Lab 12 - BERT ¶

- Language model
- Open source ML based framework for NLP
- Bidirectional Encoder

Use pretrained BERT models for various NLP tasks like question answering, sentiment analysis, and text classification. Fine tune BERT for a specific task and compare its performance with traditional models.

In [2]:
```
import torch
```

In [3]:
```
from transformers import BertTokenizer, BertForSequenceClassification
from torch.nn import functional as F
```

In [4]:
```
# Load pre-trained BERT model and tokenizer
model_name = 'bert-base-uncased'
tokenizer = BertTokenizer.from_pretrained(model_name)
model = BertForSequenceClassification.from_pretrained(model_name)
```

```
Some weights of BertForSequenceClassification were not initialized from the model checkpoint at bert-base-uncased and are newly initial
ized: ['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.
```

In [5]:
```
# Define sentiment labels
sentiment_labels = {0: 'Negative', 1: 'Neutral', 2: 'Positive'}

def predict_sentiment(text):
    # Tokenize the text and convert to tensor
    inputs = tokenizer(text, return_tensors='pt', padding=True, truncation=True)

    # Forward pass through the model
    outputs = model(**inputs)

    # Get predicted logits and softmax probabilities
    logits = outputs.logits
    probabilities = F.softmax(logits, dim=1)

    # Get predicted sentiment label
    predicted_label = torch.argmax(probabilities, dim=1).item()
    predicted_sentiment = sentiment_labels[predicted_label]

    return predicted_sentiment, probabilities
```

In [6]:
```
text = "I didn't enjoy the movie, it was horrible"
sentiment, probabilities = predict_sentiment(text)
print(f"Predicted Sentiment: {sentiment}")
print("Probabilities:", probabilities)
```

```
Predicted Sentiment: Neutral
Probabilities: tensor([[0.2782, 0.7218]], grad_fn=<SoftmaxBackward0>)
```

In [7]:
```
from transformers import pipeline
sa = pipeline('text-classification', model=model,tokenizer=tokenizer)
sa("""I didn't enjoyed the movie, it was very horrible""")
```

Out[7]:
```
[{'label': 'LABEL_1', 'score': 0.7198613882064819}]
```

In [ ]:

```python
import pandas as pd
import torch
from transformers import BertTokenizer, BertForSequenceClassification
from torch.utils.data import TensorDataset, DataLoader
from sklearn.preprocessing import LabelEncoder

# Load the dataset
data = pd.read_csv('dataset.csv')

# Encode the text data
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
encoded_data = tokenizer.batch_encode_plus(
    data['text'].values,
    add_special_tokens=True,
    return_attention_mask=True,
    padding='max_length',
    max_length=512,
    return_tensors='pt'
)

# Encode labels
label_encoder = LabelEncoder()
labels = label_encoder.fit_transform(data['label'].values)
labels = torch.tensor(labels)

# Create input tensors
input_ids = encoded_data['input_ids']
attention_masks = encoded_data['attention_mask']

# Create the dataset and dataloader
dataset = TensorDataset(input_ids, attention_masks, labels)
dataloader = DataLoader(dataset, batch_size=16, shuffle=True)

# Load the pre-trained BERT model
model = BertForSequenceClassification.from_pretrained(
    'bert-base-uncased',
    num_labels=len(set(data['label'])),  # Number of unique labels
    output_attentions=False,
    output_hidden_states=False
)

# Set the device (CPU or GPU)
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model.to(device)

# Define the optimizer and loss function
optimizer = torch.optim.Adam(model.parameters(), lr=2e-5)
loss_fn = torch.nn.CrossEntropyLoss()

# Training loop
epochs = 3
for epoch in range(epochs):
    model.train()
    for batch in dataloader:
        # Unpack the batch
        b_input_ids, b_attention_mask, b_labels = tuple(t.to(device) for t in batch)

        # Forward pass
        outputs = model(b_input_ids, token_type_ids=None, attention_mask=b_attention_mask, labels=b_labels)
        loss = outputs.loss

        # Backward pass and optimization
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    print(f'Epoch {epoch+1}/{epochs}, Loss: {loss.item()}')

# Evaluation
model.eval()
eval_accuracy = 0
eval_data = TensorDataset(input_ids, attention_masks, labels)
eval_dataloader = DataLoader(eval_data, batch_size=16)

for batch in eval_dataloader:
    b_input_ids, b_attention_mask, b_labels = tuple(t.to(device) for t in batch)

    with torch.no_grad():
        logits = model(b_input_ids, token_type_ids=None, attention_mask=b_attention_mask)[0]

    logits = logits.detach().cpu().numpy()
    label_ids = b_labels.to('cpu').numpy()
    eval_accuracy += (logits.argmax(axis=-1) == label_ids).mean()

eval_accuracy /= len(eval_dataloader)
print(f'Evaluation Accuracy: {eval_accuracy*100:.2f}%')
```

```
C:\Users\ahana\AppData\Local\Programs\Python\Python311\Lib\site-packages\transformers\utils\generic.py:311: UserWarning: torch.utils._p
ytree._register_pytree_node is deprecated. Please use torch.utils._pytree.register_pytree_node instead.
  torch.utils._pytree._register_pytree_node(
C:\Users\ahana\AppData\Local\Programs\Python\Python311\Lib\site-packages\transformers\utils\generic.py:311: UserWarning: torch.utils._p
ytree._register_pytree_node is deprecated. Please use torch.utils._pytree.register_pytree_node instead.
  torch.utils._pytree._register_pytree_node(
Some weights of BertForSequenceClassification were not initialized from the model checkpoint at bert-base-uncased and are newly initial
ized: ['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.
```

In [ ]: