fine_tuning_BERT

June 23, 2025

1 Fine Tuning BERT For Paraphrase Classification

Fine tuning BERT for paraphrase classification using the Microsoft Research Paraphrase Classification dataset.

2 Part 1: Fine-Tuning BERT

[1]:
```
import torch
import numpy as np
from datasets import load_dataset
from transformers import AutoModelForSequenceClassification, AutoTokenizer,␣
 ↪Trainer, TrainingArguments
import evaluate

# Check if MPS is available and set the device
if torch.backends.mps.is_available():
    device = torch.device("mps")
    print("Using Apple Silicon (MPS) backend.")
elif torch.cuda.is_available():
    device = torch.device("cuda")
    print("Using NVIDIA CUDA backend.")
else:
    device = torch.device("cpu")
    print("Using CPU backend.")

# Load the full dataset dictionary (train and validation splits)
dataset_dict = load_dataset("nyu-mll/glue", "mrpc")

# Load the model and tokenizer
model = AutoModelForSequenceClassification.from_pretrained("bert-base-uncased",␣
 ↪num_labels=2).to(device)
tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")

# Tokenize the dataset, truncate and pad text
def encode(examples):
    return tokenizer(examples["sentence1"], examples["sentence2"],␣
 ↪truncation=True, padding="max_length")
```

```
tokenized_datasets = dataset_dict.map(encode, batched=True)

# Rename the label column to labels and remove unnecessary columns
tokenized_datasets = tokenized_datasets.map(lambda examples: {"labels":␣
 ↪examples["label"]}, batched=True)
tokenized_datasets = tokenized_datasets.remove_columns(["sentence1",␣
 ↪"sentence2", "idx", "label"])
tokenized_datasets.set_format("torch")

# Get the separate train and validation datasets
train_dataset = tokenized_datasets["train"]
eval_dataset = tokenized_datasets["validation"]

# Define the metric computation function
metric = evaluate.load("glue", "mrpc")

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

# Define Training Arguments
training_args = TrainingArguments(
    output_dir="./mrpc-results",       # Directory to save the model and results
    logging_dir='./mrpc-logs',         # Directory for logs
    num_train_epochs=5,                # Total number of training epochs
    per_device_train_batch_size=16,    # Batch size for training
    per_device_eval_batch_size=16,     # Batch size for evaluation

    # --- Settings for Tracking Metrics ---
    eval_strategy="epoch",             # Run evaluation at the end of each epoch
    logging_strategy="steps",          # Log metrics during training
    logging_steps=50,                  # Log training loss every 50 steps

    # --- Settings for Saving the Model ---
    save_strategy="epoch",             # Save a checkpoint at the end of each␣
 ↪epoch
    load_best_model_at_end=True,       # Load the best model found during␣
 ↪training
    metric_for_best_model="accuracy", # Use accuracy to determine the best model
)

# 3. Initialize the Trainer
trainer = Trainer(
    model=model,
    args=training_args,
```

```
    train_dataset=train_dataset,
    eval_dataset=eval_dataset,
    tokenizer=tokenizer,
    compute_metrics=compute_metrics,
)

# 4. Start Training
print("Starting training...")
trainer.train()
print("Training complete!")

# 5. Save the final best model
final_model_path = "./final_mrpc_model"
print(f"Saving the best model to {final_model_path}")
trainer.save_model(final_model_path)
print("Model saved successfully.")
```

```
Using NVIDIA CUDA backend.

Some weights of BertForSequenceClassification were not initialized from the
model checkpoint at bert-base-uncased and are newly initialized:
['classifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it
for predictions and inference.
/tmp/ipykernel_73705/801247998.py:68: FutureWarning: `tokenizer` is deprecated
and will be removed in version 5.0.0 for `Trainer.__init__`. Use
`processing_class` instead.
  trainer = Trainer(

Starting training…

<IPython.core.display.HTML object>

Training complete!
Saving the best model to ./final_mrpc_model
Model saved successfully.
```

3 Part 2: Debugging Issues

I ran into some small issues with properly installing and importing some of the required modules to train the model. These were quite trivial to solve however.

A larger issue that I encountered was with training time. Although I have a MacBook that is MPS compatible the training time for me was extremely slow and it was also causing issues with the performance of my machine outside of the training task. Fortunately I have a desktop with a CUDA enabled graphics card, so I was able to figure out how to ensure the training was completed using CUDA and my GPU which reduced my training time by 91.6%. This allowed me to complete more training epochs in a reasonable amount of time and reduce loss significantly from 11.3% to 1.5%.

