

# Project: Fine-Tuning BERT

## Part 1: Fine-Tuning BERT

### Model 1:

```
from transformers import BertTokenizer, BertForSequenceClassification
from transformers import Trainer, TrainingArguments
from datasets import load_dataset
import torch

# Load the dataset
dataset = load_dataset('stanfordnlp/imdb')

# Load the BERT tokenizer
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')

# Tokenize the data
def tokenize_function(examples):
    return tokenizer(examples['text'], padding="max_length", truncation=True, max_length=128)

tokenized_datasets = dataset.map(tokenize_function, batched=True)

# Prepare data for PyTorch
tokenized_datasets = tokenized_datasets.rename_column("label", "labels")
tokenized_datasets.set_format("torch", columns=["input_ids", "attention_mask", "labels"])

train_dataset = tokenized_datasets["train"].shuffle(seed=42).select(range(2000)) # Use a subset for quick training
test_dataset = tokenized_datasets["test"].shuffle(seed=42).select(range(500))

# Load the pre-trained BERT model
model = BertForSequenceClassification.from_pretrained('bert-base-uncased', num_labels=2)

# To reduce validation loss since it is increasing
# Define training arguments
training_args = TrainingArguments(
    output_dir="./results",
    eval_strategy="epoch",
    learning_rate=2e-5,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    num_train_epochs=3,
    weight_decay=0.01,
    logging_dir='./logs',
    logging_steps=10,
    save_steps=10,
)

# Define a Trainer instance
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=test_dataset,
)

# Train the model
trainer.train()

# Evaluate the model
eval_result = trainer.evaluate()
print(f"Evaluation results: {eval_result}")
```

[375/375 06:30, Epoch 3/3]

Epoch	Training Loss	Validation Loss
1	0.363000	0.395811
2	0.258800	0.434042
3	0.176400	0.406961

```
/opt/anaconda3/lib/python3.12/site-packages/torch/utils/data/dataloader.py:683: UserWarning: 'pin_memory' argument is set as true but not supported on MPS now, then device pinned memory won't be used.
  warnings.warn(warn_msg)
/opt/anaconda3/lib/python3.12/site-packages/torch/utils/data/dataloader.py:683: UserWarning: 'pin_memory' argument is set as true but not supported on MPS now, then device pinned memory won't be used.
  warnings.warn(warn_msg)
/opt/anaconda3/lib/python3.12/site-packages/torch/utils/data/dataloader.py:683: UserWarning: 'pin_memory' argument is set as true but not supported on MPS now, then device pinned memory won't be used.
  warnings.warn(warn_msg)
```

[32/32 00:09]


```
Evaluation results: {'eval_loss': 0.40696120262145996, 'eval_runtime': 9.1805, 'eval_samples_per_second': 54.463, 'eval_steps_per_second': 3.486, 'epoch': 3.0}
```

The results show that while training loss steadily decreased over the epochs, the validation loss initially increased and only slightly decreased afterward, remaining relatively high. This pattern suggests overfitting, where the model is learning the training data too well and failing to generalize effectively to unseen data. Additionally, all training took over 6.5 minutes to complete.

## Part 2: Debugging Issues

While fine-tuning BERT, I observed that although the training loss consistently decreased over time, the validation loss did not follow the same trend; instead, it initially increased and remained relatively high. This indicated that the model was overfitting: learning the training data too well and struggling to generalize to new examples. To address this, I began by reducing the number of epochs from 3 to 2 in Model 2. This helped lower the validation loss slightly but unexpectedly increased the training time to over 9 minutes.

### Model 2:



Epoch	Training Loss	Validation Loss
1	0.374100	0.383969
2	0.245700	0.367452


In Model 3, I aimed to improve both training time and model performance. I increased the batch size to 32 while keeping the number of epochs at 2. This reduced the time slightly to 8:50 minutes, but the validation loss worsened in the second epoch, suggesting potential underfitting or instability. With Model 4, I tried increasing the batch size further and returned to 3 epochs, hoping to balance learning and efficiency. However, this significantly increased the training time to 14:28 minutes. Although the training loss dropped dramatically, the validation loss rose to 0.59, only slightly improving to 0.57 in the final epoch, further evidence of overfitting.

