ANLP

Assignment-3

Report

Akshit Sharma
2021101029

Theory Questions

Concept of soft prompts

Q: How does the introduction of ″soft prompts″ address the limitations of discrete text prompts in large language models?

Ans: Soft prompts address the limitations of discrete text prompts by using continuous embeddings instead of fixed token sequences. This allows for more fine-grained control and optimization during training, enabling models to better capture task-specific information without being constrained by the rigid structure of natural language tokens. Soft prompts also reduce prompt length dependency and can generalize across tasks more effectively.

Q: Why might soft prompts be considered a more flexible and efficient approach for task-specific conditioning?

Ans: Soft prompts are considered more flexible and efficient for task-specific conditioning because they operate in the continuous embedding space, allowing for smoother optimization and adaptation to various tasks. Unlike discrete text prompts, which rely on pre-defined tokens, soft prompts can be fine-tuned directly, enabling efficient learning with fewer parameters. They also offer better task generalization and can be applied across different tasks without manual prompt engineering, saving computational resources and time.

Scaling and Efficiency in Prompt Tuning

Q: How does the efficiency of prompt tuning relate to the scale of the language model?

Ans: The efficiency of prompt tuning improves with the scale of the language model because larger models have richer representations and more capacity to capture task-specific nuances from small changes in the input embeddings. As the model size increases, prompt tuning allows fine-tuning only the soft prompts while keeping the rest of the model frozen, reducing the number of parameters that need updating. This approach is highly parameter-efficient, especially in large models, as it leverages the pre-trained knowledge without the computational overhead of retraining the entire model.

Q: Discuss the implications of this relationship for future developments in large-scale language models and their adaptability to specific tasks.

Ans: The relationship between prompt tuning efficiency and model scale implies that as language models grow larger, they become more adaptable and task-specific without requiring full fine-tuning. This opens up several future developments:

- **Scalability and Specialization:** Larger models can be fine-tuned for a wide range of tasks using minimal additional parameters via prompt tuning, making them highly scalable for specialized applications without needing extensive computational resources.
- **Cost-Effective Customization:** Since only the prompts are tuned, fine-tuning becomes more cost-effective and faster, allowing developers to adapt massive language models to specific tasks or domains with reduced hardware and time costs.
- **Model Reusability:** Pre-trained large models can be reused across different applications by simply adjusting the soft prompts. This enhances the modularity and reusability of models, fostering innovation and rapid deployment in diverse fields.
- **Democratizing Access:** Efficient prompt tuning lowers the barrier to accessing and utilizing large models, enabling smaller organizations or individual

researchers to fine-tune large models for their needs without massive computational infrastructure.

- **Improved Multi-task Learning:** The flexibility of soft prompts suggests that large models could handle multiple tasks concurrently by switching between soft prompts, potentially leading to more generalized systems that can perform various tasks with minimal overhead.

Understanding LoRA

Q: What are the key principles behind Low-Rank Adaptation (LoRA) in fine-tuning large language models?

Ans: Low-Rank Adaptation (LoRA) reduces the number of trainable parameters when fine-tuning large language models by factorizing weight updates into low-rank matrices. Instead of updating the full weight matrices, LoRA inserts low-rank matrices that approximate the weight updates, thus reducing memory and computational overhead while retaining model performance. The core principles are:

- **Decomposing weight updates:** Represent the weight update as a low-rank product of two smaller matrices.
- **Parameter efficiency:** Only a fraction of the parameters need to be trained.
- **No interference with original weights:** The original pre-trained weights remain frozen, and only the low-rank adapters are optimized.

Q: How does LoRA improve upon traditional fine-tuning methods regarding efficiency and performance?

Ans: LoRA improves upon traditional fine-tuning methods in the following ways:

- **Parameter Efficiency:** Traditional fine-tuning adjusts all model weights, while LoRA only trains a small set of low-rank matrices, reducing the number of trainable parameters and memory usage.
- **Reduced Computation:** By factorizing weight updates, LoRA reduces the computational cost of fine-tuning, making it faster and more efficient, especially for large language models.
- **Performance Retention:** Despite its parameter efficiency, LoRA maintains comparable performance to full fine-tuning by preserving the expressiveness of the model through low-rank approximations.
- **Modularity:** LoRA allows multiple tasks to be fine-tuned independently by adding separate low-rank adapters, which can be applied selectively without interfering with the pre-trained model weights.

Theoretical Implications of LoRA

Q: Discuss the theoretical implications of introducing low-rank adaptations to the parameter space of large language models.

