STT LAB Assignment - 11

Repo

Name	Roll number
Parthiv Patel	23110237
Aryan Solanki	23110049

```
# Load .tsv files
train_df = pd.read_csv("train.tsv", sep='\t')
test_df = pd.read_csv("test.tsv", sep='\t')


train_data, val_data = train_test_split(train_df, test_size=0.2, random_state=42)

# Check results
print("Training set:", train_data.shape)
print("Validation set:", val_data.shape)
```

[25] Python

```
Training set: (5535, 2)
Validation set: (1384, 2)
```

```python
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split

# Load with column names manually set
train_df = pd.read_csv("train.tsv", sep='\t', header=None, names=["sentence", "label"])
test_df = pd.read_csv("test.tsv", sep='\t', header=None, names=["sentence", "label"])

# Split 20% for validation
train_data, val_data = train_test_split(train_df, test_size=0.2, random_state=42)

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

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

[26] Python

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

```python
import torch.nn as nn
import torch


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
```

Python

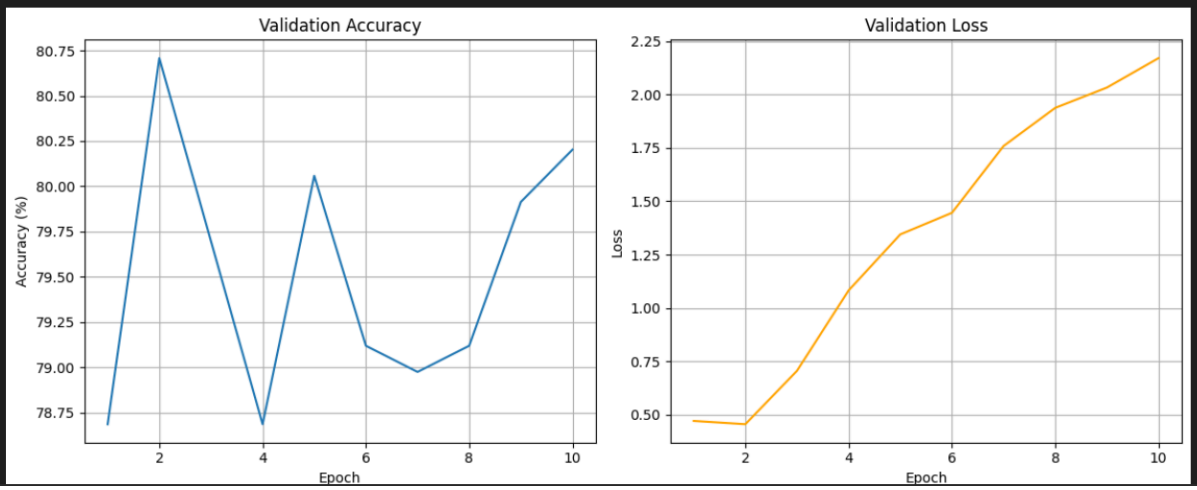
```python
model = MLP(input_dim=10000)

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

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


"""
(10000 * 512 + 512) = 5,120,512
(512 * 256 + 256)   =   131,328
(256 * 128 + 128)   =    32,896
(128 * 64 + 64)     =     8,256
(64 * 2 + 2)        =       130
----------------------------
Total               = 5,293,122

# 5,293,122 trainable parameters
"""
```

Python

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

```python
train_losses, val_losses, val_accuracies = [], [], []

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: 0.6082 | Val Loss: 10.3604 | Val Acc: 78.68%
Epoch 2 | Train Loss: 0.2775 | Val Loss: 10.0236 | Val Acc: 80.71%
Epoch 3 | Train Loss: 0.0827 | Val Loss: 15.5300 | Val Acc: 79.70%
Epoch 4 | Train Loss: 0.0207 | Val Loss: 23.8182 | Val Acc: 78.68%
Epoch 5 | Train Loss: 0.0076 | Val Loss: 29.5870 | Val Acc: 80.06%
Epoch 6 | Train Loss: 0.0056 | Val Loss: 31.8124 | Val Acc: 79.12%
Epoch 7 | Train Loss: 0.0028 | Val Loss: 38.6993 | Val Acc: 78.97%
Epoch 8 | Train Loss: 0.0006 | Val Loss: 42.6305 | Val Acc: 79.12%
Epoch 9 | Train Loss: 0.0005 | Val Loss: 44.7238 | Val Acc: 79.91%
Epoch 10 | Train Loss: 0.0001 | Val Loss: 47.7574 | Val Acc: 80.20%
```




```
Original Model Results:
Accuracy: 80.56%
Model Size: 20.20 MB
Inference Time: 58.56 ms
Number of parameters: 5,293,122
```

```
Dynamic Quantization Results:
Accuracy: 80.49%
Model Size: 5.06 MB
Inference Time: 94.78 ms
Total trainable parameters (dynamic): 0
```

```
Half Precision Results:
Accuracy: 80.56%
Model Size: 10.10 MB
Inference Time: 14319.69 ms
Total trainable parameters (FP16): 5,293,122
```

S.I.	Model Name	Test Accuracy (Out of 100)	Storage (In MB)	Number of parameters using in-build function	Inference time (In ms)
1.	Original	80.56%	20.20 MB	5,293,122	58.56 ms
2.	Dynamic	80.49%	5.06 MB	0	94.78 ms
3.	Half	80.56%	10.10 MB	5,293,122	14319.69 ms

Time for Half is more as it lacks support on CPU.