1 Question 1

What is the role of the square mask in our implementation? What about the positional encoding?

When we perform the language modeling task, the model is self-supervised, as it treats the input text both as the data and the supervision. Thus, for predicting the next word in a sequence, the information of what is coming next must remain hidden. In our code, the **square mask** (`src_mask`) ensures that future tokens are masked out while processing the "current token", so that predictions are made based only on the current and previous tokens. By doing so, the autoregressive property is maintained.

On the other hand, **positional encoding** is necessary when using transformers. Indeed, unlike traditional RNNs, transformers process tokens in parallel so they don't have a built-in mechanism to process sequential order. The positional encoding allows the model to learn about the relative positions of the tokens in a sequence, which is important to capture relevant information. Without positional encodings, the transformer would treat all tokens independently, and the model wouldn't learn the positions or order of the sequence. Moreover, the positional encoding allows transformers to capture long-range dependencies in sequences efficiently.

2 Question 2

Why do we have to replace the classification head? What is the main difference between the *language modeling* and the *classification* tasks?

One has to replace the classification head simply because of the tasks specificities. The language modeling and classification tasks differ through their respective objectives and therefore through their distinct output requirements.
For language modeling, the primary goal is to predict the next word in a sequence by assigning probabilities to each word in the vocabulary. The expected output is a probability distribution over the entire vocabulary. As for classification, the objective is to categorise data into different classes (in our case there are 2 classes), therefore the expected output is a probability distribution over the 2 classes: whether the book review is positive or negative.
Replacing the classification head is therefore essential to align the model's output with the specific task: for classification, the classification head projects onto 2 dimensions, and for the language modeling, we can see it as a classification head projecting onto `ntoken` dimensions (the length of the vocabulary).

3 Question 3

How many trainable parameters does the model have in the case of *language modeling* task and *classification* task. Please detail your answer.

In our model, we have set the following parameters:

```
ntokens = 50001    # the size of vocabulary
nhid = 200         # hidden dimension
nlayers = 4        # the number of Transformer layer
nhead = 2          # the number of heads in the multiheadattention models
```

and

```
nclasses = 50001   # for language modeling
nclasses = 2       # for classification
```

Now, to determine the total number of trainable parameters in the model for each task, let's look at the number of these parameters in each layer of the model, one by one.

• **Embedding layer**:
In the embedding block, we have $\texttt{ntokens} \times \texttt{nhid}$ parameters.
Thus, we obtain for this layer: $\texttt{ntokens} \times \texttt{nhid} = 50,001 \times 200 = \mathbf{10,000,200}$ trainable parameters.

• **Positional encoding**:
This part is non-trainable, as it is characterised by a fixed function. So we have **0** trainable parameters.

• **Transformer layers**:
There are $\texttt{nlayers} = 4$ Transformer layers in the model. For each one of them, we have:

- Self-attention:
This section is defined by several linear operations, for the queries, keys and values: W_Q, W_K, W_V that all three have dimension $\texttt{nhid} \times \texttt{nhid}$ and have associated biases of size $\texttt{nhid}$ each. The number of trainable parameters for the self-attention of one transformer layer is thus equal to: $\texttt{nhid} \times \texttt{nhid} \times 3 + \texttt{nhid} \times 3 = 200 \times 200 \times 3 + 200 \times 3 = 120,000 + 600 = 120,600$.

We then have the projection back to the $\texttt{nhid}$ dimension represented by W_0 of dimension $\texttt{nhid} \times \texttt{nhid}$ and its associated bias of size $\texttt{nhid}$. This provides us with $\texttt{nhid} \times \texttt{nhid} + \texttt{nhid} = 200 \times 200 + 200 = 40,200$ additional parameters.
In total, for self-attention, there are $120,600 + 40,200 = 160,800$ learnable parameters.

- Feed-forward:

The feed-forward of a Transformer layer include several steps:

- Normalization 1: We have dimensions of weights $= \texttt{nhid} = 200$, and bias $= \texttt{nhid} = 200$, so we have 400 parameters.
- Linear 1: Again, we have a weight matrix and bias vector jointly of dimension $\texttt{nhid} \times \texttt{nhid} + \texttt{nhid} = 200 \times 200 + 200 = 40,200$ parameters. This is usually a higher dimension projection, but in this case we remain in the same dimensions.
- ReLu function, for example: no trainable parameter.
- Linear 2: same as linear 1, $40,200$ parameters to learn. This is usually the projection back down to the original dimension.
- Normalization 2: same as normalization 1, so we have 400 parameters.

