Question Answering Fine-Tuning Report

Link to github repository: https://github.com/Kaishoo-git/dsa4213_assignment3/tree/master

1. Dataset Selection

I chose the **SQuAD v1.1** dataset for this project since it is a **widely recognized benchmark** for extractive question answering.
Version 2.0 includes examples with *no possible answer*, but this aspect is not directly relevant for the current objective, which focuses on **answer extraction** rather than answerability classification.
Hence, **SQuAD v1.1** provides a cleaner and more direct benchmark for evaluating model performance on span prediction.

---

2. Model Choice

The base model used was `mrm8488/bert-tiny-finetuned-squadv2`.
This model was selected because: - It was **already fine-tuned on SQuAD v2**, which is closely related to SQuAD v1.1.
- It is **lightweight (`bert-tiny`)**, making it feasible to fine-tune with limited computational resources. - Its underlying architecture is **BERT**, which is particularly well-suited for **extractive question answering** tasks due to its bidirectional context encoding and strong performance on span-level token prediction.

In essence, the model inherits the core inductive bias of BERT for **matching questions and contexts** and identifying the most relevant answer span.

---

3. Training Configuration

Due to computational constraints, the training setup prioritized efficiency and convergence monitoring:

Parameter	Value	Notes
Dataset	SQuAD v1.1	Standard QA benchmark
Epochs	10	Upper bound, early stopping used
Early Stopping	Enabled	Monitors `eval_loss`, patience = 2
Learning Rate	3e-3	Slightly higher due to fewer epochs
Batch Size	2	With gradient accumulation = 4
Precision	FP32	FP16 disabled due to hardware limits

Training employed **early stopping based on validation loss (`eval_loss`)**, ensuring that overfitting was avoided even with a relatively high learning rate and small model.

---

4. Methods Compared

Two parameter-efficient fine-tuning methods were tested:

Method	Description
LoRA (Low-Rank Adaptation)	Inserts trainable low-rank matrices into attention layers, directly modifying the model's internal representations.
Prompt Tuning	Learns a small set of "soft prompt" embeddings prepended to the model's input, indirectly influencing predictions.

---

5. Results

Metric	Pre-trained	LoRA	Prompt Tuning
F1	0.89%	32.6%	1.91%
Exact Match	0.0%	24.0%	0.0%
BERTSCORE	10.12%	72.2%	58.6%

---

6. Key Findings

- **LoRA outperformed prompt tuning** in terms of both convergence speed and final evaluation metrics (F1 and EM).

- This is expected: LoRA **directly adapts internal weight matrices**, modifying how the model computes token-level attention and contextual relevance.

- Prompt tuning, on the other hand, **only affects input embeddings**, influencing model behavior more **implicitly** and requiring more data or training steps to reach comparable performance.

In broader terms, this aligns with the understanding that: > Full fine-tuning and LoRA both optimize the model's *log-likelihood* more directly, while prompt tuning operates via input perturbation and is thus less expressive under limited training budgets.

7. Summary

Aspect	Decision	Rationale
Dataset	SQuAD v1.1	Benchmark without "no-answer" noise
Model	`bert-tiny` (SQuAD v2 fine-tuned)	Lightweight and task-aligned
Fine-tuning	LoRA and Prompt Tuning	Compare efficiency and performance
Result	LoRA superior	Direct weight adaptation more effective