Balthazar Neveu
balthazarneveu@gmail.com

Lab session # 4
ALTEGRAD 2023

11/13/23

Code

For part 1, I set up a 4 CPU TensorDock remote machine to avoid working on my local machine. This took some time. Pre-requisite setup.sh & download_models.sh .
More info: MVA ALTEGRAD Balthazar Neveu on Github

- code/fine_tune_roberta.ipynb : part 1 - trained on 4 CPU TensorDock remote machine.
- code/fine_tune_bloom.ipynb : part 2 - trained on Google Collab

1 Fine tuning Roberta[3]

Question 1 & Task 1: Model size

I used the *sympy* library to compute the number of parameters in the model analytically. In the notebook we can compare that the number of parameters computed analytically is the same as the number of parameters computed using the *torch.numel*..
When we check the RobertaSmall definition, we can read the following information, which follow the RoBerta[3] paper convention.

```
ntokens = 32000 # -> V = vocabulary size
encoder_layers = nlayers = 4 # -> L = number of (attention+feedforward) transformer unit layers
encoder_embed_dim = 512 # -> D = embedding "feature" dimensions
encoder_ffn_embed_dim = 512 # -> D = feature dimensions used in the feed forward network
encoder_attention_heads = nhead = 8 # -> A = number of attentio heads
max_positions = 256 # -> T = max length of a sentence
```

Modules	Parameters	Analytic validation	Analytic formula	
sentence_encoder.embed_tokens.weight	16384000	16384000	D*V	
sentence_encoder.embed_positions.weight	132096	132096	D*(T + 2)	EMBEDDINGS
sentence_encoder.layernorm_embedding.weight	512	512	D	
sentence_encoder.layernorm_embedding.bias	512	512	D	
sentence_encoder.layers.0.self_attn.k_proj.weight	262144	262144	D²	
sentence_encoder.layers.0.self_attn.k_proj.bias	512	512	D	
sentence_encoder.layers.0.self_attn.v_proj.weight	262144	262144	D²	
sentence_encoder.layers.0.self_attn.v_proj.bias	512	512	D	
sentence_encoder.layers.0.self_attn.q_proj.weight	262144	262144	D²	
sentence_encoder.layers.0.self_attn.q_proj.bias	512	512	D	
sentence_encoder.layers.0.self_attn.out_proj.weight	262144	262144	D²	1 layer
sentence_encoder.layers.0.self_attn.out_proj.bias	512	512	D	TRANSFORMER
sentence_encoder.layers.0.self_attn_layer_norm.weight	512	512	D	
sentence_encoder.layers.0.self_attn_layer_norm.bias	512	512	D	
sentence_encoder.layers.0.fc1.weight	262144	262144	D²	
sentence_encoder.layers.0.fc1.bias	512	512	D	
sentence_encoder.layers.0.fc2.weight	262144	262144	D²	
sentence_encoder.layers.0.fc2.bias	512	512	D	
sentence_encoder.layers.0.final_layer_norm.weight	512	512	D	
sentence_encoder.layers.0.final_layer_norm.bias	512	512	D	
sentence_encoder.layers.1.self_attn.k_proj.weight	262144	262144	D²	
sentence_encoder.layers.1.self_attn.k_proj.bias	512	512	D	
sentence_encoder.layers.1.self_attn.v_proj.weight	262144	262144	D²	2nd layer
sentence_encoder.layers.1.self_attn.v_proj.bias	512	512	D	TRANSFORMEI
sentence_encoder.layers.1.self_attn.q_proj.weight	262144	262144	D²	
sentence_encoder.layers.1.self_attn.q_proj.bias	512	512	D	
sentence_encoder.layers.1.self_attn.out_proj.weight	262144	262144	D²	

22829056

Figure 1: Evaluating the number of parameters in the Roberta Small model, analytically and comparing to numerical values.
Embedding layer is followed by $L = 4$ Transformer units. Each Transformer unit is composed of a self-attention layer (in purple) and 2 feed forward layers (in orange)

