Aim

To develop and compare neural network models for sentiment analysis of movie reviews using various architectures including Simple Neural Network (SNN), Convolutional Neural Network (CNN), and Long Short-Term Memory Network (LSTM).

Objective

1. Preprocess text data from the IMDb movie reviews dataset.
2. Implement SNN, CNN, and LSTM models for sentiment analysis.
3. Evaluate and compare the performance of these models.
4. Deploy the best-performing model on unseen data for sentiment prediction.

Summary

This project focuses on building and evaluating different neural network models for sentiment analysis using IMDb movie reviews. The project involves preprocessing text data, creating word embeddings using GloVe, and training three types of neural networks: SNN, CNN, and LSTM. The performance of each model is evaluated, and the best-performing model is used to predict sentiments on unseen movie reviews. The final model and predictions are saved for future use.

Tools and Libraries Used

- **Google Colab**
- **Python**
- **Pandas**: Data manipulation and analysis
- **NumPy**: Numerical computations
- **Keras**: Building and training neural network models
- **Matplotlib**: Plotting graphs
- **Seaborn**: Data visualization
- **NLTK**: Natural language processing
- **Sklearn**: Train-test split and evaluation

Procedure

1. **Mount Google Drive**:

CODE:

```
from google.colab import drive

drive.mount('/content/drive')
```

2. **Set Working Directory**:

CODE:

```
%cd /content/drive/My Drive/1_LiveProjects/Project8_SentimentAnalysis_with_NeuralNetwork
!ls
```

3. **Import Libraries**:

CODE:

```
import pandas as pd
import numpy as np
import re
import nltk
from nltk.corpus import stopwords
from keras.preprocessing.text import one_hot, Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Activation, Dropout, Dense, Flatten, GlobalMaxPooling1D, Embedding, Conv1D, LSTM
from sklearn.model_selection import train_test_split
import seaborn as sns
import matplotlib.pyplot as plt
```

4. **Load and Explore Dataset**:

CODE:

```
movie_reviews = pd.read_csv("a1_IMDB_Dataset.csv")
sns.countplot(x='sentiment', data=movie_reviews)
```

5. **Preprocess Text Data**:

CODE:

```
TAG_RE = re.compile(r'<[^>]+>')

def remove_tags(text):
    return TAG_RE.sub('', text)
```

```
nltk.download('stopwords')


def preprocess_text(sen):
    sentence = sen.lower()
    sentence = remove_tags(sentence)
    sentence = re.sub('[^a-zA-Z]', ' ', sentence)
    sentence = re.sub(r"\s+[a-zA-Z]\s+", ' ', sentence)
    sentence = re.sub(r'\s+', ' ', sentence)
    pattern = re.compile(r'\b(' + r'|'.join(stopwords.words('english')) + r')\b\s*')
    sentence = pattern.sub('', sentence)
    return sentence

X = [preprocess_text(sen) for sen in movie_reviews['review']]
y = np.array(list(map(lambda x: 1 if x=="positive" else 0, movie_reviews['sentiment'])))
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42)
```

6. **Prepare Embedding Layer**:

CODE:

```
word_tokenizer = Tokenizer()
word_tokenizer.fit_on_texts(X_train)
X_train = word_tokenizer.texts_to_sequences(X_train)
X_test = word_tokenizer.texts_to_sequences(X_test)
vocab_length = len(word_tokenizer.word_index) + 1
maxlen = 100
X_train = pad_sequences(X_train, padding='post', maxlen=maxlen)
X_test = pad_sequences(X_test, padding='post', maxlen=maxlen)

embeddings_dictionary = dict()
glove_file = open('a2_glove.6B.100d.txt', encoding="utf8")
for line in glove_file:
```

```
    records = line.split()
    word = records[0]
    vector_dimensions = asarray(records[1:], dtype='float32')
    embeddings_dictionary[word] = vector_dimensions
glove_file.close()


embedding_matrix = zeros((vocab_length, 100))
for word, index in word_tokenizer.word_index.items():
    embedding_vector = embeddings_dictionary.get(word)
    if embedding_vector is not None:
        embedding_matrix[index] = embedding_vector
```

