ECE657

Next Word Prediction

Setting seed and importing libraries:

TODO: Import the required libraries

```
import os
import numpy as np
import tensorflow as tf
import random
from google.colab import drive
import re
drive.mount('/content/drive/')
%cd /content/drive/MyDrive/ECE657
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split

from tensorflow.keras.preprocessing.sequence import pad_sequences
```

```
Drive already mounted at /content/drive/; to attempt to forcibly remount, call drive.mount("/content/drive/", force_remount=True).
/content/drive/MyDrive/ECE657
```

Adding random seed

```
# Set environment variables
os.environ['PYTHONHASHSEED'] = str(25)
os.environ['TF_DETERMINISTIC_OPS'] = '1'
os.environ['TF_CUDNN_DETERMINISTIC'] = '1'

# Set seed values
np.random.seed(25)
tf.random.set_seed(25)
random.seed(25)
```

Next Word Prediction:

Step1:

TODO: Read and Preprocess the dataset

```
path = 'alice.txt'
text = ""

# TODO: Load and preprocess the text

with open(path, 'r', encoding='utf-8') as file:
    text = file.read()
# Convert to lowercase
text = text.lower()
# Remove punctuation and special characters
text = re.sub(r'[^\w\s]', '', text)
```

```
print(len(text))
```

```
140269
```

ECE657

Step2:

TODO: Using tokenizers

```
# TODO: Tokenize the text

tokenizer = Tokenizer()
tokenizer.fit_on_texts([text])

# Calculate the total number of unique words
total_words = len(tokenizer.word_index) + 1  # Adding 1 to account for the tokenizer's indexing method

# Print the total number of words
print(f"Total number of unique words: {total_words}")
```

```
Total number of unique words: 2751
```

```
print(total_words)
```

```
2751
```

Step3:

TODO: Feature Engineering

```
# TODO: Create input sequences
lines = text.split('\n')

# Initialize a list to hold n-gram sequences
input_sequences = []

# Generate n-gram sequences
for line in lines:
    token_list = tokenizer.texts_to_sequences([line])[0]
    for i in range(1, len(token_list)+1):
        n_gram_sequence = token_list[:i]
        input_sequences.append(n_gram_sequence)

# Identify the maximum sequence length
max_sequence_length = max([len(seq) for seq in input_sequences])

# Pad sequences to the maximum length
padded_sequences = pad_sequences(input_sequences, maxlen=max_sequence_length, padding='pre')
```

```
print(len(input_sequences))
```

```
26410
```

ECE657

Step3:

TODO: Storing features and labels

```
[ ] input_sequences = np.array(padded_sequences)
    predictors, labels = input_sequences[:, :-1], input_sequences[:, -1]

    # One-hot encode the labels
    labels = to_categorical(labels, num_classes=total_words)

    # Split the data into training and validation sets
    X_train, X_val, y_train, y_val = train_test_split(predictors, labels, test_size=0.2, random_state=25)

    print(f"Training features shape: {X_train.shape}")
    print(f"Validation features shape: {X_val.shape}")
    print(f"Training labels shape: {y_train.shape}")
    print(f"Validation labels shape: {y_val.shape}")
```

```
Training features shape: (21128, 15)
Validation features shape: (5282, 15)
Training labels shape: (21128, 2751)
Validation labels shape: (5282, 2751)
```

Step4:

Summary:

TODO: Building our model

```
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense

# Build the first model
model = Sequential()
model.add(Embedding(input_dim=total_words, output_dim=100, input_length=max_sequence_length - 1))
model.add(LSTM(150))
model.add(Dense(total_words, activation='softmax'))

# Compile the model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

# Print the model summary
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 15, 100)           275100

 lstm (LSTM)                 (None, 150)               150600

 dense (Dense)               (None, 2751)              415401

=================================================================
Total params: 841101 (3.21 MB)
Trainable params: 841101 (3.21 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

ECE657

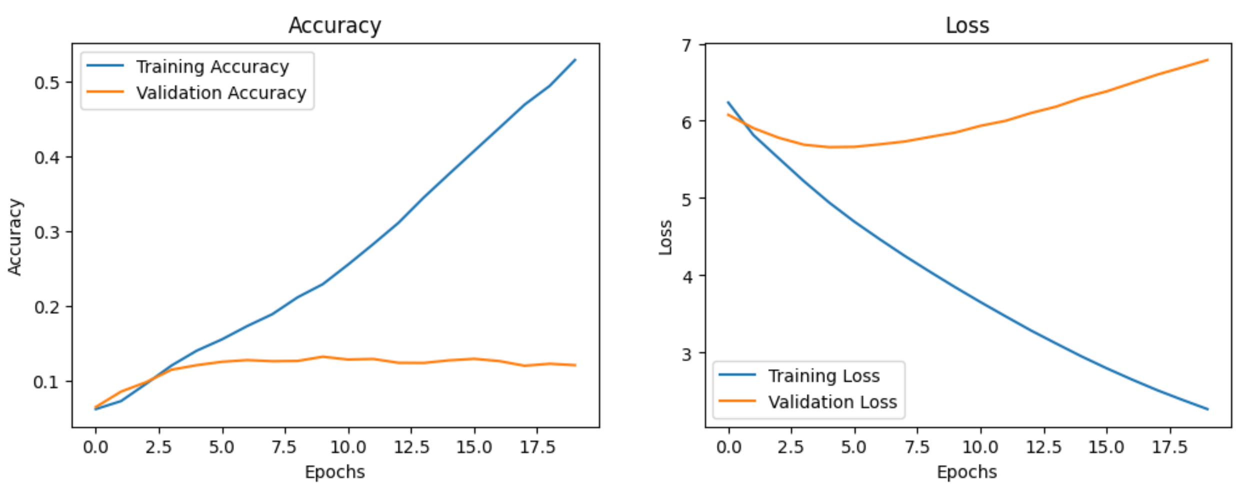
Training:

TODO: Model training

```python
# TODO: Train your model

# Train the model
history = model.fit(X_train, y_train, epochs=20, validation_data=(X_val, y_val), verbose=1)
```

```
Epoch 1/20
661/661 [==============================] - 38s 54ms/step - loss: 6.2359 - accuracy: 0.0615 - val_loss: 6.0777 - val_accuracy: 0.0644
Epoch 2/20
661/661 [==============================] - 39s 59ms/step - loss: 5.8140 - accuracy: 0.0723 - val_loss: 5.9046 - val_accuracy: 0.0850
Epoch 3/20
661/661 [==============================] - 35s 54ms/step - loss: 5.5150 - accuracy: 0.0956 - val_loss: 5.7807 - val_accuracy: 0.0977
Epoch 4/20
661/661 [==============================] - 40s 60ms/step - loss: 5.2178 - accuracy: 0.1199 - val_loss: 5.6905 - val_accuracy: 0.1144
Epoch 5/20
661/661 [==============================] - 36s 54ms/step - loss: 4.9420 - accuracy: 0.1398 - val_loss: 5.6589 - val_accuracy: 0.1204
Epoch 6/20
661/661 [==============================] - 35s 53ms/step - loss: 4.6945 - accuracy: 0.1550 - val_loss: 5.6633 - val_accuracy: 0.1250
Epoch 7/20
661/661 [==============================] - 38s 58ms/step - loss: 4.4684 - accuracy: 0.1728 - val_loss: 5.6956 - val_accuracy: 0.1272
Epoch 8/20
661/661 [==============================] - 35s 53ms/step - loss: 4.2507 - accuracy: 0.1888 - val_loss: 5.7322 - val_accuracy: 0.1257
Epoch 9/20
661/661 [==============================] - 35s 53ms/step - loss: 4.0456 - accuracy: 0.2113 - val_loss: 5.7914 - val_accuracy: 0.1261
Epoch 10/20
661/661 [==============================] - 40s 61ms/step - loss: 3.8456 - accuracy: 0.2289 - val_loss: 5.8480 - val_accuracy: 0.1318
Epoch 11/20
661/661 [==============================] - 36s 54ms/step - loss: 3.6539 - accuracy: 0.2551 - val_loss: 5.9352 - val_accuracy: 0.1280
Epoch 12/20
661/661 [==============================] - 36s 54ms/step - loss: 3.4687 - accuracy: 0.2827 - val_loss: 6.0001 - val_accuracy: 0.1287
Epoch 13/20
661/661 [==============================] - 41s 62ms/step - loss: 3.2867 - accuracy: 0.3112 - val_loss: 6.1006 - val_accuracy: 0.1236
Epoch 14/20
661/661 [==============================] - 36s 54ms/step - loss: 3.1178 - accuracy: 0.3450 - val_loss: 6.1833 - val_accuracy: 0.1234
Epoch 15/20
661/661 [==============================] - 42s 64ms/step - loss: 2.9521 - accuracy: 0.3764 - val_loss: 6.2946 - val_accuracy: 0.1268
Epoch 16/20
661/661 [==============================] - 35s 53ms/step - loss: 2.7976 - accuracy: 0.4074 - val_loss: 6.3802 - val_accuracy: 0.1289
Epoch 17/20
661/661 [==============================] - 35s 53ms/step - loss: 2.6525 - accuracy: 0.4383 - val_loss: 6.4881 - val_accuracy: 0.1259
Epoch 18/20
661/661 [==============================] - 39s 58ms/step - loss: 2.5146 - accuracy: 0.4695 - val_loss: 6.5964 - val_accuracy: 0.1197
Epoch 19/20
661/661 [==============================] - 35s 53ms/step - loss: 2.3892 - accuracy: 0.4947 - val_loss: 6.6908 - val_accuracy: 0.1223
Epoch 20/20
661/661 [==============================] - 40s 60ms/step - loss: 2.2686 - accuracy: 0.5294 - val_loss: 6.7869 - val_accuracy: 0.1204
```

Visualization:



Yes, the model is overfitting

Model 1 shows substantial overfitting, with a significant disparity between training and validation performance.

ECE657

The architecture, while capable of learning the training data, does not generalize well to unseen data, highlighting the need for techniques to mitigate overfitting, such as regularization, dropout layers, or more complex architectures.

My Model (LayerNormalization):

```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense, Bidirectional, Dropout, LayerNormalization
import matplotlib.pyplot as plt

# Build the improved model with LayerNormalization
improved_model = Sequential()
improved_model.add(Embedding(input_dim=total_words, output_dim=100, input_length=max_sequence_length - 1))
improved_model.add(Bidirectional(LSTM(150, return_sequences=True)))
improved_model.add(Dropout(0.2))
improved_model.add(LayerNormalization())
improved_model.add(LSTM(100))
improved_model.add(Dense(total_words, activation='softmax'))

# Compile the improved model
improved_model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

# Print the improved model summary
improved_model.summary()

# Train the improved model
improved_history = improved_model.fit(X_train, y_train, epochs=20, validation_data=(X_val, y_val), verbose=1)

# Plot training and validation accuracy and loss for the improved model
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(improved_history.history['accuracy'], label='Training Accuracy')
plt.plot(improved_history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Improved Model Accuracy')

plt.subplot(1, 2, 2)
plt.plot(improved_history.history['loss'], label='Training Loss')
plt.plot(improved_history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.title('Improved Model Loss')

plt.show()

# Check if the improved model is overfitting
if min(improved_history.history['val_loss']) < min(improved_history.history['loss']):
    print("The improved model is not overfitting.")
else:
    print("The improved model is overfitting.")
```

Summary:

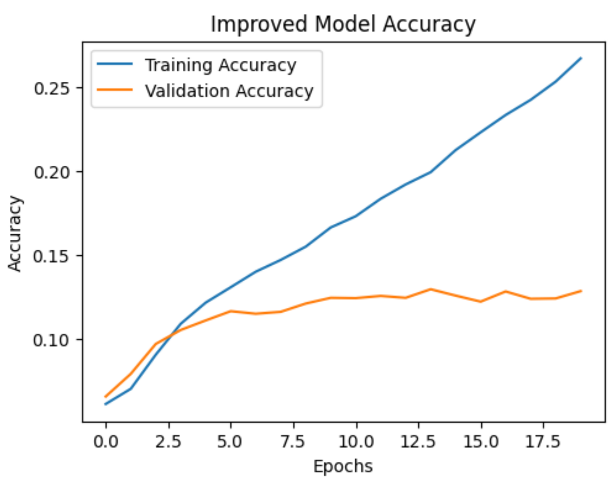
```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, 15, 100)           275100

 bidirectional (Bidirection  (None, 15, 300)           301200
 al)

 dropout (Dropout)           (None, 15, 300)           0

 layer_normalization (Layer  (None, 15, 300)           600
 Normalization)

 lstm_2 (LSTM)               (None, 100)               160400

 dense_1 (Dense)             (None, 2751)              277851

=================================================================
Total params: 1015151 (3.87 MB)
Trainable params: 1015151 (3.87 MB)
Non-trainable params: 0 (0.00 Byte)
```

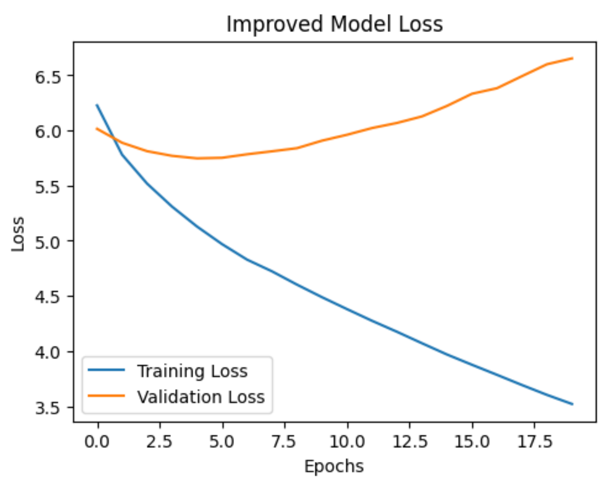
ECE657

Training:

```
____________________________________________________________________
Epoch 1/20
661/661 [==============================] - 100s 137ms/step - loss: 6.2274 - accuracy: 0.0612 - val_loss: 6.0151 - val_accuracy: 0.0657
Epoch 2/20
661/661 [==============================] - 88s 134ms/step - loss: 5.7810 - accuracy: 0.0702 - val_loss: 5.8889 - val_accuracy: 0.0791
Epoch 3/20
661/661 [==============================] - 89s 134ms/step - loss: 5.5187 - accuracy: 0.0905 - val_loss: 5.8126 - val_accuracy: 0.0969
Epoch 4/20
661/661 [==============================] - 89s 135ms/step - loss: 5.3097 - accuracy: 0.1090 - val_loss: 5.7704 - val_accuracy: 0.1053
Epoch 5/20
661/661 [==============================] - 94s 143ms/step - loss: 5.1289 - accuracy: 0.1215 - val_loss: 5.7474 - val_accuracy: 0.1109
Epoch 6/20
661/661 [==============================] - 88s 133ms/step - loss: 4.9697 - accuracy: 0.1307 - val_loss: 5.7526 - val_accuracy: 0.1164
Epoch 7/20
661/661 [==============================] - 92s 139ms/step - loss: 4.8291 - accuracy: 0.1399 - val_loss: 5.7843 - val_accuracy: 0.1149
Epoch 8/20
661/661 [==============================] - 84s 128ms/step - loss: 4.7228 - accuracy: 0.1469 - val_loss: 5.8114 - val_accuracy: 0.1161
Epoch 9/20
661/661 [==============================] - 93s 141ms/step - loss: 4.6022 - accuracy: 0.1548 - val_loss: 5.8398 - val_accuracy: 0.1210
Epoch 10/20
661/661 [==============================] - 91s 137ms/step - loss: 4.4883 - accuracy: 0.1662 - val_loss: 5.9078 - val_accuracy: 0.1244
Epoch 11/20
661/661 [==============================] - 87s 131ms/step - loss: 4.3802 - accuracy: 0.1729 - val_loss: 5.9613 - val_accuracy: 0.1242
Epoch 12/20
661/661 [==============================] - 82s 124ms/step - loss: 4.2745 - accuracy: 0.1833 - val_loss: 6.0222 - val_accuracy: 0.1255
Epoch 13/20
661/661 [==============================] - 91s 138ms/step - loss: 4.1747 - accuracy: 0.1918 - val_loss: 6.0686 - val_accuracy: 0.1244
Epoch 14/20
661/661 [==============================] - 89s 135ms/step - loss: 4.0708 - accuracy: 0.1991 - val_loss: 6.1277 - val_accuracy: 0.1295
Epoch 15/20
661/661 [==============================] - 86s 130ms/step - loss: 3.9688 - accuracy: 0.2123 - val_loss: 6.2230 - val_accuracy: 0.1257
Epoch 16/20
661/661 [==============================] - 85s 128ms/step - loss: 3.8756 - accuracy: 0.2228 - val_loss: 6.3331 - val_accuracy: 0.1221
Epoch 17/20
661/661 [==============================] - 87s 132ms/step - loss: 3.7845 - accuracy: 0.2332 - val_loss: 6.3834 - val_accuracy: 0.1282
Epoch 18/20
661/661 [==============================] - 90s 136ms/step - loss: 3.6920 - accuracy: 0.2421 - val_loss: 6.4927 - val_accuracy: 0.1238
Epoch 19/20
661/661 [==============================] - 89s 134ms/step - loss: 3.6026 - accuracy: 0.2530 - val_loss: 6.6006 - val_accuracy: 0.1240
Epoch 20/20
661/661 [==============================] - 88s 133ms/step - loss: 3.5202 - accuracy: 0.2668 - val_loss: 6.6537 - val_accuracy: 0.1284
```

Visualization:





Comparison:

- Model 2 showed an improvement over Model 1 in terms of overfitting and generalization, although it was more computationally intensive and had a slower training process.
- The use of Bidirectional LSTM, Dropout, and Layer Normalization in Model 2 contributed to slightly better validation performance, suggesting these techniques helped mitigate overfitting compared to the simpler LSTM-based Model 1.
- For further improvement, additional techniques like tuning dropout rates, increasing the number of epochs, or exploring more advanced architectures such as Attention mechanisms could be explored.

Q6.6:

TODO: Generate text