So for the feed-forward, there are $400 + 40,200 + 40,200 + 400 = 81,200$ learnable parameters.
Thus, there are $160,800 + 81,200 = 242,000$ trainable parameters for a single transformer layer. In total, for all the Transformer layers, we have $\texttt{nlayers} \times 242,000 = 4 \times 242,000 = \mathbf{968,000}$ trainable parameters.

• **Classification Head**:
The trainable parameters in this last linear layer are represented by weights $W \in \mathbb{R}^{\texttt{nclasses} \times \texttt{nhid}}$ and bias $b \in \mathbb{R}^{\texttt{nclasses}}$, so there are $\texttt{nclasses} \times \texttt{nhid} + \texttt{nclasses}$ parameters to learn.
For language modeling, this is equal to $50,001 \times 200 + 50,001 = \mathbf{10,050,020}$.
For classification, this is equal to $2 \times 200 + 2 = \mathbf{402}$.

Finally, now that we have looked into each layer, we can compute the TOTAL.

- Language modeling:
  Total number of trainable parameters in the model: $10,000,200 + 968,000 + 10,050,020 = \boxed{\mathbf{21,018,401}}$.
- Classification:
  Total number of trainable parameters in the model: $10,000,200 + 968,000 + 402 = \boxed{\mathbf{10,968,602}}$.

To verify these numbers, I have found the following function on *Stack Overflow*, which allows me to print exactly how many trainable parameters there are in the model, in the various steps:

```python
from prettytable import PrettyTable

def count_parameters(model):
    table = PrettyTable(["Modules", "Parameters"])
    total_params = 0
    for name, parameter in model.named_parameters():
        if not parameter.requires_grad:
            continue
        params = parameter.numel()
        table.add_row([name, params])
        total_params += params
    print(table)
    print(f"Total Trainable Params: {total_params}")
    return total_params
```

This has given me the outputs in Figures 1a and respectively 1b for the model with language modeling task, and respectively the one with binary classification task. We can see that the printed total number of parameters coincides with the result we have found.

Modules	Parameters
base.encoder.weight	10000200
base.transformer_encoder.layers.0.self_attn.in_proj_weight	120000
base.transformer_encoder.layers.0.self_attn.in_proj_bias	600
base.transformer_encoder.layers.0.self_attn.out_proj.weight	40000
base.transformer_encoder.layers.0.self_attn.out_proj.bias	200
base.transformer_encoder.layers.0.linear1.weight	40000
base.transformer_encoder.layers.0.linear1.bias	200
base.transformer_encoder.layers.0.linear2.weight	40000
base.transformer_encoder.layers.0.linear2.bias	200
base.transformer_encoder.layers.0.norm1.weight	200
base.transformer_encoder.layers.0.norm1.bias	200
base.transformer_encoder.layers.0.norm2.weight	200
base.transformer_encoder.layers.0.norm2.bias	200
base.transformer_encoder.layers.1.self_attn.in_proj_weight	120000
base.transformer_encoder.layers.1.self_attn.in_proj_bias	600
base.transformer_encoder.layers.1.self_attn.out_proj.weight	40000
base.transformer_encoder.layers.1.self_attn.out_proj.bias	200
base.transformer_encoder.layers.1.linear1.weight	40000
base.transformer_encoder.layers.1.linear1.bias	200
base.transformer_encoder.layers.1.linear2.weight	40000
base.transformer_encoder.layers.1.linear2.bias	200
base.transformer_encoder.layers.1.norm1.weight	200
base.transformer_encoder.layers.1.norm1.bias	200
base.transformer_encoder.layers.1.norm2.weight	200
base.transformer_encoder.layers.1.norm2.bias	200
base.transformer_encoder.layers.2.self_attn.in_proj_weight	120000
base.transformer_encoder.layers.2.self_attn.in_proj_bias	600
base.transformer_encoder.layers.2.self_attn.out_proj.weight	40000
base.transformer_encoder.layers.2.self_attn.out_proj.bias	200
base.transformer_encoder.layers.2.linear1.weight	40000
base.transformer_encoder.layers.2.linear1.bias	200
base.transformer_encoder.layers.2.linear2.weight	40000
base.transformer_encoder.layers.2.linear2.bias	200
base.transformer_encoder.layers.2.norm1.weight	200
base.transformer_encoder.layers.2.norm1.bias	200
base.transformer_encoder.layers.2.norm2.weight	200
base.transformer_encoder.layers.2.norm2.bias	200
base.transformer_encoder.layers.3.self_attn.in_proj_weight	120000
base.transformer_encoder.layers.3.self_attn.in_proj_bias	600
base.transformer_encoder.layers.3.self_attn.out_proj.weight	40000
base.transformer_encoder.layers.3.self_attn.out_proj.bias	200
base.transformer_encoder.layers.3.linear1.weight	40000
base.transformer_encoder.layers.3.linear1.bias	200
base.transformer_encoder.layers.3.linear2.weight	40000
base.transformer_encoder.layers.3.linear2.bias	200
base.transformer_encoder.layers.3.norm1.weight	200
base.transformer_encoder.layers.3.norm1.bias	200
base.transformer_encoder.layers.3.norm2.weight	200
base.transformer_encoder.layers.3.norm2.bias	200
classifier.decoder.weight	10000200
classifier.decoder.bias	50001

