Mathis EMBIT
mathis.embit@gmail.com

Lab session # 3
ALTEGRAD 2023

10/31/23

1 Question 1

- We need a square mask to prevent the self-attention layer from looking at future words in the sequence. This is because we want the output sequence to be sampled auto-regressively. Using a mask also allows parallel computation during training.
- As the attention layer do not process words sequentially we need to find a way of precising the word order to the network. We do this by adding the position of each word in the sequence to their embedding. That is called positional encoding. Used with attention mechanism, positional encoding performs better than sequential models. It is crucial to precise words positions to the network one way or another because it allows to contextualize depending on the position. This way the network can understand different meanings of a same word.

2 Question 2

- We have to replace the classification head because we often have a lot more data and obtain a more generalized model when training for text prediction. The aim of the transformer is to understand natural language so we can guess that there is a non empty intersection between classifying text and generating text with a transformer. We then train a big model to generate text and fine tune it on a smaller amount of data for a specific task for example sentiment analysis. We do this by changing the head of the model and keeping the pretrained base.
- Language modeling consists in predicting patterns in natural language, typically predicting a probability over the next word of a sentence, whereas classification tasks focus on assigning predefined labels to input text based on learned patterns from labeled data.

3 Question 3

We have those hyperparameters:

- V = **ntokens** is the size of vocabulary
- D = **nhid** is the dimension of the embeddings computed from the feedforward network model in nn.TransformerEncoder
- L = **nlayers** is the number of nn.TransformerEncoderLayer in nn.TransformerEncoder
- H = **nhead** is the number of heads in the multi-head attention models
- C = **nclasses** is the number of classes (= **ntokens** for language modelling task)

We also add that a attention head dimensionality is : $d_{head} = \frac{D}{H}$.
From these hyperparameters we can compute the number of trainable parameters:

- To embed we have $V \times D$ trainable parameters
- A multi head attention layer contains the (key, query, value) linear projection of size $H \times 3 \times (D \times d_{head} + d_{head})$ and a final concatenation linear projection of size $D \times D$ for a total of $H \times 3 \times D \times d_{head} + D^2$ trainable parameters
- An add and norm layer contains $2 \times D$ trainable parameters
- A feed forward network contains $D^2 + D$ trainable parameters
- We have L encoder layers. Each encoder layer contains a multi head attention, followed by a norm layer, followed by a feed forward network, followed by a second norm layer: $L \times (H \times 3 \times (D \times d_{head} + d_{head}) + D^2 + 2 \times D + D^2 + D + 2 \times D)$ trainable parameters
- Finally, the classifier head contains a linear layer of $D \times C + C$ trainable parameters

We have **Total number of trainable parameters**
$= V \times D + L \times (H \times 3 \times (D \times d_{head} + d_{head}) + D^2 + 2 \times D + D^2 + D + 2 \times D) + D \times C + C$
$= V \times D + L \times (H \times 3 \times (D \times \frac{D}{H} + \frac{D}{H}) + D^2 + 2 \times D + D^2 + D + 2 \times D) + D \times C + C$
$= V \times D + L \times (3 \times (D^2 + D)) + D^2 + 2 \times D + D^2 + D + 2 \times D) + D \times C + C$

- For **language modeling task**, we have those values:
  - V = 50,001
  - D = 200
  - L = 4
  - H = 2
  - C = 50,001

  Which leads to a total of **20,613,801** trainable parameters.

  From my python code, I computed 10,968,200 trainable parameters in the base model and 10,050,201 in the classifier head. Which leads to a total of **21,018,401** trainable parameters for the whole language modelling task model. It's not so far but I probably made a mistake in my formula.

- For **classification task**, we have those parameters:
  - V = 50,001
  - D = 200
  - L = 4
  - H = 2
  - C = 2

  Which leads to a total of **10,564,002** trainable parameters.

  From my python code, I computed 10,968,200 trainable parameters in the base model and 402 in the classifier head. Which leads to a total of **10,968,602** trainable parameters for the whole classification task model. It's not so far but I probably made a mistake in my formula.

4 Question 4

Here is the graph I obtain:

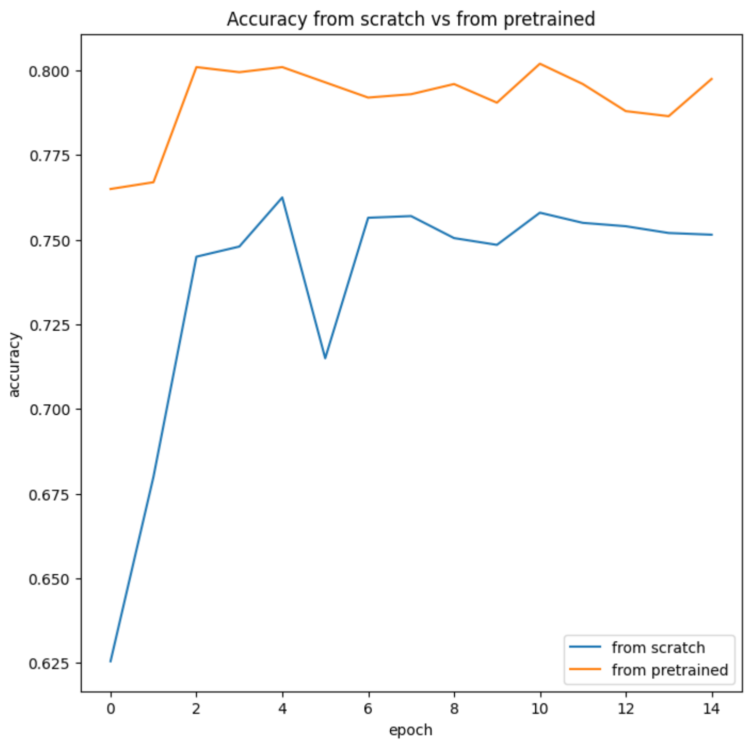

Figure 1: Evolution of the accuracy of the model trained from scratch vs pretrained

We observe that the pretrained model has already a good accuracy at the begenning, and then need fewer epochs than the one trained from scratch to reach a given accuracy. Eventhough the pretrained model performs better, the model trained from scratch drastically improves its accuracy during the first 4 epochs before stabilizing from the fourth to the fifteenth.

5 Question 5

The objective we used consists in the prediction of the next token given the previous tokens in a sequence. The sequential nature of such a model makes training and sampling computationally expensive and time-consuming. Furthermore it limits their ability to parallelize efficiently. Beyond these computational considerations, the context of our model is quiet limited. It feels natural to generate from left to right but a machine can benefit from having a larger context.

In ”BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding” [1], researchers introduce a masked language model. Its objective is to predict randomly masked token from an input sentence. It performs this task by using the context of the whole sentence. Thus, the main difference is that masked language model use the whole sentence as context and is no more autoregressive werease our model use context from the past without looking at the future words coming in the sentence.

Other problems can be pointed out in autoregressive objective. Their is a strong bias in the sampling as the model has to rely on its previous prediction to predicts the next token. The longer the generation is, the more amplified the bias. Relying on its own prediction also makes the model sensitive to perturbations which is a clear lack of robustness.

References

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