Project Documentation

Analysis of Neural Networks for Sentiment Analysis on IMDb Movie Reviews

Information Retrieval Project

Members:
Hasan Zahid 21k-4872
Abdul Rehman 21k-4662

Presented to:
Dr. Muhammad Rafi

Dated: 4th June, 2024

Introduction

This project aims to analyze and compare the performance of different neural network models for sentiment analysis on IMDb movie reviews. The neural network architectures evaluated in this study are Convolutional Neural Network (CNN), Long Short-Term Memory Network (LSTM), and a Hybrid LSTM-CNN architecture.

Objectives

- To preprocess IMDb movie review data for sentiment analysis.
- To implement and train three different neural network models (CNN, LSTM, Hybrid LSTM-CNN).
- To evaluate and compare the performance of these models using accuracy metrics.

Data Description

The dataset used in this project consists of IMDb movie reviews. Each review is labeled with its corresponding sentiment (positive or negative). The dataset is preprocessed to convert text reviews into numerical representations suitable for input into neural network models.

Data Conversion

The dataset is initially stored in directories with separate folders for training and testing data, each containing positive and negative reviews. The following code snippet loads the dataset from these files, converts it to a DataFrame, and saves it as a CSV file for easier handling.

```python
import pyprind
import pandas as pd
import os

basepath = r'C:\Users\WINDOWS10\OneDrive\Desktop\IR\project\aclImdb'
csv_path = 'movie_reviews.csv'
# Load Dataset
labels = {'pos': 1, 'neg': 0}
pbar = pyprind.ProgBar(50000)

data = []
for s in ('test', 'train'):
    for l in ('pos', 'neg'):
        path = os.path.join(basepath, s, l)
        for file in os.listdir(path):
            with open(os.path.join(path, file), 'r', encoding='utf-8') as infile:
                txt = infile.read()
                data.append([txt, labels[l]])
                pbar.update()

df = pd.DataFrame(data, columns=['review', 'sentiment'])

# Save to CSV
df.to_csv(csv_path, index=False, encoding='utf-8')
print("Data loaded from files and saved to CSV.")
```

Methodology

Data Preprocessing

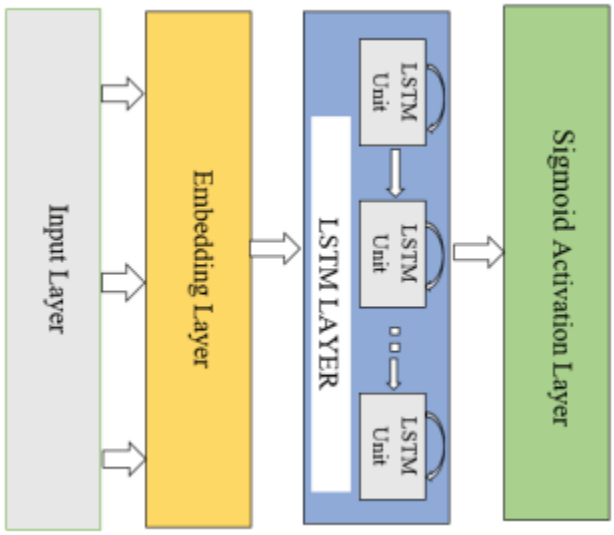
1. **Tokenization**: Convert the text data into sequences of integers using the `Tokenizer` from Keras.
2. **Padding**: Ensure all sequences have the same length by padding them to a fixed length (600 in this case).

```python
max_features=5000
tokenizer = Tokenizer(num_words=max_features, split=' ')
tokenizer.fit_on_texts(data['review'].values)
X = tokenizer.texts_to_sequences(data['review'].values)
X = pad_sequences(X,maxlen=600)
```

Model Architectures

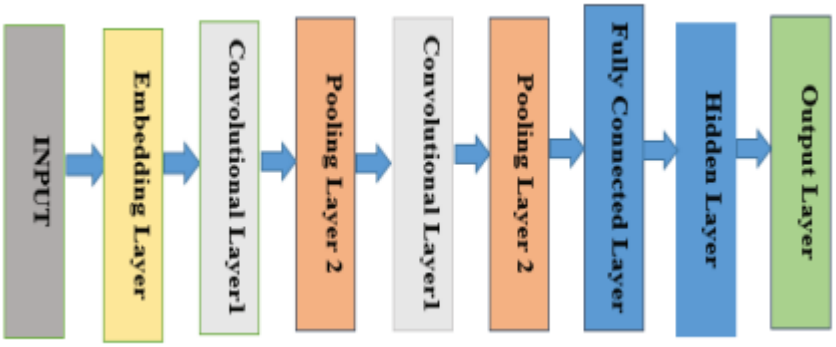
Three different neural network architectures were implemented and trained:

1. LSTM Model



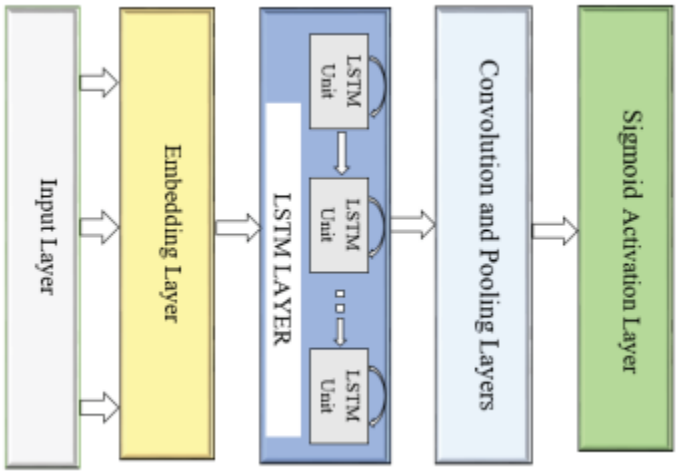

```
model = Sequential()
model.add(Embedding(max_features, embed_dim,input_length = X.shape[1]))
model.add(SpatialDropout1D(0.4))
model.add(LSTM(lstm_out, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
print(model.summary())
```

2. CNN Model



```
# Define the model
model_cnn = Sequential()
model_cnn.add(Embedding(5000, 128, input_length=X.shape[1]))  # Embedding layer
model_cnn.add(SpatialDropout1D(0.4))  # Spatial Dropout for regularization
model_cnn.add(Conv1D(filters=64, kernel_size=3, padding='same', activation='relu'))  # Convolutional Layer 1
model_cnn.add(MaxPooling1D(pool_size=2))  # Pooling Layer 1
model_cnn.add(Conv1D(filters=64, kernel_size=3, padding='same', activation='relu'))  # Convolutional Layer 2
model_cnn.add(MaxPooling1D(pool_size=2))  # Pooling Layer 2
model_cnn.add(Flatten())  # Flatten layer
model_cnn.add(Dense(256, activation='relu'))  # Fully Connected Layer
model_cnn.add(Dropout(0.5))  # Dropout for regularization
model_cnn.add(Dense(128, activation='relu'))  # Hidden Layer
model_cnn.add(Dense(1, activation='sigmoid'))  # Output layer with sigmoid activation for binary classification
```

3. Hybrid LSTM-CNN Model




```
# Define the model
model_cnn = Sequential()
model_cnn.add(Embedding(5000, 128, input_length=X.shape[1]))  # Embedding layer
model_cnn.add(SpatialDropout1D(0.4))  # Spatial Dropout for regularization
model_cnn.add(LSTM(128, dropout=0.2, recurrent_dropout=0.2, return_sequences=True))  # LSTM layer
model_cnn.add(Conv1D(filters=64, kernel_size=3, padding='same', activation='relu'))  # Convolutional layer
model_cnn.add(MaxPooling1D(pool_size=2))  # Pooling layer
model_cnn.add(Flatten())  # Flatten layer to convert 2D to 1D
model_cnn.add(Dense(256, activation='relu'))  # Fully connected layer
model_cnn.add(Dropout(0.5))  # Dropout for regularization
model_cnn.add(Dense(1, activation='sigmoid'))  # Output layer with sigmoid activation
```

Model Training

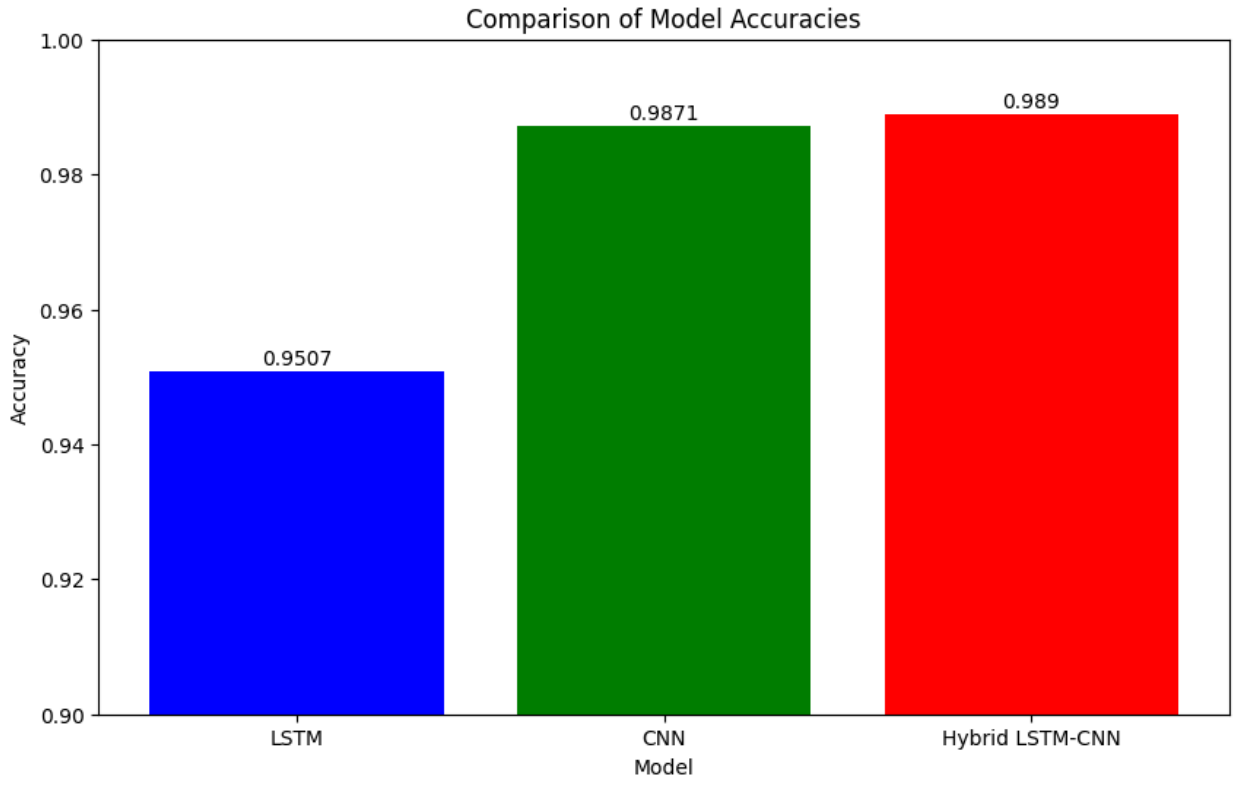
The models were trained on the preprocessed dataset using an 80-20 train-test split. The training process involved fitting the models to the training data and validating their performance on the test data.