7. **Model Training**:
    - **Simple Neural Network**:

CODE:

```
snn_model = Sequential()
embedding_layer = Embedding(vocab_length, 100, weights=[embedding_matrix], input_length=maxlen, trainable=False)
snn_model.add(embedding_layer)
snn_model.add(Flatten())
snn_model.add(Dense(1, activation='sigmoid'))
snn_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['acc'])
snn_model_history = snn_model.fit(X_train, y_train, batch_size=128, epochs=6, verbose=1, validation_split=0.2)
score = snn_model.evaluate(X_test, y_test, verbose=1)
```

    - **Convolutional Neural Network**:

CODE:

```
cnn_model = Sequential()
embedding_layer = Embedding(vocab_length, 100, weights=[embedding_matrix], input_length=maxlen, trainable=False)
cnn_model.add(embedding_layer)
```

```
cnn_model.add(Conv1D(128, 5, activation='relu'))

cnn_model.add(GlobalMaxPooling1D())

cnn_model.add(Dense(1, activation='sigmoid'))

cnn_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['acc'])

cnn_model_history = cnn_model.fit(X_train, y_train, batch_size=128, epochs=6, verbose=1,
validation_split=0.2)

score = cnn_model.evaluate(X_test, y_test, verbose=1)
```

- **Recurrent Neural Network (LSTM)**:

CODE:

```
lstm_model = Sequential()

embedding_layer = Embedding(vocab_length, 100, weights=[embedding_matrix], input_length=maxlen,
trainable=False)

lstm_model.add(embedding_layer)

lstm_model.add(LSTM(128))

lstm_model.add(Dense(1, activation='sigmoid'))

lstm_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['acc'])

lstm_model_history = lstm_model.fit(X_train, y_train, batch_size=128, epochs=6, verbose=1,
validation_split=0.2)

score = lstm_model.evaluate(X_test, y_test, verbose=1)
```

8. **Model Performance Visualization**:

CODE:

```
plt.plot(snn_model_history.history['acc'])

plt.plot(snn_model_history.history['val_acc'])

plt.title('model accuracy')

plt.ylabel('accuracy')

plt.xlabel('epoch')

plt.legend(['train','test'], loc='upper left')

plt.show()


plt.plot(snn_model_history.history['loss'])
```

```
plt.plot(snn_model_history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train','test'], loc='upper left')
plt.show()
```

9. **Save and Load Model**:

CODE:

```
lstm_model.save(f"./c1_lstm_model_acc_{round(score[1], 3)}.h5", save_format='h5')
# from keras.models import load_model
# model_path ='./c1_lstm_model_acc_0.856.h5'
# pretrained_lstm_model = load_model(model_path)
```

10. **Predict Sentiments on Unseen Data**:

CODE:

```
sample_reviews = pd.read_csv("a3_IMDb_Unseen_Reviews.csv")
unseen_reviews = sample_reviews['Review Text']
unseen_processed = [preprocess_text(review) for review in unseen_reviews]
unseen_tokenized = word_tokenizer.texts_to_sequences(unseen_processed)
unseen_padded = pad_sequences(unseen_tokenized, padding='post', maxlen=maxlen)
unseen_sentiments = lstm_model.predict(unseen_padded)
sample_reviews['Predicted Sentiments'] = np.round(unseen_sentiments*10, 1)
sample_reviews.to_csv("./c2_IMDb_Unseen_Predictions.csv", sep=',', encoding='UTF-8')
```

Highlights

- **Data Preprocessing**: Effective cleaning and preprocessing of text data to remove HTML tags, punctuations, and stopwords.
- **Word Embeddings**: Utilization of GloVe embeddings for converting words into vector representations.
- **Multiple Model Architectures**: Implementation and comparison of SNN, CNN, and LSTM models.
- **Performance Visualization**: Plotting training and validation accuracy and loss to analyze model performance.

Conclusion

The project successfully demonstrated the implementation of neural network models for sentiment analysis. Among the models tested, the LSTM network showed the best performance in terms of accuracy. The final LSTM model was deployed to predict sentiments on new movie reviews, showcasing its practical application in real-world scenarios.