

Project Description: Next Word Prediction Using LSTM

Project Overview:

This project aims to develop a deep learning model for predicting the next word in a given sequence of words. The model is built using Long Short-Term Memory (LSTM) networks, which are well-suited for sequence prediction tasks. The project includes the following steps:

- 1- Data Collection: We use the text of Shakespeare's "Hamlet" as our dataset. This rich, complex text provides a good challenge for our model.
- 2- Data Preprocessing: The text data is tokenized, converted into sequences, and padded to ensure uniform input lengths. The sequences are then split into training and testing sets.
- 3- Model Building: An LSTM model is constructed with an embedding layer, two LSTM layers, and a dense output layer with a softmax activation function to predict the probability of the next word.
- 4- Model Training: The model is trained using the prepared sequences, with early stopping implemented to prevent overfitting. Early stopping monitors the validation loss and stops training when the loss stops improving.
- 5- Model Evaluation: The model is evaluated using a set of example sentences to test its ability to predict the next word accurately.
- 6- Deployment: A Streamlit web application is developed to allow users to input a sequence of words and get the predicted next word in real-time.

Data Collection

```
In [ ]: import nltk
import pandas as pd
import tensorflow as tf

import warnings
warnings.filterwarnings('ignore')

from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.model_selection import train_test_split
```

Data Preprocessing

```
In [3]: # Credits for data: https://www.gutenberg.org/files/1661/1661-0.txt
with open('1661-0.txt', 'r', encoding='utf-8') as file:
    text = file.read()

tokenizer = Tokenizer()
tokenizer.fit_on_texts([text])
total_words = len(tokenizer.word_index) + 1
total_words
```

```
Out[3]: 8932
```

```
In [10]: list(tokenizer.word_index.items())[:10]
```

```
Out[10]: [('the', 1),
('and', 2),
('to', 3),
('of', 4),
('a', 5),
('i', 6),
('''', 7),
('in', 8),
('that', 9),
('it', 10)]
```

```
In [5]: input_sequences = []
for line in text.split('\n'):
    token_list = tokenizer.texts_to_sequences([line])[0]
    for i in range(1, len(token_list)):
        n_gram_sequence = token_list[:i+1]
        input_sequences.append(n_gram_sequence)
```

```
In [6]: input_sequences[:10]
```

```
Out[6]: [[145, 4790],  
         [145, 4790, 1],  
         [145, 4790, 1, 1020],  
         [145, 4790, 1, 1020, 4],  
         [145, 4790, 1, 1020, 4, 128],  
         [145, 4790, 1, 1020, 4, 128, 34],  
         [145, 4790, 1, 1020, 4, 128, 34, 45],  
         [145, 4790, 1, 1020, 4, 128, 34, 45, 611],  
         [145, 4790, 1, 1020, 4, 128, 34, 45, 611, 2235],  
         [145, 4790, 1, 1020, 4, 128, 34, 45, 611, 2235, 2236]]
```

```
In [12]: # Pad Sequences  
max_sequence_len = max([len(x) for x in input_sequences])
```

```
In [13]: max_sequence_len
```

```
Out[13]: 20
```

```
In [14]: input_sequences = pad_sequences(input_sequences, maxlen=max_sequence_len, padding='pre')  
input_sequences
```

```
Out[14]: array([[ 0,  0,  0, ...,  0, 145, 4790],  
                 [ 0,  0,  0, ..., 145, 4790,  1],  
                 [ 0,  0,  0, ..., 4790,  1, 1020],  
                 ...,  
                 [ 0,  0,  0, ...,  3, 360,  83],  
                 [ 0,  0,  0, ..., 360,  83, 358],  
                 [ 0,  0,  0, ..., 83, 358, 1673]])
```

```
In [15]: # Split the data into predictors and Label  
X = input_sequences[:, :-1]  
y = input_sequences[:, -1]
```

```
In [16]: X
```

```
Out[16]: array([[ 0,  0,  0, ...,  0,  0, 145],  
                 [ 0,  0,  0, ...,  0, 145, 4790],  
                 [ 0,  0,  0, ..., 145, 4790,  1],  
                 ...,  
                 [ 0,  0,  0, ..., 8931,  3, 360],  
                 [ 0,  0,  0, ...,  3, 360,  83],  
                 [ 0,  0,  0, ..., 360,  83, 358]])
```

```
In [17]: y
```

```
Out[17]: array([4790, 1, 1020, ..., 83, 358, 1673])
```

```
In [18]: y = tf.keras.utils.to_categorical(y, num_classes=total_words)
```

```
In [19]: # Split train test  
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
In [21]: # Define early stopping  
from tensorflow.keras.callbacks import EarlyStopping  
early_stopping = EarlyStopping(monitor='val_loss',  
                               patience=5,  
                               restore_best_weights=True)
```

```
In [22]: from tensorflow.keras.models import Sequential  
from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout, GRU  
  
model = Sequential([  
    Embedding(input_dim = total_words, output_dim = 100, input_length=max_sequence_len-1),  
    LSTM(150, return_sequences=True),  
    Dropout(0.4), # Increased dropout  
    LSTM(100),  
    Dense(100, activation='relu'),  
    Dropout(0.4), # Increased dropout  
    Dense(total_words, activation='softmax')  
])
```

```
In [23]: from tensorflow.keras.optimizers import Adam  
  
# Use a much smaller Learning rate  
optimizer = Adam(learning_rate=0.0001)  
  
model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'])
```

```
In [ ]: history = model.fit(  
        X_train,  
        y_train,  
        epochs=50,  
        verbose=1,
```

```

        validation_data=(X_test, y_test),
        callbacks=[early_stopping]
    )

Epoch 1/150
2541/2541 95s 38ms/step - accuracy: 0.0516 - loss: 6.5116 - val_accuracy: 0.0529 - val_loss: 6.4201
Epoch 2/150
2541/2541 107s 42ms/step - accuracy: 0.0526 - loss: 6.3337 - val_accuracy: 0.0533 - val_loss: 6.4200
Epoch 3/150
2541/2541 103s 41ms/step - accuracy: 0.0594 - loss: 6.2330 - val_accuracy: 0.0632 - val_loss: 6.3732
Epoch 4/150
2541/2541 107s 42ms/step - accuracy: 0.0651 - loss: 6.1381 - val_accuracy: 0.0645 - val_loss: 6.3791
Epoch 5/150
2541/2541 106s 42ms/step - accuracy: 0.0664 - loss: 6.0573 - val_accuracy: 0.0659 - val_loss: 6.3766
Epoch 6/150
2541/2541 105s 41ms/step - accuracy: 0.0690 - loss: 5.9958 - val_accuracy: 0.0678 - val_loss: 6.3870
Epoch 7/150
2541/2541 106s 42ms/step - accuracy: 0.0707 - loss: 5.9426 - val_accuracy: 0.0713 - val_loss: 6.3920
Epoch 8/150
2541/2541 105s 41ms/step - accuracy: 0.0737 - loss: 5.8954 - val_accuracy: 0.0727 - val_loss: 6.4076

