Assignment – 4

Understanding LoRA:

Q-1. Review the concept, benefits, and mechanism of Low-Rank Adaptation (LoRA) for adapting pretrained models.

Ans:

Concept: LoRA is a machine learning technique that modifies pretrained model’s parameters to better suit specific, often smaller, datasets. It achieves this by adjusting onlya small, low-rank subset of the model’s parameters.

Benefits:

Efficiency: As I mentioned, LoRA trains only a small number of parameters, it is importantly faster and requires less memory than traditional fine-tuning methods. This makes it more memory-efficient and potentially allows for faster adaptation over the specific task.

Flexibility: By using low-rank matrices, LoRA enables selective adaptation of different parts of the model, allowing fine-tuning to focus on specific aspects relevant to the target task.

Mechanism: LoRA technique freezes the weights of pre-trained model and include the learnable rank decomposition matrices into each of the layers of the Transformer architecture. This reduction in trainable parameters facilitates downstream tasks while maintaining or improving model quality.

Q-2. Discuss the suitability of pretrained language models for code-related QA tasks and the advantages of using LoRA for fine-tuning.

Ans:

Suitability of Pretrained Models:

Pretrained language models (LLMs) have already showed the very good performance across various natural language understanding tasks. Their ability to capture context and semantics makes them suitable for code-related question-answering (QA) tasks. For example, code related QA system for "flytech/python-codes-25k" dataset. However, fine-tuning large LLMs on specific code-related tasks can be computationally very difficult. This is because of the big number of learnable parameters. That is because, at every iteration, large numbers of parameters need to be updated.

Advantages of Using LoRA for Fine-Tuning:

Parameter Reduction: LoRA significantly reduces the big number of learnable parameters, making fine-tuning more feasible even for large models.

Quality Retention: Despite having fewer parameters, LoRA performs better than traditional fine-tuning in terms of model quality.

Efficiency: LoRA’s efficiency allows for faster adaptation, which is very crucial for code-related QA tasks where responsiveness matters a lot.

No Additional Inference Latency: Unlike adapters, LoRA introduces no additional inference latency.

Dataset Preparation:

Q-1. Provide an overview of the "flytech/python-codes-25k"dataset, focusing on its structure and relevance for a QA system.

Ans:

The “flytech/python-codes-25k” dataset is a quite large collection of around 49.6K Python code entries specifically designed for tasks like, code related question-answering. It’s structured into four key features:

Instruction: A concise task description or user query.

Input: Any initial data or parameters the code might need.

Output: The expected result after the code execution or expected LLM model output.

Text: A combination of the instruction, input, and output fields that are mentioned above.

This given dataset is particularly significant for question-answering systems. It contains a diverse set of coding tasks and their corresponding solutions. These tasks cover a wide range of programming challenges. By using this dataset, we can train language models to understand and generate code based on natural language instructions.

Some key points about the dataset:

Task Variety: The dataset includes a rich collection of coding problems, each with its own unique requirements. These tasks span different domains and levels of complexity.

Instructions: There are 24,580 unique instructions provided alongside the coding tasks. These instructions serve as examples of how human can naturally express their coding requirements in natural language.

Inputs and Outputs: For each task, there are corresponding input examples (3,666 in total) and output solutions (24,581 in total). These inputs and outputs allow us to study the relationship between natural language descriptions and the actual code produced.

Q-2 Describe necessary preprocessing steps, including tokenization and encoding strategies for code snippets.

Ans:

Preprocessing Steps for Code Snippets: Before using the code snippets in natural language models, they need to be preprocessed.

Cleaning:

- Remove comments to focus on the executable code.
- Also remove unnecessary string like “python\n” at the beginning of code.
- Normalize Whitespace: Standardize the use of spaces and tabs for consistency.
- Remove duplicate and empty entries.
- Keep only import columns “instruction” and “output”

Eliminate Non-Code Elements:

- Remove non-code text or metadata available in code.

Tokenization:

- This step involves breaking down the code into its basic components or ‘tokens’. These tokens can be keywords, operators, variables, etc. For Python code, this means identifying for, while, if statements, and other Python-specific syntax too.

Sequence Padding:

- Fixed-Length Sequences: Pad the sequences to the maximum length to create uniform input size for models.

Convert to Vectors:

- Transform the processed tokens into numerical vectors that can be fed into machine learning models. These preprocessing steps are crucial for preparing code snippets for a QA system, as they help in reducing noise, standardizing input, and extracting meaningful features that can be used for accurate predictions and responses.

These preprocessing steps are essential for transforming the raw code into a format that will be effectively used for training an LLM model that can act like an QA system.

