Assignment 3

Cecilia Nyberg

December 3, 2024

1 Comparison of fine-tuning methods

Feature Extraction freezes the parameters for all layers except for the last layer, the classification layer. The parameters for the last layer are retrained for the specific task and data. LoRA on the other hand updates targeted weights in the model that ideally have the most impact on the output for the specific task. Often the type of layers that are targeted are attention layers [1]. The way LoRA works is by storing large matrices of weight updates in decomposed smaller matrices (with rank r) and then merging these with the existing parameter weights. This drastically reduces the number of trainable parameters compared to full re-training, but it still updates more weights compared to Feature Extraction. This means feature extraction is more resource efficient and have lower complexity than LoRA while LoRA is more flexible and can result in a more specialized model for the task.

1.1 Comparsion of results

The loss per 100 steps during training for the model using feature extraction can be seen in figure 1. As can be seen, the loss starts att 1.09 and steadily goes down to 0.459. The reduction is larger for the first 1000 steps, indicating that it might have been enough to train the model for three epochs.

Step	Training Loss
100	1.087400
200	1.037200
300	0.922800
400	0.814300
500	0.677100
600	0.589700
700	0.525600
800	0.507700
900	0.489600
1000	0.473200
1100	0.474500
1200	0.468800
1300	0.461000
1400	0.451500
1500	0.459000

Figure 1: Loss for Feature Extraction training

The loss per 300 steps for training the model using LoRA can be seen in figure 2. This was because the model took so long to train so the logging was done less often in order to save time. Since the loss was only logged per 300 steps, and the model ran for a total of 927 steps, the loss was only updated three times. This can be seen since the loss is the same for epochs 1 and 2, and not included for epoch 5. As can be seen, the loss starts out much lower than for the feature extraction. Since the first calculated loss for the model using LoRA was calculated after 300 steps, this has to be compared with the loss

for step 300 of the feature extraction model. For the feature extraction model, the loss was 0.92 at 300 steps, and for LoRA it was 0.69. Then the loss drops drastically to 0.26 for the LoRA model, and eventually ends up at 0.21 at step 900, much lower than 0.46 which was the final value for the Feature extraction model.

Epoch	Training Loss	Validation Loss
0	No log	No log
1	0.694900	No log
2	0.694900	No log
3	0.264200	No log
4	0.213500	No log

Figure 2: Loss for LoRA training

As can be seen in table 1.1, the evaluation accuracy for the Feature Extraction is 0.81 and the F1 score is also 0.81, while for LoRA the results are higher, with an accuracy and F1 score of about 0.93 for both.

Method	**Accuracy**	**F1-Score**
Feature Extraction	0.81432	0.81415
LoRA	0.92852	0.92852

Table 1: Evaluation Metrics for Feature Extraction and LoRA

LoRA shows better results for all metrics (loss, accuracy, F1). This is expected since LoRA updates a larger number of parameters and is a more flexible method. By updating more weights, the model becomes more specialized on the specific task, which usually leads to better performance. This proved to be true in this case too. However, it comes with the downside of higher complexity and requiring more resources. The batch size had to be drastically reduced to save memory usage. Training the model took almost twice the time when using LoRA compared to feature extraction.

2 Model design

For both methods, the following training arguments were used:

$$\begin{aligned} \text{Learning Rate} &= 1 \times 10^{-4} \\ \text{Number of Epochs} &= 5 \\ \text{Warmup Steps} &= 500 \\ \text{Weight Decay} &= 0.01 \end{aligned}$$

Then for only the Feature Extraction, the following parameters were also used:

$$\begin{aligned} \text{Batch size} &= 64 \\ \text{Logging steps} &= 100 \end{aligned}$$

For LoRA, some changes had to be made. So for LoRA the following parameters were used:

$$\begin{aligned}
\text{Batch size} &= 16 \\
\text{Logging steps} &= 300 \\
\text{Gradient accumulation steps} &= 8 \\
\text{fp16} &= \text{True} \\
\text{Target modules} &= [\text{"}c_fc\text{"}, \text{"}c_proj\text{"}] \\
\text{Task type} &= \text{"SEQ_CLASSIFICATION"}
\end{aligned}$$

2.1 Discussion of parameter settings

When training the LoRA model, the memory usage was a very limiting factor. This influenced the possible parameter settings greatly. The batch size had to be kept very low, fp16 was set to true and only a few target modules could be chosen.

The chosen target modules were fully connected layers and projection layers, since these are basic layers to update for binary tasks and can be beneficial to target for classification tasks. It could have been beneficial to target more types of layers, such as attention layers, but because of computational complexity this was not possible.

When training the LoRA model, the memory kept getting full so the batch size had to be reduced to 16. A small batch size, like batch size 16, can have some effects on the models performance. Using a small batch size can introduce noise and make it harder for the model to converge since the model has fewer data points to calculate the gradient on. To reduce the impact of the small batch size, gradient accumulation was used. Gradient accumulation makes the model average the gradient over several batches before updating the model. Therefore, the parameter updates become more stable. The gradient accumulation was set to 8, resulting in a simulated batch size of 16*8=128.

The warmup steps of 500 means that the model starts with a very low learning rate and then increases it over the first 500 steps until it reaches the constant value (in this case set to 10^{-4}). This can improve the training since the parameters can be very incorrect from the start, and large gradients from these poorly initialized parameters can have a negative impact on convergence. Therefore, the gradients are multiplied with a very low learning rate at first to reduce the impact of noisy gradients at early stages. For this specific case, however, a warmup step of 500 might be too large. For the LoRA model, the total steps were only 927. This means that for more than half the training steps a very low learning rate was used. It could have been more beneficial to use a lower warmup step for the model to learn faster.

The task size was set to sequence classification task because this was the task for the model.

The rank parameter had to be kept low to reduce complexity. It would have been interesting to see what effect a higher value would have. A higher value would probably result in a better performance since this results in a larger number of weights being updated.

The learning rate could have been set to a higher value in order for the model to converge faster. However, it can also lead to exploding gradients. The best way to know is to try out different settings. This was not done because of the computational complexity and the time it took to train the models. It could have been beneficial to use hyperparameter optimization, for example using random search.

Unfortunately, the validation loss and accuracy could not be printed when training the models. This is very important in order to determine how many epochs to run and to see if the model starts to overfit. However, the high test accuracy after five epochs shows that the model has learned efficiently and correctly. LoRA proved to be much more beneficial than feature extraction by reaching an accuracy and F1 score of 93%. However, there are still many possible improvements that could yield an even higher accuracy and F1 score.

In conclusion, my ideal configuration (assuming I have more resources like memory and GPUs) would use LoRA rather than feature extraction. The training parameter settings would be similar to the ones used but the target modules would also include attention layers and a higher rank, for example r=16, would be used to get an even more flexible and specialized model. However, if the model was to be used for other tasks in the future, updating more layers and weights could reduce generalizability. Additionally, the training parameters would be fine-tuned by analyzing the model during training. A separate validation set would have been used to track overfitting in order to determine the number of epochs. Alternatively, early stopping could have been used to stop the training if the validation accuracy starts going down.

References

[1] Kailash Thiyagarajan. Fine-tuning large language models with lora: Demystifying efficient adaptation, Jan 2024. Medium article, accessed Dec 3, 2024.