Name – Rutik Kothwala
Subject – Introduction To Deep Learning
UID – U01868702

Final Project - Sentiment Analysis using Recurrent Neural Networks (RNNs) for Movie Reviews

Sentiment Analysis

As the name suggests, it means to identify the view or emotion behind a situation. It basically means to analyze and find the emotion or intent behind a piece of text or speech or any mode of communication.

In this documents , we will focus on the sentiment analysis of text data.

We, humans, communicate with each other in a variety of languages, and any language is just a mediator or a way in which we try to express ourselves. And, whatever we speak or write, has a sentiment associated with it. It might be positive or negative or it might be neutral as well.

This document will guide you through the process of training a machine learning model to categorize movie reviews as either positive or negative, depending on the content of the review text.

Introduction:

In this step-by-step guide, we will walk through the process of building a Sentiment Analysis model using Recurrent Neural Networks (RNNs) to classify movie reviews as positive or negative. Sentiment analysis is a popular Natural Language Processing (NLP) task that involves determining the sentiment or emotion expressed in a piece of text. We will use Python, TensorFlow, and Keras to implement the RNN model and analyze the results.

What is IMDB Dataset?

The IMDB dataset consists of movie reviews from the IMDb website, along with labels indicating whether each review is “positive” or “negative” based on the reviewer’s opinion. It is designed for binary sentiment classification it has total of 50,000 reviews in it.

```
In [2]: df_reviews= pd.read_csv('C:/Users/rutik/Downloads/IMDB Dataset.csv')

In [3]: df_reviews.head()
```

Out[3]:

	review	sentiment
0	One of the other reviewers has mentioned that ...	positive
1	A wonderful little production. The...	positive
2	I thought this was a wonderful way to spend ti...	positive
3	Basically there's a family where a little boy ...	negative
4	Petter Mattei's "Love in the Time of Money" is...	positive

Import libraries

```
In [1]:
import numpy as np
import pandas as pd
import tensorflow as tf
import re
import matplotlib.pyplot as plt
import seaborn as sns
import pickle
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential,load_model
from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout
from sklearn.model_selection import train_test_split
```

First, we need to import the required libraries. TensorFlow is an open-source machine learning framework developed by Google, and Keras is a high-level neural networks API that runs on top of TensorFlow. We also import the IMDB dataset from Keras, which contains movie reviews along with their associated sentiment labels (positive or negative). We will use this dataset to train and test our RNN model.

Preprocessing the Data

mapping the sentiment labels to a binary representation The purpose of this mapping is to quantify and categorize the emotional content within the text, facilitating further analysis or machine learning applications that benefit from numerical representations.

```
In [4]: df_reviews.isnull().sum()

Out[4]: review       0
        sentiment    0
        dtype: int64

In [5]: # Map sentiment labels to numerical values
        sentiment_mapping = {'positive': 1, 'negative': 0}
        df_reviews['sentiment'] = df_reviews['sentiment'].map(sentiment_mapping)

In [6]: df_reviews
```

Out[6]:

	review	sentiment
0	One of the other reviewers has mentioned that ...	1
1	A wonderful little production. The...	1
2	I thought this was a wonderful way to spend ti...	1
3	Basically there's a family where a little boy ...	0
4	Petter Mattei's "Love in the Time of Money" is...	1
...	...	...
49995	I thought this movie did a down right good job...	1
49996	Bad plot, bad dialogue, bad acting, idiotic di...	0
49997	I am a Catholic taught in parochial elementary...	0
49998	I'm going to have to disagree with the previou...	0
49999	No one expects the Star Trek movies to be high...	0

50000 rows × 2 columns

The next step is to clean the text data by removing any irrelevant or noisy information. This typically involves converting the text to lowercase, removing special characters, punctuation, and other non-alphanumeric characters that may not contribute to sentiment analysis.

```
In [7]: # Convert text to lowercase and remove special characters
        def clean_text(text):
            text = text.lower()
            text = re.sub(r'[^a-zA-Z0-9\s]', '', text)
            return text

In [8]: df_reviews['review'] = df_reviews['review'].apply(clean_text)

In [9]: df_reviews
```

Out[9]:

	review	sentiment
0	one of the other reviewers has mentioned that ...	1
1	a wonderful little production br br the filmin...	1
2	i thought this was a wonderful way to spend ti...	1
3	basically theres a family where a little boy j...	0
4	petter matteis love in the time of money is a ...	1
...	...	...
49995	i thought this movie did a down right good job...	1
49996	bad plot bad dialogue bad acting idiotic direc...	0
49997	i am a catholic taught in parochial elementary...	0
49998	im going to have to disagree with the previous...	0
49999	no one expects the star trek movies to be high...	0

50000 rows × 2 columns

Splitting the Data and Tokenize it.