```

In [26]: `model.summary()`

Model: "sequential"

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 19, 100)	893,200
lstm (LSTM)	(None, 19, 150)	150,600
dropout (Dropout)	(None, 19, 150)	0
lstm_1 (LSTM)	(None, 100)	100,400
dense (Dense)	(None, 100)	10,100
dropout_1 (Dropout)	(None, 100)	0
dense_1 (Dense)	(None, 8932)	902,132

Total params: 6,169,298 (23.53 MB)

Trainable params: 2,056,432 (7.84 MB)

Non-trainable params: 0 (0.00 B)

Optimizer params: 4,112,866 (15.69 MB)

GRU

```

In [27]: model_gru = Sequential(
    [
        Embedding(total_words, 100, input_length=max_sequence_len-1),
        GRU(150, return_sequences=True),
        Dropout(0.2),
        GRU(100),
        Dense(total_words, activation='softmax')
    ]
)
model_gru.compile(loss="categorical_crossentropy", optimizer='adam', metrics=['accuracy'])

```

```

In [28]: history_gru = model_gru.fit(
    X_train,
    y_train,
    epochs=20,
    verbose=1,
    validation_data=(X_test, y_test),
    callbacks=[early_stopping]
)

```

```

Epoch 1/20
2541/2541 97s 36ms/step - accuracy: 0.0614 - loss: 6.4302 - val_accuracy: 0.0807 - val_loss: 6.0685
Epoch 2/20
2541/2541 95s 37ms/step - accuracy: 0.1050 - loss: 5.6977 - val_accuracy: 0.1195 - val_loss: 5.8233
Epoch 3/20
2541/2541 87s 34ms/step - accuracy: 0.1337 - loss: 5.3146 - val_accuracy: 0.1313 - val_loss: 5.7462
Epoch 4/20
2541/2541 81s 32ms/step - accuracy: 0.1514 - loss: 5.0269 - val_accuracy: 0.1387 - val_loss: 5.7717
Epoch 5/20
2541/2541 79s 31ms/step - accuracy: 0.1640 - loss: 4.7787 - val_accuracy: 0.1404 - val_loss: 5.8185
Epoch 6/20
2541/2541 80s 32ms/step - accuracy: 0.1749 - loss: 4.5568 - val_accuracy: 0.1395 - val_loss: 5.8727
Epoch 7/20
2541/2541 85s 33ms/step - accuracy: 0.1888 - loss: 4.3544 - val_accuracy: 0.1420 - val_loss: 5.9079
Epoch 8/20
2541/2541 82s 32ms/step - accuracy: 0.2034 - loss: 4.1742 - val_accuracy: 0.1410 - val_loss: 5.9739

```

```
In [29]: def predict_next_word(model, tokenizer, text_sequence, max_sequence_len):
    token_list = tokenizer.texts_to_sequences([text_sequence])[0]

    if len(token_list) >= max_sequence_len - 1:
        token_list = token_list[-(max_sequence_len - 1):]
    token_list = pad_sequences([token_list], maxlen=max_sequence_len-1, padding='pre')
    predicted = model.predict(token_list, verbose=0)
    predicted_word_index = np.argmax(predicted, axis=-1)[0]

    for word, index in tokenizer.word_index.items():
        if index == predicted_word_index:
            return word
    return None
```

```
In [38]: input_text = "I will leave if they"
predicted_word = predict_next_word(model_gru, tokenizer, input_text, max_sequence_len)
print(f"Input: '{input_text}' -> \nPredicted next word: '{predicted_word}'")
```

Input: 'I will leave if they' ->
Predicted next word: 'have'

```
In [39]: model_gru.save('next_word_gru_model.h5')

# Save the model
import pickle
with open('tokenizer.pkl', 'wb') as f:
    pickle.dump(tokenizer, f, protocol=pickle.HIGHEST_PROTOCOL)
```

WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.