Model Fine-Tuning with LoRA:

Q-1 Select a suitable pretrained language model and justify the choice based on its architecture and expected performance on code-related QA tasks.

Ans:

Choice of Pretrained Language Model:

The "Salesforce/codet5-base" model is a suitable choice for a language model that specializes in understanding programming syntax and generating code in different computer languages. That is why it's best suitable for my python code-related question-answering (QA) tasks here.

Architecture:

This T5 model is basically an encoder-decoder based (Text-to-Text Transfer Transformer) architecture. T5 is also known for its effectiveness in handling natural language processing tasks. The "base" in its name indicates that it's a medium-sized version of the model, offering a balance between performance and resource usage. The T5 model turns every task that involves text into a single format where both the information you put in and get out are in the form of text sentences. This design makes it flexible and powerful for a variety of tasks, including those related to code.

Pretraining on Code: Before being fine-tuned for specific tasks, the codet5-base model was pretrained on a large corpus of code from different programming languages like java, Ruby, PHP and Go. This pretraining helps the model learn the syntax and semantics of python code, which is essential for understanding programming-related questions and generating accurate answers.

Expected Performance: Due to its pretraining on diverse coding datasets, the codet5-base model is expected to perform well on python code dataset **"flytech/python-codes-25k"**, code-related QA tasks. It can understand context within code snippets, recognize programming patterns, and generate explanations or solutions that are syntactically correct and semantically meaningful.

Q-2 Detail the integration of LoRA, specifying the adaptation process and adjustments made to the model for the QA task.

Ans:

LoRA Integration: As mentioned earlier, LoRA is a technique used to adapt large pretrained models like Salesforce/codet5-base for specific tasks with minimal additional parameters. It works by adding small, trainable changes to the existing model without altering the original pretrained weights. This allows the model to learn new tasks effectively while retaining the knowledge it gained during pretraining.

Adaptation Process: The adaptation process involves inserting low-rank matrices into the model's attention mechanism. These matrices are much smaller than the model's original parameters, making them easier to train. The low-rank matrices are applied to specific parts of the attention mechanism, namely the key (k), query (q), value (v), and output (o) projections. This targeted approach allows the model to adjust its behavior for the QA task without needing to retrain the entire model.

Adjustments Made:

The LoRA configuration which has been adapted with the model codet5-base:

r=4: This is the rank of the low-rank matrices, which determines their size. A smaller rank means fewer parameters to train.

lora_alpha=32: This scaling factor increases the influence of the low-rank matrices on the model's behavior.

lora_dropout=0.01: This is the dropout rate for the LoRA layers, which helps prevent overfitting by randomly setting a portion of the weights to zero during training.

target_modules=["k","q","v","o"]: These are the parts of the attention mechanism that the low-rank matrices will modify.

Model Adaptation:

For the adaption of the LoRA, I have used PEFT library. The get_peft_model function from this lirary is called with the base model and the LoRA configuration as arguments. This function integrates the LoRA adaptation into the model and make it ready for training on the QA task.

Trainable Parameters: The print_trainable_parameters function shows that only a small fraction (0.395%) of the model’s parameters will be trained, which indicates that LoRA is a parameter-efficient method. The model retains most of its original structure, with only a small part being adapted for the new task.

Overall, LoRA allows the Salesforce/codet5-base model to become more specialized for the QA task by learning from new data and adjusting its responses accordingly, all while keeping the training process more efficient and focused.

Training and Evaluation:

Q-1 Outline the training process, including configurations related to LoRA, learning rate settings, and QA-specific adaptations.

Ans:

For the training of LoRA adapted, I have chosen only first 100 samples only due to limitation of the computational resources that I have and for less training time. The training process for the QA task using LoRA involves specific parameter choices that are important for the model's performance. These parameters and their values are given below.

Learning Rate (0.001): The learning rate is a key hyperparameter in the training of neural networks. It controls how much the model's weights should be updated during training. A value of 0.001 is chosen as it's small enough to allow the model to gradually learn from the training data without making drastic changes that could lead to poor generalization. It's a common starting point for fine-tuning tasks.

Batch Size (8 for both training and evaluation): The batch size determines how many examples the model sees before it updates its weights. A size of 8 is a balance between computational efficiency that I have and the ability to generalize well. Smaller batch sizes often lead to more stable and reliable training, but it will increase the training time of my model.

Evaluation Strategy ("steps"): Evaluating the model at regular intervals (every 100 steps) allows for frequent monitoring of the model's performance on the validation dataset that I have provided. This helps in understanding how well the model is learning and generalizing to new data.

Number of Training Epochs (100): Setting the number of epochs to 100 ensures that the model has sufficient exposure to the training data. This number has been chosen to give the model ample opportunity to learn the specific task of the QA system without overfitting.

LoRA Parameters (r=4, lora_alpha=32, lora_dropout=0.01):

r=4: A lower rank for the matrices keeps the number of additional parameters small, which is important for maintaining the efficiency of the model.

lora_alpha=32: A higher scaling factor increases the impact of the LoRA layers, allowing the model to make more significant adjustments during fine-tuning.

lora_dropout=0.01: A small dropout rate helps prevent overfitting by introducing some randomness in the training process.

These parameter choices have been chosen by me after hyperparameter tunning as well as with my best knowledge to check the model's performance to learn coding-based QA from the training data. I trained this model for 100 epochs with only 100 samples because of having less the computational resources. The loss value at every epoch (100 steps) is given below.

[500/500 03:01, Epoch 100/100]

Step	Training Loss	Validation Loss
100	0.884200	0.820719
200	0.123800	0.920829
300	0.041200	0.967752
400	0.022900	1.004380
500	0.016300	1.017744

```
TrainOutput(global_step=500, training_loss=0.2176723017692566,
metrics={'train_runtime': 183.8049, 'train_samples_per_second': 21.762,
'train_steps_per_second': 2.72, 'total_flos': 90795626496000.0, 'train_loss':
0.2176723017692566, 'epoch': 100.0})
```

Q-2 Evaluate the fine-tuned model using appropriate metrics, comparing its performance with a baseline model.

Ans:

For the evaluation of Lora adapted model (salesforce/codet5-base + Lora layer) trained on the python dataset and baseline model (salesforce/codet5-base), I have used two metrics BLUE and ROUGE and tested it with first four instruction or in other words prompt from the training dataset. They both has been measured by comparing the code generated by trained lora model and baseline model. The performance of both lora adapted trained model and baseline model has been described below,

1. BLUE Score of LoRA model:

```
Prompt 1: Help me set up my daily to-do list!

Generated Code: tasks = []
while True:
    task = input('Enter a task or type 'done' to finish: ')
    if task == 'done': break
    tasks.append(task)
print(f'Your to-do list for today: {tasks}')

BLEU Score: 1.0000
--------------------------------------------------------------------------------
Prompt 2: Create a shopping list based on my inputs!
Generated Code: shopping_list = {}
while True:
    item = input('Enter an item or type 'done' to finish: ')
    if item == 'done': break
    quantity = input(f'Enter the quantity for {item}: ')
    shopping_list[item] = quantity
print(f'Your shopping list: {shopping_list}')

BLEU Score: 1.0000
--------------------------------------------------------------------------------
Prompt 3: Calculate how much time I spend on my phone per week!
Generated Code: total_time = 0
for i in range(1, 8):
    time = float(input(f'Enter phone usage in hours for day {i}: '))
    total_time += time
print(f'You spend approximately {total_time} hours per week on your
phone.')

BLEU Score: 1.0000
--------------------------------------------------------------------------------
Prompt 4: Help me split the bill among my friends!
Generated Code: total_bill = float(input('Enter the total bill amount: '))
friends_count = int(input('Enter the number of friends: '))
per_person = total_bill / friends_count
print(f'Each person should pay {per_person}')

BLEU Score: 1.0000
--------------------------------------------------------------------------------
Average BLEU Score of Lora model: 1.0
```

2. ROUGE Score of LoRA model:

```
Rouge Scores of Lora Model:

Prompt 1: Help me set up my daily to-do list!
Rouge Score: [{'rouge-1': {'r': 1.0, 'p': 1.0, 'f': 0.999999995}, 'rouge-
2': {'r': 1.0, 'p': 1.0, 'f': 0.999999995}, 'rouge-l': {'r': 1.0, 'p':
1.0, 'f': 0.999999995}}]
--------------------------------------------------------------------
Prompt 2: Create a shopping list based on my inputs!
Rouge Score: [{'rouge-1': {'r': 1.0, 'p': 1.0, 'f': 0.999999995}, 'rouge-
2': {'r': 1.0, 'p': 1.0, 'f': 0.999999995}, 'rouge-l': {'r': 1.0, 'p':
1.0, 'f': 0.999999995}}]
--------------------------------------------------------------------
Prompt 3: Calculate how much time I spend on my phone per week!
Rouge Score: [{'rouge-1': {'r': 1.0, 'p': 1.0, 'f': 0.999999995}, 'rouge-
2': {'r': 1.0, 'p': 1.0, 'f': 0.999999995}, 'rouge-l': {'r': 1.0, 'p':
1.0, 'f': 0.999999995}}]
--------------------------------------------------------------------
Prompt 4: Help me split the bill among my friends!
Rouge Score: [{'rouge-1': {'r': 1.0, 'p': 1.0, 'f': 0.999999995}, 'rouge-
2': {'r': 1.0, 'p': 1.0, 'f': 0.999999995}, 'rouge-l': {'r': 1.0, 'p':
1.0, 'f': 0.999999995}}]
--------------------------------------------------------------------
```

3. BLUE Score of Baseline model:

```
Prompt 1: Help me set up my daily to-do list!
Generated Code: I'm not sure how to
BLEU Score: 0.0000
--------------------------------------------------------------------
Prompt 2: Create a shopping list based on my inputs!
Generated Code:  function ( ) {
BLEU Score: 0.0000
--------------------------------------------------------------------
Prompt 3: Calculate how much time I spend on my phone per week!
Generated Code:  public function calculateTime (
BLEU Score: 0.0000
--------------------------------------------------------------------
Prompt 4: Help me split the bill among my friends!
Generated Code: public class Bill {
BLEU Score: 0.0000
--------------------------------------------------------------------
Average BLEU Score of Baseline model: 3.061619493425024e-234
```

4. ROUGE Score of Baseline model:

```
Rouge Scores of Baseline Model:

Prompt 1: Help me set up my daily to-do list!
Rouge Score: [{'rouge-1': {'r': 0.04, 'p': 0.2, 'f': 0.066666663888889},
'rouge-2': {'r': 0.0, 'p': 0.0, 'f': 0.0}, 'rouge-l': {'r': 0.04, 'p':
0.2, 'f': 0.066666663888889}}]
--------------------------------------------------------------------
Prompt 2: Create a shopping list based on my inputs!
Rouge Score: [{'rouge-1': {'r': 0.0, 'p': 0.0, 'f': 0.0}, 'rouge-2': {'r':
0.0, 'p': 0.0, 'f': 0.0}, 'rouge-l': {'r': 0.0, 'p': 0.0, 'f': 0.0}}]
--------------------------------------------------------------------
Prompt 3: Calculate how much time I spend on my phone per week!
Rouge Score: [{'rouge-1': {'r': 0.0, 'p': 0.0, 'f': 0.0}, 'rouge-2': {'r':
0.0, 'p': 0.0, 'f': 0.0}, 'rouge-l': {'r': 0.0, 'p': 0.0, 'f': 0.0}}]
--------------------------------------------------------------------
Prompt 4: Help me split the bill among my friends!
Rouge Score: [{'rouge-1': {'r': 0.0, 'p': 0.0, 'f': 0.0}, 'rouge-2': {'r':
0.0, 'p': 0.0, 'f': 0.0}, 'rouge-l': {'r': 0.0, 'p': 0.0, 'f': 0.0}}]
```

LoRA-Adapted Model:

- Here, I achieved a BLEU score of 1.00 for all four prompts that I tested from training dataset. It indicates an exact match between the model generated code and the reference code.
- The ROUGE scores also show a perfect or near-perfect match across all the evaluated aspects (recall, precision, and f-measure), further confirming the high quality of the generated code.

Baseline Model:

- Received a BLEU score of 0.00 for all prompts, suggesting no overlap with the reference code, which simply indicates a failure to generate relevant code.
- The ROUGE scores are also lower, with the highest f-measure being only 0.0667, showing that the baseline model's outputs have little to no similarity with the expected code.

The LoRA-adapted model's perfect scores suggest that it has learned the task exceptionally well, at least for the given dataset. However, such high scores, especially a consistent BLEU score of 1.00, are unusual in practice and might indicate overfitting, especially considering the small training sample size of only 100.

Q-3 Analyze the results, focusing on improvements or limitations introduced by LoRA in the context of programming-related QA.

Ans:

The small dataset size (100 samples) that I have used here for training the LoRA-adapted model is not sufficient to generalize the findings. A larger and more diverse dataset would be necessary to validate the model's true performance. It may not behave well on unseen data. So, I will need to train it for more data as well as try to play around the different parameters of lora layers like different size of lora matrices and dropout rate too.

The perfect scores of the LoRA model might not reflect real-world performance and may mis behave on unseen dataset. So, that should be kept in mind for me for the future improvements. It's important to test the model on a separate, unseen test set to evaluate its true generalization capabilities.

In conclusion, while the LoRA-adapted model shows promising results according to the metrics, the evaluation context raises questions about the robustness and practicality of these results. Further testing with a larger dataset and a more competitive baseline would provide a clearer picture of the model's capabilities.