1 Question 1

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

In our implementation, we defined a method *generate_square_subsequent_mask* in the TransformerModel class. This method takes as input a parameter *sz* that we will later set to take the value of the sequences' length in a batch. It then outputs a square matrix of size *sz* filled with 1 on the lower triangular part and $-\infty$ for the entries above the diagonal.
This method is later called when computing the Self-Attention: applying the so-defined mask before the softmax (cf. figure 1) allows to disregard every entry in the matrix product $QK^\top$ that corresponds to a token located in an ulterior position in the sequence.

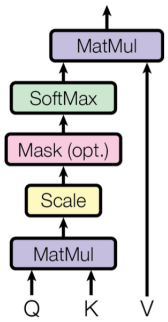

Figure 1: Scaled-Dot Product Attention [3]

Indeed, as the self-attention layer is only allowed to attend to earlier positions in the output sequence when training the Decoder part of the Transformer, setting the 'future tokens' to a large negative value modify the initial attention score matrix. Passing through the softmax function, those positions then practically get a value of 0, ensuring no attention is paid to them.

In the Transformer architecture, the mechanism of self-attention does not have any notion of the position or order of tokens. This is in contrast to recurrent architectures like LSTMs or GRUs, where the input sequence's order inherently influences the computations. Because sequences do have an order that is often vital to understanding them (e.g., "The hunter killed the deer" vs. "The deer killed the hunter"), the Transformer needs a way to consider token positions. Positional Encoding is introduced to give the Transformer model some information about the relative positions of tokens in a sequence. The main idea is to add a unique positional encoding to each token's embedding in a sequence so that the model can consider the order of tokens in its computations. This is done before feeding it to the self-attention layer, and it completely disregards the token's embedding value: the only thing that matters is the position of the token in the given sequence.

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?

Depending on the task we give to the model, the classification head needs to be changed. Indeed, when performing *language modeling*, the objective for the model is to continue the sequence it has been given as input: it iteratively generates tokens until predicting the one corresponding to the end of sentence. This can thus be considered as a classification task, with as many classes as there are entries in the vocabulary it has been trained on. The softmax applied to the output of the linear layer allow to get a probability vector of vocabulary's size: a greedy selection can then be used to predict the next most probable token.

The actual classification task considered here is sentiment analysis: for a set of book reviews, the model has to predict if each review is positive or negative. In this case, the number of classes is reduced to 2. It is thus

obvious that the linear layer has to be trained differently: the number of classes is different and the weights will be different since it is a different task.
We note as well that the *language modeling* task is a self-supervised learning one since it uses its input to predict other parts of its input, while the *classification* task has a binary label associated to each training example.

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.

For the base model, the trainable parameters (discarding the biases) are as follow:

- Embedding layer: $n_{tokens} \times n_{hid}$ (with n_{tokens} being the size of the vocabulary and n_{hid} the dimension of the embedding).
- Positional Encoding layer: No trainable parameters, n_{hid} fixed values added to the output of the embedding layer.
- Self-Attention layer: $n_k \times n_{hid}$ for the Query, $n_k \times n_{hid}$ for the Keys and $n_v \times n_{hid}$ for the values (the weights are the same for each token). Here, $n_k = n_v = n_{hid}/n_{head}$.
- Multi-Head Attention: $n_{head} \times w_{sal} + n_{hid} \times n_{hid}$ (with w_{sal} being the number of parameters in one Self-Attention layer, the second term corresponding to a projection after concatenation of the self-heads).
- Feed-forward layer: $n_{in} \times n_{ff} + n_{ff} \times n_{out}$. Here, $n_{in} = n_{ff} = n_{out} = n_{hid}$.
- Decoder: $n_{layers} \times w_{dec}$ (with w_{dec} being the number of parameters in one Decoder layer).

$P = n_{token} \times n_{hid} + n_{layers} \times \left(3 \times n_{hid}^2 + n_{hid}^2 + 2 \times n_{hid}^2\right) = n_{token} \times n_{hid} + 6 \times N \times n_{hid}^2$.

