Assignment 3 Part 2: Fine-Tuning Theory and Practice - Practical Fine-Tuning Session

By Ajay Sethuraman

Tasks:

Fine-tune the distilbert-base-uncased model for text classification using Hugging Face, as demonstrated in the lesson. Complete the following steps:

- Environment Setup: Write the commands to install the required libraries and verify GPU availability.
- Preprocessing Data: Demonstrate how to load and preprocess the IMDB dataset for tokenization.
- Model Training: Define the training arguments and use Hugging Face's Trainer to fine-tune the model.
- Save and Evaluate: Save the fine-tuned model and evaluate its accuracy on the test set.

```
# Step 1: Setting Up the Environment
!pip install torch tensorflow transformers datasets scikit-learn

import torch
print("GPU Available:", torch.cuda.is_available())
```

```
# Step 2: Load the Pre-Trained Model
from transformers import AutoModelForSequenceClassification, AutoTokenizer

model_name = "distilbert-base-uncased"
model = AutoModelForSequenceClassification.from_pretrained(model_name, num_labels=2)
tokenizer = AutoTokenizer.from_pretrained(model_name)
```

```
/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 secre
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(
config.json: 100%  483/483 [00:00<00:00, 37.4kB/s]
model.safetensors: 100%  268M/268M [00:01<00:00, 224MB/s]
Some weights of DistilBertForSequenceClassification were not initialized from the model checkpoint at distilbert-base-uncased and are ne
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
tokenizer_config.json: 100%  48.0/48.0 [00:00<00:00, 3.80kB/s]
vocab.txt: 100%  232k/232k [00:00<00:00, 2.88MB/s]
tokenizer.json: 100%  466k/466k [00:00<00:00, 5.86MB/s]
```

```
# Step 3: Prepare the Dataset
from datasets import load_dataset

dataset = load_dataset("imdb")

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

tokenized_dataset = dataset.map(preprocess_function, batched=True)
```

```
README.md: 100%  7.81k/7.81k [00:00<00:00, 811kB/s]
train-00000-of-00001.parquet: 100%  21.0M/21.0M [00:00<00:00, 196MB/s]
test-00000-of-00001.parquet: 100%  20.5M/20.5M [00:00<00:00, 211MB/s]
unsupervised-00000-of-00001.parquet: 100%  42.0M/42.0M [00:00<00:00, 193MB/s]
Generating train split: 100%  25000/25000 [00:00<00:00, 106991.43 examples/s]
Generating test split: 100%  25000/25000 [00:00<00:00, 109100.84 examples/s]
Generating unsupervised split: 100%  50000/50000 [00:00<00:00, 105986.76 examples/s]
Map: 100%  25000/25000 [00:24<00:00, 960.74 examples/s]
Map: 100%  25000/25000 [00:22<00:00, 1055.16 examples/s]
Map: 100%  50000/50000 [00:48<00:00, 1076.37 examples/s]
```

```
# Step 4: Set Up the Trainer
from transformers import Trainer, TrainingArguments

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

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_dataset["train"],
```

```
    eval_dataset=tokenized_dataset["test"],
)
```

```
/usr/local/lib/python3.11/dist-packages/transformers/training_args.py:1575: FutureWarning: `evaluation_strategy` is deprecated and will
  warnings.warn(
```

```
# Step 5: Fine-Tune the Model
trainer.train()

# Step 6: Save the Fine-Tuned Model
model.save_pretrained("./fine_tuned_model")
tokenizer.save_pretrained("./fine_tuned_model")

# Step 7: Evaluate the Fine-Tuned Model
results = trainer.evaluate()
print("Evaluation Results:", results)
```

```
wandb: WARNING The `run_name` is currently set to the same value as `TrainingArguments.output_dir`. If this was not intended, please spe
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: ajaykrsnaa (ajaykrsnaa-cognizant) to https://api.wandb.ai. Use `wandb login --relogin` to force relogin
```

Tracking run with wandb version 0.19.8
Run data is saved locally in `/content/wandb/run-20250314_211859-jam2e2iu`
Syncing run **./results** to Weights & Biases (docs)
View project at https://wandb.ai/ajaykrsnaa-cognizant/huggingface
View run at https://wandb.ai/ajaykrsnaa-cognizant/huggingface/runs/jam2e2iu
[4689/4689 1:20:26, Epoch 3/3]

Epoch	Training Loss	Validation Loss
1	0.228400	0.221130
2	0.154200	0.239806
3	0.091500	0.282230

[3125/3125 06:16]

```
Evaluation Results: {'eval_loss': 0.2822296917438507, 'eval_runtime': 376.6907, 'eval_samples_per_second': 66.367, 'eval_steps_per_secon
```

```
# Step 8: Detailed Performance Metrics
from sklearn.metrics import classification_report

predictions = trainer.predict(tokenized_dataset["test"])
y_pred = predictions.predictions.argmax(axis=1)
y_true = tokenized_dataset["test"]["label"]
print(classification_report(y_true, y_pred))

```

```
              precision    recall  f1-score   support

           0       0.94      0.92      0.93     12500
           1       0.92      0.94      0.93     12500

    accuracy                           0.93     25000
   macro avg       0.93      0.93      0.93     25000
weighted avg       0.93      0.93      0.93     25000
```

Challenges Faced:

- Limited computational resources made training slower without GPU acceleration.
- Handling large datasets required memory-efficient tokenization and batch processing.
- Hyperparameter tuning required multiple iterations to improve accuracy.

Potential Improvements:

- Using a larger dataset or domain-specific data could enhance performance.

- Adjusting learning rates and epochs could optimize model convergence.
- Exploring alternative architectures like RoBERTa or GPT-based models for better accuracy.