Ans: Introducing low-rank adaptations (LoRA) to the parameter space of large language models (LLMs) has several theoretical implications:

- **Parameter Space Compression:** LoRA reduces the dimensionality of weight updates by approximating them with low-rank matrices. This creates a compressed subspace for optimization, effectively regularizing the model and preventing overfitting by limiting the degrees of freedom in weight adjustments.
- **Mitigating Catastrophic Forgetting**: Since LoRA freezes the original pre-trained weights and only fine-tunes a small set of low-rank matrices, the risk of catastrophic forgetting (where a model loses knowledge of previous tasks) is reduced. The original knowledge encoded in the pre-trained model is preserved, while task-specific updates are captured by the adapters.

- **Efficient Task-Specific Adaptation:** LoRA focuses on fine-tuning only task-relevant subspaces, implying that many large model weights are not critical for task-specific performance. Theoretically, this suggests that a model's parameter space can be highly over-parameterized, and fine-tuning can be effectively constrained to lower-rank transformations without losing performance.

Q: How does this affect the expressiveness and generalization capabilities of the model compared to standard fine-tuning?

Ans: LoRA's approach to fine-tuning affects the expressiveness and generalization capabilities of large language models (LLMs) in the following ways:

1. **Expressiveness**

- **Preserved Expressiveness:** LoRA preserves the full expressiveness of the model because it keeps the pre-trained weights intact and only introduces small, task-specific low-rank adaptations. This means the underlying capacity of the model is not compromised, unlike some pruning or quantization techniques that may reduce expressiveness by directly altering core model parameters.

Selective Adaptation: By focusing on low-rank matrices for updates, LoRA efficiently adjusts only the necessary components for a new task, while leveraging the rich, pre-trained knowledge in the original weights. This ensures that the model remains expressive enough to handle complex patterns in both the pre-trained and fine-tuned contexts.

2. **Generalization**

- **Implicit Regularization:** LoRA's low-rank updates act as a form of regularization. Because it limits the number of trainable parameters, it reduces the risk of overfitting to the fine-tuning dataset, which can enhance the model's ability to generalize better to unseen data. Standard fine-tuning, on the other hand, updates all model weights, which increases the risk of overfitting, especially in cases where the fine-tuning dataset is small or domain-specific.

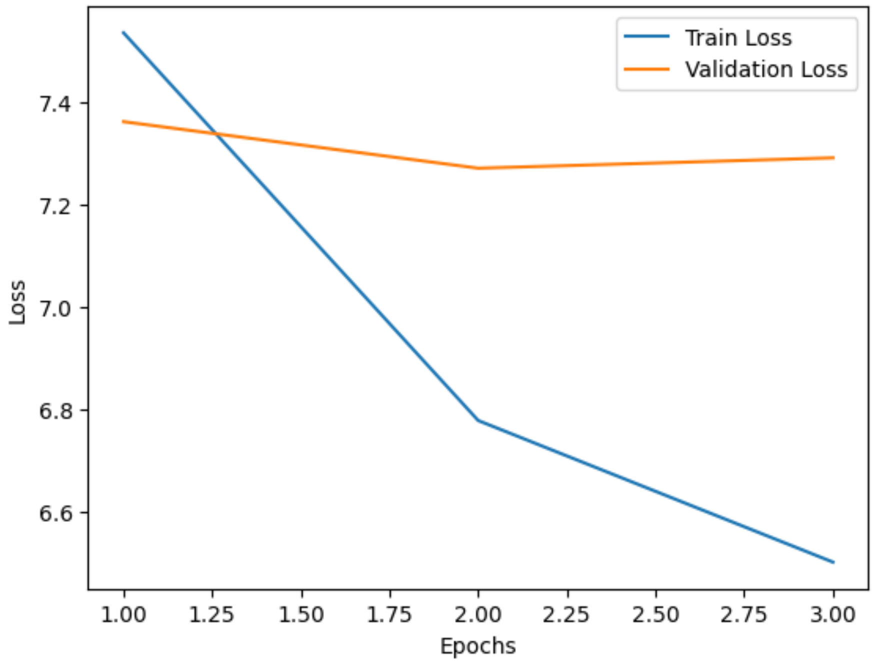
- **Task-Specific Adaptability:** LoRA's modular nature allows different low-rank adapters for different tasks, without needing to retrain or interfere with the original model weights. This enables efficient task-specific fine-tuning and potentially better generalization across different tasks compared to standard fine-tuning, which requires retraining the entire model for each new task.

