T.Z.Zhang
YuanPei College
Peking University

November 13, 2024

NLP-DL — Assignment2 Report

1 Task2: Training Script

Following the instructions in the notes, we conducted $3 \times 3 \times 5 = 45$ experiments across various datasets, including `restaurant_sup`, `acl_sup`, and `agnews_sup`, and fine-tuned several pre-trained models: `roberta-base`, `bert-base-uncased`, and `allenai/scibert_scivocab_uncased`.

We first present the learning curves for one experiment (averaged over five different random seeds). The mean performance across runs is shown as a thick line, with shaded regions representing the maximum and minimum values across these runs (see Figure 1), and see Appendix 3.1 for training hyper-parameters and details.

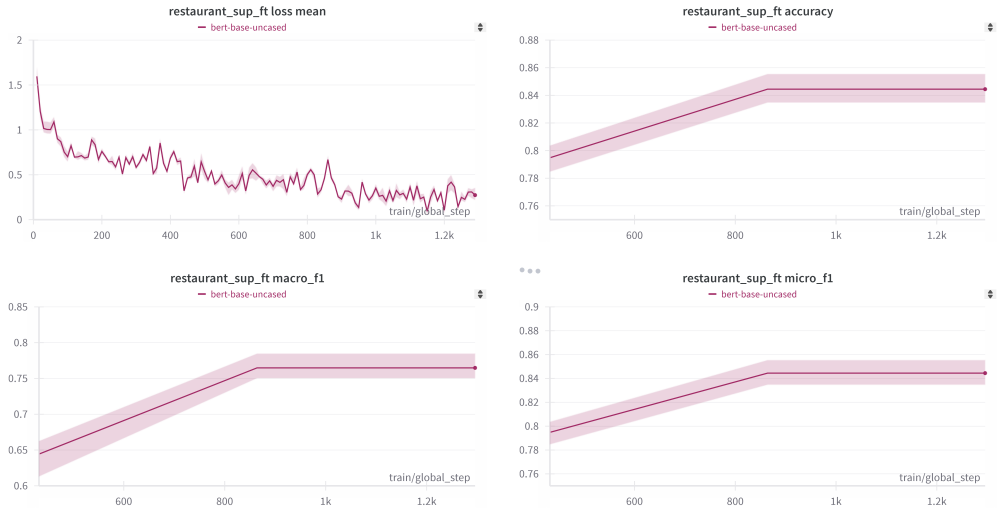

Figure 1: The loss curve, accuracy curve, macro_f1 curve and micro_f1 curve of a bert-base-uncased model fine-tuned on the restaurant_sup dataset. (the X axis of the loss curve is step, and we trained each setting for three epochs)

After three epochs, we observe a performance decline on the validation set, indicating that setting `epoch_num` to 3 is appropriate. The training loss curve supports this choice, as it converges well by the end of the third epoch.

We present our fine-tuning results in Table 1 and Table 2. As shown, the `allenai` model achieves the best performance on both the `restaurant_sup` and `acl_sup` datasets, while the `roberta` model performs best on the `agnews_sup` dataset. Additionally, we observe that the training loss does not fully reflect model performance on the validation and test sets. For example, although the `bert` model has the lowest training loss on the `agnews_sup` dataset, it underperforms in terms of accuracy and other metrics.

	restaurant_sup				acl_sup			
models	train_loss	accuracy	macro_f1	micro_f1	train_loss	accuracy	macro_f1	micro_f1
allenai	0.2809	0.8076	0.6972	0.8076	**0.5151**	**0.7884**	**0.6986**	**0.7884**
bert	0.2732	0.8444	0.7648	0.8444	0.7372	0.7136	0.4019	0.7136
roberta	**0.2568**	**0.8560**	**0.7904**	**0.8560**	0.7499	0.7338	0.5187	0.7338

Table 1: The fine-tune result of the three pre-train models on restaurant_sup dataset and acl_sup dataset.

	agnews_sup			
models	train_loss	accuracy	macro_f1	micro_f1
allenai	0.1756	0.9210	0.9195	0.9210
bert	**0.1632**	0.9165	0.9148	0.9165
roberta	0.2606	**0.9286**	**0.9270**	**0.9286**

Table 2: The fine-tune result of the three pre-train models on agnews_sup dataset.

