Parameter Efficient Fine Tuning (PEFT) - LoRA and QLoRA

Problem Statement

In the realm of machine learning, fine-tuning large language models (LLMs) can be computationally expensive and resource-intensive. Traditional fine-tuning methods require updating all model parameters, which is not always feasible for large models due to memory constraints and long training times. **Parameter Efficient Fine Tuning (PEFT)** offers a solution by enabling efficient fine-tuning with minimal computational overhead. Two prominent techniques in PEFT are **Low-Rank Adaptation (LoRA)** and **Quantized Low-Rank Adaptation (QLoRA)**.

Technical Stack

To implement PEFT using LoRA and QLoRA, we will use the following technical stack:

- **Python**: Programming language for implementation.
- **PyTorch**: Deep learning framework.
- **Transformers**: Hugging Face library for working with LLMs.
- **Datasets**: Hugging Face library for dataset management.
- **Scikit-learn**: For evaluation metrics.

Steps to be Followed

1. Setup Environment

First, we need to set up our environment by installing the necessary libraries.

```
!pip install torch transformers datasets scikit-learn
```

2. Load Pre-trained Model and Dataset

Next, we load a pre-trained model and a dataset for fine-tuning.

```
from transformers import AutoModelForSequenceClassification, AutoTokenizer
from datasets import load_dataset

model_name = "bert-base-uncased"
model = AutoModelForSequenceClassification.from_pretrained(model_name)
tokenizer = AutoTokenizer.from_pretrained(model_name)
dataset = load_dataset("imdb")
```

3. Implement LoRA

LoRA introduces low-rank matrices to adapt the model weights efficiently. We will define a LoRA layer and integrate it into our model.

```
import torch.nn as nn

class LoRALayer(nn.Module):
    def __init__(self, input_dim, output_dim, rank):
        super(LoRALayer, self).__init__()
        self.rank = rank
        self.A = nn.Parameter(torch.randn(input_dim, rank))
        self.B = nn.Parameter(torch.randn(rank, output_dim))

    def forward(self, x):
        return x + torch.matmul(torch.matmul(x, self.A), self.B)

# Integrate LoRA into the model
for name, param in model.named_parameters():
    if "weight" in name:
        input_dim, output_dim = param.shape
        lora_layer = LoRALayer(input_dim, output_dim, rank=4)
        param.requires_grad = False
        model.add_module(name + "_lora", lora_layer)
```

4. Implement QLoRA

QLoRA combines quantization with LoRA to further reduce memory usage. We will quantize the model weights and apply LoRA.

```
import torch.quantization as quant

# Quantize model weights
```

```
model.qconfig = quant.get_default_qconfig("fbgemm")
quant.prepare(model, inplace=True)
quant.convert(model, inplace=True)

# Apply LoRA as shown in the previous step
```

5. Fine-tune the Model

Now, we can fine-tune the model using the IMDB dataset.

```
from transformers import Trainer, TrainingArguments

training_args = TrainingArguments(
    output_dir="./results",
    evaluation_strategy="epoch",
    learning_rate=2e-5,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    num_train_epochs=3,
    weight_decay=0.01,
)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=dataset["train"],
    eval_dataset=dataset["test"],
    tokenizer=tokenizer,
)

trainer.train()
```

6. Evaluate the Model

Finally, we evaluate the fine-tuned model.

```
from sklearn.metrics import accuracy_score

predictions = trainer.predict(dataset["test"]).predictions
pred_labels = predictions.argmax(axis=1)
true_labels = dataset["test"]["label"]
```

```
accuracy = accuracy_score(true_labels, pred_labels)
print(f"Accuracy: {accuracy:.4f}")
```

Conclusion

Parameter Efficient Fine Tuning (PEFT) using LoRA and QLoRA provides a practical approach to fine-tuning large language models with reduced computational resources. By introducing low-rank adaptations and quantization, we can achieve efficient model updates without compromising performance. This technique is particularly useful for deploying LLMs in resource-constrained environments.