Name: Prabhu Satya dev velpula

Student ID: 23093818

Long Short-Term Memory (LSTM) Networks

1. Introduction

Long Short-Term Memory (LSTM) networks are a specialized form of recurrent neural network (RNN) used to handle the problems of normal RNNs in managing long-term dependencies. RNNs suffer from vanishing and exploding gradients for long sequences, which can cause it to be hard to learn useful temporal relations. LSTMs address this by employing memory cells that can selectively forget and remember.

LSTMs are currently a core application in natural language processing (NLP), speech recognition, and time-series forecasting. LSTMs are particularly suited for sentiment analysis, machine translation, and stock market forecasting (Ketkar, 2017). In this article, we discuss:

- LSTM architecture and operations

- LSTM applications to sequential data

- Step-by-step deployment with Keras to conduct sentiment analysis on the IMDB movie reviews dataset

2. LSTM Model

LSTMs are memory cells that help in long-term dependency management. LSTMs differ from regular RNNs on hidden states in that they employ a list of gates to control the flow of information.

Forget Gate

This gate regulates what to discard from the cell state. It considers the previous hidden state and current input, passes it through a sigmoid function, and outputs a value between 0 and 1. The closer to 0, the more information forgotten; the closer to 1, the more information retained.

Input Gate

The input gate decides what new information is to be added to the cell state. It uses a sigmoid activation function to determine relevance and a tanh function to calculate a new candidate value to be added.

Output Gate

This gate decides which part of the cell state must be passed on to the next hidden state. It uses a sigmoid function to threshold critical information and a tanh function to normalize the output.

These operations allow LSTMs to preserve valuable information in long sequences and remove redundant information (Demirag, 2024).

3. Application of LSTM

LSTMs are applied in many sequential tasks like:

- Time-Series Forecasting: Forecasting future trends from past trends, for example, stock prices or weather.
- Natural Language Processing: Text classification, sentiment analysis, and machine translation procedures.
- Speech-to-Text: Voice assistant speech to text and automatic transcription.
- Handwriting Recognition: Handwritten character recognition in OCR.

4. Python Implementation of LSTM

We implement an LSTM model using Keras here for sentiment analysis on the IMDB movie reviews dataset. The dataset consists of 50,000 positive and negative labeled reviews.

```
# Step 1: Load and Preprocess the Data

from tensorflow.keras.datasets import imdb

from tensorflow.keras.preprocessing.sequence import pad_sequences

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout

import numpy as np

import matplotlib.pyplot as plt
```

```
# Load IMDB dataset
(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=10000)

# Pad sequences to have even length
X_train = pad_sequences(X_train, maxlen=500)
X_test = pad_sequences(X_test, maxlen=500)

# Step 2: Build the LSTM Model
# Build the LSTM model
model = Sequential()

# Embedding layer to discover representations of words
model.add(Embedding(input_dim=10000, output_dim=128, input_length=500))

# LSTM layer with 100 units
model.add(LSTM(100))

# Dropout layer to combat overfitting
model.add(Dropout(0.5))

# Fully connected output layer with sigmoid activation for binary classification
model.add(Dense(1, activation='sigmoid'))

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

# Step 3: Train the LSTM Model
```

Train the LSTM model

history = model.fit(X_train, y_train, epochs=3, batch_size=64, validation_data=(X_test, y_test))

```
Epoch 1/3
391/391 ━━━━━━━━━━━━━━━━━━━━ 677s 2s/step - accuracy: 0.7107 - loss: 0.5435 - val_accuracy: 0.8418 - val_loss: 0.3733
Epoch 2/3
391/391 ━━━━━━━━━━━━━━━━━━━━ 683s 2s/step - accuracy: 0.8619 - loss: 0.3322 - val_accuracy: 0.8372 - val_loss: 0.3848
Epoch 3/3
391/391 ━━━━━━━━━━━━━━━━━━━━ 647s 2s/step - accuracy: 0.8810 - loss: 0.2949 - val_accuracy: 0.8346 - val_loss: 0.3795
```

Step 4: Evaluate the Model

Evaluate the model on the test data

loss, accuracy = model.evaluate(X_test, y_test)

print(f"Test Accuracy: {accuracy * 100:.2f}%")

```
Test accuracy: 83.46%
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 149s 190ms/step - accuracy: 0.8323 - loss: 0.3860
Test accuracy: 83.46%
```

Step 5: Visualize Training History

Plot training & validation accuracy

plt.plot(history.history['accuracy'])

plt.plot(history.history['val_accuracy'])