Upon further analysis of the three pre-trained models, we observe that the `allenai` model is expected to perform better on scientific topics, as it was pre-trained on a specialized scientific corpus. Additionally, since `roberta` was trained on a larger dataset than `bert`, it is anticipated to perform better than the original `bert` model. Our test results confirm that `roberta` generally outperforms `bert`, and the `allenai` model achieves remarkable results on the `acl_sup` dataset. A closer inspection of `acl_sup` reveals that it contains a substantial amount of academic data, aligning with our expectations for the `allenai` model's strengths.

2 Task3: Parameter-efficient Fine-tuning (PEFT)

During our implementation of adapters, we observed that the performance of RoBERTa with adapters was notably poor on the acl_sup dataset. The training curves across different datasets are shown in Figure 2.

From the results, we observe that the RoBERTa model combined with adapters does not fully converge when fine-tuned on the acl_sup dataset. A closer analysis across the three datasets reveals that acl_sup has the largest number of classes and the most asymmetric distribution of corpus themes. In the standard fine-tuning setting, we also observed that RoBERTa performs poorly on the acl_sup dataset compared to other pre-trained models. We therefore attribute the poor performance mainly to the inherent incompatibility of the RoBERTa model with this downstream task on acl_sup, making it challenging for the adapter technique to sufficiently align the pre-trained model with the task. Table 3 provides a performance comparison between standard fine-tuning and PEFT with adapters.

	datasets	**restaurant_sup**	**acl_sup**	**agnews_sup**
training loss	*fine-tune*	**0.2568**	**0.7499**	**0.2606**
	PEFT with adapter	0.6944	1.4091	0.3581
accuracy	*fine-tune*	**0.8560**	**0.7884**	**0.9286**
	PEFT with adapter	0.7542	0.5107	0.9226
macro_f1	*fine-tune*	**0.7904**	**0.6986**	**0.9270**
	PEFT with adapter	0.5057	0.1126	0.9207
micro_f1	fine-tune	**0.8560**	**0.7884**	**0.9286**
	PEFT with adapter	0.7542	0.5107	0.9226

Table 3: The performance comparison between the fine-tune setting and the PEFT with adapter setting.

