Basile Terver
basile.terver@polytechnique.edu

# Lab session # 3
ALTEGRAD 2023

10/31/23

## 1 Question 1

### 1.1 Mask

In a Transformer, the self-attention operation takes as input a sequence $x$ and outputs a sequence $y$ of the same length, where $y$ is computed as $\sum_j w_{i,j}x_j$ and $w_{i,j}$ is the weight produced as a function of $x_i$ and $x_j$ by the self-attention mechanism. As described in [3] (Section 3.2.3), the decoder is made of self-attention layers that must prevent "leftward" information flow to preserve the auto-regressive property, that is the property that the decoder consumes previously generated symbols as additional input when generating the next output. Therefore, the decoder's self-attention layers allows each position to attend to all positions up to and including that position. This is done using an attention mask. An attention mask is used to "zero out" some attention weights in order to determine which entries of the input can be attended to during training. Three types of masks are depicted in Figure 1 from [2].

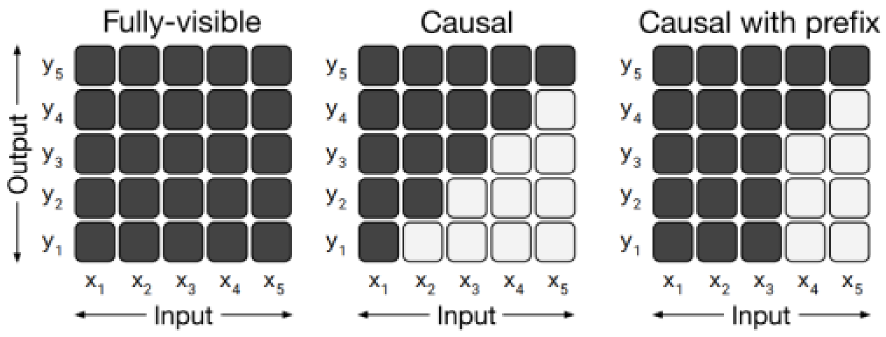

Figure 3: Matrices representing different attention mask patterns. The input and output of the self-attention mechanism are denoted $x$ and $y$ respectively. A dark cell at row $i$ and column $j$ indicates that the self-attention mechanism is allowed to attend to input element $j$ at output timestep $i$. A light cell indicates that the self-attention mechanism is *not* allowed to attend to the corresponding $i$ and $j$ combination. Left: A fully-visible mask allows the self-attention mechanism to attend to the full input at every output timestep. Middle: A causal mask prevents the $i$th output element from depending on any input elements from "the future". Right: Causal masking with a prefix allows the self-attention mechanism to use fully-visible masking on a portion of the input sequence.

Figure 1: Illustration of three masks.

Our code produces a triangular mask similar to the causal mask of 1 which sets to $-\infty$ all values in the input which correspond to "illegal" connections.
Thus, our model is a simpler version of the one of [3]. We do not have different transformer layers for the encoding and decoding. We just have positional encoding of the input encoder and 4 layers of transformers with a "decoder" (i.e. causal) mask, this corresponds to the right part in Figure 2 but without the encoder-decoder attention sublayer in each of our $N = 4$ layers.

### 1.2 Positional encoding

As explained in [3] (Section 3.5), since our model does not use recurrence (RNNs for example) and no convolution, we must ensure to give it some information about the positions of the tokens in the sequence it is given as input.
For various reasons, a single number, such as the index value, is not used to represent position in transformer models. The indices can grow large in magnitude for long sequences. Normalizing the index value between 0 and 1 can create problems for variable length sequences since they would be normalized differently.

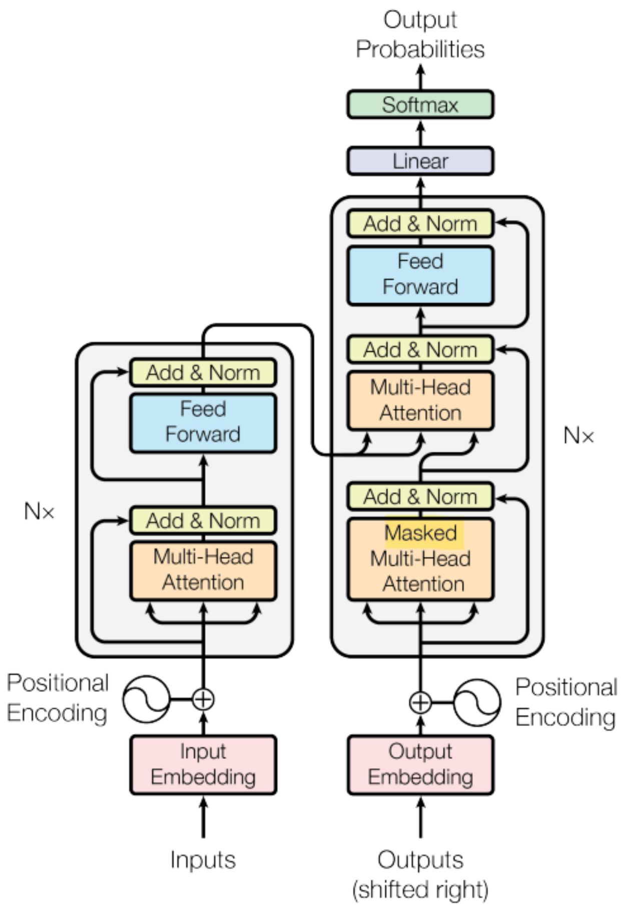

Figure 2: The Transformer architecture. The part framed in red corresponds to what we use in this lab.

To circumvent this, transformers use wave frequencies to encode position. Indeed, the position is mapped to a vector. The positional encoding layer outputs a matrix, where each row represents an encoded object of the sequence with its positional information.
In [3], they use sine and cosine functions of different frequencies

$$\begin{aligned} PE_{(pos,2i)} &= \sin(pos/10000^{2i/d_{model}}) \\ PE_{(pos,2i+1)} &= \cos(pos/10000^{2i/d_{model}}) \end{aligned} \tag{1}$$

where $pos$ is the position and $i$ is the dimension. Each dimension of the positional encoding thus corresponds to a sinusoid.

## 2 Question 2

In this notebook, the `TransformerModel()` has its last layer (module) named `ClassificationHead` but we will use it as a Text Prediction head in the pretraining of Task 4 as we will build the model with `nclasses=ntokens`. Indeed, with such a number of "classes", the Transformer will output a probability distribution of vocabulary size, that is it will predict the next token of the output sentence recursively.
A real classification head is when `nclasses = 2`. The task is very different from language modeling. Language modeling consists in predicting an output sequence from an input sequence of tokens autoregressively ("seq2seq" model as in first lab). This involves understanding context, links between grammatical structures. A model trained on a language modeling task can be used for text completion, generation, in addition to just translation.
Conversely, sentence classification consists in outputting whether the input sentence is positive or negative about what it reviews.
Pretraining is done in "language modeling mode", then the last layer is changed to a classification head and the weights of the pretrained model "transferred" to accelerate training on the classification task, as we will see in Task 7.

# 3 Question 3

## 3.1 Base model

As stated in the Pytorch documentation about the TransformerEncoderLayer() function, it is an based on the paper [3]. A Transformer encoder layer is made up of a self-attention layer and a simple, position-wise fully connected feedforward network.
In Section 3.2.2 of this paper, the number of parameters of each multi-attention head is given by the parameter matrices $W_i^Q \in \mathbb{R}^{d_{model} \times d_k}, W_i^K \in \mathbb{R}^{d_{model} \times d_k}, W_i^V \in \mathbb{R}^{d_{model} \times d_v}$ and $W^O \in \mathbb{R}^{hd_v \times d_{model}}$. The multi-head attention is given by

$$\begin{aligned} \text{Multihead(Q,K,V)} &= \text{Concat}(\text{head}_1, \dots, \text{head}_h) W^O \\ \text{where head}_i &= \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \end{aligned} \tag{2}$$

We employ $h$ parallel attention layers, or heads. For each of these we use $d_k = d_v = d_{model}/h$.
In each layer, after the attention sub-layer, the feedforward consists of two linear transformations with a ReLU activation, that is

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2 \tag{3}$$

The dimensionality of input and output is $d_{model}$ so $W_1, W_2 \in \mathbb{R}^{d_{model} \times d_{model}}$.
After both the self-attention and the feedforward sublayers, the "Add and norm" sublayer does not add any trainable parameters.
In our code `nhid` and `nhead` correspond respectively to $d_{model}$ and $h$ from the paper.
The parameters of the base model should therefore be, for each of the transformer encoder layers $W_i^Q \in \mathbb{R}^{d_{model} \times d_k}, W_i^K \in \mathbb{R}^{d_{model} \times d_k}, W_i^V \in \mathbb{R}^{d_{model} \times d_v}, W^O \in \mathbb{R}^{hd_v \times d_{model}}$ for $i \in \{1, \dots, h\}$ and $W_1, W_2$. This yields

$$\begin{aligned} &N \times \left[h\Big(d_{model}d_k + d_{model}d_k + d_{model}d_v\Big) + hd_vd_{model} + 2\big(d_{model}^2 + d_{model}\big)\right] \\ &= N \times \left[\underbrace{3d_{model}^2}_{attention} + \underbrace{d_{model}^2}_{concat} + \underbrace{2\big(d_{model}^2 + d_{model}\big)}_{feedforward}\right] \end{aligned} \tag{4}$$

trainable parameters.
Printing all named trainable parameters of our base model, we realize that they are some more parameters, not present in the article [3], such as biases associated to $W^O, W^Q, W^V, W^K$, that is $N$ times $4d_{model}$ more parameters. Moreover, each "Add and Norm" layer has $4d_{model}$ trainable parameters, so that is $4Nd_{model}$ more parameters.
In addition to this transformer model, one should not forget the embedding layer which maps the input of size the number of tokens $n_{tokens}$ to the feature vector of size $d_{model}$, which adds $n_{tokens}d_{model}$ parameters.
In the end, the base model has

$$N \times \left[\underbrace{3(d_{model}^2 + d_{model})}_{attention} + \underbrace{d_{model}^2 + d_{model}}_{concat} + \underbrace{2\big(d_{model}^2 + d_{model}\big)}_{feedforward} + \underbrace{4d_{model}}_{norm}\right] + \underbrace{n_{tokens}d_{model}}_{embedding} \tag{5}$$

trainable parameters.

## 3.2 Classification

In the classification task, the output of the base model, which is of size $d_{model}$, goes through a fully connected feedforward layer to yield an output of size $n_{classes}$, which is 2 in the case of sentiment analysis. Thus, the classification layer has $n_{classes}d_{model} + n_{classes}$ parameters.

## 3.3 Language modeling

In the language modeling task, the output of the base model also goes through a feedforward layer to yield an output of size $n_{tokens}$, which is 2 in the case of sentiment analysis. Thus, the language modeling layer has $n_{tokens}d_{model} + n_{tokens}$ parameters.

### 3.4 Numerical results

We have $N = 4$ Transformer Encoder Layers, $d_{model} = 200$ with 2-headed attention heads ($h = 2$) and a vocabulary of size $n_{tokens} = 50001$.This yields

- $4[6(40000 + 200) + 800] + 50001 \times 200 = 10968200$ parameters for the base model.
- In ”classification mode”, the last layer has 402 parameters.
- In ”language modeling mode”, the last layer has 10050201 parameters.

## 4 Question 4

We can see, in Figure 3, a performance gap between the pre-trained model, which has better accuracy on the validation set throughout the training epochs. The training we perform here is for a classification task, which, as seen in Question 1, is very different from the language modeling task on which the model has been pre-trained.
Yet, throughout 15 training epochs on the classification task, it seems that the accuracy of the ”from sratch” model is not about to catch up with the accuracy of the pre-trained model. Both accuracy curves seem to plateau, the ”from sratch” curve at 0.75 and the pre-trained one at 0.8 accuracy.
This can be explained by the fact that language models acquire a high-level language representation which allows, as a corollary to perform specific tasks such as sentence classification.
Another fact supporting this hypothesis is that the accuracy curve of the pre-trained only seems to have very slightly increases throughout the training epochs (on the classification task). This shows that the first training epoch has done most of the work to adapt the pre-trained language model to this new classification task.
Sometimes, to avoid many training epochs on a particular task (here sentence classification), that can be called ”sub-task” of an another task (here language modeling), using a pre-trained model on the more general task is computationally more efficient.

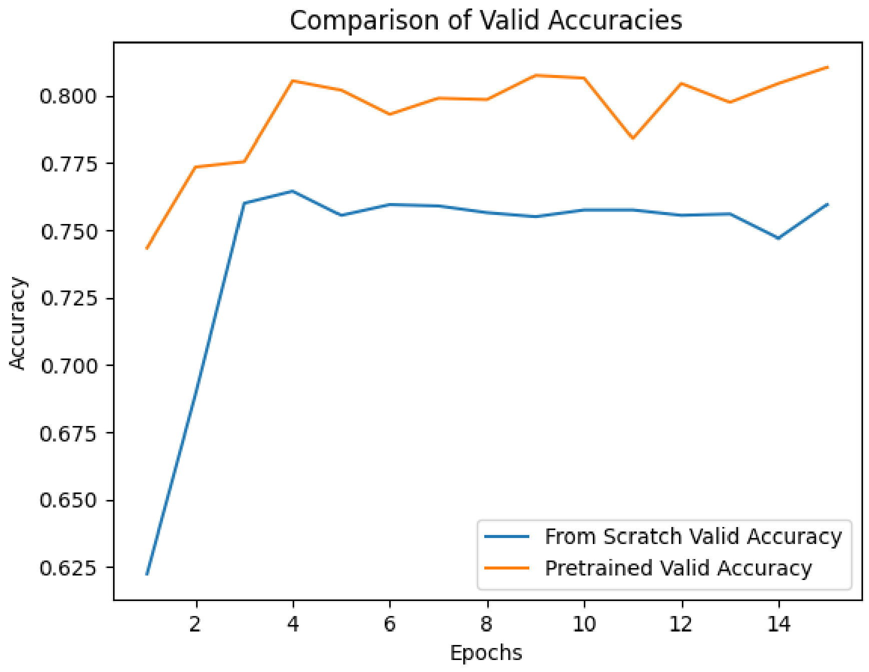

Figure 3: Plot of the evolution of the accuracy of the pre-trained and the ”from scratch” models on the validation set during training on the classification task.

## 5 Question 5

One of the limitations of the language modeling objective used in [3] is the lack of bidirectionality. The model is trained to predict the next word in a sequence given the previous words, essentially in a left-to-right manner, because of the causal mask discussed in Question 1. This limits the ability of the model to capture dependencies that might exist between the left and right contexts within a sentence, as the model does not have access to the

entire context during training. Consequently, this may hinder the understanding of long-range dependencies and relationships within the text.
On the other hand, the masked language model (MLM) objective used in BERT [1] allows the model to consider bidirectional context during pre-training. Specifically, BERT masks some of the words in the input and trains the model to predict the masked words based on the surrounding context, which enables the model to understand the context of a word by considering both its left and right contexts. This bidirectional approach helps BERT to capture more nuanced relationships and dependencies within the text, making it more effective for various downstream NLP tasks.

## References

[1] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[2] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer, 2023.

[3] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.