Import Dependancies

In [ ]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
import re
import nltk
from nltk.tokenize import word_tokenize
from collections import Counter
from keras.preprocessing.sequence import pad_sequences
import matplotlib.pyplot as plt
import numpy as np
import torch
import torch.nn as nn
from torch.utils.data import DataLoader, TensorDataset
import torch.optim as optim
import torch.nn.functional as F
```

In [ ]:
```
class SentimentClassifier(nn.Module):
    def __init__(self, vocab_size, embedding_dim, hidden_dim, output_dim):
        super(SentimentClassifier, self).__init__()
        self.embedding = nn.Embedding(vocab_size, embedding_dim)
        self.fc = nn.Linear(embedding_dim, hidden_dim)
        self.relu = nn.ReLU()
        self.out = nn.Linear(hidden_dim, output_dim)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        embedded = self.embedding(x)
        hidden = self.fc(embedded.mean(dim=1))
        hidden = self.relu(hidden)
        output = self.out(hidden)
        return self.sigmoid(output)
```

In [ ]:
```
# Load dataset
df = pd.read_csv('./imdb.csv')

# Splitting the dataset into training, validation, and testing sets
train_df, temp_df = train_test_split(df, test_size=0.3, random_state=42)  #
val_df, test_df = train_test_split(temp_df, test_size=0.5, random_state=42)

# Cleaning Data
def clean_text(text):
    # Add whitespace around punctuation
    text = re.sub(r'([.,!?()])', r' \1 ', text)
    # Remove non-punctuation symbols
    text = re.sub(r'[^a-zA-Z.,!?() ]', '', text)
    return text.strip()

# Apply cleaning to all splits
train_df['cleaned_text'] = train_df['review'].apply(clean_text)
val_df['cleaned_text'] = val_df['review'].apply(clean_text)
test_df['cleaned_text'] = test_df['review'].apply(clean_text)
```

In [ ]:

```python
# Map string labels to numeric values: 'negative' to 0 and 'positive' to 1
label_map = {'negative': 0, 'positive': 1}
train_df['sentiment'] = train_df['sentiment'].map(label_map).astype(float)
val_df['sentiment'] = val_df['sentiment'].map(label_map).astype(float)
test_df['sentiment'] = test_df['sentiment'].map(label_map).astype(float)

# Download NLTK tokenizer model
nltk.download('punkt')

# Tokenization
tokenized_reviews = [word_tokenize(review.lower()) for review in train_df['c
val_tokenized = [word_tokenize(review.lower()) for review in val_df['cleaned
test_tokenized = [word_tokenize(review.lower()) for review in test_df['clean

# Build Vocabulary
word_counts = Counter(word for review in tokenized_reviews for word in revie
vocabulary = {word: i + 1 for i, (word, _) in enumerate(word_counts.most_com

# Convert Text to Integer Sequences
reviews_int = [[vocabulary[word] for word in review if word in vocabulary] f
val_reviews_int = [[vocabulary[word] for word in review if word in vocabular
test_reviews_int = [[vocabulary[word] for word in review if word in vocabula
review_lengths = [len(review) for review in tokenized_reviews]

# Find a suitable max length
percentile = 95
max_len = int(np.percentile(review_lengths, percentile))

# Padding Sequences
padded_reviews = pad_sequences(reviews_int, maxlen=max_len, padding='post',
val_padded_reviews = pad_sequences(val_reviews_int, maxlen=max_len, padding=
test_padded_reviews = pad_sequences(test_reviews_int, maxlen=max_len, paddir
```

```
[nltk_data] Downloading package punkt to /Users/nickparov/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

In [ ]:

```python
# Parameters
vocab_size = 10000 + 1 # +1 for padding token
embedding_dim = 100  # common choice for embedding dimension
hidden_dim = 64 # mid-range for hidden dimension
output_dim = 1 # binary

# init model
model = SentimentClassifier(vocab_size, embedding_dim, hidden_dim, output_di

# Parameters
batch_size = 32

train_inputs = torch.tensor(padded_reviews, dtype=torch.long)
val_inputs = torch.tensor(val_padded_reviews, dtype=torch.long)
test_inputs = torch.tensor(test_padded_reviews, dtype=torch.long)

# labels
train_labels = torch.tensor(train_df['sentiment'].values, dtype=torch.float3
val_labels = torch.tensor(val_df['sentiment'].values, dtype=torch.float32)
```

```
test_labels = torch.tensor(test_df['sentiment'].values, dtype=torch.float32)

# Train DataLoader
train_data = TensorDataset(train_inputs, train_labels)
train_loader = DataLoader(train_data, batch_size=batch_size, shuffle=True)

# Validation DataLoader
val_data = TensorDataset(val_inputs, val_labels)
val_loader = DataLoader(val_data, batch_size=batch_size, shuffle=True)

# Test DataLoader
test_data = TensorDataset(test_inputs, test_labels)
test_loader = DataLoader(test_data, batch_size=batch_size, shuffle=False)
```