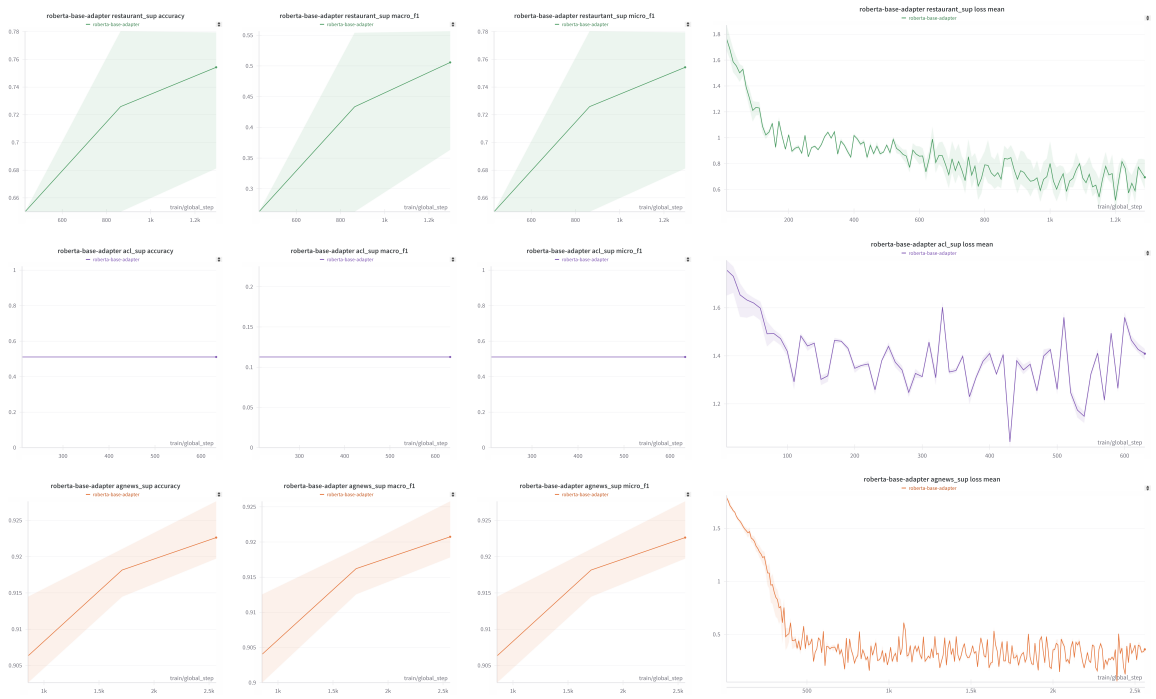

Figure 2: The accuracy curve, macro_f1 curve, micro_f1 curve and loss curve of a roberta model using adapter fine-tuned on different datasets (restaurant_sup, acl_sup and agnews_sup) for 3 epochs.

Further analysis involves calculating the GPU memory used in both the standard fine-tuning setting and the PEFT with adapters setting. Given that the `roberta-base` model has approximately 3 billion parameters and we are using FP32 by default, the memory required for the model itself is:

$$3 \times 10^9 \times 4 \text{ bytes} = 12 \text{ GB}.$$

When using the Adam optimizer, memory usage triples due to the additional storage for momentum and variance estimates, requiring:

$$12 \text{ GB} \times 3 = 36 \text{ GB}.$$

Additionally, the gradients require as much memory as the model parameters, adding another 12 GB. Activation memory, which is also substantial for large models, accounts for approximately 10 GB for a sequence length of 512. This brings the total GPU memory requirement to:

$$12 \text{ GB} + 36 \text{ GB} + 12 \text{ GB} + 10 \text{ GB} = 70 \text{ GB}.$$

In contrast, with PEFT using adapters, we observed through monitoring that the GPU memory required by this method is only about 13.17 GB, representing a significant reduction compared to the 70 GB estimated for standard fine-tuning.

Code of this project available at: https://github.com/nameistzzhang/2024Autumn-NLP-Assignment2.git

3 Appendix

3.1 Training Details

Here we paste our default hyper-parameters, which is used in all the experiments above:

```
@dataclass
class TrainArguments(TrainingArguments):
    learning_rate: float = field(default=2e-5,
                                 metadata={"help": "learning rate"})
    num_train_epochs: int = field(default=3,
                                  metadata={"help": "training epoch"})
    weight_decay: float = field(default=0.01,
                                metadata={"help": "weight decay"})

    output_dir: str = field(default="./results",
                            metadata={"help": "path to the output"})
    eval_strategy: str = field(default="epoch",
                               metadata={"help": "evaluation strategy, can be steps or epoch"})
    save_strategy: str = field(default="epoch",
                               metadata={"help": "save strategy, can be steps or epoch"})
    per_device_train_batch_size: int = field(default=8,
                                             metadata={"help": "training batch size"})
    per_device_eval_batch_size: int = field(default=8,
                                            metadata={"help": "evaluation batch size"})
    logging_dir: str = field(default="./logs",
                             metadata={"help": "logginh directory"})
    load_best_model_at_end: bool = field(default=True,
                                         metadata={"help": "load the best model"})

    logging_steps: int = field(default=10,
                               metadata={"help": "wandb hyperparameter: logging steps"})
    save_steps: int = field(default=0,
                            metadata={"help": "wandb hyperparameter: save steps"})
    report_to: str = field(default="wandb",
                           metadata={"help": "report to wandb for logging"})
```

Submitted by T.Z.Zhang on November 13, 2024.