1 Role of the square mask and the positional encoding in our implementation

We use the square mask to allow the model to differentiate between actual input and padding and also to restrict the model from considering future tokens during the sequence processing. With the self-attention mechanism, each position in the input sequence can focus on others, guided by attention weights. To maintain the autoregressive nature of many sequential tasks, we apply a mask to these weights. This is done by setting the upper triangular part (including the diagonal) with negative infinity or a large negative value. Simultaneously, the lower triangular part is filled with zeros. This setup ensures that, during attention weight computation and the subsequent softmax operation, values in the upper triangular part become negligible, effectively preventing information flow from future positions.

On the other hand, positional encoding plays a key role in providing the model with context about the input sequence. Unlike RNN architectures that process sentences sequentially, transformers handle each token in parallel, lacking inherent knowledge of token order. This is where positional encoding steps in, supplementing input embeddings with information about the positions of tokens in a sentence. By doing so, it enables the transformer to capture sequential patterns and effectively manage variable-length sequences. In essence, both the square mask and positional encoding contribute to the robustness and effectiveness of our implementation.

2 Why do we need to replace the classification head? Difference between Language modeling and classification

During fine-tuning in transfer learning, especially with language models, we need to adapt the pre-trained model if we want to do a classification task. Generally in the pre-training phase the language model is trained to predict the next token in a sequence based on its context, we need to replace the classification head (the final layer of the neural network responsible for producing the output) if we are doing classification task because the objective is different (instead of predicting the next token, the model needs to classify the input sequence into predefined categories.) We then modify it to produce the appropriate number of class probabilities or scores.

- Language modeling:
  - Objective: The main goal of language modeling is to predict the probability distribution of the next word or token in a sequence given the context of preceding words
  - : The input in language modeling is the context of a sequence to predict the next element. The input is a sequence of words, and the model learns to capture syntatic and semantic relationships between words

  For example, given the input “The sky is”, the model might predict “blue” as the next word.

- Classification task
  - Objective: The main goal here is to categorize or classify input sequences or documents into predifined classes or labels. This is a supervised learning task.
  - Classification models take in a document or sequence of text and aim to assign it to one of several predefined classes
  - In the training step, the model requires here labeld training data where each input is associated with a specific class or label. The model learns to map input features to class labels during training. In prediction step, the model outputs a class label indicating the category to which the input sequence belongs.

  For example, given a movie review, the model might classify it as positive or negative

To summarize, language modeling is an unsupervised task where the goal is to predict the next token is a sequence whereas classification tasks aim at categorizing sequences into predifined classes.

3 Number of trainable parameters in language modeling task and classification task

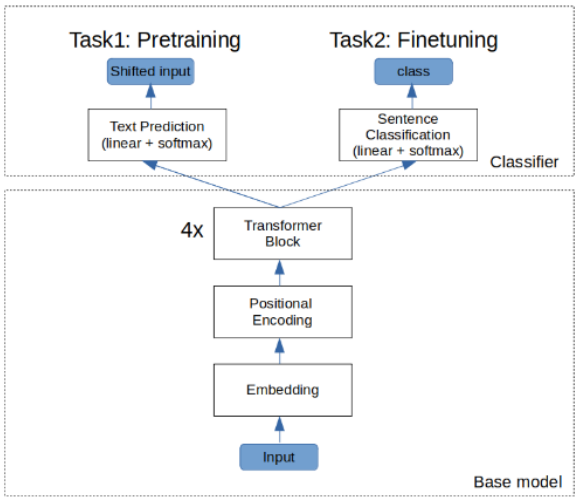

Figure 1: Visualization of the model"

In our implementation, we utilize the encoder for the language modeling task and an additional classification layer for the classification task. This section provides a detailed breakdown of the architecture to calculate the number of trainable parameters.

1. **Input Embeddings:** At this stage, the input sequence is tokenized, and each token is represented as an embedding vector of dimension N_{hid}. These vectors are learnable parameters. Therefore, the number of parameters at this stage is given by $P_{embeddings} = N_{tokens} \times N_{hid}$.
2. **Positional Encoding:** This step involves fixed matrices or vectors and does not contribute any learnable parameters.
3. **Encoder:** The encoder consists of several components:
   (a) **Multi-Head Attention Mechanism:** To compute the attention scores, three matrices (W_Q, W_K, and W_V) of size $N_{hid} \times N_{hid}$ are created. These matrices transform the input embeddings to Q', K', and V' through a dot product with Q, K and V. The number of attention heads does not influence the number of learnable parameters. We also have a weight matrix for the linear projection that has a shape of (N_{hid}, N_{hid}) Thus, the number of parameters at this stage is $P_{attention} = 4 \times N_{hid} \times N_{hid}$.
   (b) **"Add & Norm" Layer:** This layer contains learnable scale (alpha) and shift (beta) parameters. Therefore, the number of parameters at this stage is $P_{norm} = 2 \times N_{hid}$.
   (c) **Feed-Forward Network:** This network consist of 2 linear layers containing each a weight matrix of size $N_{hid} \times N_{hid}$ and a bias vector of size N_{hid}, yielding $P_{ffn} = 2 \times N_{hid}^2 + N_{hid}$ parameters.
   (d) **"Add & Norm" layer:** This second one follows the feed-forward network, contributing another $P_{norm2} = 2 \times N_{hid}$ parameters.

