# PFFT using LoRA

Did Supervised Fine Tuning :Parameter-Efficient Fine-Tuning (PEFT), specifically using techniques like LoRA (Low-Rank Adaptation) for fine-tuning large language models with reduced computational cost.

* torch: PyTorch library for tensor computation and model training.
* datasets: A library by Hugging Face for loading and processing datasets.
* peft: Library for parameter-efficient fine-tuning techniques, such as LoRA.
* transformers: Hugging Face library for transformers models.
* trl: Library for reinforcement learning and fine-tuning transformers.
* Other imports are for logging, data manipulation, and setting environment variables.

**Setting Environment Variables**

os.environ["TOKENIZERS\_PARALLELISM"] = "True"

import torch

from datasets import load\_dataset

from peft import LoraConfig, PeftModel, prepare\_model\_for\_kbit\_training

from transformers import (

AutoModelForCausalLM,

AutoTokenizer,

BitsAndBytesConfig,

TrainingArguments,

)

from trl import SFTTrainer

import logging

import pandas as pd

from datetime import datetime

import os

**Model Dataset Path**

model\_name = "meta-llama/Meta-Llama-3-8B-Instruct"

log\_path = "/home/ankur/projects/llm\_test/Akshay/fine\_tuning"

finetuned\_path = "/home/ankur/projects/llm\_test/Akshay/fine\_tuning"

train\_dataset\_json = "/home/ankur/projects/llm\_test/Akshay/fine\_tuning/bvr\_training\_set\_v2.json"

eval\_dataset\_json = "/home/ankur/projects/llm\_test/Akshay/fine\_tuning/bvr\_eval\_set\_v2.json"

**Quantization Configuration**

quant\_config = BitsAndBytesConfig(

load\_in\_4bit=True,

bnb\_4bit\_use\_double\_quant=True,

bnb\_4bit\_quant\_type="nf4",

bnb\_4bit\_compute\_dtype=torch.bfloat16

)

Configures the model to use 4-bit quantization, a technique to reduce model size and improve inference speed.

**Loading the Model**

model = AutoModelForCausalLM.from\_pretrained(

model\_name,

quantization\_config=quant\_config,

device\_map="auto",

attn\_implementation="flash\_attention\_2"

)

Loads a pre-trained causal language model with the specified quantization settings and device mapping.

**Loading the Tokenizer**

tokenizer = AutoTokenizer.from\_pretrained(model\_name, trust\_remote\_code=True)

if tokenizer.pad\_token is None:

tokenizer.pad\_token = tokenizer.eos\_token

tokenizer.padding\_side = "right"

Loads the tokenizer for the specified model, ensuring it has a padding token and sets the padding side to right.

**Loading and Preparing Datasets**

train\_dataset = load\_dataset("json", data\_files=train\_dataset\_json, split="train")

train\_dataset = train\_dataset.map(lambda x: {"formatted\_chat": tokenizer.apply\_chat\_template(x['messages'], tokenize=False, add\_generation\_prompt=False)})

eval\_dataset = load\_dataset("json", data\_files=eval\_dataset\_json, split="train")

eval\_dataset = eval\_dataset.map(lambda x: {"formatted\_chat": tokenizer.apply\_chat\_template(x['messages'], tokenize=False, add\_generation\_prompt=False)})

Loads training and evaluation datasets from JSON files and applies a formatting template using the tokenizer.

**Calculating Maximun Token Length**

train\_token = (tokenizer(train\_dataset['formatted\_chat'])).get('input\_ids')

train\_token\_max = max(len(token) for token in train\_token)

test\_token = (tokenizer(eval\_dataset['formatted\_chat'])).get('input\_ids')

test\_token\_max = max(len(token) for token in test\_token)

max\_token = max(test\_token\_max, train\_token\_max)

print(max\_token) #1536

Tokenizes the formatted chats in the datasets and calculates the maximum token length to set sequence length limits for training.

**Re-Loading the Model with Quantization and Preparation for Training**

bnb\_config = BitsAndBytesConfig(

load\_in\_4bit=True,

bnb\_4bit\_use\_double\_quant=True,

bnb\_4bit\_quant\_type="nf4",

bnb\_4bit\_compute\_dtype=torch.bfloat16

)

model = AutoModelForCausalLM.from\_pretrained(

model\_name,

quantization\_config=bnb\_config,

device\_map="auto",

attn\_implementation="flash\_attention\_2"

)

model = prepare\_model\_for\_kbit\_training(model)

Loads the model again with the same quantization settings and prepares it for k-bit (quantized) training.

**Configuring PEFT with LoRA**

peft\_config = LoraConfig(

lora\_alpha=16,

lora\_dropout=0.05,

r=16,

bias="none",

task\_type="CAUSAL\_LM",

target\_modules=['k\_proj', 'q\_proj', 'v\_proj', 'o\_proj', "gate\_proj", "down\_proj", "up\_proj"]

)

LoRA config parameters:

1. lora\_alpha: A scaling factor for LoRA. It determines how much the LoRA layers contribute to the final model's predictions.
2. lora\_dropout: Dropout rate for LoRA layers. Helps prevent overfitting.
3. r: Rank of the LoRA adaptation. Determines the complexity of the LoRA layers.
4. bias: Specifies whether to use bias in LoRA layers. "none" means no bias is used.
5. task\_type: Type of task, set to "CAUSAL\_LM" (causal language modeling).
6. target\_modules: List of model modules to which LoRA will be applied. These are typically the projection layers in the transformer architecture.

**Setting Training Arguments**

training\_arguments = TrainingArguments(

output\_dir=f"{finetuned\_path}/llama\_3\_snippets\_v2",

logging\_dir = f"{log\_path}/llama\_3\_snippets\_v2",

num\_train\_epochs= 4,

max\_steps = -1,

per\_device\_train\_batch\_size=2,

per\_device\_eval\_batch\_size=2,

gradient\_accumulation\_steps=6,

eval\_accumulation\_steps = 4,

optim="paged\_adamw\_8bit",

save\_strategy="epoch",

logging\_strategy="epoch",

evaluation\_strategy="epoch",

learning\_rate= 2e-4,

fp16= False,

bf16= True,

group\_by\_length= True,

disable\_tqdm=False,

overwrite\_output\_dir=True,

save\_total\_limit=2,

load\_best\_model\_at\_end=True,

report\_to="tensorboard"

)

Specifies training arguments such as batch size, learning rate, number of epochs, and evaluation strategy.

Training Hyperparameters:

1. output\_dir: Directory where the trained model and checkpoints will be saved.
2. logging\_dir: Directory where the logs will be stored.
3. num\_train\_epochs: Number of epochs to train the model.
4. max\_steps: Maximum number of training steps. If set to -1, it is ignored in favor of num\_train\_epochs.
5. per\_device\_train\_batch\_size: Batch size per device (GPU/CPU) for training.
6. per\_device\_eval\_batch\_size: Batch size per device (GPU/CPU) for evaluation.
7. gradient\_accumulation\_steps: Number of updates steps to accumulate the gradients before performing a backward/update pass.
8. eval\_accumulation\_steps: Number of predictions steps to accumulate before moving the tensors to the CPU.
9. optim: Optimizer type, in this case, "paged\_adamw\_8bit", which is an 8-bit Adam optimizer variant.
10. save\_strategy: Strategy to save checkpoints, set to "epoch" to save after each epoch.
11. logging\_strategy: Strategy to log training progress, set to "epoch" to log after each epoch.
12. evaluation\_strategy: Strategy to evaluate the model, set to "epoch" to evaluate after each epoch.
13. learning\_rate: Learning rate for the optimizer.
14. fp16: Boolean to use 16-bit (mixed) precision training.
15. bf16: Boolean to use bfloat16 precision training.
16. group\_by\_length: Group sequences of similar lengths together to optimize training.
17. disable\_tqdm: Disable the tqdm progress bar.
18. overwrite\_output\_dir: Overwrite the content of the output directory.
19. save\_total\_limit: Maximum number of checkpoints to keep. Older checkpoints will be deleted.
20. load\_best\_model\_at\_end: Load the best model at the end of training based on evaluation metrics.
21. report\_to: The logging platform to use (e.g., "tensorboard").

**Initializing the Trainer**

trainer = SFTTrainer(

model=model,

dataset\_text\_field="formatted\_chat",

train\_dataset=train\_dataset,

eval\_dataset=eval\_dataset,

peft\_config=peft\_config,

max\_seq\_length=max\_token,

tokenizer=tokenizer,

args=training\_arguments,

)

Initializes the trainer with the model, datasets, tokenizer, and training arguments.

The SFTTrainer is a specialized trainer for supervised fine-tuning of transformers. Here's a detailed explanation of each parameter passed to the SFTTrainer:

1. model:

- The pre-trained model to be fine-tuned. In this case, it is a model loaded with 4-bit quantization and prepared for LoRA-based training.

2. dataset\_text\_field:

- The name of the field in the dataset that contains the text to be used for training. Here, formatted\_chat is the field containing the pre-processed text data.

3. train\_dataset:

- The dataset used for training. This dataset has been loaded, processed, and formatted to include the formatted\_chat field.

4. eval\_dataset:

- The dataset used for evaluation. Similar to train\_dataset, this dataset has been prepared with the formatted\_chat field for evaluation purposes.

5. peft\_config:

- The configuration for parameter-efficient fine-tuning (PEFT) using LoRA. This includes settings like lora\_alpha, lora\_dropout, and the target modules where LoRA will be applied.

6. max\_seq\_length:

- The maximum sequence length for the input tokens. This is set to max\_token, which was calculated earlier based on the longest sequence length in the training and evaluation datasets.

7. tokenizer:

- The tokenizer used to preprocess and tokenize the text data. The tokenizer is essential for converting text to input IDs that the model can process.

8. args:

- The training arguments encapsulated in a TrainingArguments instance. These arguments include configurations for learning rate, batch size, number of epochs, logging, and other training parameters.

The SFTTrainer is configured to:

- Model: Fine-tune the specified pre-trained model using the training and evaluation datasets.

- Data: Use formatted\_chat field from both training and evaluation datasets.

- LoRA Config: Apply LoRA configurations for parameter-efficient fine-tuning.

- Tokenizer: Tokenize the text data for training.

- TrainingArguments: Follow specified training arguments for the entire fine-tuning process.

This setup ensures that the fine-tuning process is efficient, taking advantage of LoRA to reduce the number of trainable parameters while maintaining model performance, and utilizing 4-bit quantization to further reduce the computational load. The SFTTrainer simplifies the fine-tuning process by managing the training loop, evaluation, and checkpointing according to the provided configurations.

**Training and Saving the Model**

trainer.train()

trainer.save\_model()

Trains the model and saves the trained model.

**Cleaning Up and Loading the Fine-Tuned Model**

del modeldel trainerimport gc

gc.collect()

gc.collect()

torch.cuda.empty\_cache()

device\_map = {"": 0}

model = AutoPeftModelForCausalLM.from\_pretrained(

training\_arguments.output\_dir,

low\_cpu\_mem\_usage=True,

return\_dict=True,

torch\_dtype=torch.float16,

device\_map=device\_map,

)

Deletes the current model and trainer instances, clears GPU memory, and loads the fine-tuned model.

**Merging and Saving the Final Model**

Merges the LoRA-adapted model with the base model and saves the final model and tokenizer.

merged\_model = model.merge\_and\_unload()

merged\_model.save\_pretrained(f"{finetuned\_path}/llama3-8b\_bvr\_v2", safe\_serialization=True)

tokenizer.save\_pretrained(f"{finetuned\_path}/llama3-8b\_bvr\_v2")

Merges the LoRA-adapted model with the base model and saves the final model and tokenizer.