Tokenization is the process of splitting the text into individual words or tokens. Th is step is crucial as it converts the text into a format suitable for further analysis. E ach word or token represents a feature that the model can learn from.

In [10]:
```
# Split data into input text and sentiment label
X = df_reviews['review'].values
y = df_reviews['sentiment'].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [11]:
```
# Tokenize the text data
max_words = 25000
tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(X)
```

In [12]:
```
# Convert text to sequences
X_train_sequences = tokenizer.texts_to_sequences(X_train)
X_test_sequences = tokenizer.texts_to_sequences(X_test)
```

To ensure that all sequences have the same length, we pad or truncate them to a fi xed length. Padding is typically done by adding zeros to the end of shorter sequen ces, while truncating removes excess words from longer sequences.

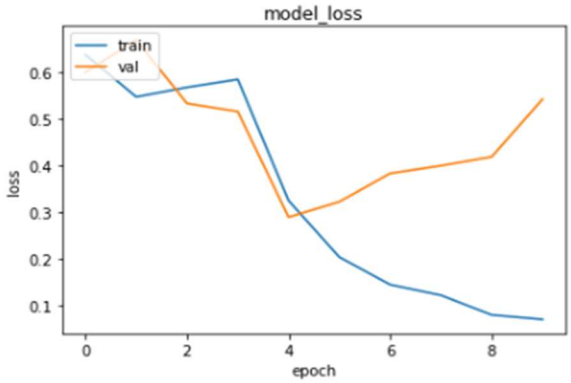
In [13]:
```
# Pad sequences to have consistent length
max_sequence_length = 200  # You can choose an appropriate length based on your data
X_train_padded = pad_sequences(X_train_sequences, maxlen=max_sequence_length, padding='post')
X_test_padded = pad_sequences(X_test_sequences, maxlen=max_sequence_length, padding='post')
```

Model Architecture and compilation.

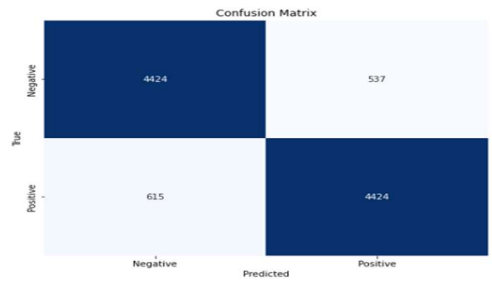
We have used sequential model with Long Short-Term Memory (LSTM) Recurrent Neural Network for binary classification using Keras. It consists of an embedding la yer, an LSTM layer with 64 units, a dense layer with 32 units and ReLU activation, a dropout layer, and a final dense layer with sigmoid activation for binary output. Th e model is compiled with binary crossentropy loss and Adam optimizer.

```
In [16]: M

# Build the LSTM Recurrent Neural Network
embedding_dim = 100
lstm_units = 64

model = Sequential()
model.add(Embedding(input_dim=vocab_size, output_dim=embedding_dim, input_length=max_sequence_lengt
model.add(LSTM(units=lstm_units))
model.add(Dense(32, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))  # Output layer with binary classification

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

model.summary()
```

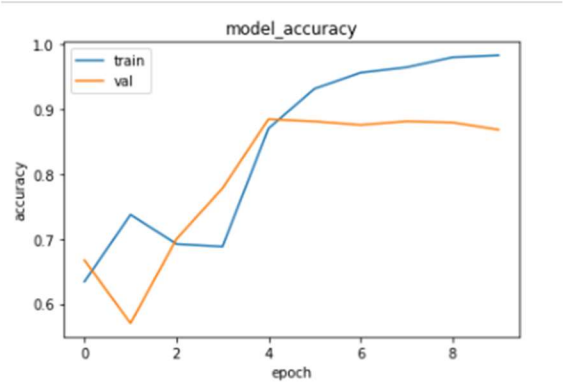
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 200, 100)          18163200

 lstm (LSTM)                 (None, 64)                42240

 dense (Dense)               (None, 32)                2080

 dropout (Dropout)           (None, 32)                0

 dense_1 (Dense)             (None, 1)                 33

=================================================================
Total params: 18,207,553
Trainable params: 18,207,553
Non-trainable params: 0
_________________________________________________________________
```



	precision	recall	f1-score	support
0	0.88	0.89	0.88	4961
1	0.89	0.88	0.88	5039
accuracy			0.88	10000
macro avg	0.88	0.88	0.88	10000
weighted avg	0.88	0.88	0.88	10000



Model Evaluation



Here we can see that our model works very well on both training data set and test data set with accuracy of 0.88, Avg recall 0.88 and f1 score of 0.88.