We conclude that the trainable parameters for the base model is:

$$Trainable\_parameters_{BaseModel} = P_{embeddings} + N_{layers} \times (P_{attention} + P_{norm} + P_{ffn} + P_{norm2})$$

1. **Language modelling:**
   For the language modelling task, we have a weight matrix of size (N_{hid}, N_{tokens}). Hence the number of trainable parameters is: $N_{hid} \times N_{tokens}$ We have:

$$\begin{aligned} Trainable\_parameters_{Modelling} &= P_{embeddings} + N_{layers} \times (P_{attention} + P_{norm} + P_{ffn} + P_{norm2}) + N_{hid} \times N_{tokens} \\ &= N_{tokens} \times N_{hid} + N_{layers} \times (4 \times N_{hid} \times N_{hid} \\ &\quad + 2 \times N_{hid} + 2 \times N_{hid}^2 + N_{hid} + 2 \times N_{hid}) + N_{hid} \times N_{tokens} \\ Trainable\_parameters_{Modelling} &= N_{tokens} \times N_{hid} + N_{layers} \times (6 \times N_{hid}^2 + 5 \times N_{hid}) + N_{hid} \times N_{tokens} \end{aligned}$$

   With $N_{tokens} = 50001, N_{hid} = 200, N_{layers} = 4$, we have:
   $Trainable\_parameters_{Modelling} = 20,964,400$

2. **Classification task:**
For the classification task, the input to the linear layer is of size $(N_{hid},)$ and the ouput is of size $N_{classes}$, then we have a trainable weight matrix of size $(N_{hid}, N_{classes})$. Hence there are $N_{hid} \times N_{classes} + N_{classes}$ trainable parameters. Hence the number of trainable parameters is: $N_{hid} \times N_{tokens}$ We have:

$$\begin{aligned} Trainable\_parameters_{classification} &= P_{embeddings} + N_{layers} \times (P_{attention} + P_{norm} + P_{ffn} + P_{norm2}) + N_{hid} \times N_{classes} \\ &= N_{tokens} \times N_{hid} + N_{layers} \times (4 \times N_{hid} \times N_{hid} \\ &\quad + 2 \times N_{hid} + 2 \times N_{hid}^2 + N_{hid} + 2 \times N_{hid}) + N_{hid} \times N_{classes} + N_{classes}) \\ Trainable\_parameters_{classification} &= N_{tokens} \times N_{hid} + N_{layers} \times (6 \times N_{hid}^2 + 5 \times N_{hid}) + N_{hid} \times N_{classes} + N_{classes} \end{aligned}$$

With $N_{tokens} = 50001, N_{hid} = 200, N_{layers} = 4$, we have:
$Trainable\_parameters_{Classification} = 10,964,602$

4 Question 4

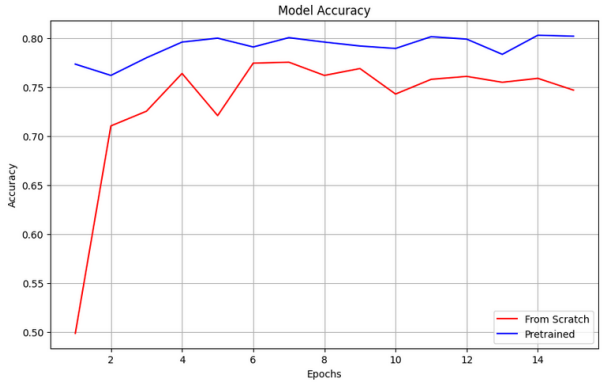

Figure 2: Visualization of the results of the pre-trained model and the model trained from scratch"

We can see on the plot that the pre-trained performed better than the model trained from scratch. Also we can notice that the pre-trained got a good starting compared to the model trained from scratch. This is very likely because its weighted were tuned or trained.

5 Limitations of the language modeling model, compared to the masked language model objective introduced in Bert

One of the limitations of our model is that it is unidirectional, meaning that it processes the text from left to right or right to left. This might limit the understanding of the full context from the model. For instance, in a sentence where future context is important for understanding the meaning of a current word, a unidirectional model might struggle to make accurate predictions.

On the other hand, BERT (Bidirectional Encoder Representations from Transformers) [1] introduces a new training objective known as the masked language model. In this approach, some percentage of the input tokens are masked at random, and then the model is trained to predict those masked tokens based on their context. Importantly, unlike traditional unidirectional language models , BERT takes into account both the left and the right context of a token during training. This means it uses both previous and future words in the context to predict the target word. This gives BERT a more comprehensive understanding of the text context.

References

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