Total Trainable Params: 21018401

(a) Language modeling

Modules	Parameters
base.encoder.weight	10000200
base.transformer_encoder.layers.0.self_attn.in_proj_weight	120000
base.transformer_encoder.layers.0.self_attn.in_proj_bias	600
base.transformer_encoder.layers.0.self_attn.out_proj.weight	40000
base.transformer_encoder.layers.0.self_attn.out_proj.bias	200
base.transformer_encoder.layers.0.linear1.weight	40000
base.transformer_encoder.layers.0.linear1.bias	200
base.transformer_encoder.layers.0.linear2.weight	40000
base.transformer_encoder.layers.0.linear2.bias	200
base.transformer_encoder.layers.0.norm1.weight	200
base.transformer_encoder.layers.0.norm1.bias	200
base.transformer_encoder.layers.0.norm2.weight	200
base.transformer_encoder.layers.0.norm2.bias	200
base.transformer_encoder.layers.1.self_attn.in_proj_weight	120000
base.transformer_encoder.layers.1.self_attn.in_proj_bias	600
base.transformer_encoder.layers.1.self_attn.out_proj.weight	40000
base.transformer_encoder.layers.1.self_attn.out_proj.bias	200
base.transformer_encoder.layers.1.linear1.weight	40000
base.transformer_encoder.layers.1.linear1.bias	200
base.transformer_encoder.layers.1.linear2.weight	40000
base.transformer_encoder.layers.1.linear2.bias	200
base.transformer_encoder.layers.1.norm1.weight	200
base.transformer_encoder.layers.1.norm1.bias	200
base.transformer_encoder.layers.1.norm2.weight	200
base.transformer_encoder.layers.1.norm2.bias	200
base.transformer_encoder.layers.2.self_attn.in_proj_weight	120000
base.transformer_encoder.layers.2.self_attn.in_proj_bias	600
base.transformer_encoder.layers.2.self_attn.out_proj.weight	40000
base.transformer_encoder.layers.2.self_attn.out_proj.bias	200
base.transformer_encoder.layers.2.linear1.weight	40000
base.transformer_encoder.layers.2.linear1.bias	200
base.transformer_encoder.layers.2.linear2.weight	40000
base.transformer_encoder.layers.2.linear2.bias	200
base.transformer_encoder.layers.2.norm1.weight	200
base.transformer_encoder.layers.2.norm1.bias	200
base.transformer_encoder.layers.2.norm2.weight	200
base.transformer_encoder.layers.2.norm2.bias	200
base.transformer_encoder.layers.3.self_attn.in_proj_weight	120000
base.transformer_encoder.layers.3.self_attn.in_proj_bias	600
base.transformer_encoder.layers.3.self_attn.out_proj.weight	40000
base.transformer_encoder.layers.3.self_attn.out_proj.bias	200
base.transformer_encoder.layers.3.linear1.weight	40000
base.transformer_encoder.layers.3.linear1.bias	200
base.transformer_encoder.layers.3.linear2.weight	40000
base.transformer_encoder.layers.3.linear2.bias	200
base.transformer_encoder.layers.3.norm1.weight	200
base.transformer_encoder.layers.3.norm1.bias	200
base.transformer_encoder.layers.3.norm2.weight	200
base.transformer_encoder.layers.3.norm2.bias	200
classifier.decoder.weight	400
classifier.decoder.bias	2

