question 4

We observe in Figure 1 that the pre-trained model outperforms the model when it is trained from scratch. That is because a pre-trained model will not start from a random initialization of parameters and only worry about the parameters of the classification head. Furthermore we observe that the pretrained model reaches a good accuracy much faster than a model that is trained from scratch. This is especially useful when dealing with very big models that have a lot of parameters.

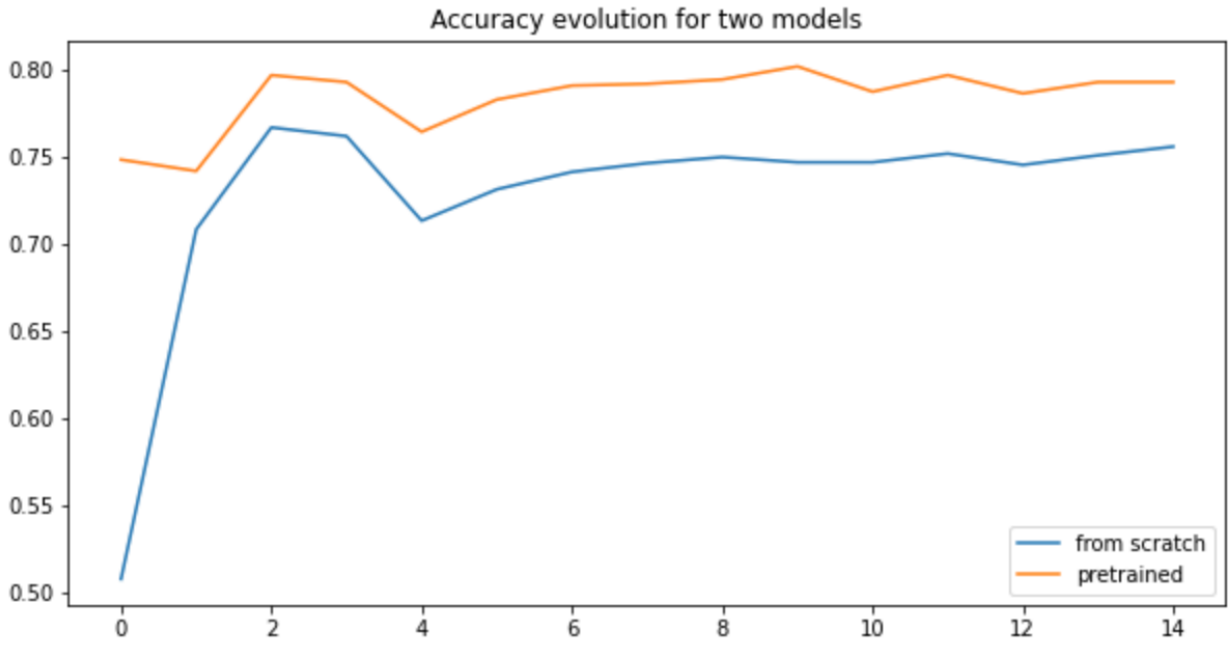

Figure 1: The evolution of accuracy for the pre-trained model and the model trained from scratch.

Question 5

In our model we use a square mask which as described in Question 1 prevents the model from seeing the ”future” of the sequence of words. This is good for tasks that involve next word prediction, however this is sub-optimal for tasks like Question-Answering since the model would need to know the full context. Indeed in BERT [1], the masked language model (MLM) randomly masks some of the tokens from the input, and by trying to predict the original vocabulary id of the masked token the model learns a good representation of the left and right context. Furthermore the Next sentence prediction (NSP) also allows the model to get an understanding of the order of two pairs of sentences which proves to be very useful for Question-Answering tasks.

References

[1] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

```
Model(
  (base): TransformerModel(
    (encoder): Embedding(50001, 200)
    (pos_encoder): PositionalEncoding(
      (dropout): Dropout(p=0, inplace=False)
    )
    (transformer_encoder): TransformerEncoder(
      (layers): ModuleList(
        (0): TransformerEncoderLayer(
          (self_attn): MultiheadAttention(
            (out_proj): NonDynamicallyQuantizableLinear(in_features=200, out_features=200, bias=True)
          )
          (linear1): Linear(in_features=200, out_features=200, bias=True)
          (dropout): Dropout(p=0, inplace=False)
          (linear2): Linear(in_features=200, out_features=200, bias=True)
          (norm1): LayerNorm((200,), eps=1e-05, elementwise_affine=True)
          (norm2): LayerNorm((200,), eps=1e-05, elementwise_affine=True)
          (dropout1): Dropout(p=0, inplace=False)
          (dropout2): Dropout(p=0, inplace=False)
        )
        (1): TransformerEncoderLayer(
          (self_attn): MultiheadAttention(
            (out_proj): NonDynamicallyQuantizableLinear(in_features=200, out_features=200, bias=True)
          )
          (linear1): Linear(in_features=200, out_features=200, bias=True)
          (dropout): Dropout(p=0, inplace=False)
          (linear2): Linear(in_features=200, out_features=200, bias=True)
          (norm1): LayerNorm((200,), eps=1e-05, elementwise_affine=True)
          (norm2): LayerNorm((200,), eps=1e-05, elementwise_affine=True)
          (dropout1): Dropout(p=0, inplace=False)
          (dropout2): Dropout(p=0, inplace=False)
        )
        (2): TransformerEncoderLayer(
          (self_attn): MultiheadAttention(
            (out_proj): NonDynamicallyQuantizableLinear(in_features=200, out_features=200, bias=True)
          )
          (linear1): Linear(in_features=200, out_features=200, bias=True)
          (dropout): Dropout(p=0, inplace=False)
          (linear2): Linear(in_features=200, out_features=200, bias=True)
          (norm1): LayerNorm((200,), eps=1e-05, elementwise_affine=True)
          (norm2): LayerNorm((200,), eps=1e-05, elementwise_affine=True)
          (dropout1): Dropout(p=0, inplace=False)
          (dropout2): Dropout(p=0, inplace=False)
        )
        (3): TransformerEncoderLayer(
          (self_attn): MultiheadAttention(
            (out_proj): NonDynamicallyQuantizableLinear(in_features=200, out_features=200, bias=True)
          )
          (linear1): Linear(in_features=200, out_features=200, bias=True)
          (dropout): Dropout(p=0, inplace=False)
          (linear2): Linear(in_features=200, out_features=200, bias=True)
          (norm1): LayerNorm((200,), eps=1e-05, elementwise_affine=True)
          (norm2): LayerNorm((200,), eps=1e-05, elementwise_affine=True)
          (dropout1): Dropout(p=0, inplace=False)
          (dropout2): Dropout(p=0, inplace=False)
        )
      )
    )
  )
  (classifier): ClassificationHead(
    (decoder): Linear(in_features=200, out_features=50001, bias=True)
  )
)
```

Figure 2: The model architecture