Question 1

What is the role of square mask in our implementation ?
Masking is used to ensure that during the generation of each token in the output sequence, the model can only attend to and consider information from previous tokens and not look ahead into the future. It prevents the model from 'cheating' on the next tokens.
The square mask is a binary mask applied to the attention scores to ensure that each position in the sequence can only attend to positions before it and not to itself or positions that come after it in the sequence.
Here is how the square mask works:

- For each position in the sequence, the mask is applied to the attention scores computed during self-attention.
- The mask is typically a square matrix where the upper triangular part (including the diagonal) is set to negative infinity, and the lower triangular part is set to zero.
- This mask effectively prevents each position from attending to itself and any positions that follow it in the sequence. It enforces a left-to-right or causal relationship during the decoding process.
- After applying the mask, the softmax function is used to compute the weighted sum of the values associated with each position based on the modified attention scores.

What is the role of the positional encoding ?
Transformers differ from sequential models like RNNs and LSTMs in that they lack an inherent mechanism to naturally capture the relative positions of words in a sentence.
Positional encoding involves creating positional encoding vectors that have the same dimensions as the word embedding vectors for each word in a sequence. These positional encoding vectors are then added to the corresponding word embedding vectors. By doing this, a distinct pattern is introduced into the embedding vectors that conveys spatial information. This pattern helps the model understand the relative positions of words in a sequence.

Question 2

Why do we have to replace the classification head?
The main reason for replacing the classification head is that the requirements of language modeling and classification tasks are different. In language modeling, the goal is typically to predict the next word in a sequence given the context, and this can involve predicting a probability distribution over a large vocabulary of words. In contrast, classification tasks involve assigning a specific label or category to an input.

What is the main difference between the language modeling and the classification tasks?
Language modeling typically operates in the space of a vocabulary, where the model predicts probabilities for each word in the vocabulary. Classification tasks operate in a label space, where the model predicts the class or category to which the input belongs.

Question 3

How many trainable parameters does the model have in the case of language modeling task and in the case of classification task ?

- Language Modelling Task:
  - Embedding Layer parameters: ntokens * nhid
  - Positional Encoding Layer: has no parameters

- The Transformer Encoder Parameters:
  Multi Head Attention (MHA):
  The self-attention mechanism includes three linear transformations:
  - Linear transformation for queries: nhid x nhid
  - Linear transformation for keys: nhid x nhid
  - Linear transformation for values: nhid x nhid
  Each of these transformations has nhid * nhid parameters. If we add bias, we will have 4*(nhid * nhid + nhid) the total number of parameters in this layer.
  Layer Norm: 2 * nhid.
  Feed Forward:
  The feedforward network consists of two linear transformations with a ReLU activation in between.
  The first linear layer has nhid * nhid parameters.
  The second linear layer has nhid * nhid parameters.
  If adding bias, we will have: 2*nhid*nhid + nhid + nhid
  So the number of parameters in the Transformer Encoder Layer are: Since we have a Linear Layer for MHA et Feed Forward and nlayers, the total number of parameters will be: nlayers * (4*(nhid * nhid + nhid) + 4*nhid + 2*nhid*nhid + nhid + nhid)
- Output Layer parameters:
  In a language modeling task, we train a transformer model to predict the probability distribution of the next word in a sequence given the context. So the ouptut layer is related to the size of the vocabulary. So the number of parameters is: nhid*ntokens + ntokens
- The total number of parameters is:
  ntokens*nhid + (4*nhid*(nhid+1) + 2*nhid*(nhid+1)+4*nhid)*nlayers + ntokens*(nhid+1)

- Classification Task:
  - For a classification task using a transformer-based model, the architecture is similar to that of a language model, but with a different output head. So the number of trainable parameters remain the same excpet for the ouput layer, which we have nhid * nclasses + nclasses parameters.
  - The total number of parameters is:
    ntokens*nhid + (4*nhid*(nhid+1) + 2*nhid*(nhid+1)+4*nhid)*nlayers + nclasses*(nhid+1)

Question 4

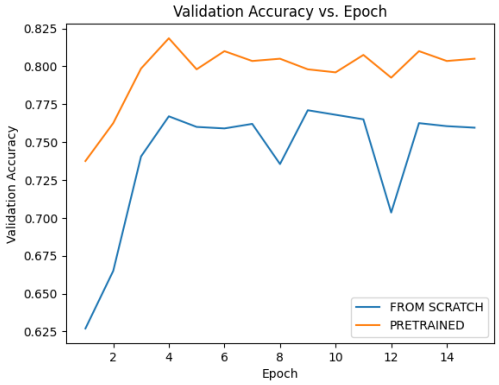

Figure 1: Validation Accuracy (Scratch VS Pretrained)

Interpret the results:
The pretrained model outperforms the model trained from scratch, validating the superiority of pretrained models. This underscores the significance of the time and resources saved during training, as these factors substantially contribute to the model's overall accuracy.

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]?

One of the limitations of unidirectional language models used in the notebook, as compared to the masked language model (MLM) presented in [1], is that unidirectional models like the left-to-right language models used in standard language models can only attend to previous tokens in the self-attention layers of the Transformer. This means that they can only incorporate information from the left side of a given token in the input sequence. This limitation is suboptimal for tasks that require a comprehensive understanding of context, especially for sentence-level tasks and token-level tasks like question answering.

In contrast, the MLM pre-training objective in BERT, which is bidirectional, randomly masks some tokens from the input and requires the model to predict the original vocabulary ID of the masked word based on its context. This bidirectional pre-training enables the model to fuse information from both the left and the right context of a token, allowing it to capture a more comprehensive understanding of the language and context. This is a significant advantage when compared to the unidirectional models, as it provides a more holistic view of the text and allows for improved performance on various natural language understanding 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*, 2018.