Kyllian Asselin de Beauville
kyllian.asselin-de-beauville@polytechnique.edu

Lab session # 4
ALTEGRAD 2023

14/11/23

1 Question 1

Given the RoBERTa [3] small architecture [1] defined as follows:

```
encoder_layers = 4
encoder_embed_dim = 512
encoder_ffn_embed_dim = 512
encoder_attention_heads = 8
max_positions = 256
```

We compute the number of parameters as follows:

1. **Embedding Layer Parameters**:

$$\begin{aligned} \#\text{ Embedding Parameters} &= \text{Vocabulary Size} \times \text{Embed Dimension} && (1)\\ &= 32000 \times 512 && (2)\\ &= 16384000 && (3)\end{aligned}$$

2. **Positional Encoding Parameters**: In the case of BERT [1], the positional encoding is learnt, not fixed. We also need to add 2 to the maximum positions due to the start and end of sentence tokens.

$$\begin{aligned} \#\text{ Positional Encoding Parameters} &= (\text{Max Positions} + 2) \times \text{Embed Dimension} && (4)\\ &= 258 \times 512 && (5)\\ &= 132096 && (6)\end{aligned}$$

3. **Multi-Head Attention Parameters per Layer**: Based on the Transformer architecture defined in *Attention Is All You Need* [4], we have $D = d_{\text{embed}}/h = 512/8 = 64$. For Q, K, V, and output projections, the parameters for each head are:

$$4 \times (\text{Embed Dimension} \times D) \quad (7)$$

With 8 attention heads, this becomes:

$$\begin{aligned} \#\text{ Attention Parameters per Layer} &= 8 \times 4 \times (512 \times 64) && (8)\\ &= 8 \times 131072 && (9)\\ &= 1048576 && (10)\end{aligned}$$

4. **Feed-Forward Network (FFN) Parameters per (Encoder) Layer**: The FFN consists of two linear layers and since Embed Dim = FFN Embed Dim, we can write:

$$\begin{aligned} \#\text{ FFN Parameters per Layer} &= 2 \times \text{FFN Embed Dimension}^2 && (11)\\ &= 2 \times 512^2 && (12)\\ &= 524288 && (13)\end{aligned}$$

5. **Total Parameters**: Considering the 4 encoder layers, the total number of parameters is:

$$\begin{aligned} \#\text{ Total Parameters} &= \#\text{ Embedding Parameters} + \#\text{ Positional Encoding Parameters}\\ &\quad + 4 \times (\#\text{ Attention Parameters per Layer} + \#\text{ FFN Parameters per Layer}) && (14)\\ &= 16384000 + 132096 + 4 \times (1048576 + 524288) && (15)\\ &= 22807552 && (16)\end{aligned}$$

Note: This calculation omits the biases and the parameters of normalization layer, as per the question's instruction.

[1] https://github.com/hadi-abdine/fairseq/blob/main/fairseq/models/roberta/model.py

2 Task 3 & 4

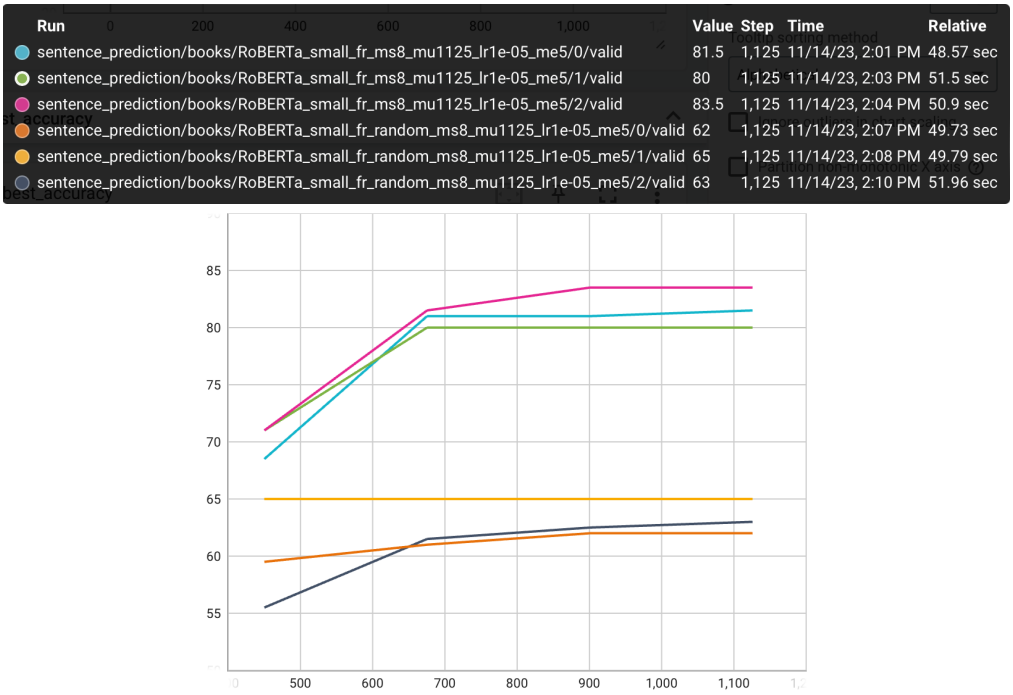

Figure 1: Accuracy of RoBERTa fine-tuning using Fairseq

Here are the results of fine-tuning using Fairseq:

$$\text{From a pre-trained model (task 3):} \quad \text{Average Accuracy} = 81.67\% \pm 1.43\% \tag{17}$$

$$\text{From scratch (task 4):} \quad \text{Average Accuracy} = 63.33\% \pm 1.25\% \tag{18}$$