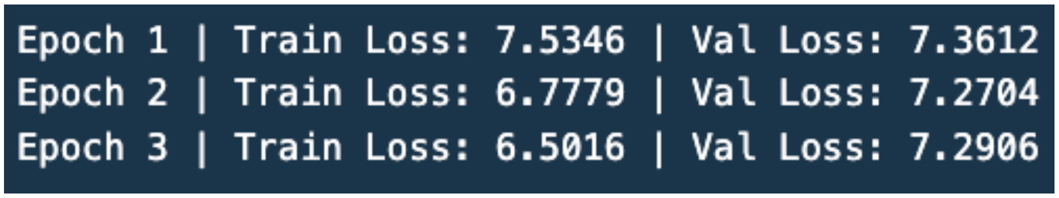
Analysis of Observations and Results

Pre-trained model performance:

'rouge1': 0.121, **'rouge2'**: 0.057, **'rougeL'**: 0.089, **'rougeLsum'**: 0.099

Full Fine Tuning (last layer only)




```
Epoch 1 | Train Loss: 7.5346 | Val Loss: 7.3612
Epoch 2 | Train Loss: 6.7779 | Val Loss: 7.2704
Epoch 3 | Train Loss: 6.5016 | Val Loss: 7.2906
```

After 3 training epochs, the model started to overfit (val set loss started increasing). We see that the loss kept decreasing for the first 3 epochs which shows that the model is able to learn the task of summarization.

<u>Hyper-parameters:</u>

Learning Rate: 5e-4

batch_size: 24

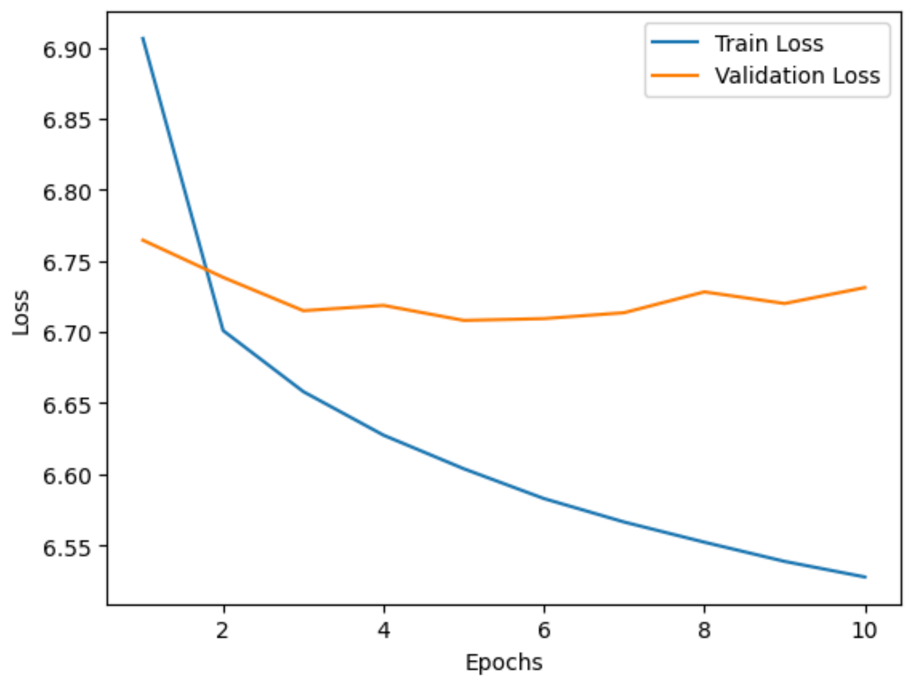
Epochs: 3

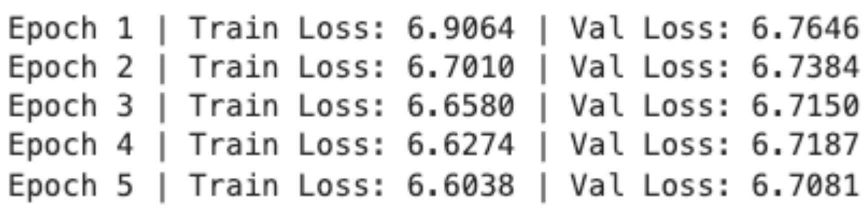
Optimizer: AdamW

<u>Performance metrics:</u>

'rouge1': 0.165**, 'rouge2':** 0.075**, 'rougeL':** 0.118**, 'rougeLsum':** 0.128

LoRA




```
Epoch 1 | Train Loss: 6.9064 | Val Loss: 6.7646
Epoch 2 | Train Loss: 6.7010 | Val Loss: 6.7384
Epoch 3 | Train Loss: 6.6580 | Val Loss: 6.7150
Epoch 4 | Train Loss: 6.6274 | Val Loss: 6.7187
Epoch 5 | Train Loss: 6.6038 | Val Loss: 6.7081
```

After 5 epochs, the model starts to overfit (loss for val set started increasing).