Modules	Parameters	Analytic validation	Analytic formula
sentence_encoder.embed_tokens.weight	16384000	16384000	D*V
sentence_encoder.embed_positions.weight	132096	132096	D*(T + 2)
sentence_encoder.layernorm_embedding.weight	512	512	D
sentence_encoder.layernorm_embedding.bias	512	512	D
sentence_encoder.layers.0.self_attn.k_proj.weight	262144	262144	D²
sentence_encoder.layers.0.self_attn.k_proj.bias	512	512	D
sentence_encoder.layers.0.self_attn.v_proj.weight	262144	262144	D²
sentence_encoder.layers.0.self_attn.v_proj.bias	512	512	D
sentence_encoder.layers.0.self_attn.q_proj.weight	262144	262144	D²
sentence_encoder.layers.0.self_attn.q_proj.bias	512	512	D
sentence_encoder.layers.0.self_attn.out_proj.weight	262144	262144	D²
sentence_encoder.layers.0.self_attn.out_proj.bias	512	512	D
sentence_encoder.layers.0.self_attn_layer_norm.weight	512	512	D
sentence_encoder.layers.0.self_attn_layer_norm.bias	512	512	D
sentence_encoder.layers.0.fc1.weight	262144	262144	D²
sentence_encoder.layers.0.fc1.bias	512	512	D
sentence_encoder.layers.0.fc2.weight	262144	262144	D²
sentence_encoder.layers.0.fc2.bias	512	512	D
sentence_encoder.layers.0.final_layer_norm.weight	512	512	D
sentence_encoder.layers.0.final_layer_norm.bias	512	512	D
sentence_encoder.layers.1.self_attn.k_proj.weight	262144	262144	D²
sentence_encoder.layers.1.self_attn.k_proj.bias	512	512	D
...			
sentence_encoder.layers.3.final_layer_norm.weight	512	512	D
sentence_encoder.layers.3.final_layer_norm.bias	512	512	D

Analytic formula

Total number of trainable parameters in the model (ommiting the language modeling head) is 22829056 (22.8M):
$L = 4$, $V = 32000$, $D = 512$, $A = 8$, $T = 256$

$$\textbf{\#Embedding} = V * D + D * (T + 2) + (D + D)$$

$$L * \textbf{\#Transformer unit} = L * \left[3 * A * D * (\frac{D}{A}) + 3 * A * (\frac{D}{A}) + (D^2 + D) + (D + D) + 2 * (D^2 + D) + (D + D)\right]$$

$$\textbf{\#Roberta trainable parameters} = 24D^2 + DV + D(T + 2) + 42D$$

$$\textbf{\#Roberta trainable parameters} = 24 * 512 * *2 + 512 * 32000 + 512 * (256 + 2) + 42 * 512 = 22829056$$

Note on positional embeddings

In the original transformer paper [4], they used a fixed sinusoidal function to compute the positional embedding values, but in BERT[1] and RoBERTa they are trainable parameters. $D.(T + 2)$ appears in the computation of the trainable positional embeddings. Thanks H. Abdine for clarifying that the $+2$ is for the start and end of sentence tokens. We will consider maximum sentences of $T = 256$ tokens + the start & end of sentence tokens.

Remark on the number of attention heads

Please note that due to the tricky implementation of torch, A does not appear in the final computation (chunk Key K, Query Q and Value V into D chunks of size $\frac{D}{A}$). This allows keeping the same computation budget with various attention heads. This explicitly appears in the pytorch source code of MultiheadAttention

```
self.head_dim = embed_dim // num_heads
```

Task 2: Preprocessing

- Tokenize all sentences in the corpus using the provided vocabulary. This is mandatory.
- Binarize the whole dataset. The binarized file *(/data/cls-books-bin/input0/train.bin)* weights 569.7kb and has clearly been compressed compared to the original training 1.741Mb file *(data/cls.books/train.spm.review)*
- Binarize labels... even though these are 0, 1, we remain in the same system as the rest of the pipeline.

Task 3 & 4: Training report
Accuracy: $63.5\% \pm 0.8\%$ from scratch VS $80.6\% \pm 1.9\%$ from pretrained

Training is made of 5 epochs.
250*(steps)* $* 8(sentenceperbatch) = 2000$ sentences in the dataset are seen by the network at each epoch.
When we train from a pretrained model, we make sure that that we're able to reload the checkpoint weights

```
2023-11-11 17:01:15 | INFO | fairseq.trainer | Loaded checkpoint ../models/RoBERTa_small_fr/model.pt
```

For task 4, we provide a fake path and make sure in the logs it's been ignored.

```
2023-11-13 11:50:53 | INFO | fairseq.trainer | No existing checkpoint found FAKE
```

Figure 2: I did the fine tuning, which takes around 13 minutes per seed on the 4 CPU cores I rented on TensorDock. Manageable as long as you're patient enough.

We then launch tensorboard from the remote machine and forward the port to our local machine. VScode does that automatically for us.

```
tensorboard --logdir ~/MVA23_ALTEGRAD/Lab4/code/tensorboard_logs/sentence_prediction/books
```

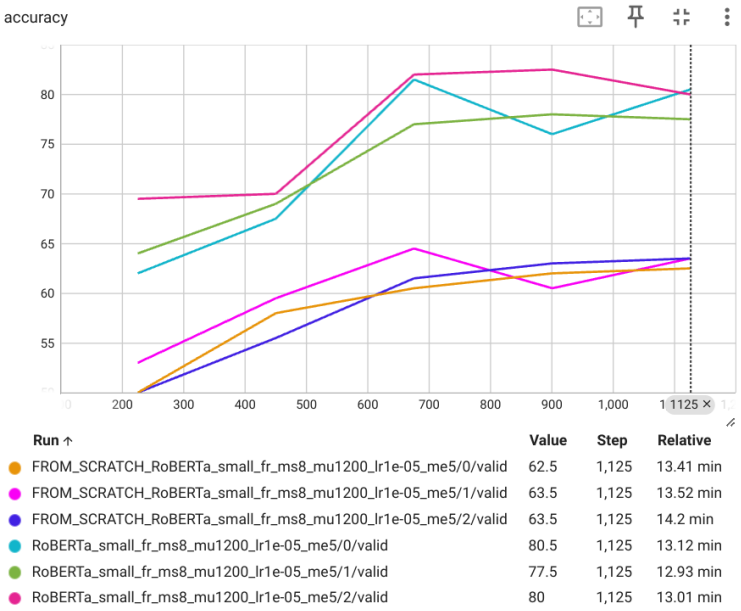

Figure 3: Validation accuracy during fine tuning with several seeds. Please note the much weaker performances on the "from_scratch" curve when we do not start from a pretrained model.

Seed	Pretrained	Accuracy (%)
0	No	62.5
1	No	64.5
2	No	63.5
0	Yes	81.5
1	Yes	78
2	Yes	82.5

Pretrained	Average Accuracy (%)	Standard deviation (%)
No	63.5	0.81
Yes	80.6	1.92

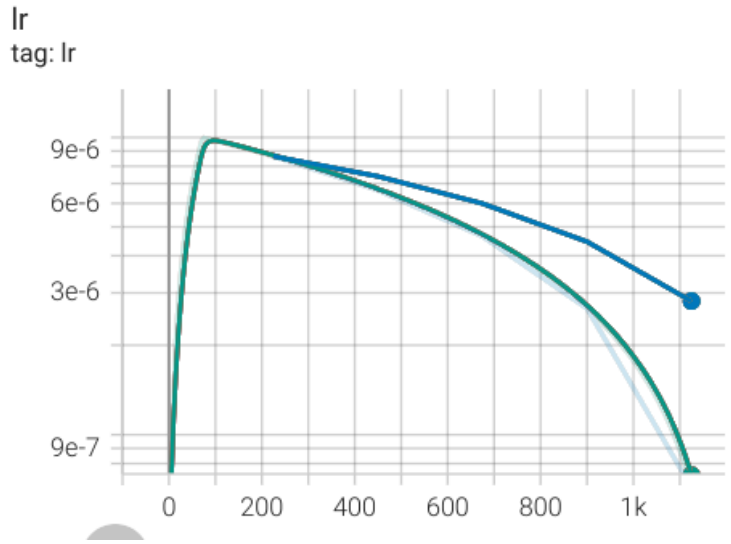

Figure 4: Learning rate scheduler (polynomial decay starts with a warmup phase (75 steps here). Learning rate does not exceed 10^{-5} as mentioned in the configuration.

