Deep Learning CS 6953 - Spring 2025

Esteban D Lopez
Shruti Karkamar
Steven Granaturov

Project 2

AI Disclaimer: OpenAI's and Google's models have been consulted for this assignment for: simple explanations of roberta and lora architectures, understand process conceptually and in simple terms, and to optimize and review code for errors.

Starter Notebook

Install and import required libraries

```
!pip install transformers datasets evaluate accelerate peft trl bitsandbytes
!pip install nvidia-ml-py3
```

```
Requirement already satisfied: transformers in /usr/local/lib/python3.11/dist-packages (4.51.3)
Collecting datasets
  Downloading datasets-3.5.0-py3-none-any.whl.metadata (19 kB)
Collecting evaluate
  Downloading evaluate-0.4.3-py3-none-any.whl.metadata (9.2 kB)
Requirement already satisfied: accelerate in /usr/local/lib/python3.11/dist-packages (1.5.2)
Requirement already satisfied: peft in /usr/local/lib/python3.11/dist-packages (0.14.0)
Collecting trl
  Downloading trl-0.16.1-py3-none-any.whl.metadata (12 kB)
Collecting bitsandbytes
  Downloading bitsandbytes-0.45.5-py3-none-manylinux_2_24_x86_64.whl.metadata (5.0 kB)
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages (from transformers) (3.18.0)
Requirement already satisfied: huggingface-hub<1.0,>=0.30.0 in /usr/local/lib/python3.11/dist-packages (from transformers) (0.30.2)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.11/dist-packages (from transformers) (2.0.2)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from transformers) (24.2)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.11/dist-packages (from transformers) (6.0.2)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.11/dist-packages (from transformers) (2024.11.6)
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from transformers) (2.32.3)
Requirement already satisfied: tokenizers<0.22,>=0.21 in /usr/local/lib/python3.11/dist-packages (from transformers) (0.21.1)
Requirement already satisfied: safetensors>=0.4.3 in /usr/local/lib/python3.11/dist-packages (from transformers) (0.5.3)
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.11/dist-packages (from transformers) (4.67.1)
Requirement already satisfied: pyarrow>=15.0.0 in /usr/local/lib/python3.11/dist-packages (from datasets) (18.1.0)
Collecting dill<0.3.9,>=0.3.0 (from datasets)
  Downloading dill-0.3.8-py3-none-any.whl.metadata (10 kB)
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages (from datasets) (2.2.2)
Collecting xxhash (from datasets)
  Downloading xxhash-3.5.0-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (12 kB)
Collecting multiprocess<0.70.17 (from datasets)
  Downloading multiprocess-0.70.16-py311-none-any.whl.metadata (7.2 kB)
Collecting fsspec<=2024.12.0,>=2023.1.0 (from fsspec[http]<=2024.12.0,>=2023.1.0->datasets)
  Downloading fsspec-2024.12.0-py3-none-any.whl.metadata (11 kB)
Requirement already satisfied: aiohttp in /usr/local/lib/python3.11/dist-packages (from datasets) (3.11.15)
Requirement already satisfied: psutil in /usr/local/lib/python3.11/dist-packages (from accelerate) (5.9.5)
Requirement already satisfied: torch>=2.0.0 in /usr/local/lib/python3.11/dist-packages (from accelerate) (2.6.0+cu124)
Requirement already satisfied: rich in /usr/local/lib/python3.11/dist-packages (from trl) (13.9.4)
Requirement already satisfied: aiohappyeyeballs>=2.3.0 in /usr/local/lib/python3.11/dist-packages (from aiohttp->datasets) (2.6.1)
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.11/dist-packages (from aiohttp->datasets) (1.3.2)
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.11/dist-packages (from aiohttp->datasets) (25.3.0)
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.11/dist-packages (from aiohttp->datasets) (1.5.0)
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.11/dist-packages (from aiohttp->datasets) (6.4.3)
Requirement already satisfied: propcache>=0.2.0 in /usr/local/lib/python3.11/dist-packages (from aiohttp->datasets) (0.3.1)
Requirement already satisfied: yarl<2.0,>=1.17.0 in /usr/local/lib/python3.11/dist-packages (from aiohttp->datasets) (1.19.0)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub<1.0,>=0.30.0->transformers) (4.13.2)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests->transformers) (3.4.1)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests->transformers) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests->transformers) (2.3.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests->transformers) (2025.1.31)
Requirement already satisfied: networkx in /usr/local/lib/python3.11/dist-packages (from torch>=2.0.0->accelerate) (3.4.2)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.11/dist-packages (from torch>=2.0.0->accelerate) (3.1.6)
Collecting nvidia-cuda-nvrtc-cu12==12.4.127 (from torch>=2.0.0->accelerate)
  Downloading nvidia_cuda_nvrtc_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl.metadata (1.5 kB)
Collecting nvidia-cuda-runtime-cu12==12.4.127 (from torch>=2.0.0->accelerate)
  Downloading nvidia_cuda_runtime_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl.metadata (1.5 kB)
Collecting nvidia-cuda-cupti-cu12==12.4.127 (from torch>=2.0.0->accelerate)
  Downloading nvidia_cuda_cupti_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl.metadata (1.6 kB)
Collecting nvidia-cudnn-cu12==9.1.0.70 (from torch>=2.0.0->accelerate)
  Downloading nvidia_cudnn_cu12-9.1.0.70-py3-none-manylinux2014_x86_64.whl.metadata (1.6 kB)
```

```
import os
import pandas as pd
import torch
from transformers import RobertaModel, RobertaTokenizer, TrainingArguments, Trainer, DataCollatorWithPadding, RobertaForSequenceClassification
from peft import LoraConfig, get_peft_model, PeftModel
from datasets import load_dataset, Dataset, ClassLabel
import pickle
import torch
torch.backends.cudnn.benchmark = True
```