<u>Hyper-parameters:</u>

Learning Rate: 5e-4

batch_size: 24

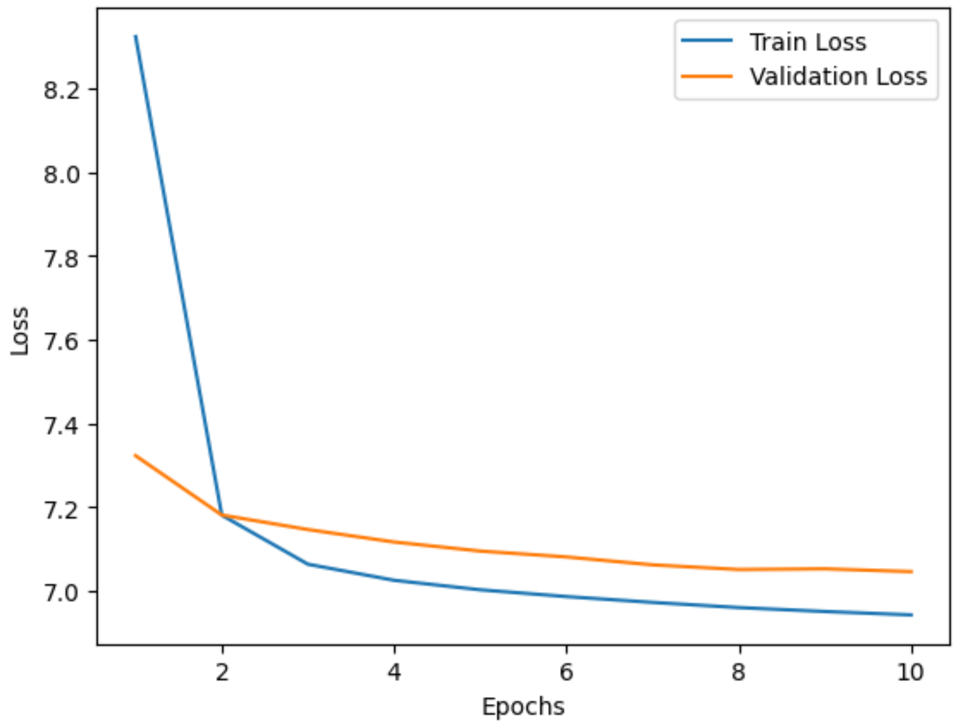
Epochs: 5

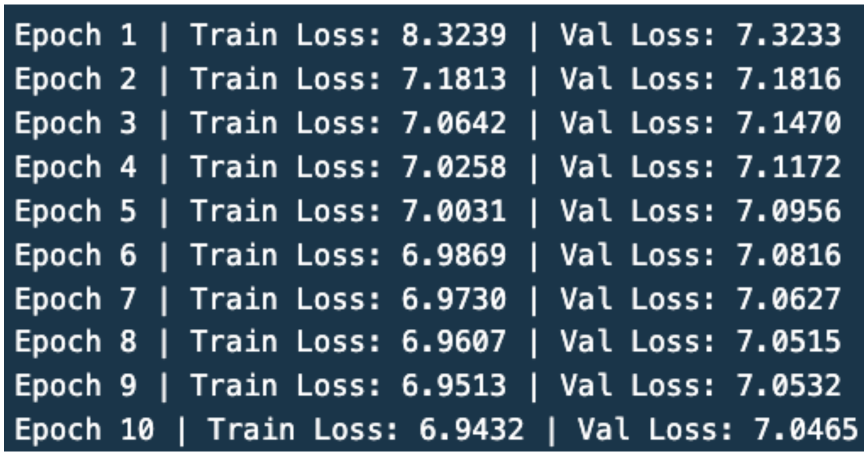
Optimizer: AdamW

<u>Performance metrics:</u>

'rouge1': 0.197**, 'rouge2':** 0.094**, 'rougeL':** 0.153**, 'rougeLsum':** 0.163