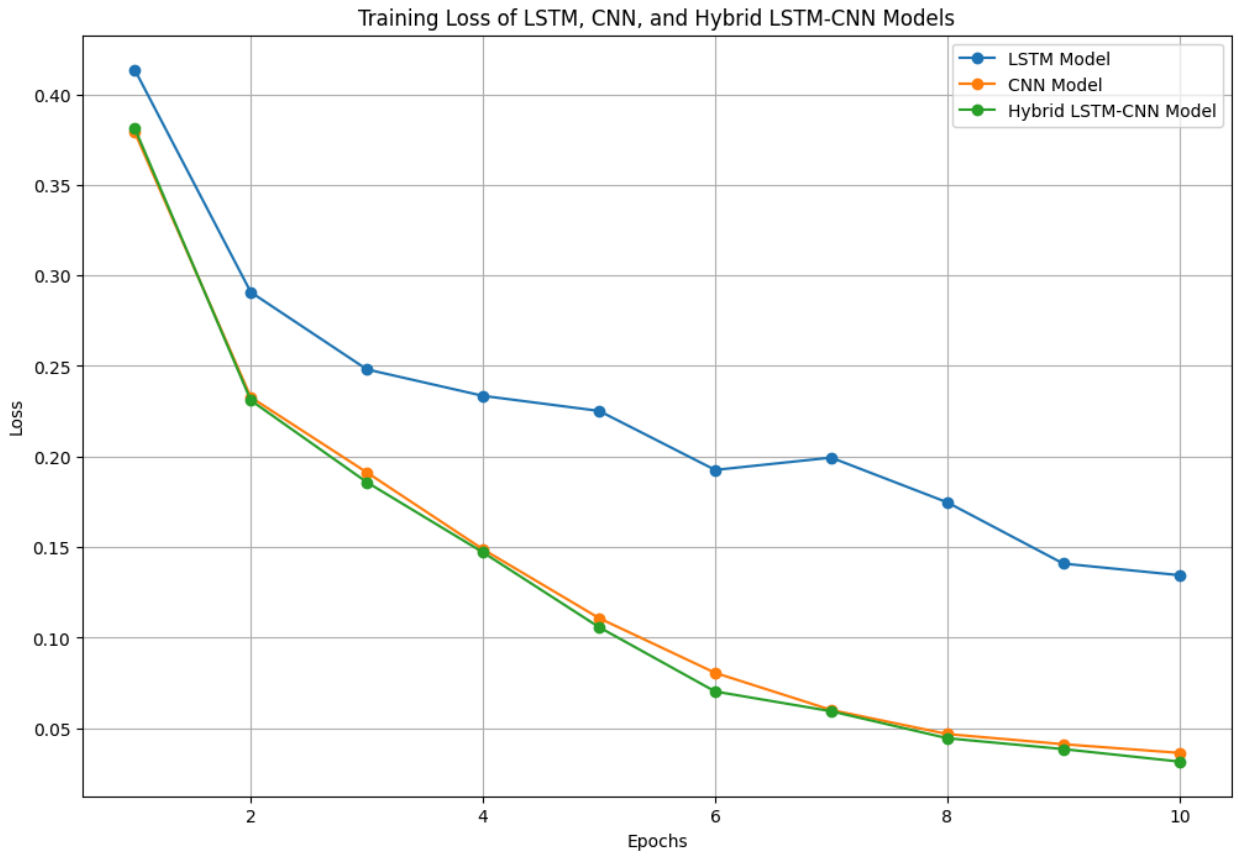
```
Y=pd.get_dummies(data['sentiment']).values
X_train, X_test, Y_train, Y_test = tts(X,Y, test_size = 0.2, random_state = 42)

batch_size = 64
model_cnn.fit(X_train, Y_train, epochs = 10, batch_size=batch_size, validation_data=(X_test,Y_test), verbose = True)
```

Evaluation

The models were evaluated based on their accuracy on the test dataset. The results for each model were compared to determine which architecture performed best for sentiment analysis on IMDb movie reviews.





Results and Discussion

Comparison of Models

LSTM Model:

Accuracy: 0.9507
Training Loss (at 10th epoch): 0.1345
Observations: The LSTM model showed a steady decrease in training loss over the epochs but had a slightly lower accuracy compared to the CNN and Hybrid models.

CNN Model:

Accuracy: 0.9871
Training Loss (at 10th epoch): 0.0364
Observations: The CNN model achieved a high accuracy with a significant reduction in training loss, indicating that it was very effective for this sentiment analysis task.

Hybrid LSTM-CNN Model:

Accuracy: 0.9890
Training Loss (at 10th epoch): 0.0316
Observations: The Hybrid LSTM-CNN model outperformed both the standalone LSTM and CNN models in terms of accuracy and training loss, showing the benefits of combining both approaches.

Conclusion

Performance: The Hybrid LSTM-CNN model provided the best performance with the highest accuracy (0.9890) and the lowest training loss (0.0316). This suggests that combining LSTM's ability to capture long-term dependencies with CNN's capability of extracting local features is highly effective for sentiment analysis.

Training Dynamics: The LSTM model, while effective, had a slower reduction in training loss and lower accuracy compared to the other models. The CNN model showed rapid improvement in both loss and accuracy but was slightly outperformed by the Hybrid model.

Recommendation: For tasks involving sentiment analysis or similar NLP tasks, the Hybrid LSTM-CNN model is recommended due to its superior performance. However, the CNN model is also a strong contender, especially when computational resources or training time are limited. The LSTM model, while not the best in this comparison, still provides valuable insights and can be preferred in scenarios where understanding long-term dependencies is crucial.

References

- Https://www.researchgate.net/publication/343046458_Performance_Analysis_of_Different_Neural_Networks_for_Sentiment_Analysis_on_IMDb_Movie_Reviews?enrichId=rgreq-126555695724815643464 7b23dd3c2c2-XXX&enrichSource=Y292ZXJQYWdlOzM0MzA0NjQ1ODtBUzo5MTQ2NDAzNDczNjEyODFAMTU5NTA3ODY2MDU5Ng%3D%3D&el=1_x_3&_esc=publicationCoverPdf