Benjamin Lapostolle
benjamin.jean.lapostolle@gmail.com

Lab session # 4
ALTEGRAD 2023

11/13/23

1 Question 1

Let's compute the number of parameters in our model. We omit the biases and the parameters of the normalization layers and of the language model head.
We have:

The embeddings layers

- The embedding of the tokens: Projet the $32,000$ tokens into a 512 space dimension, so a total of $16,384,000$ parameters.
- The embedding of the positions: We have 258 embedded into 512, so we have $132,096$ parameters.
- **In total:** $16,384,000 + 132,096 = 16,516,096$ parameters

The encoders

- Matrix K, V, Q: 3 layers of $512 \times 512 = 262,144$ parameters, so a total of $786,432$.
- Output linear transformation: $512 \times 512 = 262,144$ parameters
- Feedforward Network: two linear layers of $512 \times 512 = 262,144$ parameters so a total of $524,288$ parameters.
- **In total:** $786,432 + 262,144 + 524,288 = 1,572,864$ parameters.

In total, as we have 4 encoders, we have: $16,516,096 + 4 \times 1,572,864 = 22,807,552$ parameters. We find the same result when computing the sum of the parameters in the notebook.

2 Task 3 and Task 4

Here are the figures obtained on tensorboard for the task 3 and 41. We clearly see here that for the 3 seeds tested, we have better results when using pretrained-weights as initialisation than when using random weights. The accuracy score are not close at the end of the training phase.

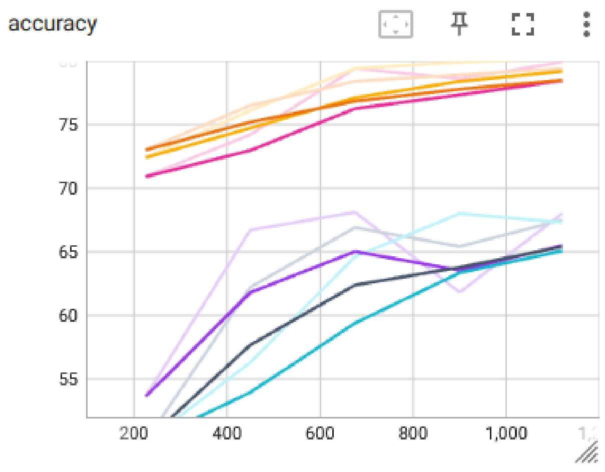

Figure 1: Evolution of the best accuracy score on the test and valid set with respect to the epochs. The blue, black and purple curves correspond to the t

3 Task 5

Here are the result obtained on tensorboard for the task 5 2.

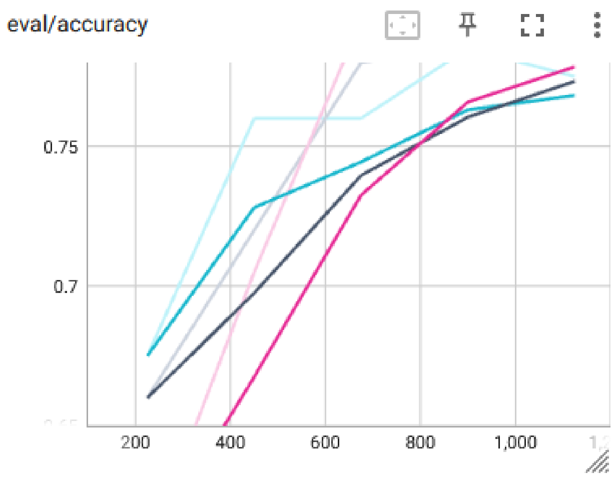

Figure 2: Evolution of the best accuracy score on the test and valid set with respect to the epochs

4 Task 7

Here are the result obtained on tensorboard for the task 5 2. We see that our model successfully train on the training set3.

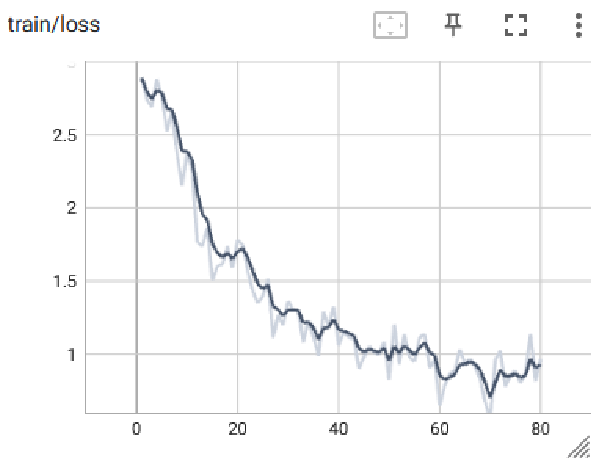

Figure 3: Evolution of the training loss

5 Question 2

Let's explain the different parameters we set in our *LoraConfig*.

Rank of the matrix

We set the parameter **r** equal to 16. This parameters correspond to the rank of the matrix used to finetune the weights of the model. In fact, when fine-tuning large models, we can assume that we do not have to update the complete matrix containing all the trainable weights, because the matrix we will have to the pretrained matrix when tuning will have a low rank. Hence, it can be approximate by a dot product between matrices of low dimension. Fine-tuning the model boils down to computing this product of matrices, hence by setting r=16 we assume that this rank is equal to 16. Increasing this parameter will result in a fine-tuning phase more consuming in computational resources. However, if we reduce too much this parameter we can have trouble fine-tuning our large model.

LoRA alpha

This parameter is used to scale the weight of the matrices. A higher value mean we assign more weight to the LoRA activatons[1].

target_modules parameter

This parameter allows the user to specify which layer does he want to fine-tune using the LoRA trick. One can fine-tune all the layers to get the closest solution from a complete fine-tuning, or just a few layers. Here, by setting target_modules=["query_key_value"], we only fine-tune the Q, K, V layers in our architecture.

dropout, bias

Usual dropout parameter used in deep architectures. It's a regularization technique to prevent overfitting by randomly setting a fraction of input units to 0 at each update during training. Bias = None means we do not use bias in our fine-tuning.

task_type

Indicates the intended task for the LoRA-equipped model, such as "CAUSAL_LM" (representing causal language modeling). This specification helps in tailoring LoRA adjustments for optimal performance on particular tasks.

References

[1] Ariel N. Lee, Cole J. Hunter, and Nataniel Ruiz. Platypus: Quick, cheap, and powerful refinement of llms. *Journal of Advanced Linguistic Models*, 2023. * Equal contribution.