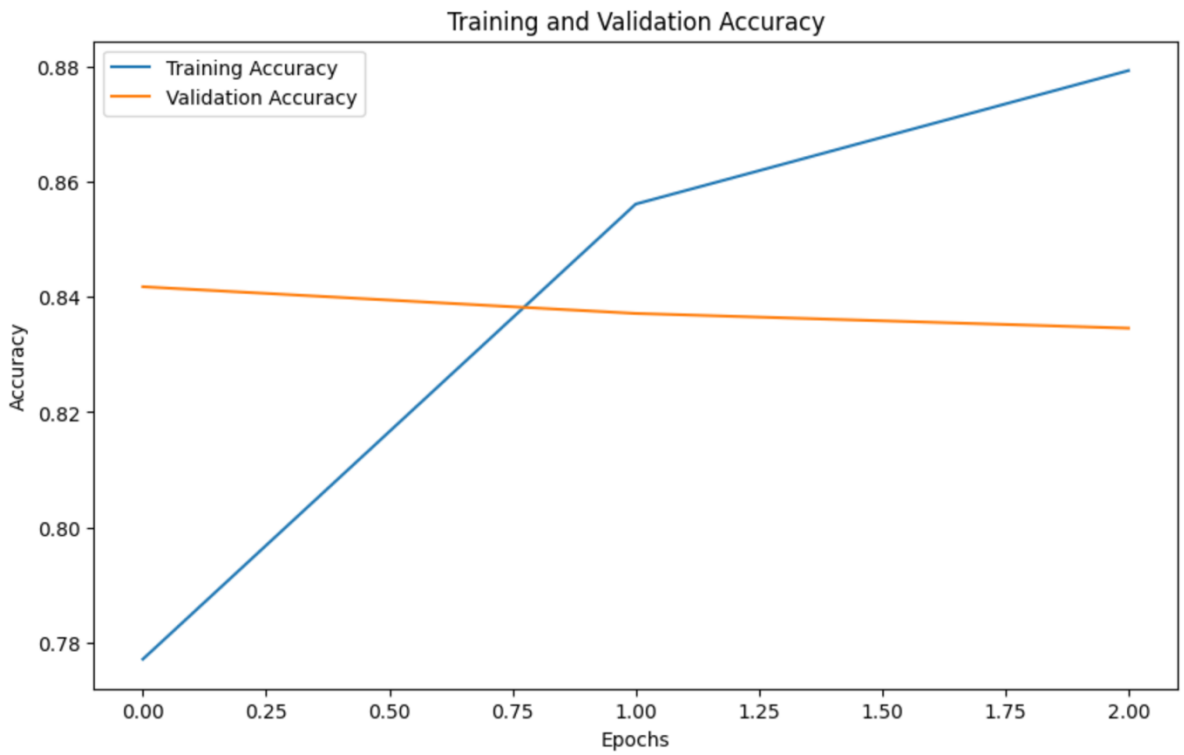
plt.title('Model Accuracy')

plt.xlabel('Epoch')

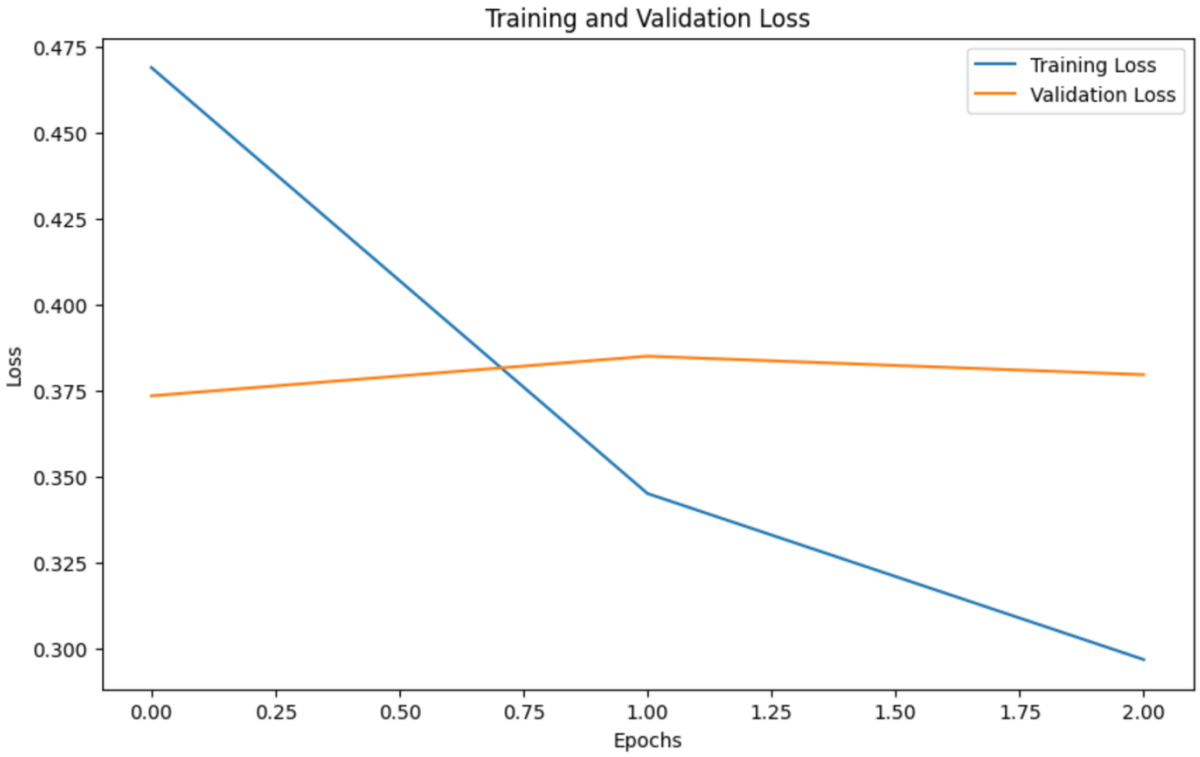
plt.ylabel('Accuracy')