4 Part 3: Evaluating the Model

We will evaluate the final model on the validation set provided with the dataset.

[4]:
```
eval_results = trainer.evaluate()

print("--- Evaluation Metrics ---")
print(f"Accuracy: {eval_results['eval_accuracy']:.4f}")
print(f"F1-Score: {eval_results['eval_f1']:.4f}")
print(f"Loss: {eval_results['eval_loss']:.4f}")
```

```
<IPython.core.display.HTML object>
```

```
--- Evaluation Metrics ---
Accuracy: 0.8652
F1-Score: 0.9063
Loss: 0.8259
```

5 Refining the Model

Now that we have a baseline for model performance we can make refinements to our learning rate, weight decay, and add a warmup schedule and compare.

[6]:
```
# Output the eval for experimental refined model
eval_results = trainer.evaluate()

print("--- Evaluation Metrics ---")
print(f"Accuracy: {eval_results['eval_accuracy']:.4f}")
print(f"F1-Score: {eval_results['eval_f1']:.4f}")
print(f"Loss: {eval_results['eval_loss']:.4f}")
```

```
<IPython.core.display.HTML object>
```

```
--- Evaluation Metrics ---
Accuracy: 0.8603
F1-Score: 0.8966
Loss: 0.3824
```

6 Analysis of Refinement

6.1 1. Initial Model Performance (Before Refinement)

— Evaluation Metrics — - Accuracy: 0.8652 - F1-Score: 0.9063 - Loss: 0.8259

Based on the initial results, we attempted to refine the model to improve generalization. I hypothesized that the default learning rate might be too high, causing the model to converge too quickly and potentially overfit. I made the following hyperparameter adjustments: - Reduced learning rate from `5e-5` to `2e-5`. - Added weight decay, set to `0.01` to add regularization. - Added warmup steps to introduce a period of smaller learning steps before training at defined level.

6.2 2. Refined Model Performance

— Evaluation Metrics — - Accuracy: 0.8603 - F1-Score: 0.8966 - Loss: 0.3824

6.3 3. Compare and Analyze Results

We can see that with our refinements the model has in fact become very overconfident. The log loss has decreased significantly with a more stable training process, however our accuracy and F1-Scores have both decreased slightly in comparison to the original, untuned hyperparameters. It seems that our model has learned the patterns in the training data far more specifically such that it doesn't generalize as well as the untuned model.

7 Part 4: Creative Application

For part 4 I have decided to train the DistilBERT model on a sentiment analysis problem using the Yelp Review full dataset. The goal is to classify reviews into one of five star ratings. I have chosen DistilBERT becuase it is a smaller version of BERT that retains performance well from the foundation model which will result in much faster training, reducing training time, and thereby increasing experimental iteration speeds. To optimize performance, I implemented mixed-precision training (fp16=True) to accelerate the process and an EarlyStoppingCallback to prevent overfitting.

