STTAI - Assignment 11

Group 25   [Github Link Here]

Name	Roll Number
Romit Mohane	23110279
Rudra Pratap Singh	23110281

1. Dataset Preparation (10%)

- Load the training dataset and test data.
- Use 20% of the training dataset as the validation set.

1. Dataset Preparation

```
!wget https://raw.githubusercontent.com/clairett/pytorch-sentiment-classification/master/data/SST2/train.tsv
!wget https://raw.githubusercontent.com/clairett/pytorch-sentiment-classification/master/data/SST2/test.tsv
```

Show hidden output

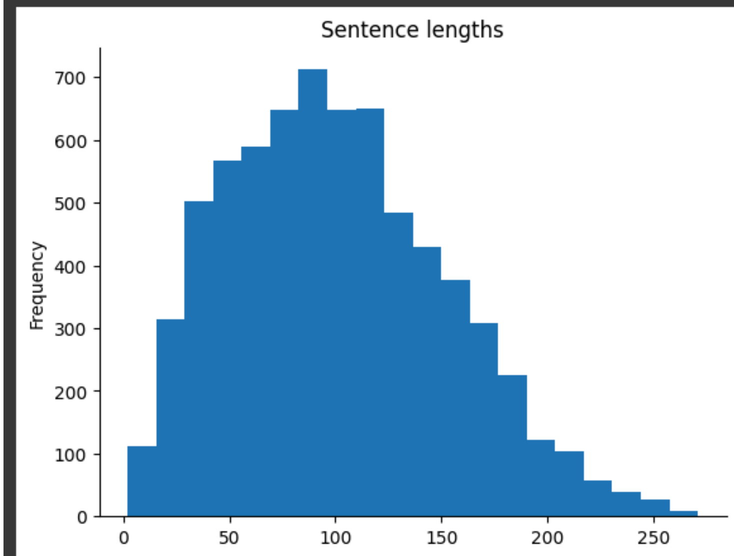
Load the training dataset and test data.

```
train_df = pd.read_csv('train.tsv', header= None, sep='\t')
test_df = pd.read_csv('test.tsv', header=None, sep='\t')
train_df.columns = ['sentence', 'label',]
test_df.columns = ['sentence', 'label',]

train_df.head()
```

	sentence	label
0	a stirring , funny and finally transporting re...	1
1	apparently reassembled from the cutting room f...	0
2	they presume their audience wo n't sit still f...	0
3	this is a visually stunning rumination on love...	1
4	jonathan parker 's bartleby should have been t...	1

```python
train_df['sentence'].str.len().plot(kind='hist', bins=20, title='Sentence lengths')
plt.gca().spines[['top', 'right',]].set_visible(False)
```



Use 20% of the training dataset as the validation set.

```python
train_df, val_df = tts(train_df, test_size=0.2)

train_df.shape, val_df.shape, test_df.shape
```

```
((5536, 2), (1384, 2), (1821, 2))
```

```python
# Initialize Bag-of-Words vectorizer
vectorizer = CountVectorizer(max_features=10000)
X_train = vectorizer.fit_transform(train_df['sentence']).toarray()
X_val = vectorizer.transform(val_df['sentence']).toarray()
y_train = train_df['label'].values
y_val = val_df['label'].values

print("X_train shape:", X_train.shape)
print("X_val shape:", X_val.shape)
```

```
X_train shape: (5536, 10000)
X_val shape: (1384, 10000)
```

2. Construct a **Multi-Layer Perceptron (MLP) model**. **(10%)**

- The parameter should be with:
  - hidden_sizes=[512, 256, 128, 64]
  - Output should have two labels.
  - With the following architecture:

```
class MLP(nn.Module):
    def __init__(self, input_dim, hidden_1=512,hidden_2=256,hidden_3=128,hidden_4=64, output_dim=2):
        super(MLP, self).__init__()
        self.fc1 = nn.Linear(input_dim, hidden_1)
        self.fc2 = nn.Linear(hidden_1, hidden_2)
        self.fc3 = nn.Linear(hidden_2, hidden_3)
        self.fc4 = nn.Linear(hidden_3, hidden_4)
        self.fc5 = nn.Linear(hidden_4, output_dim)
        self.relu = nn.ReLU()
        self.dropout = nn.Dropout(0.3)

    def forward(self, x):
        x = self.fc1(x)
        x = self.relu(x)
        x = self.dropout(x)
        x = self.fc2(x)
        x = self.relu(x)
        x = self.dropout(x)
        x = self.fc3(x)
        x = self.relu(x)
        x = self.dropout(x)
        x = self.fc4(x)
        x = self.relu(x)
        x = self.dropout(x)
        x = self.fc5(x)
        return x
```

```
MLP(
  (fc1): Linear(in_features=10000, out_features=512, bias=True)
  (fc2): Linear(in_features=512, out_features=256, bias=True)
  (fc3): Linear(in_features=256, out_features=128, bias=True)
  (fc4): Linear(in_features=128, out_features=64, bias=True)
  (fc5): Linear(in_features=64, out_features=2, bias=True)
  (relu): ReLU()
  (dropout): Dropout(p=0.3, inplace=False)
)
```

- Count the number of trainable parameters in the model using the automated function (reference).

Count the number of trainable parameters in the model using the automated function.

```
[ ]  model = MLP(input_dim=10000)

     # Count trainable parameters
     num_params = sum(p.numel() for p in model.parameters() if p.requires_grad)

     print(f"Total trainable parameters: {num_params:,}")
```

```
Total trainable parameters: 5,293,122
```

3. Train the model with 10 epochs and create the best-performing model (checkpoint.pt) (10%)

- Plot the validation accuracy + loss (epochs vs accuracy-loss).

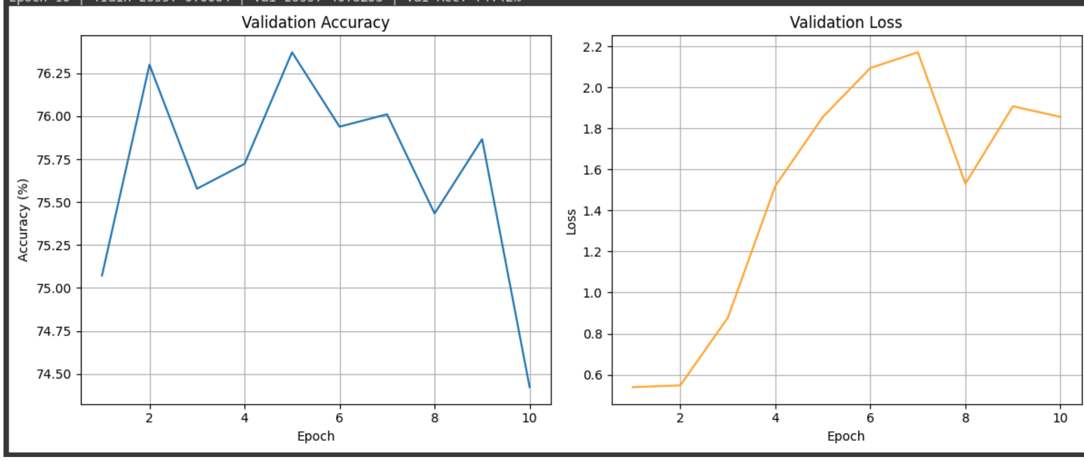
```
for epoch in range(10):
    model.train()
    running_loss = 0.0
    for batch_x, batch_y in train_loader:
        optimizer.zero_grad()
        outputs = model(batch_x)
        loss = criterion(outputs, batch_y)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
    avg_train_loss = running_loss / len(train_loader)
    train_losses.append(avg_train_loss)

    # Validation
    model.eval()
    val_loss = 0.0
    correct = 0
    total = 0
    with torch.no_grad():
        for batch_x, batch_y in val_loader:
            outputs = model(batch_x)
            loss = criterion(outputs, batch_y)
            val_loss += loss.item()
            _, predicted = torch.max(outputs, 1)
            correct += (predicted == batch_y).sum().item()
            total += batch_y.size(0)
    acc = 100 * correct / total
    val_losses.append(val_loss / len(val_loader))
    val_accuracies.append(acc)

    print(f"Epoch {epoch+1} | Train Loss: {avg_train_loss:.4f} | Val Loss: {val_loss:.4f} | Val Acc: {acc:.2f}%")

    # Save best model
    if acc > best_val_acc:
        best_val_acc = acc
        torch.save(model.state_dict(), "checkpoint.pt")
```

```
Epoch 1 | Train Loss: 0.5938 | Val Loss: 11.8571 | Val Acc: 75.07%
Epoch 2 | Train Loss: 0.2567 | Val Loss: 12.0529 | Val Acc: 76.30%
Epoch 3 | Train Loss: 0.0679 | Val Loss: 19.2941 | Val Acc: 75.58%
Epoch 4 | Train Loss: 0.0186 | Val Loss: 33.4290 | Val Acc: 75.72%
Epoch 5 | Train Loss: 0.0050 | Val Loss: 40.8227 | Val Acc: 76.37%
Epoch 6 | Train Loss: 0.0031 | Val Loss: 46.0707 | Val Acc: 75.94%
Epoch 7 | Train Loss: 0.0018 | Val Loss: 47.7548 | Val Acc: 76.01%
Epoch 8 | Train Loss: 0.0143 | Val Loss: 33.6602 | Val Acc: 75.43%
Epoch 9 | Train Loss: 0.0053 | Val Loss: 41.9656 | Val Acc: 75.87%
Epoch 10 | Train Loss: 0.0064 | Val Loss: 40.8253 | Val Acc: 74.42%
```



4. Now use:

1. **Dynamic Quantization with INT4 or INT8 (Link: here) (20%)**
   a. Use the torch.quantization.quantize_dynamic()
2. **Half precision (20%)**
   a. Use the .half() function. (Reference: here)

```
# Apply dynamic quantization (INT8, not INT4)
model_dynamic = torch.quantization.quantize_dynamic(
    model_fp32, {nn.Linear}, dtype=torch.qint8
)
```

```
Dynamic Quantization Results:
Accuracy: 76.37%
Model Size: 5.06 MB
Inference Time: 368.41 ms
```

```
model_fp32 = MLP(input_dim=X_train.shape[1])
model_fp32.load_state_dict(torch.load("checkpoint.pt"))
model_fp16 = model_fp32.half()
model_fp16.eval()
```

```
Half Precision Results:
Accuracy: 76.37%
Model Size: 10.10 MB
Inference Time: 0.75 ms
```

5. Fill the table for different quantization techniques. (30%)

S.I.	Model Name	Accuracy	Storage	Inference Time
1	Original (CPU Runtime)	78.97%	20.2 MB	272 ± 54 ms
2	Original (T4 GPU Runtime)	76.37%	20.2 MB	3.93 ms ± 61.8 μs
3	Dynamic Quantization (CPU Runtime)	78.90%	5.06 MB	213 ± 50 ms
3	Dynamic Quantization (T4 GPU Runtime)	Not Supported	-	-
4	Half (CPU Runtime)	78.97%	10.1 MB	2120 ± 152 ms
5	Half (T4 GPU Runtime)	76.37%	10.1 MB	634 μs ± 5.91 μs