1 Question 1

The **square mask** ensures that a token can only attend to preceding tokens or itself, but not to tokens that come after it in the sequence while being processed. Suppose we have a sequence of tokens: [x1, x2, x3, x4]. When token x3 is processed during self-attention, it can attend to tokens x1, x2, and itself due to the causality mask. However, it cannot attend to token x4, enforcing causality. This is crucial in tasks like language modeling and text generation, where the model should not have access to future information. The ”generate square subsequent mask method” creates this square mask, ensuring that each token only attends to earlier tokens in the sequence (or itself), but not to future tokens. This is especially important in causal language modeling tasks. The ”src mask” is passed as an argument to the Transformer encoder, allowing the mask to be applied to the multi-head attention mechanism to enforce the causal structure.

Positional encoding is a technique used in transformers to incorporate the sequential order of tokens into the input embeddings. Since transformers inherently lack the ability to capture token order, positional encoding explicitly provides information about the positions of tokens in a sequence. In the TransformerModel class, this encoding is applied in the forward method, right after converting the tokens into embeddings. This allows the embeddings to carry position information, enabling the transformer to account for the sequence structure of the data. The positional encoding is designed to be both learnable and structured, ensuring that each position in the sequence has a unique and distinctive representation.

2 Question 2

The classification head is the last layer of a neural network that will classify the data. It takes as an input the representation learn by the model and output a prediction (probability for each class). The classification head is often replaced in the Transformer model since it allows the model to be adapted to a new task or dataset that may have a different output space than the original pre-trained model. For example, BERT (a pre-trained transformer model) is trained for a specific task. If we want to use the same model for a different task, the output layer (that's to say the classification head) must be replaced.

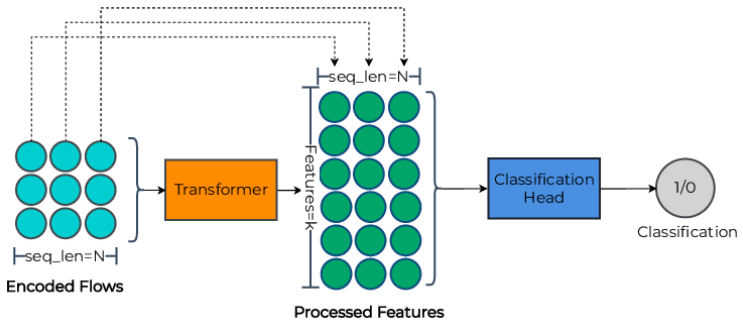

Figure 1: Vizualisation of the model

In Language modeling task, the goal is to predict the next token in a sequence given the previous tokens, whereas in a classification task, the goal is to assigne a label or category to the input data.

3 Question 3

We combine the notations of Attention is all you need and notations from the notebook. Let's count the number of training parameters.

First, the Embedding module *nn.Embedding(ntoken, nhid)* takes as an input parameters ntoken, nhid. It results that the corresponding matrix is size $\mathbb{R}^{\text{ntoken,nhid}}$.

Secondly, if we denote by **lengthM** the maximum length of a sentence, the positional encoding *PositionalEncoding(nhid, dropout)* is size $\mathbb{R}^{\text{lengthM,nhid}}$.

Then, the Transformer block is divided in four blocks : multi-head, feed-forward and two layer of normalization.

- **MultiHead** : MultiHead (Q, K, V) = Concat($\text{head}_1, ..., \text{head}_{\text{nhead}}$) W_0 with head_i = Attention(Q W_i^Q, K W_i^K,V W_i^v). Note that $W_0 \in \mathbb{R}^{\text{nhid} \times \text{nhid}}$ and $W_i^Q, W_i^K, W_i^V \in \mathbb{R}^{\text{nhid} \times \frac{\text{nhid}}{\text{nhid}}}$. Thus, the total number of trainable parameters is : $(\text{nhid}^2 + \text{nhid}) + 3 \times \text{nhid} \times (\text{nhid} \times \text{nhid} + \text{nhid}) = 4 \times (\textbf{nhid}^2 + \textbf{nhid})$.
- **FeedForward** : In addition to attention sub-layers, each of the layers in the encoder and decoder contains a fully connected feed-forward network, which is applied to each position separately and identically. This consists of two linear transformations with a **ReLU activation** in between. $\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$. We also know that dmodel = nhid and dff = nhid, since that, the number of trainable parameters is: $2(\text{nhid}^2 + \text{nhid})$.
- **Normalization** : The normalization layer consists in a multiplication and an addition by two vectors of size dmodel (= nhid). Since that, the number of trainable parameters is $2 \times$ nhid.

As so, the total number of training parameters for the transformer block is : $4 \times (N_{\text{MultiHead}} + N_{\text{FeedForward}} + N_{\text{Normalization}}) = 24\text{nhid}^2 + 40\text{nhid}$.

We can now compute $N_{\text{base model}} = \text{ntoken} \times \text{nhid} + \text{lengthM} \times \text{nhid} + 24\text{nhid}^2 + 40\text{nhid} = \text{nhid}(40 + \text{ntoken} + \text{lengthM} + 24\text{nhid})$.

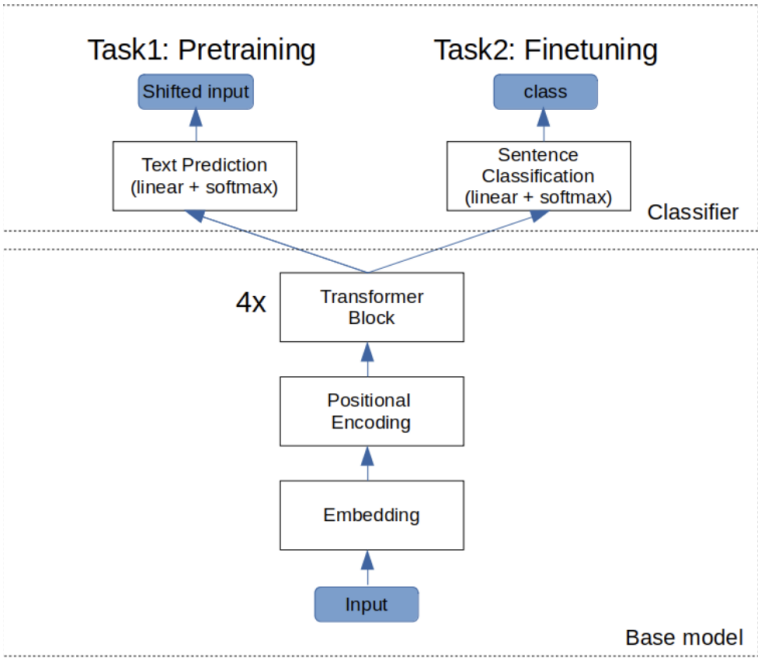

Figure 2: Vizualisation of the model

3.1 Language modelling task

The linear layer for text prediction has dimensions $(\text{nhid}, \text{ntokens})$. The total number of parameters of the language modelling task is :

$$\boxed{\text{nhid (40 + 2 ntoken + lengthM + 24 nhid)}}$$

.

3.2 Classification task

The linear layer for text prediction has dimensions $(\text{nhid}, \text{nclass})$. The total number of parameters of the language modelling task is :

$$\boxed{\text{nhid (40 + nclasses + ntoken + lengthM + 24 nhid)}}$$

.

4 Question 4

The two validation accuracy curves are increasing against epochs. The pre-trained model has a higher validation accuracy. This difference was expected. Generally, pre-trained models tend to generalize better to new data. Their generalization ability is valuable where data is not perfectly representative of the entire problem space. Models trained from scratch are more susceptible to overfitting the training dataset.

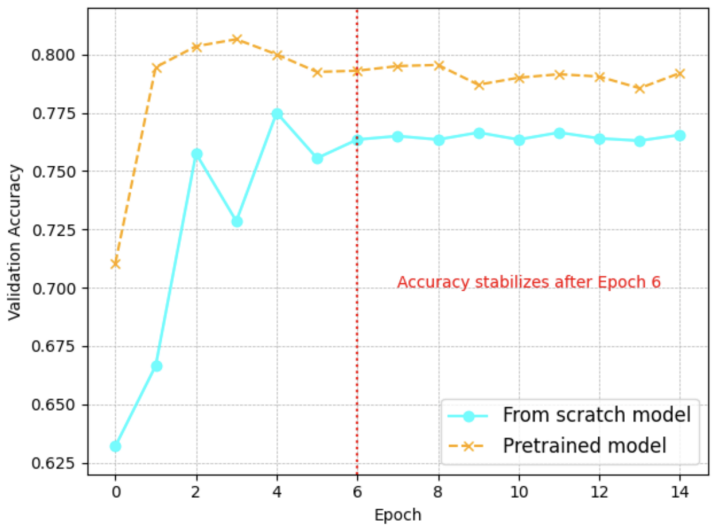

Figure 3: Validation Accuracy for Pretrained vs Scratch Model

5 Question 5

One of the limitations of the language modelling objective we used in this notebook is the lack of ability to capture the sentiment information. Our model learns during the pretraining to predict the next word in a sequence based on the context. While it is able to capture grammar, syntax, and some level of semantics, it doesn't inherently learn sentiment information. Sentiment analysis requires a different kind of knowledge that may not be present in the pretraining objective.

Models like **BERT**, introduced in Bert: Pre-training of deep bidirectional transformers for language understanding, in contrast, use a bidirectional language modeling objective. BERT is pre-trained by masking words in a sentence and predicting the masked words using both the left and right context. This bidirectional approach allows BERT to capture more nuanced and contextually relevant information, including sentiment-related cues. Moreover, BERT can be fine-tuned on specific tasks, such as sentiment analysis, by adding a classification head on top of the pre-trained transformer. This fine-tuning process, where the model is trained on labeled sentiment data, allows it to learn task-specific patterns, including positive or negative sentiments expressed in the text. Fine-tuning not only retains the general language understanding from the pretraining phase but also adapts the model to the specific nuances of the sentiment task.