[10]:
```
from transformers import EarlyStoppingCallback

# Check if MPS is available and set the device
if torch.backends.mps.is_available():
    device = torch.device("mps")
    print("Using Apple Silicon (MPS) backend.")
elif torch.cuda.is_available():
    device = torch.device("cuda")
    print("Using NVIDIA CUDA backend.")
else:
    device = torch.device("cpu")
    print("Using CPU backend.")

dataset = load_dataset("yelp_review_full")
model_name = "distilbert-base-uncased"

# There are 5 stars so the number of labels is 5
model = AutoModelForSequenceClassification.from_pretrained(model_name,␣
 ↪num_labels=5)
tokenizer = AutoTokenizer.from_pretrained(model_name)

def encode(examples):
    # dataset has a single 'text' field
    return tokenizer(examples["text"], truncation=True, padding="max_length",␣
 ↪max_length=512)

# tokenize the dataset
```

```
tokenized_datasets = dataset.map(encode, batched=True)
tokenized_datasets = tokenized_datasets.map(lambda examples: {"labels":␣
 ↪examples["label"]}, batched=True)
tokenized_datasets = tokenized_datasets.remove_columns(["text", "label"])
tokenized_datasets.set_format("torch")

train_dataset = tokenized_datasets["train"]
test_dataset = tokenized_datasets["test"]

metric = evaluate.load("accuracy")

training_args = TrainingArguments(
    output_dir="./yelp-distilbert-results",
    num_train_epochs=3,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=64,
    learning_rate=2e-5,
    weight_decay=0.01,
    warmup_steps=500,
    eval_strategy="epoch",
    save_strategy="epoch",
    load_best_model_at_end=True,
    metric_for_best_model="accuracy",
    fp16=True,  # Enable Mixed Precision Training
)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=test_dataset,
    tokenizer=tokenizer,
    compute_metrics=compute_metrics,a
    callbacks=[EarlyStoppingCallback(early_stopping_patience=3)] # Stop if␣
 ↪metric doesn't improve for 3 evaluations
)

# Start Training
print("Starting training...")
trainer.train()
print("Training complete!")

# Save the final best model
final_model_path = "./final_yelp_model"
print(f"Saving the best model to {final_model_path}")
trainer.save_model(final_model_path)
print("Model saved successfully.")
```

```
Using NVIDIA CUDA backend.

Some weights of DistilBertForSequenceClassification were not initialized from
the model checkpoint at distilbert-base-uncased and are newly initialized:
['classifier.bias', 'classifier.weight', 'pre_classifier.bias',
'pre_classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it
for predictions and inference.

Map:   0%|          | 0/50000 [00:00<?, ? examples/s]

Map:   0%|          | 0/50000 [00:00<?, ? examples/s]

/tmp/ipykernel_73705/4140540532.py:51: FutureWarning: `tokenizer` is deprecated
and will be removed in version 5.0.0 for `Trainer.__init__`. Use
`processing_class` instead.
  trainer = Trainer(

Starting training…

<IPython.core.display.HTML object>

Training complete!
Saving the best model to ./final_yelp_model
Model saved successfully.
```

[12]:
```python
# Check if MPS is available and set the device
if torch.backends.mps.is_available():
    device = torch.device("mps")
    print("Using Apple Silicon (MPS) backend.")
elif torch.cuda.is_available():
    device = torch.device("cuda")
    print("Using NVIDIA CUDA backend.")
else:
    device = torch.device("cpu")
    print("Using CPU backend.")

dataset = load_dataset("yelp_review_full")
model_name = "distilbert-base-uncased"

# There are 5 stars so the number of labels is 5
model = AutoModelForSequenceClassification.from_pretrained(model_name,␣
 ↪num_labels=5)
tokenizer = AutoTokenizer.from_pretrained(model_name)

def encode(examples):
    # dataset has a single 'text' field
    return tokenizer(examples["text"], truncation=True, padding="max_length",␣
 ↪max_length=512)

# tokenize the dataset
```

```
tokenized_datasets = dataset.map(encode, batched=True)
tokenized_datasets = tokenized_datasets.map(lambda examples: {"labels":␣
 ↪examples["label"]}, batched=True)
tokenized_datasets = tokenized_datasets.remove_columns(["text", "label"])
tokenized_datasets.set_format("torch")

train_dataset = tokenized_datasets["train"]
test_dataset = tokenized_datasets["test"]

metric = evaluate.load("accuracy")

training_args = TrainingArguments(
    output_dir="./yelp-distilbert-results",
    num_train_epochs=4,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    learning_rate=5e-5,
    warmup_steps=250,
    eval_strategy="epoch",
    save_strategy="epoch",
    load_best_model_at_end=True,
    metric_for_best_model="accuracy",
    fp16=True,  # Enable Mixed Precision Training
)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=test_dataset,
    tokenizer=tokenizer,
    compute_metrics=compute_metrics,
    callbacks=[EarlyStoppingCallback(early_stopping_patience=3)] # Stop if␣
 ↪metric doesn't improve for 3 evaluations
)

# Start Training
print("Starting training...")
trainer.train()
print("Training complete!")

# Save the final best model
final_model_path = "./final_yelp_model"
print(f"Saving the best model to {final_model_path}")
trainer.save_model(final_model_path)
print("Model saved successfully.")
```

```
Using NVIDIA CUDA backend.

Some weights of DistilBertForSequenceClassification were not initialized from
the model checkpoint at distilbert-base-uncased and are newly initialized:
['classifier.bias', 'classifier.weight', 'pre_classifier.bias',
'pre_classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it
for predictions and inference.
/tmp/ipykernel_73705/678575060.py:48: FutureWarning: `tokenizer` is deprecated
and will be removed in version 5.0.0 for `Trainer.__init__`. Use
`processing_class` instead.
  trainer = Trainer(

Starting training…

<IPython.core.display.HTML object>

Training complete!
Saving the best model to ./final_yelp_model
Model saved successfully.
```

[ ]: