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) (
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) (
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.
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
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests->transform
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) (
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests->transformers) (
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)
Collecting nvidia-cublas-cu12==12.4.5.8 (from torch>=2.0.0->accelerate)
```

```
import os
import pandas as pd
import torch
from transformers import RobertaModel, RobertaTokenizer, TrainingArguments, Trainer, DataCollatorWithPadding, RobertaForSequence
from peft import LoraConfig, get_peft_model, PeftModel
from datasets import load_dataset, Dataset, ClassLabel
import pickle
```

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
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, 810kB/s]
train-00000-of-00001.parquet: 100%	18.6M/18.6M [00:00<00:00, 65.4MB/s]
test-00000-of-00001.parquet: 100%	1.23M/1.23M [00:00<00:00, 101MB/s]
Generating train split: 100%	120000/120000 [00:00<00:00, 374434.91 examples/s]
Generating test split: 100%	7600/7600 [00:00<00:00, 301571.50 examples/s]
tokenizer_config.json: 100%	25.0/25.0 [00:00<00:00, 2.90kB/s]
vocab.json: 100%	899k/899k [00:00<00:00, 3.61MB/s]
merges.txt: 100%	456k/456k [00:00<00:00, 36.1MB/s]
tokenizer.json: 100%	1.36M/1.36M [00:00<00:00, 4.10MB/s]
config.json: 100%	481/481 [00:00<00:00, 56.2kB/s]
Map: 100%	120000/120000 [00:56<00:00, 2209.82 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
WARNING:huggingface_hub.file_download:Xet Storage is enabled for this repo, but the 'hf_xet' package is not installed. Falli
```

model.safetensors: 100% 499M/499M [00:01<00:00, 368MB/s]

```
Some weights of RobertaForSequenceClassification were not initialized from the model checkpoint at roberta-base and are newl
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)
  )
)
```

LoRA Configuration

LoraConfig (TaskType.SEQ_CLS) that injects low-rank (r=6) adapter matrices with scaling α=6 and dropout=0.1 into each attention “query,” “key,” and “value” projection, leaving all original weights frozen.

Printing the Trainable Parameters

```
from peft import get_peft_model, LoraConfig, TaskType

lora_config = LoraConfig(
    task_type=TaskType.SEQ_CLS,
    r=6,
    lora_alpha=6,
    lora_dropout=0.1,
    target_modules=["query", "key", "value"],
    bias="none",
)

model = get_peft_model(model, lora_config)
model.print_trainable_parameters()
```

```
trainable params: 925,444 || all params: 125,574,152 || trainable%: 0.7370
```

Dataset

Importing the dataset and splitting it into train and test

tokenize_fn to truncate/pad each text to a max length of 256 tokens, and applies it in batched fashion to both train and validation sets.

Dynamic padding batch-pad sequences on the fly during training/evaluation.

```
from datasets import load_dataset

dataset = load_dataset("ag_news")

split_dataset = dataset["train"].train_test_split(test_size=0.1, seed=42)
train_dataset = split_dataset["train"]
val_dataset = split_dataset["test"]

def tokenize_fn(examples):
    return tokenizer(examples["text"], truncation=True, max_length=256)

train_dataset = train_dataset.map(tokenize_fn, batched=True)
val_dataset = val_dataset.map(tokenize_fn, batched=True)

train_dataset.set_format(type="torch", columns=["input_ids", "attention_mask", "label"])
val_dataset.set_format(type="torch", columns=["input_ids", "attention_mask", "label"])

from transformers import DataCollatorWithPadding
data_collator = DataCollatorWithPadding(tokenizer=tokenizer)
```

```
Map: 100%  108000/108000 [00:48<00:00, 2365.60 examples/s]
Map: 100%  12000/12000 [00:06<00:00, 2082.34 examples/s]
```

Hyper parameter

loading the “accuracy” metric and defined compute_metrics to convert raw model logits into predicted classes (argmax) and compute accuracy against true labels.

save outputs in ./results,

run evaluation every logging_steps=100 training steps (eval_strategy="steps"),

limit training to max_steps=1600 (with num_train_epochs=1 as a fallback),

use AdamW (optim="adamw_torch") at lr=1e-5,

batch sizes of 16/64 for train/eval,

and automatically reload the checkpoint with the lowest eval_loss.

Trainer that ties together your LoRA-wrapped model, the training args, the compute_metrics function, your tokenized train/validation datasets, and the padding collator—ready to call .train() and .evaluate()

```
import numpy as np
import evaluate
from transformers import TrainingArguments, Trainer, EarlyStoppingCallback

accuracy_metric = evaluate.load("accuracy")

def compute_metrics(eval_pred):
    logits, labels = eval_pred
    predictions = np.argmax(logits, axis=1)
    return accuracy_metric.compute(predictions=predictions, references=labels)

training_args = TrainingArguments(
    output_dir="./results",
    report_to=None,
    eval_strategy="steps",
    logging_steps=100,
    learning_rate=1e-5,
    max_steps=1600,
```

```
    num_train_epochs=1,
    use_cpu=False,
    dataloader_num_workers=4,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=64,
    optim="adamw_torch",
    gradient_checkpointing=False,
    gradient_checkpointing_kwargs={'use_reentrant': True},
    load_best_model_at_end=True,
    metric_for_best_model="eval_loss",
    greater_is_better=False
)

trainer = Trainer(
    model=model,
    args=training_args,
    compute_metrics=compute_metrics,
    train_dataset=train_dataset,
    eval_dataset=val_dataset,
    data_collator=data_collator,

)
```

```
Downloading builder script: 100%                    4.20k/4.20k [00:00<00:00, 513kB/s]
No label_names provided for model class `PeftModelForSequenceClassification`. Since `PeftModel` hides base models input argu
```

Training the Roberta base model

```
trainer.train()

eval_results = trainer.evaluate()
print("Validation results:", eval_results)
```

```
wandb: WARNING The `run_name` is currently set to the same value as `TrainingArguments.output_dir`. If this was not intended
wandb: Using wandb-core as the SDK backend.  Please refer to https://wandb.me/wandb-core for more information.
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: ppm7517 (ppm7517-new-york-university) to https://api.wandb.ai. Use `wandb login --relogin` to
Tracking run with wandb version 0.19.9
Run data is saved locally in /content/wandb/run-20250421_202939-1byevbl7
Syncing run ./results to Weights & Biases (docs)
View project at https://wandb.ai/ppm7517-new-york-university/huggingface
View run at https://wandb.ai/ppm7517-new-york-university/huggingface/runs/1byevbl7
[1600/1600 26:31, Epoch 0/1]
```

Step	Training Loss	Validation Loss	Accuracy
100	1.387000	1.380269	0.260083
200	1.377100	1.371749	0.271500
300	1.368900	1.361618	0.567333
400	1.356600	1.349600	0.601667
500	1.342100	1.332246	0.730250
600	1.314800	1.304521	0.808917
700	1.285500	1.264468	0.872083
800	1.230400	1.202307	0.875250
900	1.148700	1.104583	0.870500
1000	1.031400	0.973451	0.868333
1100	0.894400	0.815740	0.875333
1200	0.778700	0.699303	0.879083
1300	0.698800	0.627558	0.881083
1400	0.661100	0.587032	0.881667
1500	0.615000	0.565399	0.881667
1600	0.608200	0.558541	0.881750

```
[188/188 01:20]
Validation results: {'eval_loss': 0.5585409998893738, 'eval_accuracy': 0.88175, 'eval_runtime': 80.9434, 'eval_samples_per_s
```

Running an inference on the test_unlabelled.pkl file to and creating the submissions.csv file to upload on kaggle

```
import pickle
import pandas as pd
from datasets import Dataset

test_data = pickle.load(open("test_unlabelled.pkl", "rb"))

test_df = pd.DataFrame(test_data)
print("Sample of the Unlabeled Test Set:")
print(test_df.head())

test_dataset = Dataset.from_pandas(test_df)

def tokenize_fn_unlabelled(examples):
    return tokenizer(examples["text"], truncation=True, max_length=256)

test_dataset = test_dataset.map(tokenize_fn_unlabelled, batched=True)
test_dataset.set_format(type="torch", columns=["input_ids", "attention_mask"])

predictions_output = trainer.predict(test_dataset)
logits = predictions_output.predictions
predictions = np.argmax(logits, axis=-1)
```

```
if "id" in test_df.columns:
    ids = test_df["id"]
elif "ID" in test_df.columns:
    ids = test_df["ID"]
else:
    ids = list(range(len(test_df)))

submission_df = pd.DataFrame({
    "ID": ids,
    "Label": predictions
})

submission_filename = "submission.csv"
submission_df.to_csv(submission_filename, index=False)
print(f"Submission file saved as {submission_filename}")
```

```
Sample of the Unlabeled Test Set:
                                                text
0  Remains of New Species of Hobbit-Sized Human F...
1  Iran to cease negotiations with EU in case of ...
2  Israel levels new accusations against Syria Wi...
3  Enevo a Silicon Valley startup create self-pow...
4  NBA owners have imposed a luxury tax change on...
Map: 100%                                   8000/8000 [00:04<00:00, 1747.91 examples/s]
Submission file saved as submission.csv
```

Checking the unlabbeled pkl file to check if the unpickling is done correctly

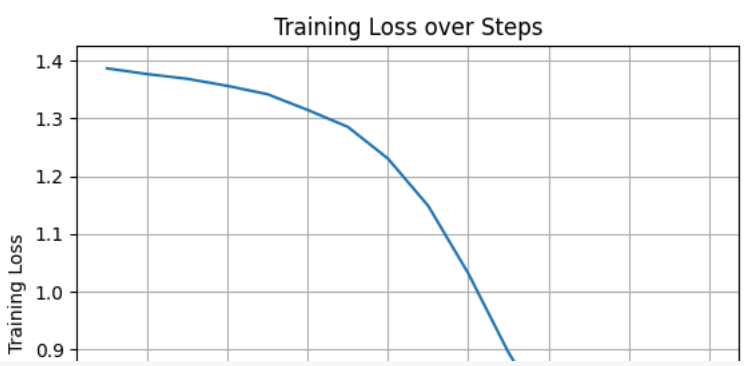
```
import pickle

pickle_filename = "/content/test_unlabelled.pkl"

with open(pickle_filename, "rb") as file:
    data = pickle.load(file)

print("Dataset information:")
print(data)

print("\nFirst few examples:")

num_examples_to_display = 5
for i in range(num_examples_to_display):
    print(f"Example {i+1}:")
    print(data[i])
    print("------")

import pandas as pd
try:
    df = pd.DataFrame(data)
    print("\nData preview using pandas:")
    print(df.head())
except Exception as e:
    print("\nCould not convert data to a DataFrame:", e)
```

```
Dataset information:
Dataset({
    features: ['text'],
    num_rows: 8000
})

First few examples:
Example 1:
{'text': 'Remains of New Species of Hobbit-Sized Human Found Scientists in Australia have found a new species of hobbit-size
------
Example 2:
{'text': 'Iran to cease negotiations with EU in case of dead end A top Iranian official said Sunday that Iran would withdraw
------
Example 3:
{'text': 'Israel levels new accusations against Syria Without acknowledging responsibility for the car-bombing death of a Ha
------
Example 4:
{'text': 'Enevo a Silicon Valley startup create self-powered battery and another new company building project creating low-C
------
```

```
Example 5:
{'text': "NBA owners have imposed a luxury tax change on US based player draft stocks in talks to buy European buy-up under
------

Data preview using pandas:
                                                text
0  Remains of New Species of Hobbit-Sized Human F...
1  Iran to cease negotiations with EU in case of ...
2  Israel levels new accusations against Syria Wi...
3  Enevo a Silicon Valley startup create self-pow...
4  NBA owners have imposed a luxury tax change on...
```

Plotting the Graphs

```
import matplotlib.pyplot as plt

history = trainer.state.log_history

train_steps = [e["step"] for e in history if "loss" in e]
train_loss  = [e["loss"] for e in history if "loss" in e]

plt.figure()
plt.plot(train_steps, train_loss)
plt.xlabel("Training Step")
plt.ylabel("Training Loss")
plt.title("Training Loss over Steps")
plt.grid(True)
plt.show()

eval_steps = [e["step"]          for e in history if "eval_accuracy" in e]
eval_acc   = [e["eval_accuracy"] for e in history if "eval_accuracy" in e]

plt.figure()
plt.plot(eval_steps, eval_acc)
plt.xlabel("Training Step")
plt.ylabel("Validation Accuracy")
plt.title("Validation Accuracy over Steps")
plt.grid(True)
plt.show()
```




```
import matplotlib.pyplot as plt

history = trainer.state.log_history

train_entries = [e for e in history if "loss" in e and "eval_loss" not in e]
train_steps   = [e["step"] for e in train_entries]
train_loss    = [e["loss"] for e in train_entries]

eval_entries  = [e for e in history if "eval_loss" in e]
eval_steps    = [e["step"] for e in eval_entries]
eval_loss     = [e["eval_loss"] for e in eval_entries]

plt.figure()
plt.plot(train_steps, train_loss, label="Train Loss")
plt.plot(eval_steps,  eval_loss,  label="Validation Loss")
plt.xlabel("Training Step")
plt.ylabel("Loss")
plt.title("Training vs. Validation Loss")
```