Fine Tunning and Optimization

We fine tune our sequential model by implementing a Gated Recurrent Unit (GRU) architecture. The model incorporates an Embedding Layer, GRU Layer, Dense Lay er with ReLU activation, Dropout Layer for regularization, and a final Dense Layer f or binary classification. Compiled with binary crossentropy loss and Adam optimize r, this model is designed for fine-tuning and optimization.

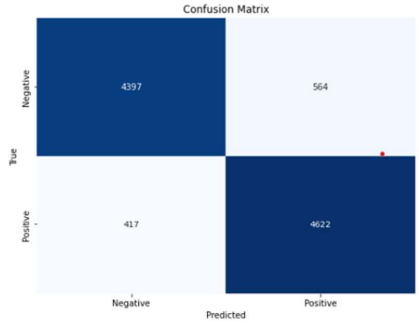
Fine Tunning and Optimization

```
In [27]: from tensorflow.keras.layers import GRU

         # Build a GRU Recurrent Neural Network
         embedding_dim = 100
         gru_units = 64

         model4 = Sequential()
         model4.add(Embedding(input_dim=max_words, output_dim=embedding_dim, input_length=max_sequence_length))
         model4.add(GRU(units=gru_units))
         model4.add(Dense(64, activation='relu'))
         model4.add(Dropout(0.5))
         model4.add(Dense(1, activation='sigmoid'))

         # Compile the model
         model4.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

         model4.summary()
```

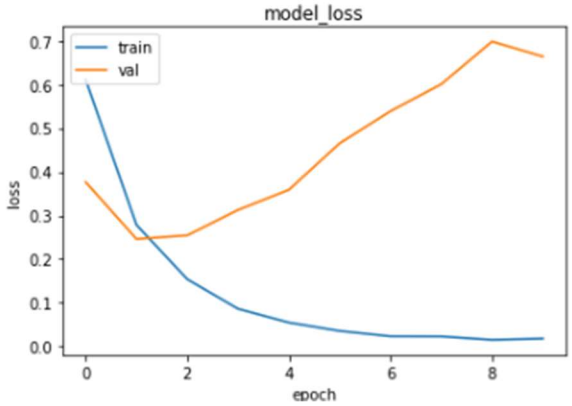
```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, 200, 100)          2500000

 gru (GRU)                   (None, 64)                31872

 dense_2 (Dense)             (None, 64)                4160

 dropout_1 (Dropout)         (None, 64)                0

 dense_3 (Dense)             (None, 1)                 65

=================================================================
Total params: 2,536,097
Trainable params: 2,536,097
Non-trainable params: 0
_________________________________________________________________
```

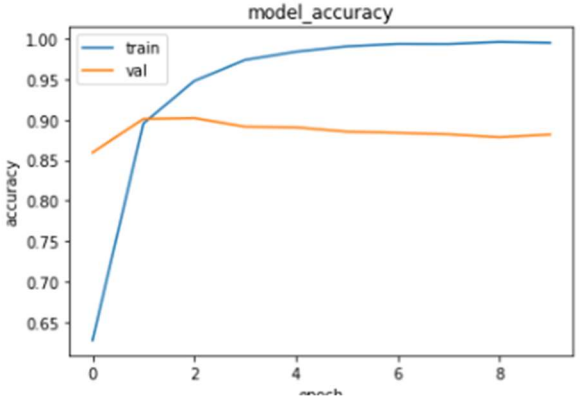
```
Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.89      0.90      4961
           1       0.89      0.92      0.90      5039

    accuracy                           0.90     10000
   macro avg       0.90      0.90      0.90     10000
weighted avg       0.90      0.90      0.90     10000
```

Model Evaluation



Here we can see that after fine tuning our model accuracy increase to 90% , recall increase to 91 and f1 score to 90.



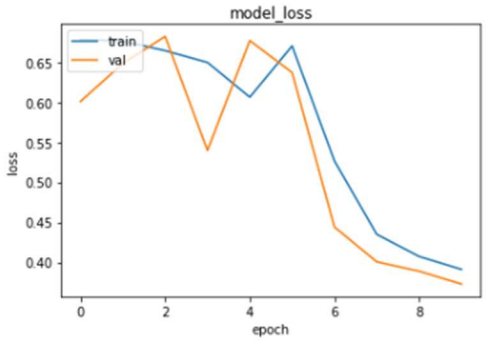


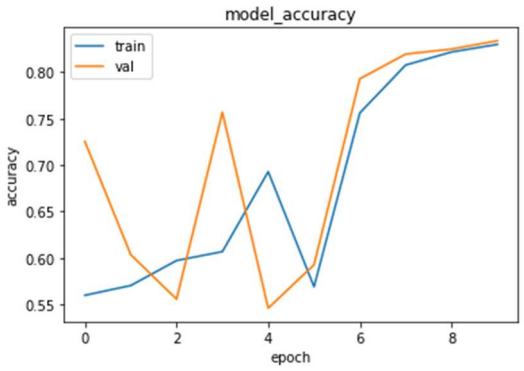
Word Embedding with use of pre trained model Glove.

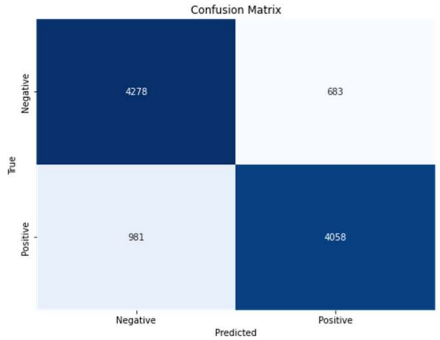
Here we have used pretarined model glove By utilizing GloVe embeddings, the model can benefit from the con textual understanding of words, contributing to improved performance in natural language processing tasks. T he pre-trained nature of GloVe embeddings eliminates the need for the model to learn representations from s cratch, enhancing efficiency and effectiveness in various NLP applications.

```python
# Step 1: Download GloVe embeddings
# You can download GloVe embeddings from: https://nlp.stanford.edu/projects/glove/

# Load GloVe embeddings into memory
glove_path = "C:/Users/rutik\Downloads/glove.6B.50d.txt"
embedding_index = {}
with open(glove_path, encoding='utf-8') as f:
    for line in f:
        values = line.split()
        word = values[0]
        coefs = np.asarray(values[1:], dtype='float32')
        embedding_index[word] = coefs

# Step 2: Create an embedding matrix
embedding_matrix = np.zeros((vocab_size, embedding_dim))
for word, i in tokenizer.word_index.items():
    embedding_vector = embedding_index.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector

# Step 3: Modify our model to use GloVe embeddings
model5 = Sequential()
model5.add(Embedding(input_dim=vocab_size, output_dim=embedding_dim, weights=[embedding_matrix], inpu
model5.add(LSTM(units=lstm_units))
model5.add(Dense(32, activation='relu'))
model5.add(Dropout(0.5))
model5.add(Dense(1, activation='sigmoid'))

# Compile the model
model5.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

model5.summary()
```

Model Evaluation







By embedding it in our initial model we get the almost same accuracy of 83% but time training time and performance efficiency improved a lot

Conclusion

In this document, we explored the fascinating world of sentiment analysis using deep learning with TensorFlow and recurrent neural networks . Our primary objective was to build a model capable of accurately predicting the sentiment of movie reviews, distinguishing between positive and negative sentiments.

Throughout the journey, we took a step-by-step approach, starting with data collection and preprocessing. We carefully cleaned and tokenized the movie reviews, creating a vocabulary that mapped each unique word to a numerical index. This allowed us to convert the text data into a suitable format for deep learning.

Next, we constructed our RNN model with different architectures, leveraging the power of LSTM layers and GRU layers to capture long-range dependencies in the sequential data. The embedding layer helped us convert integer-encoded text sequences into dense vectors that captured the underlying semantic relationships between words.Also look at the power of pretained model Glove.

During the training phase, we witnessed the model's gradual improvement as it learned from the training data. The training loss decreased with each epoch, and the training accuracy increased, demonstrating the model's ability to learn and generalize from the provided examples.

In conclusion, our sentiment analysis model using TensorFlow and RNNs proved to be a powerful tool for deciphering sentiments in movie reviews. With its application in customer feedback analysis, movie recommendation systems, and market sentiment monitoring, sentiment analysis continues to hold immense potential in various domain