Prompt Tuning





```
Epoch 1 | Train Loss: 8.3239 | Val Loss: 7.3233
Epoch 2 | Train Loss: 7.1813 | Val Loss: 7.1816
Epoch 3 | Train Loss: 7.0642 | Val Loss: 7.1470
Epoch 4 | Train Loss: 7.0258 | Val Loss: 7.1172
Epoch 5 | Train Loss: 7.0031 | Val Loss: 7.0956
Epoch 6 | Train Loss: 6.9869 | Val Loss: 7.0816
Epoch 7 | Train Loss: 6.9730 | Val Loss: 7.0627
Epoch 8 | Train Loss: 6.9607 | Val Loss: 7.0515
Epoch 9 | Train Loss: 6.9513 | Val Loss: 7.0532
Epoch 10 | Train Loss: 6.9432 | Val Loss: 7.0465
```

Hyper-parameters:

Learning Rate: 5e-4

batch_size: 24

Epochs: 10

Optimizer: AdamW

Performance metrics:

'rouge1': 0.145**, 'rouge2':** 0.068**, 'rougeL':** 0.107**, 'rougeLsum':** 0.118

Analysis of performance

We observe that all the fine tuning techniques improve from the pre-trained model, evident from the improvement seen in the ROUGE scores (average) for test set, calculated before and after fine tuning. The best performing model (based on average ROUGE score on test set) is LoRA, followed by traditional fine tuning and then prompt tuning.

Training Loss

Traditional Fine-tuning (Lowest Loss): Full LM head updates allow for detailed task adaptation, leading to the lowest training loss and higher risk of overfitting.

LoRA (Moderate Loss): Low-rank updates constrain parameter adjustments, preventing perfect fitting, resulting in slightly higher training loss than traditional fine-tuning.

Prompt Tuning (Highest Loss): Only soft prompts are updated, limiting the model's flexibility and increasing training loss due to fewer parameters being optimized.

<u>Validation Loss</u>

LoRA (Lowest Loss): Low-rank updates prevent overfitting, leading to the best generalization and lowest validation loss.

Prompt Tuning (Moderate Loss): Limited overfitting due to fixed model weights; soft prompts improve validation performance but generalization is restricted.

Traditional Fine-tuning (Highest Loss): More prone to overfitting, causing high validation loss due to excessive adaptation to training data.

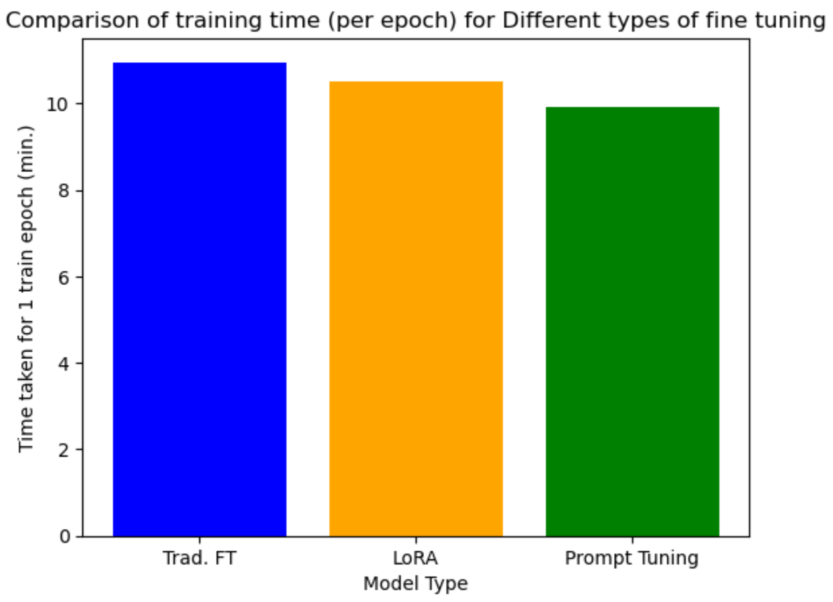
<u>Average test set ROUGE scores</u>

Overfitting in Traditional Fine-tuning: Lowest training loss but highest validation loss due to overfitting, leading to poor generalization and lower ROUGE scores.

LoRA's Balance: Low-rank updates prevent overfitting while maintaining generalization, leading to the <u>best test performance</u> (highest ROUGE) and lowest validation loss.

Prompt Tuning's Limitation: Limited to prompt adjustments, restricting task adaptation. It avoids overfitting but offers weaker performance (lower ROUGE) due to lack of flexibility.