When performing language modeling, we add a linear output layer with $n_{hid} \times n_{classes} = n_{hid} \times n_{tokens}$.
When performing sentiment analysis, we add a linear output layer with $n_{hid} \times n_{classes} = n_{hid} \times 2$.

4 Question 4

Interpret the results.

We trained two models on the supervised classification task of sentiment analysis: one model was trained from scratch using randomly initialized weights, while the second model was initialized with the weights of a model pretrained for language modeling and only the weights of the classification head were trained.

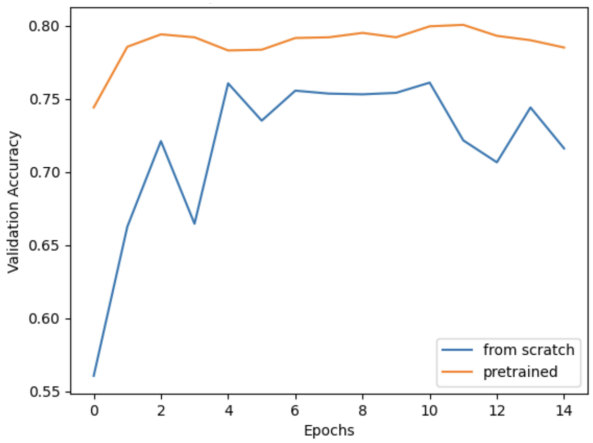

Figure 2: Validation accuracies' comparison of pretrained model vs. trained from scratch

After training for a few epochs, we observe a distinct gap in performance between the models. The one benefiting from pretrained Transformer's weights starts with an accuracy around 75% while the other one started with an accuracy of 50% corresponding to random guessing between the two classes. The pretrained model consistently outperforms the model trained from scratch, with a gap in accuracy between 5 and 10%

after 15 epochs.

We draw several conclusions regarding transfer learning from this analysis and from [2]. Training large Transformers from scratch can be computationally expensive and time-consuming. Pre-training on a general task and then fine-tuning on a smaller dataset for a specific task can thus accelerate the work and make it more computationally efficient: we see that if we wanted the model trained from scratch to have the same performance, it would require more training. Moreover, when dealing with supervised tasks with limited training examples, transfer learning can then prove to be efficient since the model could concentrate on learning the specific asked task while benefiting from the "semantic knowledge" of a pretrained model.

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].

In the previous question, we witnessed the performance of a pretrained model on binary classification for sentiment analysis. The accuracy obtained was close to 80% after a few epochs, which seemed relatively good but improvable. To perform the classification task, we used the Decoder part of a Transformer model pretrained on language modeling. The training of such model relies on selecting the output associated to the last token of a sequence and minimizing a loss function based on this output and the class we are supposed to predict. The rationale behind it is that in the Decoder part of a Transformer, a mask is used in the self-attention layer to prevent tokens from accessing the next ones and maintain an auto-regressive setting. Thus, taking the output of the last token is the only way to have information of all the past tokens already predicted: the last token of the sequence is the only one having access to the entirety of the sequence.

However, it is clear that the meaning of a word is highly contextual (e.g., the word bank in "the bank raised interest rates" and "the bank of the river" has a totally different meaning). An embedding relying solely on past tokens seems thus limited: to fully comprehend the information carried by a word, it makes sense that the model would need the whole sentence.

In [1], a new model called BERT is introduced. It deeply relies on the previous work made on Transformers, but this time the architecture and objectives are made differently. The Transformer was built with text translation as primary objective: an Encoder-Decoder architecture with a mask then made sense. For BERT, the objective was to train a model that would develop an acute semantic sense and capture complex language relations, and then use this base model for numerous *downtasks* (equivalent of transfer learning for specific supervised tasks). In this logic, the Decoder was considered no longer needed, but a bidirectional representation was introduced. To simplify, the idea was to give the model a sequence with masked tokens randomly chosen. The model would then try to predict the masked tokens based on every other token of the sequence (in both directions). This method enhanced the understanding of word meaning, and allowed BERT to achieve SOTA performance on several NLP supervised tasks benchmarks.

References

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

[2] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. 2018.

[3] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.