```
def generate_text(model, tokenizer, seed_text, next_words, max_sequence_len, temperature=1.0):
    result = seed_text
    for _ in range(next_words):
        # Tokenize the current text
        token_list = tokenizer.texts_to_sequences([result])[0]
        # Pad the tokenized text to the required sequence length
        token_list = pad_sequences([token_list], maxlen=max_sequence_len-1, padding='pre')
        # Predict the logits for the next word
        predictions = model.predict(token_list, verbose=0)
        # Adjust the logits by the temperature parameter
        predictions = np.asarray(predictions).astype('float64')
        predictions = np.log(predictions + 1e-7) / temperature
        exp_preds = np.exp(predictions)
        predictions = exp_preds / np.sum(exp_preds)
        # Sample the next word's index based on these probabilities
        probas = np.random.multinomial(1, predictions[0], 1)
        next_index = np.argmax(probas)
        # Map the index back to the corresponding word using the tokenizer
        next_word = tokenizer.index_word[next_index]
        # Append the word to the current text
        result += " " + next_word
    return result

# Assuming tokenizer and model are already defined and trained
seed_text = "Forest is"
next_words = 10

# Generate text with temperature 0.05
generated_text_low_temp = generate_text(model, tokenizer, seed_text, next_words, max_sequence_len, temperature=0.05)
print(f"Generated text with temperature 0.05: {generated_text_low_temp}")

# Generate text with temperature 1.5
generated_text_high_temp = generate_text(model, tokenizer, seed_text, next_words, max_sequence_len, temperature=1.5)
print(f"Generated text with temperature 1.5: {generated_text_high_temp}")
```

```
Generated text with temperature 0.05: Forest is that the mock turtle went on the trumpet and looked
Generated text with temperature 1.5: Forest is next different theyre a violent mary family silence and its
```

Conclusion:

Contextual Importance: Stop words play a critical role in maintaining sentence structure and meaning. They ensure grammatical integrity and the natural flow of language, which is essential for generating coherent text.

Sequential Dependencies: Text generation models depend on the sequential nature of language. Stop words provide necessary context and establish dependencies between words, crucial for understanding and predicting subsequent words in a sequence.

Training Data Completeness: Removing stop words can result in incomplete training data, hindering the model's ability to learn proper language usage and structure. This can affect the overall performance of the text generation model.

Real-World Application: Text generation models are expected to produce human-like text, which naturally includes stop words. This is especially important in applications like chatbots and automated content creation, where fluent and natural language is essential.

In conclusion, retaining stop words in text generation tasks is crucial as they are integral to producing coherent, grammatically correct, and natural-sounding language.