In [ ]:
```
# Define optimizer and loss function
optimizer = optim.Adam(model.parameters(), lr=0.001)
criterion = nn.BCELoss()

num_epochs = 8

for epoch in range(num_epochs):
    # Training Phase
    model.train()
    total_loss, total, correct = 0, 0, 0

    for inputs, labels in train_loader:
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs.squeeze(), labels)
        loss.backward()
        optimizer.step()

        total_loss += loss.item()
        predicted = (outputs.squeeze() > 0.5).float()
        correct += (predicted == labels).sum().item()
        total += labels.size(0)

    train_loss = total_loss / len(train_loader)
    train_accuracy = correct / total
    print(f'Epoch {epoch+1}/{num_epochs}, Train Loss: {train_loss:.4f}, Trai

    # Validation Phase
    model.eval()
    val_loss, val_correct, val_total = 0, 0, 0
    with torch.no_grad():
        for inputs, labels in val_loader:
            outputs = model(inputs)
            loss = criterion(outputs.squeeze(), labels)
            val_loss += loss.item()
            predicted = (outputs.squeeze() > 0.5).float()
            val_correct += (predicted == labels).sum().item()
            val_total += labels.size(0)

    val_loss /= len(val_loader)
```

```python
    val_accuracy = val_correct / val_total
    print(f'Epoch {epoch+1}/{num_epochs}, Validation Loss: {val_loss:.4f}, V
```

```
Epoch 1/8, Train Loss: 0.5962, Train Accuracy: 0.6543
Epoch 1/8, Validation Loss: 0.3987, Validation Accuracy: 0.8391
Epoch 2/8, Train Loss: 0.3382, Train Accuracy: 0.8609
Epoch 2/8, Validation Loss: 0.3087, Validation Accuracy: 0.8735
Epoch 3/8, Train Loss: 0.2734, Train Accuracy: 0.8897
Epoch 3/8, Validation Loss: 0.2917, Validation Accuracy: 0.8751
Epoch 4/8, Train Loss: 0.2434, Train Accuracy: 0.9019
Epoch 4/8, Validation Loss: 0.2644, Validation Accuracy: 0.8925
Epoch 5/8, Train Loss: 0.2209, Train Accuracy: 0.9130
Epoch 5/8, Validation Loss: 0.2735, Validation Accuracy: 0.8865
Epoch 6/8, Train Loss: 0.2054, Train Accuracy: 0.9206
Epoch 6/8, Validation Loss: 0.3199, Validation Accuracy: 0.8607
Epoch 7/8, Train Loss: 0.1924, Train Accuracy: 0.9247
Epoch 7/8, Validation Loss: 0.2559, Validation Accuracy: 0.8967
Epoch 8/8, Train Loss: 0.1808, Train Accuracy: 0.9301
Epoch 8/8, Validation Loss: 0.2655, Validation Accuracy: 0.8944
```

In [ ]:

```python
# Model Testing
model.eval()
test_loss, test_correct, test_total = 0, 0, 0

with torch.no_grad():
    for inputs, labels in test_loader:
        outputs = model(inputs)
        loss = criterion(outputs.squeeze(), labels)
        test_loss += loss.item()
        predicted = (outputs.squeeze() > 0.5).float()
        test_correct += (predicted == labels).sum().item()
        test_total += labels.size(0)

test_loss /= len(test_loader)
test_accuracy = test_correct / test_total
print(f'Test Loss: {test_loss:.4f}, Test Accuracy: {test_accuracy:.4f}')


# Extract the embedding layer weights
embedding_weights = model.embedding.weight.data

# Calculate the influence of each word
word_influence = torch.norm(embedding_weights, dim=1)

# Create a dictionary of words and their corresponding influence
word_influence_dict = {word: influence.item() for word, influence in zip(voc

# Sort words by influence
sorted_words = sorted(word_influence_dict.items(), key=lambda x: x[1], rever

# Get top 20 influential words for positive and negative reviews
# higher values indicate positive influence and lower values indicate negati
top_positive_words = sorted_words[:20]
top_negative_words = sorted_words[-20:]

# Print top 20 influential words for positive reviews
```

```python
print("Top 20 Positive Words:")
for word, influence in top_positive_words[:20]:
    print(f"{word}: {influence}")

# Print top 20 influential words for negative reviews
print("\nTop 20 Negative Words:")
for word, influence in top_negative_words[:20]:
    print(f"{word}: {influence}")
```

```
Test Loss: 0.2616, Test Accuracy: 0.9005
Top 20 Positive Words:
entire: 26.370708465576172
kind: 25.498929977416992
camera: 23.64904022216797
credits: 21.489665985107422
men: 20.365657806396484
appropriate: 20.28223419189453
page: 20.166799545288086
giving: 20.06629753112793
views: 19.575176239013672
green: 19.46065902709961
boyfriend: 19.23826026916504
mysterious: 19.13362693786621
beautiful: 19.074342727661133
weapons: 18.98100471496582
wasted: 18.938899993896484
mean: 18.158227920532227
sounds: 18.05463218688965
loves: 17.826236724853516
sea: 17.820823669433594
insane: 17.72930908203125

Top 20 Negative Words:
sunday: 8.13501262664795
producers: 8.122499465942383
marshall: 8.121447563171387
made: 8.105531692504883
washing: 8.09601879119873
stupid: 8.087364196777344
accidentally: 8.057730674743652
enemy: 8.051525115966797
sigh: 8.036587715148926
newcomer: 8.004074096679688
shows: 7.994915008544922
june: 7.987314224243164
symphony: 7.984182357788086
economic: 7.974999904632568
thrill: 7.962036609649658
alexandra: 7.954861640930176
misses: 7.931981086730957
worries: 7.739133834838867
multiple: 7.527898788452148
predator: 6.8740081787109375
```

In [ ]: