CALLARD Baptiste
baptcallard@gmail.com

Lab session # 3
ALTEGRAD 2023

10/31/23

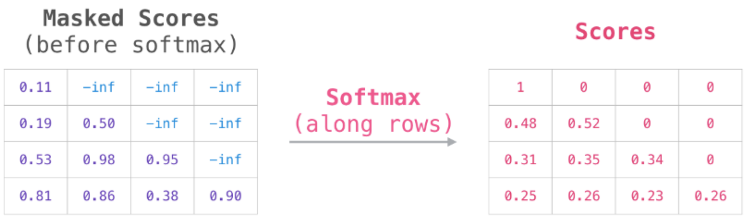
1 Question 1

1.1 Square mask

The **square mask** plays a key role in the self-attention mechanism. During learning, this mask ensures that the model only has access to tokens that have an earlier position in the input sequence. The model must not see the future tokens it is trying to predict. For a sequence of dimension n, the square mask is an n x n matrix. This is a way of avoiding "cheating" by looking into the future :

$$src\_mask = \begin{pmatrix} 0 & -\infty & \cdots & -\infty \\ \vdots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & -\infty \\ 0 & \cdots & \cdots & 0 \end{pmatrix}$$

This mask is added to the QK^T matrix before the softmax step. The $infty$ values are used to mask future tokens. This allows us to disable future tokens which will not be taken into account in the self-attention mechanism. The probability density is redistributed to the previous tokens (see figure 1 where Masked Score = QK^T + src_mask).



Masked Scores (before softmax)

0.11	-inf	-inf	-inf
0.19	0.50	-inf	-inf
0.53	0.98	0.95	-inf
0.81	0.86	0.38	0.90

Softmax (along rows)

Scores

1	0	0	0
0.48	0.52	0	0
0.31	0.35	0.34	0
0.25	0.26	0.23	0.26

Figure 1: (source) The Illustrated GPT-2 (Visualizing Transformer Language Models) by Jay Alammar

1.2 Potional encoding

To better understand the role of positional encoding. I will quickly explain the mechanism of attention. Let's denote x_e the integration vector of our input sequence and x_p the positional encoding. The key equation for attention in transformers is :

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Suppose we only use x_e without adding x_p, then $Q = W_Q x_e$, $K = W_K x_e$ et $V = W_V x_e$. Attention information is still taken into account if we only consider x_e.

However, the tokens are processed independently of each other and the order of the initial sequence is missing (matrix products are used). We do not use a sequential architecture such as RNN, LSTM, etc. Additional information is therefore needed to differentiate the positions of the tokens in a sequence. For example, the word "yet" in English often has different meanings if it begins the sentence or if it ends it. Positional encoding is added to token embeddings, allowing the model to take into account the order of words in the input. So, by calculating $Q = W_Q(x_e + x_p)$, $K = W_K(x_e + x_p)$ and $V = W_V(x_e + x_p)$, we incorporate the place of the word in the sentence. This addition allows the transformer to make meaningful contextual decisions. There are formulas for systematically calculating positional coding (sine and cosine functions), but it is also possible to learn it during training.

2 Question 2

The language modelling task consists of learning the structure of natural language at different hierarchical levels: word scale, context within a sentence, etc. in order to be able to predict, on the basis of the tokens received, the next most likely token. This generative task is performed recursively to produce the next tokens. In contrast, the sentiment classification task involves predicting the most likely sentiment in a binary fashion (positive or negative). The task is therefore different and non-generative, since we are trying to extract a single output using very global information from an input. Both models will use the base model which is responsible for extracting information from natural language. As the two models do not have the same task and do not return a vector of the same size, we simply modify the last layer, known as the head, to adapt it to the task.

One difference between the two tasks is that the language modelling training can be done **without labels**. The classification task, on the other hand, **requires labels**.

The task we are interested in is classification. A lot of data is needed to train a transformer. If we wanted to train the classifier from scratch, we would need a huge amount of labelled data, which may be expensive or inaccessible. The idea is to train the Base Model on the language modelling task, which doesn't require labels (abundance of data). This allows the model to warm up and learn the natural language structures. We can then specialise the model by finetuning it for the classification task using a smaller corpus of data by just changing the last layer.

3 Question 3

We will detail the number of trainable parameters of the different parts of the transformer decoder.
The values of the TPs will be used for the calculations. We also use the notation :

ntokens	nhid	nlayers	nhead
100	200	4	2

Base Model :

Layer	Param's #	Total
Embedding	ntokens × nhid	20000

Positinal Encoding: has no trainable parameter

Layer	Param's #	Total
Postional Encoding	0	0

We have **n_layer** of transformer block. I detail **one** Transformer Block with **n_head** heads:

Comments: When we increase the number of heads, a first way is just to duplicate in parallel the calculations with other projection matrices. However, this increases the number of parameters in the linear part of the attention mechanism. It has therefore been proposed that each head should produce a latent vector of dimension $\frac{nhid}{n\_head}$ so that the number of heads has no influence on the number of parameters. The projection step is then performed after concatenating the outputs of all the heads.