### Model 3:



Epoch	Training Loss	Validation Loss
1	0.217900	0.391609
2	0.182300	0.409156

### Model 4:



Epoch	Training Loss	Validation Loss
1	0.118700	0.594535
2	0.112400	0.591848
3	0.037700	0.575140

Finally, in Model 5, I applied a different approach: reducing the `max_length` parameter to 100 tokens, combined with the settings from Model 2 (batch size 16, 2 epochs). This led to a much shorter training time of just 2:59 minutes and a training loss of 0.31, paired with a training loss of 0.38, a more balanced result. Overall, the best-performing model in terms of validation loss was Model 2, with a training loss of 0.24 and a validation loss of 0.36. It offered a good tradeoff between generalization and efficiency, despite the longer training time.

## Model 5:

```
#EDIT CHANGED IT TO USE MAX_LENGTH = 100 WORDS INSTEAD OF 128
#BERT (Bidirectional Encoder Representations from Transformers) using the Hugging Face library.

from transformers import BertTokenizer, BertForSequenceClassification
from transformers import Trainer, TrainingArguments
from datasets import load_dataset
import torch

# Load the dataset
dataset = load_dataset('stanfordnlp/imdb')

# Load the BERT tokenizer
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')

# Tokenize the data
def tokenize_function(examples):
    return tokenizer(examples['text'], padding="max_length", truncation=True, max_length=100)

tokenized_datasets = dataset.map(tokenize_function, batched=True)

# Prepare data for PyTorch
tokenized_datasets = tokenized_datasets.rename_column("label", "labels")
tokenized_datasets.set_format("torch", columns=["input_ids", "attention_mask", "labels"])

train_dataset = tokenized_datasets["train"].shuffle(seed=42).select(range(2000)) # Use a subset for quick training
test_dataset = tokenized_datasets["test"].shuffle(seed=42).select(range(500))

# Load the pre-trained BERT model
model = BertForSequenceClassification.from_pretrained('bert-base-uncased', num_labels=2)
```

```
# EDIT: CHANGED NUM EPOCHS TO 2 and BATCH SIZE TO 16 but using 100 words
# To reduce validation loss since it is increasing
# Define training arguments
training_args = TrainingArguments(
    output_dir="./results",
    eval_strategy="epoch",
    learning_rate=2e-5,
    per_device_train_batch_size=32,
    per_device_eval_batch_size=32,
    num_train_epochs=2,
    weight_decay=0.01,
    logging_dir='./logs',
    logging_steps=10,
    save_steps=10,
)

# Define a Trainer instance
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=test_dataset,
)

#Train the model
trainer.train()

# Evaluate the model
eval_result = trainer.evaluate()
print(f"Evaluation results: {eval_result}")
```

[126/126 02:59, Epoch 2/2]

Epoch	Training Loss	Validation Loss
1	0.456000	0.413930
2	0.316700	0.388465

### Part 3: Evaluating the Model


Performance metrics results Model 2 before refining:

- Accuracy: 0.84
- F1 Score: 0.8412698412698413

Performance metrics results Model 2 after refining (training on 3000, max\_len 100, per\_device\_train\_batch\_size=32, per\_device\_eval\_batch\_size=32):

- Accuracy: 0.85
- F1 Score: 0.847870182555781

After debugging, I increased the training sample size to 3,000, an adjustment I hadn't considered earlier, and reduced the max token length to 100. These changes led to an improvement in performance, with Accuracy reaching 0.844 and F1 Score at 0.845. I also experimented with different batch sizes, which yielded results similar to earlier models. Overall, the refined model showed better training and validation loss than Model 2 and outperformed the initial fine-tuning attempt, which had shown signs of overfitting due to rising validation loss across epochs.

 [564/564 07:16, Epoch 3/3]

Epoch	Training Loss	Validation Loss
1	0.434800	0.378524
2	0.301800	0.400809
3	0.104100	0.512561

## Part 4: Creative Application

The NLP task that I decided to focus on was classifying tweets as positive or negative. Before fine-tuning BERT on the Sentiment140 dataset, I began by analyzing the data to better adapt the model to its structure. Since the dataset source didn't clearly specify the number of samples or typical text lengths, I wrote code to inspect these aspects. This helped me confirm that the dataset contained 1.6 million samples and that the average tweet length was approximately 22 tokens, with a maximum of 99. Based on this, I selected a max\_length of 70 tokens to capture nearly all tweets without excessive padding. Given that Twitter language is often informal, inconsistent, and fast-evolving, I prioritized model generalization over pure training accuracy and F1 score. This is because a model trained too specifically may fail to perform well on newer or differently phrased tweets. Throughout my process, I experimented with key hyperparameters, reducing epochs to 2 to prevent overfitting, testing different batch sizes, and comparing cased vs. uncased models. Ultimately, the uncased BERT model with a batch size of 16 and 2 epochs delivered the best generalization, as it achieved strong validation performance without signs of overfitting across epochs. Evaluation metrics for this model showed an accuracy of 0.824 and an F1 score of 0.829, confirming that the model not only learned the sentiment classification task effectively but also generalized well to unseen data.

[250/250 05:07, Epoch 2/2]

Epoch	Training Loss	Validation Loss
1	0.486400	0.434240
2	0.356300	0.437475

Evaluation results: {'eval\_loss': 0.4374752938747406, 'eval\_runtime': 8.3209, 'eval\_samples\_per\_second': 60.09, 'eval\_steps\_per\_second': 3.846, 'epoch': 2.0}  
Accuracy: 0.824  
F1 Score: 0.8294573643410853

```

#batch 16, epochs 2,
# Load the dataset
dataset = load_dataset('stanfordnlp/sentiment140')

# Map sentiment labels: 0 = negative, 4 = positive → convert 4 to 1
def map_labels(example):
    example["sentiment"] = 0 if example["sentiment"] == 4 else 1
    return example

dataset = dataset.map(map_labels)

tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')

def tokenize_function(examples):
    return tokenizer(examples['text'], padding="max_length", truncation=True, max_length=70)

tokenized_datasets = dataset.map(tokenize_function, batched=True)
tokenized_datasets = tokenized_datasets.rename_column("sentiment", "labels")
tokenized_datasets.set_format("torch", columns=["input_ids", "attention_mask", "labels"])

# Sample a subset for training/testing to speed things up
train_dataset = tokenized_datasets["train"].shuffle(seed=42).select(range(2000))
test_dataset = tokenized_datasets["train"].shuffle(seed=123).select(range(500))

# Load the BERT model for sequence classification (binary)
model = BertForSequenceClassification.from_pretrained('bert-base-uncased', num_labels=2)

# Define training arguments
training_args = TrainingArguments(
    output_dir="./results",
    eval_strategy="epoch",
    learning_rate=2e-5,
    per_device_train_batch_size= 16,
    per_device_eval_batch_size= 16,
    num_train_epochs= 2,
    weight_decay=0.01,
    logging_dir='./logs',
    logging_steps=10,
    save_steps=10,
)

# Define a Trainer instance
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=test_dataset,
)

trainer.train()

eval_result = trainer.evaluate()
print(f"Evaluation results: {eval_result}")
# Make predictions and compute metrics
predictions = trainer.predict(test_dataset)
preds = torch.argmax(torch.tensor(predictions.predictions), axis=1)
labels = predictions.label_ids

accuracy = accuracy_score(labels, preds)
f1 = f1_score(labels, preds)

print(f"Accuracy: {accuracy}")
print(f"F1 Score: {f1}")

```