Load Tokenizer and Preprocess Data

```
base_model = 'roberta-base'

dataset = load_dataset('ag_news', split='train')
tokenizer = RobertaTokenizer.from_pretrained(base_model)

def preprocess(examples):
    tokenized = tokenizer(examples['text'], truncation=True, padding=True)
    return tokenized
```

```
tokenized_dataset = dataset.map(preprocess, batched=True,  remove_columns=["text"])
tokenized_dataset = tokenized_dataset.rename_column("label", "labels")
```

```
/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secret in your Google Colab and resta
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
README.md: 100%    8.07k/8.07k [00:00<00:00, 819kB/s]
train-00000-of-00001.parquet: 100%    18.6M/18.6M [00:00<00:00, 61.9MB/s]
test-00000-of-00001.parquet: 100%    1.23M/1.23M [00:00<00:00, 108MB/s]
Generating train split: 100%    120000/120000 [00:00<00:00, 388996.71 examples/s]
Generating test split: 100%    7600/7600 [00:00<00:00, 302981.75 examples/s]
tokenizer_config.json: 100%    25.0/25.0 [00:00<00:00, 2.68kB/s]
vocab.json: 100%    899k/899k [00:00<00:00, 9.19MB/s]
merges.txt: 100%    456k/456k [00:00<00:00, 31.4MB/s]
tokenizer.json: 100%    1.36M/1.36M [00:00<00:00, 34.1MB/s]
config.json: 100%    481/481 [00:00<00:00, 58.0kB/s]
Map: 100%    120000/120000 [00:59<00:00, 2081.93 examples/s]
```

```
# Extract the number of classess and their names
num_labels = dataset.features['label'].num_classes
class_names = dataset.features["label"].names
print(f"number of labels: {num_labels}")
print(f"the labels: {class_names}")

# Create an id2label mapping
# We will need this for our classifier.
id2label = {i: label for i, label in enumerate(class_names)}

data_collator = DataCollatorWithPadding(tokenizer=tokenizer, return_tensors="pt")
```

```
number of labels: 4
the labels: ['World', 'Sports', 'Business', 'Sci/Tech']
```

Load Pre-trained Model

Set up config for pretrained model and download it from hugging face

```
model = RobertaForSequenceClassification.from_pretrained(
    base_model,
    id2label=id2label)
model
```

```
Xet Storage is enabled for this repo, but the 'hf_xet' package is not installed. Falling back to regular HTTP download. For better performance, install the package with
WARNING:huggingface_hub.file_download:Xet Storage is enabled for this repo, but the 'hf_xet' package is not installed. Falling back to regular HTTP download. For better
model.safetensors: 100%    499M/499M [00:02<00:00, 269MB/s]
Some weights of RobertaForSequenceClassification were not initialized from the model checkpoint at roberta-base and are newly initialized: ['classifier.dense.bias', 'cl
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
RobertaForSequenceClassification(
  (roberta): RobertaModel(
    (embeddings): RobertaEmbeddings(
      (word_embeddings): Embedding(50265, 768, padding_idx=1)
      (position_embeddings): Embedding(514, 768, padding_idx=1)
      (token_type_embeddings): Embedding(1, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (encoder): RobertaEncoder(
      (layer): ModuleList(
        (0-11): 12 x RobertaLayer(
          (attention): RobertaAttention(
            (self): RobertaSdpaSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): RobertaSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): RobertaIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=True)
            (intermediate_act_fn): GELUActivation()
          )
          (output): RobertaOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
    )
  )
  (classifier): RobertaClassificationHead(
    (dense): Linear(in_features=768, out_features=768, bias=True)
    (dropout): Dropout(p=0.1, inplace=False)
    (out_proj): Linear(in_features=768, out_features=4, bias=True)
  )
)
```

Anything from here on can be modified

```
# Split the original training set
split_datasets = tokenized_dataset.train_test_split(test_size=640, seed=42)
train_dataset = split_datasets['train']
eval_dataset = split_datasets['test']
```

Setup LoRA Config

Setup PEFT config and get peft model for finetuning

We apply LoRA to efficiently fine-tune the model while significantly reducing the number of trainable parameters:

- **Rank (`r=7`)**: Balances model adaptability with limited complexity.
- **Alpha (`α=15`)**: Controls adjustment strength; chosen to moderately impact original weights.
- **Dropout (`0.09`)**: Helps prevent overfitting.
- **Target Modules (`query`, `value`, `key`)**: Crucial attention layers where fine-tuning is most beneficial.

```
# PEFT Config
peft_config = LoraConfig(
    r=7,
    lora_alpha=15,
    lora_dropout=0.09,
    bias='none',
    target_modules=['query', 'value','key'],
    task_type="SEQ_CLS",
)
```

We combine our LoRA configuration with the RoBERTa model to create a fine-tunable model (`peft_model`). This ensures only specified parts of the model (defined by LoRA) are trainable, aligning with our parameter budget.

```
peft_model = get_peft_model(model, peft_config)
for name, param in peft_model.named_parameters():
    if name.startswith("classifier"):        # both dense & out_proj
        param.requires_grad = False
peft_model
```

```
PeftModelForSequenceClassification(
  (base_model): LoraModel(
    (model): RobertaForSequenceClassification(
      (roberta): RobertaModel(
        (embeddings): RobertaEmbeddings(
          (word_embeddings): Embedding(50265, 768, padding_idx=1)
          (position_embeddings): Embedding(514, 768, padding_idx=1)
          (token_type_embeddings): Embedding(1, 768)
          (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
        (encoder): RobertaEncoder(
          (layer): ModuleList(
            (0-11): 12 x RobertaLayer(
              (attention): RobertaAttention(
                (self): RobertaSdpaSelfAttention(
                  (query): lora.Linear(
                    (base_layer): Linear(in_features=768, out_features=768, bias=True)
                    (lora_dropout): ModuleDict(
                      (default): Dropout(p=0.09, inplace=False)
                    )
                    (lora_A): ModuleDict(
                      (default): Linear(in_features=768, out_features=7, bias=False)
                    )
                    (lora_B): ModuleDict(
                      (default): Linear(in_features=7, out_features=768, bias=False)
                    )
                    (lora_embedding_A): ParameterDict()
                    (lora_embedding_B): ParameterDict()
                    (lora_magnitude_vector): ModuleDict()
                  )
                  (key): lora.Linear(
                    (base_layer): Linear(in_features=768, out_features=768, bias=True)
                    (lora_dropout): ModuleDict(
                      (default): Dropout(p=0.09, inplace=False)
                    )
                    (lora_A): ModuleDict(
                      (default): Linear(in_features=768, out_features=7, bias=False)
                    )
                    (lora_B): ModuleDict(
                      (default): Linear(in_features=7, out_features=768, bias=False)
                    )
                    (lora_embedding_A): ParameterDict()
                    (lora_embedding_B): ParameterDict()
                    (lora_magnitude_vector): ModuleDict()
                  )
                  (value): lora.Linear(
                    (base_layer): Linear(in_features=768, out_features=768, bias=True)
                    (lora_dropout): ModuleDict(
                      (default): Dropout(p=0.09, inplace=False)
                    )
                    (lora_A): ModuleDict(
                      (default): Linear(in_features=768, out_features=7, bias=False)
                    )
                    (lora_B): ModuleDict(
                      (default): Linear(in_features=7, out_features=768, bias=False)
                    )
                    (lora_embedding_A): ParameterDict()
```

To confirm our LoRA integration, we inspect and list out the model parameters explicitly set as trainable. This step ensures compliance with the project's parameter limit (≤1 million parameters).

```
print("Trainable parameters:")
for name, param in peft_model.named_parameters():
   if param.requires_grad:
      print(name)
```

```
Trainable parameters:
base_model.model.roberta.encoder.layer.0.attention.self.query.lora_A.default.weight
base_model.model.roberta.encoder.layer.0.attention.self.query.lora_B.default.weight
base_model.model.roberta.encoder.layer.0.attention.self.key.lora_A.default.weight
base_model.model.roberta.encoder.layer.0.attention.self.key.lora_B.default.weight
base_model.model.roberta.encoder.layer.0.attention.self.value.lora_A.default.weight
base_model.model.roberta.encoder.layer.0.attention.self.value.lora_B.default.weight
base_model.model.roberta.encoder.layer.1.attention.self.query.lora_A.default.weight
base_model.model.roberta.encoder.layer.1.attention.self.query.lora_B.default.weight
base_model.model.roberta.encoder.layer.1.attention.self.key.lora_A.default.weight
base_model.model.roberta.encoder.layer.1.attention.self.key.lora_B.default.weight
base_model.model.roberta.encoder.layer.1.attention.self.value.lora_A.default.weight
base_model.model.roberta.encoder.layer.1.attention.self.value.lora_B.default.weight
base_model.model.roberta.encoder.layer.2.attention.self.query.lora_A.default.weight
base_model.model.roberta.encoder.layer.2.attention.self.query.lora_B.default.weight
base_model.model.roberta.encoder.layer.2.attention.self.key.lora_A.default.weight
base_model.model.roberta.encoder.layer.2.attention.self.key.lora_B.default.weight
base_model.model.roberta.encoder.layer.2.attention.self.value.lora_A.default.weight
base_model.model.roberta.encoder.layer.2.attention.self.value.lora_B.default.weight
base_model.model.roberta.encoder.layer.3.attention.self.query.lora_A.default.weight
base_model.model.roberta.encoder.layer.3.attention.self.query.lora_B.default.weight
base_model.model.roberta.encoder.layer.3.attention.self.key.lora_A.default.weight
base_model.model.roberta.encoder.layer.3.attention.self.key.lora_B.default.weight
base_model.model.roberta.encoder.layer.3.attention.self.value.lora_A.default.weight
base_model.model.roberta.encoder.layer.3.attention.self.value.lora_B.default.weight
base_model.model.roberta.encoder.layer.4.attention.self.query.lora_A.default.weight
base_model.model.roberta.encoder.layer.4.attention.self.query.lora_B.default.weight
base_model.model.roberta.encoder.layer.4.attention.self.key.lora_A.default.weight
base_model.model.roberta.encoder.layer.4.attention.self.key.lora_B.default.weight
base_model.model.roberta.encoder.layer.4.attention.self.value.lora_A.default.weight
base_model.model.roberta.encoder.layer.4.attention.self.value.lora_B.default.weight
base_model.model.roberta.encoder.layer.5.attention.self.query.lora_A.default.weight
base_model.model.roberta.encoder.layer.5.attention.self.query.lora_B.default.weight
base_model.model.roberta.encoder.layer.5.attention.self.key.lora_A.default.weight
base_model.model.roberta.encoder.layer.5.attention.self.key.lora_B.default.weight
base_model.model.roberta.encoder.layer.5.attention.self.value.lora_A.default.weight
base_model.model.roberta.encoder.layer.5.attention.self.value.lora_B.default.weight
base_model.model.roberta.encoder.layer.6.attention.self.query.lora_A.default.weight
base_model.model.roberta.encoder.layer.6.attention.self.query.lora_B.default.weight
base_model.model.roberta.encoder.layer.6.attention.self.key.lora_A.default.weight
base_model.model.roberta.encoder.layer.6.attention.self.key.lora_B.default.weight
base_model.model.roberta.encoder.layer.6.attention.self.value.lora_A.default.weight
base_model.model.roberta.encoder.layer.6.attention.self.value.lora_B.default.weight
base_model.model.roberta.encoder.layer.7.attention.self.query.lora_A.default.weight
base_model.model.roberta.encoder.layer.7.attention.self.query.lora_B.default.weight
base_model.model.roberta.encoder.layer.7.attention.self.key.lora_A.default.weight
base_model.model.roberta.encoder.layer.7.attention.self.key.lora_B.default.weight
base_model.model.roberta.encoder.layer.7.attention.self.value.lora_A.default.weight
base_model.model.roberta.encoder.layer.7.attention.self.value.lora_B.default.weight
base_model.model.roberta.encoder.layer.8.attention.self.query.lora_A.default.weight
base_model.model.roberta.encoder.layer.8.attention.self.query.lora_B.default.weight
base_model.model.roberta.encoder.layer.8.attention.self.key.lora_A.default.weight
base_model.model.roberta.encoder.layer.8.attention.self.key.lora_B.default.weight
base_model.model.roberta.encoder.layer.8.attention.self.value.lora_A.default.weight
base_model.model.roberta.encoder.layer.8.attention.self.value.lora_B.default.weight
base_model.model.roberta.encoder.layer.9.attention.self.query.lora_A.default.weight
base_model.model.roberta.encoder.layer.9.attention.self.query.lora_B.default.weight
base_model.model.roberta.encoder.layer.9.attention.self.key.lora_A.default.weight
```

We print the total number of trainable parameters and their percentage compared to the full model, verifying our adherence to the project constraint (≤1 million parameters).

```
print('PEFT Model')
peft_model.print_trainable_parameters()
```

```
PEFT Model
trainable params: 980,740 || all params: 125,629,448 || trainable%: 0.7807
```

Training Setup

We define accuracy as our primary evaluation metric since the task (text classification) emphasizes correctness of predictions. This metric guides our model training, selection, and hyperparameter tuning decisions.

```
# To track evaluation accuracy during training
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

def compute_metrics(pred):
    labels = pred.label_ids
    preds = pred.predictions.argmax(-1)
    # Calculate accuracy
    accuracy = accuracy_score(labels, preds)
    return {
        'accuracy': accuracy
    }
```

We set detailed training parameters carefully chosen through preliminary experimentation:

- **Learning Rate:** Initial rate of 3e-5, typical for transformer fine-tuning.
- **Epochs:** Set at 6 for effective yet efficient training.
- **Batch Sizes:** Moderate sizes (train: 256, eval: 128) to make the most out of access to A100 compute power.
- **Mixed Precision (bf16=True):** Balances training speed with memory efficiency with A100.
- **Optimizer:** AdamW optimizer with weight decay (0.01) to minimize overfitting.
- **Scheduler:** Linear learning rate scheduler with warmup to improve training stability.

```
# Setup Training args
output_dir = "results"
training_args = TrainingArguments(
    output_dir=output_dir,
    report_to=None,
    eval_steps=200,
    eval_strategy='steps',
    save_strategy='steps',
    save_steps=200,
    logging_steps=100,
    learning_rate=3e-5,
    num_train_epochs=6,
    use_cpu=False,
    dataloader_num_workers=12,
    per_device_train_batch_size=256,
    per_device_eval_batch_size=128,
    optim="adamw_torch",
    weight_decay = 0.01, #regularization
    fp16 = False, #mixed precision
    bf16=True,
    lr_scheduler_type="linear", #linear decay with warmup
    warmup_steps = 500, #warmup
    gradient_checkpointing=False,
    gradient_checkpointing_kwargs={'use_reentrant':True},
    load_best_model_at_end=True,
    metric_for_best_model="accuracy",
    greater_is_better=True
)

def get_trainer(model):
      return  Trainer(
          model=model,
          args=training_args,
          compute_metrics=compute_metrics,
          train_dataset=train_dataset,
          tokenizer=tokenizer,
          eval_dataset=eval_dataset,
          data_collator=data_collator,
      )
```

Start Training

We perform a controlled hyperparameter sweep over multiple learning rates (1e-5, 3e-5, 5e-5). By systematically testing different learning rates, we identify the most effective configuration for model convergence and accuracy improvement.

```
#Hyperparameter sweep over learning rates
for lr in [1e-5, 3e-5, 5e-5]:
    # update the LR in your TrainingArguments object
    training_args.learning_rate = lr

    # re-instantiate a fresh Trainer with that new LR
    trainer = get_trainer(peft_model)

    print(f"\n=== Training with learning_rate = {lr} ===")
    trainer.train()
    print("Done.\n")
```

```
<ipython-input-28-4ee0f6f391df>:31: FutureWarning: `tokenizer` is deprecated and will be removed in version 5.0.0 for `Trainer.__init__`. Use `processing_class` instead
  return Trainer(
No label_names provided for model class `PeftModelForSequenceClassification`. Since `PeftModel` hides base models input arguments, if label_names is not given, label_na

=== Training with learning_rate = 1e-05 ===
wandb: Logging into wandb.ai. (Learn how to deploy a W&B server locally: https://wandb.me/wandb-server)
wandb: You can find your API key in your browser here: https://wandb.ai/authorize
wandb: Paste an API key from your profile and hit enter: ··········
wandb: WARNING If you're specifying your api key in code, ensure this code is not shared publicly.
wandb: WARNING Consider setting the WANDB_API_KEY environment variable, or running `wandb login` from the command line.
wandb: No netrc file found, creating one.
wandb: Appending key for api.wandb.ai to your netrc file: /root/.netrc
wandb: Currently logged in as: edl9434 (edl9434-nyu) to https://api.wandb.ai. Use `wandb login --relogin` to force relogin
```

Tracking run with wandb version 0.19.9
Run data is saved locally in `/content/wandb/run-20250418_171848-fbs2q9mt`
Syncing run **results** to Weights & Biases (docs)
View project at https://wandb.ai/edl9434-nyu/huggingface
View run at https://wandb.ai/edl9434-nyu/huggingface/runs/fbs2q9mt
[2802/2802 26:47, Epoch 6/6]

Step	Training Loss	Validation Loss	Accuracy
200	1.381500	1.375867	0.439063
400	1.324900	1.262500	0.864062
600	0.431200	0.322010	0.898438
800	0.310600	0.307428	0.904687
1000	0.304000	0.298131	0.906250
1200	0.283700	0.297219	0.909375
1400	0.276300	0.294465	0.910937
1600	0.278400	0.293399	0.907813
1800	0.269000	0.288782	0.909375
2000	0.270100	0.288606	0.909375
2200	0.267200	0.288691	0.910937
2400	0.272100	0.287556	0.910937
2600	0.266800	0.286849	0.907813
2800	0.272700	0.286880	0.907813

```
<ipython-input-28-4ee0f6f391df>:31: FutureWarning: `tokenizer` is deprecated and will be removed in version 5.0.0 for `Trainer.__init__`. Use `processing_class` instead
  return Trainer(
No label_names provided for model class `PeftModelForSequenceClassification`. Since `PeftModel` hides base models input arguments, if label_names is not given, label_na
Done.


=== Training with learning_rate = 3e-05 ===
```

[2802/2802 26:40, Epoch 6/6]

Step	Training Loss	Validation Loss	Accuracy
200	0.277700	0.290244	0.910937
400	0.267400	0.288532	0.907813
600	0.261300	0.285434	0.910937
800	[illegible]	[illegible]	0.907813
1000	0.253300	0.275518	0.910937
1200	0.242200	0.269236	0.907813
1400	0.235700	0.267763	0.909375