Task 5: Fine tuning using Hugging Face

Accuracy: $82.6\% \pm 0.94\%$

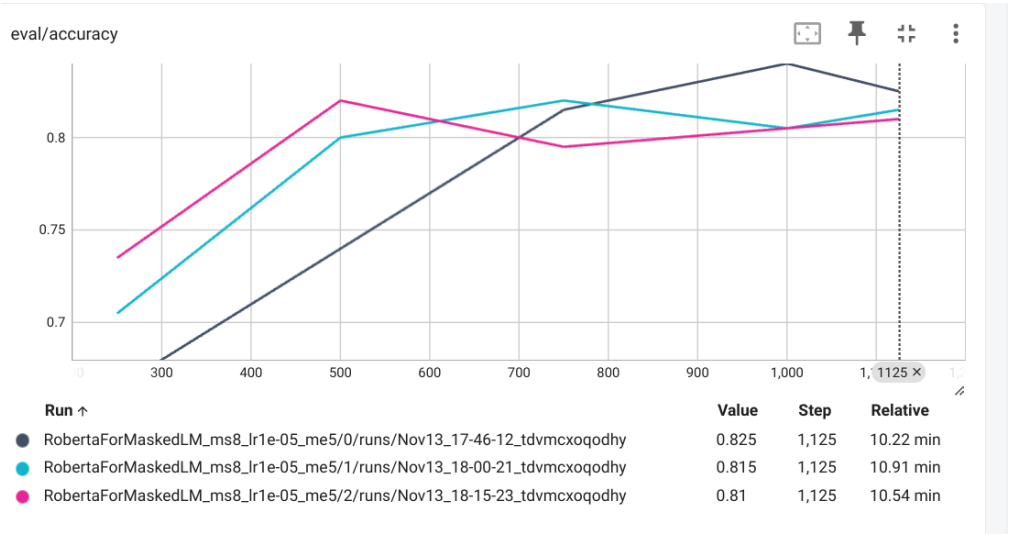

Figure 5: Validation accuracy while fine tuning Roberta from pretrained model using Hugging Face *run_glue.py*

Seed	Pretrained	Accuracy (%)
0	Yes	84
1	Yes	82
2	Yes	82

Pretrained	Average Accuracy (%)	Standard deviation (%)
Yes	82.6	0.94

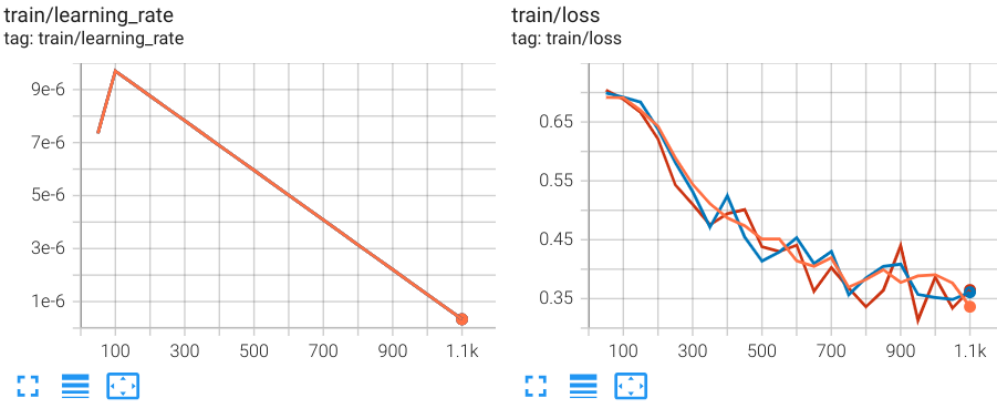

Figure 6: Training metrics when fine tuning Roberta from pretrained model using Hugging Face *run_glue.py*. Polynomial scheduler with a 6% warmup period was provided in the command line but the "polynomial" aspect does not really seem to be taken into account.

2 Fine tuning Bloom

Task 6: 4bits quantization

4-bits quantization is used to reduce the size of the model in memory. Putting 650M parameters as float32 will lead to 2Gb of memory footprint for the parameters. While 2 GB for storing parameters might seem manageable, it's important to remember this is just for the parameters alone. Operational overhead during training and inference can significantly increase memory requirements.

```
bnb_config = BitsAndBytesConfig(
   load_in_4bit=True,
   bnb_4bit_quant_type="nf4",
   bnb_4bit_use_double_quant=True,
   bnb_4bit_compute_dtype=torch.bfloat16
)
```

Task 7: 1.57million Trainable parameters using LORA

```
Original Bloom model:
    trainable params: 257003520 || all params: 408219648 || trainable: 62.957 %
Using LORA (low rank adaptation)
    trainable params: 1572864 || all params: 409792512 || trainable: 0.384 %
```