plt.legend(['Train', 'Test'], loc='upper left')

plt.show()



Validation Accuracy



Validation Loss

Visualization for predictions:

```
Review 1: Negative
Prediction Probabilities: [0.64483577 0.35516423]
--------------------------------------------------
Review 2: Positive
Prediction Probabilities: [0.01493337 0.98506665]
--------------------------------------------------
Review 3: Positive
Prediction Probabilities: [0.3632985 0.6367015]
--------------------------------------------------
Review 4: Positive
Prediction Probabilities: [0.32289892 0.6771011 ]
--------------------------------------------------
Review 5: Positive
Prediction Probabilities: [0.00217351 0.9978265 ]
--------------------------------------------------
```

5. Evaluation and Conclusion

Evaluation

After the training of the LSTM model, we evaluate its performance on the test set. The accuracy metric determines if the model correctly classifies the movie review sentiments or not. The dropout layers avoid overfitting, which consequently improves the generalization on unseen data.

Conclusion

LSTMs are highly effective for sequential data tasks and work incredibly well in tasks such as sentiment analysis, language models, and time-series prediction. Because LSTMs incorporate memory cells, the vanishing gradient problem witnessed with regular RNNs is circumvented, and the LSTMs become more adept at learning long-term dependencies without redundant computation (Ketkar, 2017).

GitHub Repository:

GitHub Link: https://github.com/Satya-dev123/Long-Short-Term-Memory-LSTM-Networks.git

6. References

1. Demirag, Yigit. (2024). Analog Alchemy: Neural Computation with In-Memory Inference, Learning and Routing.

2. Ketkar, Nikhil. (2017). Deep Learning with Python. Manning Publications.

3. TensorFlow documentation: https://www.tensorflow.org (accessed on March 20, 2024).

Appendix

```
# Step 1: Import necessary libraries

import numpy as np

import matplotlib.pyplot as plt

import tensorflow as tf

from tensorflow.keras import layers, models

from tensorflow.keras.datasets import imdb

from tensorflow.keras.preprocessing import sequence

from tensorflow.keras.utils import to_categorical


# Step 2: Load and preprocess the IMDB dataset

(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=10000)


# Step 3: Pad the sequences to ensure they all have the same length

maxlen = 500

x_train = sequence.pad_sequences(x_train, maxlen=maxlen)
```

```
x_test = sequence.pad_sequences(x_test, maxlen=maxlen)

# Step 4: Convert labels to categorical
y_train = to_categorical(y_train, 2)
y_test = to_categorical(y_test, 2)

# Step 5: Build the LSTM model
model = models.Sequential()
model.add(layers.Embedding(input_dim=10000, output_dim=128, input_length=maxlen))
model.add(layers.LSTM(100, dropout=0.2, recurrent_dropout=0.2))
model.add(layers.Dense(2, activation='softmax'))

# Step 6: Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Step 7: Train the model
history = model.fit(x_train, y_train, epochs=5, batch_size=64, validation_data=(x_test,
y_test))

# Step 8: Evaluate the model on the test data
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f"Test accuracy: {test_acc * 100:.2f}%")

# Step 9: Plot the training and validation accuracy over epochs
plt.figure(figsize=(10, 6))
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
```

```
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

# Step 10: Plot the training and validation loss over epochs
plt.figure(figsize=(10, 6))
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

# Step 11: Predict the sentiment of some sample test reviews
sample_reviews = x_test[:5]
predictions = model.predict(sample_reviews)

# Step 12: Visualize the predictions
for i, prediction in enumerate(predictions):
    print(f"Review {i+1}: {'Positive' if np.argmax(prediction) == 1 else 'Negative'}")
    print("Prediction Probabilities:", prediction)
    print("-" * 50)
```