Total Trainable Params: 10968602

(b) Classification

Figure 1: Output of the function for verification of the total number of trainable parameters for language modeling on the left and classification on the right

.

4 Question 4

Interpret the results.

The evolution of the accuracy for both methods — learning from scratch versus pre-trained model — can be seen plotted in figure 2.

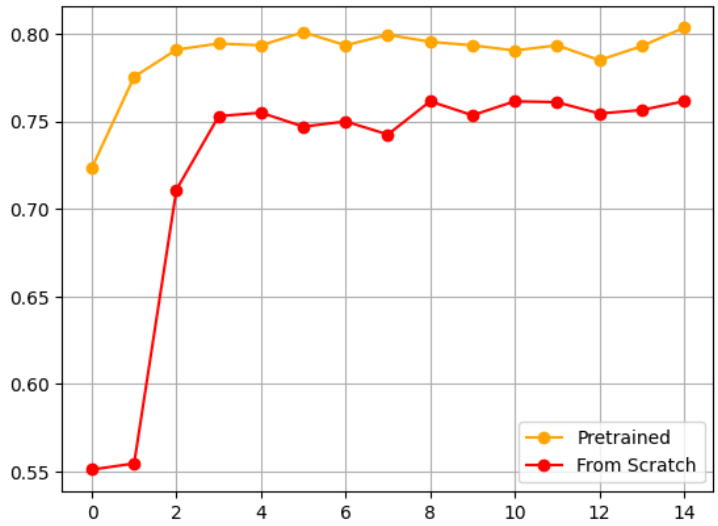

Figure 2: Evolution of the model's accuracy in function of the epoch in both settings: from scratch and pre-trained.

First of all, we can undoubtedly notice that the pre-trained model has a better accuracy than the model that has learned from scratch. However, it is important to keep in mind that the pre-trained model has an advantage, in essence, because it starts off with an accuracy of around 0.72, as it has been trained beforehand by definition, versus the model trained from scratch which starts off with a very bad accuracy of 0.55 (almost random labeling) as it has been initialised randomly, and then increases more gradually. This suggests that pre-training helps by providing a good starting point for the model.
Also, one could argue that the pre-trained model has a good accuracy but that could still be improved. Its accuracy quickly rises but then stagnates at a little under 0.8 in accuracy, which is not that promising. Maybe this is due to the fact that it has not trained on enough data, or rather because the model was not built specifically for classification tasks and could thus be improved in its architecture, which we discuss in question 5.

Overall, the plot still demonstrates the benefits of transfer learning as a way of having a better performance (and most importantly, faster training compared to training from scratch).

5 Question 5

What is one of the limitations of the language modeling objective used in this notebook, compared to the masked language model objective introduced in [1].

The main limitation of our model is its **unidirectional** nature. The model processes text from left to right, only considering previous tokens when predicting the next one. This can limit the model's ability to effectively understand the context, as well as effectively capturing the nuances of language. Having good understanding of the text is essential for objectives like ours in this Lab where we want to classify reviews. With its unidirectional approach, the model might miss important contextual information that could influence the tone or opinion of a review.
On the contrary, the BERT language model introduced in paper [1] allows the model to consider both left and right contexts simultaneously, by processing the text with a **bidirectional** architecture. Moreover, BERT's model employs a mixed strategy for masking tokens, which helps the model learn to predict missing words based on their context, which enhances the model's ability to understand language, contrarily to our unidirectional model where we potentially are less effective in classification tasks.
In summary, fine tuning a model like BERT's with a bidirectional approach would be more appropriate in our objective of classifying reviews, for the pre-training of our model, as it is more efficient in capturing contextual information of language.

References

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