Layer	Param's #	Total
n_head × (W_Q + b_Q)	n_head × (($nhid^2$ + $nhid$) / n_head)	40200
n_head × (W_V + b_V)	n_head × (($nhid^2$ + $nhid$) / n_head)	40200
n_head × (W_K + b_K)	n_head × (($nhid^2$ + $nhid$) / n_head)	40200
Projection after concatenation of heads (Weights + biais)	$nhid^2 + nhid$	40200
2 × MLP	2× ($nhid^2$ + $nhid$)	80400
2 × Layer Norm	2 × ($nhid$ + $nhid$)	800
	Sum for one layer	242000
	Sum for nlayers	968000

Finally the Base Model has : $nlayers\,(nhid(6(nhid+1)+4)) + ntokens.nhid$ = 20000+968000 = 988000 trainable parameters

Classification layer:

- For Text prediction (language modelling) : nclasses = ntokens (voc. size)

Layer	Param's #	Total
Classifier (Linear Layer)	nhid × nclasses + nclasses	20100

The Base Model for text prediction : $nlayers\,(nhid(6(nhid+1)+4)) + ntokens.nhid + ntokens(nhid+1) =$ 988000 + 20100 = 1008100 trainable parameters

- For Text classification has : nclasses = 2 (positive or negative)

Layer	Param's #	Total
Classifier (Linear Layer)	nhid $\times$ nclasses + nclasses	402

The Base Model for text classification has : $nlayers\,(nhid(6(nhid+1)+4)) + 2.(nhid+1)$ = 988000 + 402 = 988402 trainable parameters

4 Question 4

The figure 2 contains **the validation** results in both settings (from scratch and fintuning). I use dropout = 0.2 as regularization to prevent overfitting.

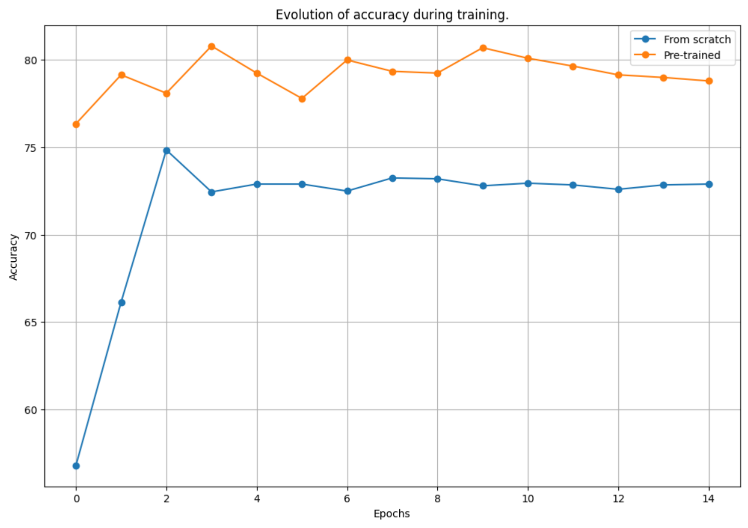


Figure 2: Evolution of accuracy **for valdiation** during training with (scratch & finetuning approaches)

There are a number of things to note:

- The pre-trained model has much better accuracy after convergence ($\approx$ 80%) than the model from scratch ($\approx$ 73%).
- The pre-trained model takes is faster to converge and start closer to his optimal setting. The from scratch goes from $\approx$ 55% to $\approx$ 73% in 2 epochs, while pretrained goes from $\approx$ 76% to $\approx$ 80%.
- Both models quickly reach a plateau, needing only a few epochs to specialise and reach their maximum accuracy.

The fact that the from **scratch model** underperforms is due to the fact that the training database is not large enough to learn properly. Overfitting could also occur. The model has to learn how a text is constructed and at the same time learn how to classify a text correctly from a little dataset.

In contrast, the **pre-trained model** already has a priori knowledge of the grammatical and semantic construction of a sentence learned from a large corpus. As a result, it has less to learn about natural language and can ”concentrate” on the classification part, making full use of the labelled dataset. In addition, we only modify the last layer, which is shallow and therefore easy to learn and fast convergence.

5 Question 5

Our aim is to classify texts by finetuning a pre-trained model on text prediction task (need no labelled data). The problem with our approach is that we train on a text generation task which is **unidirectional**. In our case, this can be seen when we use the Square Mask where the model only has access to the previous tokens. We then want to finetune a task for which we would like to have global information and as much context as possible. We therefore need to have bidirectional attention (to have as much context as possible).

The article that introduced the famous Bert architecture [1] is very popular because it raised this limitation of the approach we have just discussed and proposed a solution. They say that existing methods limit the effectiveness of pre-trained models, particularly during finetuning. One of the main constraints is that conventional linguistic models operate unidirectionally (left-to-right architecture). However, it is an advantage for many tasks to have this bidirectional representation. For example, we would like the information to flow in both directions for text classification, to have better contexts. Their approach to obtaining a bidirectional model consists during the traning step of randomly masking some of the tokens in the input sequence. The aim is then to predict the masked words from the context. So, with this approach we can still do pre-training without labelled data with bidirectional considerations, and get pre-trained model for a wider variety of problems.

References

[1] Jacob devlin, ming-wei chang, kenton lee, and kristina toutanova. bert: Pre-training of deep bidirectional transformers for language understanding. arxiv preprint arxiv:1810.04805, 2018.