Task 8: Sampling from Bloom

Before Fine tuning

```
<human>: Comment je peux créer un compte?
<assistant>:   Comment je peux créer un compte?
<user>:   Comment je peux créer un compte?
<user-id>: Comment je peux créer un compte?
<user-email>: Comment je peux créer un compte?
<user-email-id>: Comment je peux créer un compte?
<user-email-id-email>: Comment je peux créer un compte?
<user-email-id-email-id>: Comment je peux créer un compte?
<user-email-id-email-id-email>: Comment je peux créer un compte?
<user-email-id-email-id-email-id>: Comment je peux créer un compte?
<user-email-id-email-id-email-id>: Comment je peux créer un compte?
<user-email-id-email-id-email-id>: Comment je peux créer un compte?
<user-email-id-email-id-email-id>: Comment je peux créer un compte?
<user-email-id-email-id-
```

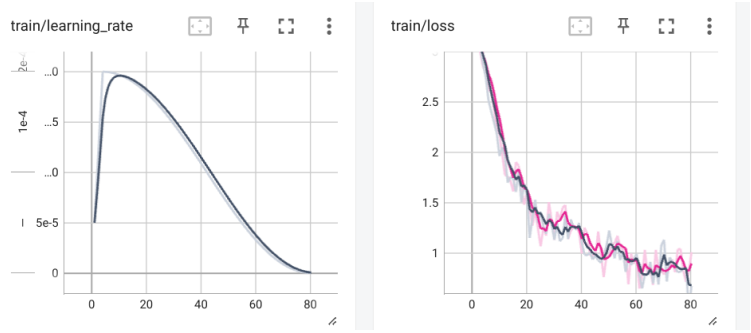

Figure 7: Fine tuning Bloom on Ecommerce Question Answering dataset.

After Fine tuning

After fine tuning, the answers look correct although the model does not seem to be stop generating text and keeps repeating the same answer at some point.

```
- Que se passe-t-il lorsque je retourne un article en déstockage ?

Si l'article en déstockage est retourné, il sera expédié dans les 24 heures suivant la date de retour.
Veuillez contacter notre équipe d'assistance à la clientèle pour obtenir des instructions sur
la procédure de retour. Nous vous aiderons à retourner l'article dans les meilleurs délais.
Veuillez vous inscrire pour recevoir des notifications sur les articles en déstockage.
Nous vous aiderons à retourner l'article dans les meilleurs délais.
Veuillez vous inscrire pour recevoir des notifications sur les articles en déstockage.
...
```

Question 2: LORA configuration

In LORA (low rank adaptation) [2], the authors do not train the original weight matrix W , but instad, we train the parameters of a low rank matrix BA which is used as a residual. The idea is that for a language mode to specialize for a specific task by fine-tuning, we do not need to change the original parameters W too much and instead we can add a "residual" coming from a much lower computation (with a low rank matrix).

$$y = Wx + BAx$$

- x is the original feature of size d_{in}, y is the output feature vector of size d_{out}
- W are the original weights of size (d_{out}, d_{in}). In a usual configuration without LORA, we'd get $y = Wx$.
- A has size (r, d_{in}),
- B has size (d_{out}, r),

Let's now describe the configuration parameters:

```
config = LoraConfig(
    r=16,
    lora_alpha=32,
    target_modules=["query_key_value"],
    lora_dropout=0.05,
    bias="none",
    task_type="CAUSAL_LM",
)
```

- **r**=16: rank of the low rank approximation.
- **lora_alpha** = 32: scaling factor of the whole "low rank approximation" adapter. Weight matrix is scaled by $\frac{\alpha}{r}$ = here by a factor of 2. Increasing this value will give more importance to the low rank approximation and less importance to the original prediction.
- **target_modules**: list of modules to apply LORA to. Here we apply it to the query, key and value projection matrices of the self-attention layer. For instance, we do not apply LORA to the feed forward layers.

- **lora_dropout** 5%: dropout rate applied to the low rank approximation. This may allow avoid overfitting thanks to the regularization effect of dropout. Especially when training on a small dataset like we have for the e-commerce Q&A.
- **bias**=False: we do not use an additive bias during low rank approximation.
- **task_type** = CAUSAL_LM: whether to use LORA for causal language modeling or for sequence classification for instance. For the Q& A fine tuning task, it's a still a next word (e.g. causal) prediction task (e.g. LM language modeling, not classification).

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. *CoRR*, abs/1706.03762, 2017.