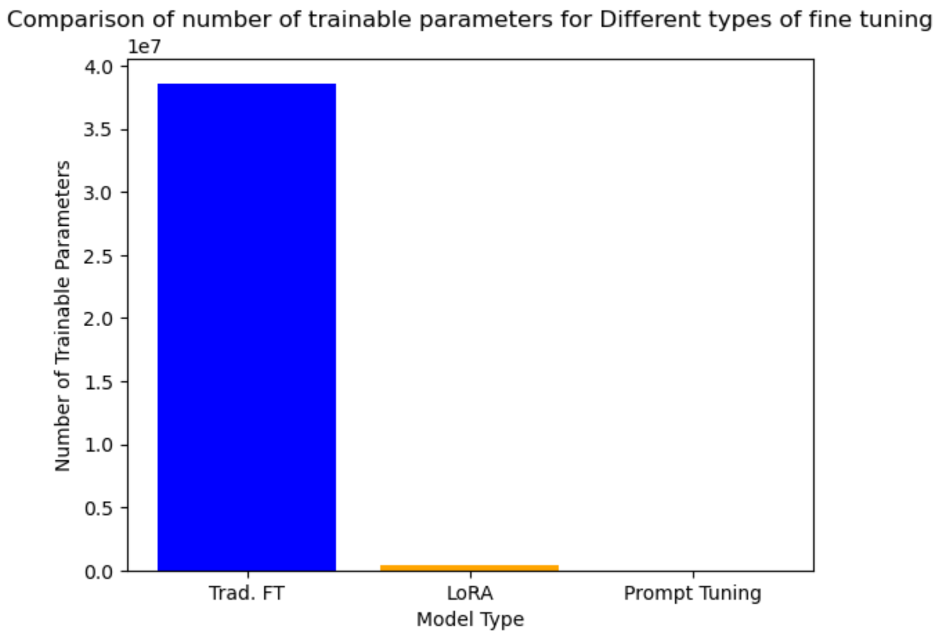
<u>Comparison of training time (per epoch)</u>



As expected, the training time per epoch is highest for the model with highest number of trainable parameters (traditional fine tuning) and lowest for the one with the lowest number of trainable parameters (prompt tuning).

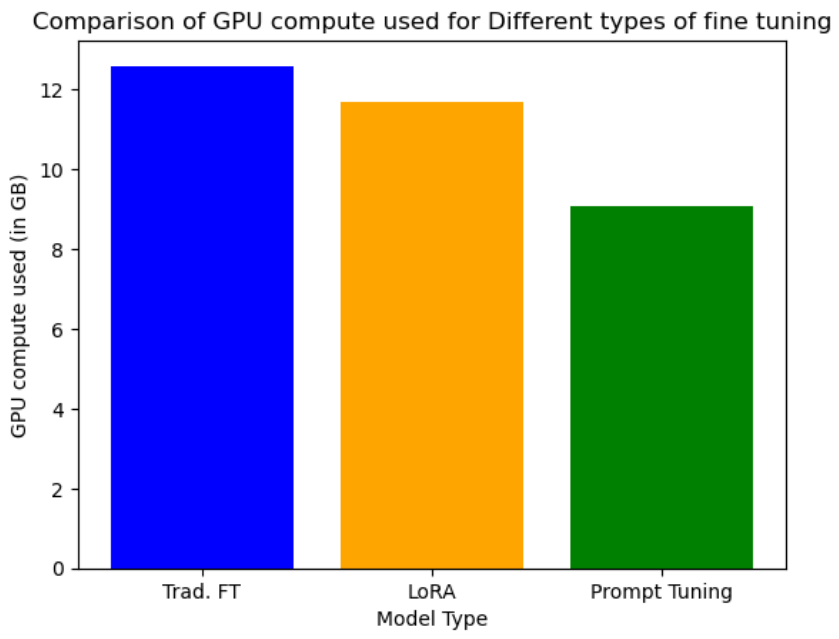
Comparison of trainable parameters

Traditional Fine Tuning: 38597376, **LoRA:** 442368, Prompt Tuning: 7680 (10 soft prompts, each 768 dimensional



Comparison of GPU compute

We observe that the maximum GPU compute is needed for the model with the highest number of triable parameters (traditional fine tuning, 12.6GB GPU VRAM), and the least for the model with lowest number of trainable parameters (prompt tuning, 9.1 GB GPU VRAM).


Comparison of GPU compute used for Different types of fine tuning
GPU compute used (in GB)
Trad. FT
LoRA
Prompt Tuning
Model Type