3 Task 5

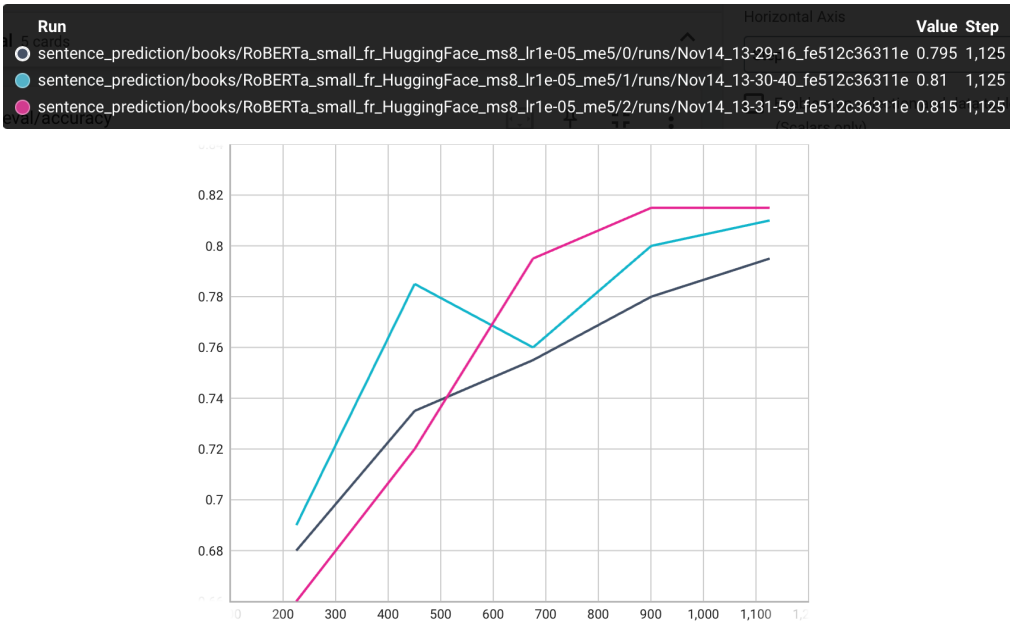

Figure 2: Accuracy of RoBERTa fine-tuning using HuggingFace

Here are the results of fine-tuning using HuggingFace:

$$\text{From a pre-trained model (task 5):} \quad \text{Average Accuracy} = 80.67\% \pm 0.85\% \tag{19}$$

4 Question 2

LoRA (Low-Rank Adaptation) [2] is a technique for fine-tuning large language models (LLMs) efficiently by keeping the original weight matrix frozen and adding additional smaller matrices, A and B, representing a low-rank approximation. This method helps maintain the generalization ability of pre-trained models while reducing the computational cost associated with full-model fine-tuning. We define the parameters used in our `LoraConfig` [2]:

- `r`: This represents the rank of decomposition matrices in LoRA. We are looking for a balance between efficiency and performance, where a higher rank can offer minimal performance boosts but at the cost of efficiency. In our configuration, we use a rank of 16.
- `lora_alpha`: Alpha scales the learned weights in LoRA. It is typically fixed to optimize the fine-tuning process. In our case, we set it to 32.
- `target_modules`: These are the model layers targeted for adaptation using LoRA. Our configuration specifically fine-tunes the "query_key_value" modules, focusing on this part of the self-attention mechanism.
- `lora_dropout`: This parameter controls the dropout probability during LoRA fine-tuning to prevent overfitting. We set this to 0.05, introducing a 5% dropout in the adapted layers.
- `bias`: This parameter specifies if the bias parameters should be trained during LoRA. Options include "none", "all", or "lora_only". We have chosen "none", indicating that no bias parameters are trained in our configuration.
- `task_type`: This specifies the type of task for the fine-tuning process. We use "CAUSAL_LM" for causal language modeling, aligning with our approach of prompt tuning.

The combination of these parameters is aimed at achieving efficient and effective fine-tuning of our LLM, balancing between model adaptability and computational constraints.

References

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

[2] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models, 2021.

[3] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach, 2019.

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

[2] https://www.anyscale.com/blog/fine-tuning-llms-lora-or-full-parameter-an-in-depth-analysis-with-llama-2