Evaluate Finetuned Model

Performing Inference on Custom Input

We create an easy-to-use function (`classify`) for making predictions on arbitrary text inputs. This function helps qualitatively verify that our model predictions align with expectations.

```
def classify(model, tokenizer, text):
     device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
     inputs = tokenizer(text, truncation=True, padding=True, return_tensors="pt").to(device)
     output = model(**inputs)

     prediction = output.logits.argmax(dim=-1).item()

     print(f'\n Class: {prediction}, Label: {id2label[prediction]}, Text: {text}')
     return id2label[prediction]
```

```
classify( peft_model, tokenizer, "Kederis proclaims innocence Olympic champion Kostas Kederis today left hospital ahead of his date with IOC inquisitors claiming his ...")
classify( peft_model, tokenizer, "Wall St. Bears Claw Back Into the Black (Reuters) Reuters - Short-sellers, Wall Street's dwindling\band of ultra-cynics, are seeing green
```

```
No label_names provided for model class `PeftModelForSequenceClassification`. Since `PeftModel` hides base models input arguments, if label_names is not given, label_na
Done.
 Class: 0, Label: World, Text: Kederis proclaims innocence Olympic champion Kostas Kederis today left hospital ahead of his date with IOC inquisitors claiming his ...

 Class: 2, Label: Business, Text: Wall St. Bears Claw Back Into the Black (Reuters) Reuters - Short-sellers, Wall Street's dwindlinand of ultra-cynics, are seeing green
'Business'
=== Training with learning_rate = 5e-05 ===
```

Step	Training Loss	Validation Loss	Accuracy
200	0.271800	0.287228	0.910937
400	0.261400	0.284725	0.909375
600	[illegible]	[illegible]	[illegible]
800	0.241500	0.275224	0.907813

Run inference on eval_dataset

We define `evaluate_model()` to run inference on our entire evaluation dataset efficiently. This function computes accuracy systematically across all evaluation samples, providing robust performance validation.

```
from torch.utils.data import DataLoader
import evaluate
from tqdm import tqdm

def evaluate_model(inference_model, dataset, labelled=True, batch_size=128, data_collator=None)
    """
    Evaluate a PEFT model on a dataset.

    Args:
        inference_model: The model to evaluate.
```

```
        dataset: The dataset (Hugging Face Dataset) to run inference on.
        labelled (bool): If True, the dataset includes labels and metrics will be computed.
                         If False, only predictions will be returned.
        batch_size (int): Batch size for inference.
        data_collator: Function to collate batches. If None, the default collate_fn is used.

    Returns:
        If labelled is True, returns a tuple (metrics, predictions)
        If labelled is False, returns the predictions.
    """
    # Create the DataLoader
    eval_dataloader = DataLoader(dataset, batch_size=batch_size, collate_fn=data_collator)
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

    inference_model.to(device)
    inference_model.eval()

    all_predictions = []
    if labelled:
        metric = evaluate.load('accuracy')

    # Loop over the DataLoader
    for batch in tqdm(eval_dataloader):
        # Move each tensor in the batch to the device
        batch = {k: v.to(device) for k, v in batch.items()}
        with torch.no_grad():
            outputs = inference_model(**batch)
        predictions = outputs.logits.argmax(dim=-1)
        all_predictions.append(predictions.cpu())

        if labelled:
            # Expecting that labels are provided under the "labels" key.
            references = batch["labels"]
            metric.add_batch(
                predictions=predictions.cpu().numpy(),
                references=references.cpu().numpy()
            )

    # Concatenate predictions from all batches
    all_predictions = torch.cat(all_predictions, dim=0)

    if labelled:
        eval_metric = metric.compute()
        print("Evaluation Metric:", eval_metric)
        return eval_metric, all_predictions
    else:
```

Evaluate Final Model Performance

We run the full evaluation function on our evaluation dataset to get an accurate and comprehensive measurement of our model's performance. This step ensures our results meet or exceed the desired baseline accuracy (≥80%).

```
# Check evaluation accuracy
_, _ = evaluate_model(peft_model, eval_dataset, True, 128, data_collator)
```

```
Downloading builder script: 100%                    4.20k/4.20k [00:00<00:00, 485kB/s]
100%|██████████| 5/5 [00:00<00:00,  6.89it/s]Evaluation Metric: {'accuracy': 0.93125}
```

Run Inference on unlabelled dataset

We load additional unlabeled data (`test_unlabelled.pkl`) provided for final predictions. This data undergoes the same preprocessing pipeline as the training data to ensure consistent input formatting.

```
#Load your unlabelled data
unlabelled_dataset = pd.read_pickle("test_unlabelled.pkl")
test_dataset = unlabelled_dataset.map(preprocess, batched=True, remove_columns=["text"])
unlabelled_dataset
```