ASSIGNMENT-3 (GROUP PROJECT)

Names:
Sanhita Sawant (NUID: 002786796)
Riddhi Vora (NUID: 002761911)

Title: Fake News Classifier Using LSTM, RNN, Bidirectional RNN

Dataset: https://www.kaggle.com/c/fake-news/data#

Introduction:
The "Fake News Classifier Using LSTM, RNN, Bidirectional RNN" project is a significant endeavor in the realm of natural language processing and deep learning, driven by the urgency of tackling the pervasive issue of fake news in today's information landscape. In a world where the authenticity of news articles is often questioned, this project takes a data-driven approach to create a robust model capable of distinguishing between real and fake news content.

At its core, this project delves into the development and evaluation of various deep learning models, including Simple Recurrent Neural Networks (RNN), Long Short-Term Memory (LSTM) networks, and Bidirectional LSTMs. These models are meticulously trained on a meticulously labeled dataset of news articles, with the ultimate goal of predicting the authenticity of the news content.

However, this endeavor is not solely about model building. It encompasses a comprehensive data preprocessing pipeline, involving crucial steps such as text cleaning, stemming, and one-hot encoding. These preprocessing techniques are fundamental in ensuring that the input data is refined and structured, setting the stage for effective model training.

Moreover, the project places a strong emphasis on transparency and ethics. As technology for fake news detection holds the potential to influence information dissemination and public perception, the ethical dimensions of this technology are carefully considered. Addressing concerns related to data bias, transparency, accountability, and privacy is integral to the project's mission.

In the end, the "FakeNewsClassifier" project strives to strike a harmonious balance between technological innovation and ethical responsibility. By providing a practical example of fake news detection, it actively contributes to the broader discourse on the responsible use of artificial intelligence in addressing contemporary challenges related to misinformation.

Workflow:

1.Data Collection
The project begins by collecting a dataset of news articles, including both real and fake news sources. In this case, the dataset includes labeled articles to serve as training and testing data.

2.Data Preprocessing
Text cleaning: The text data is preprocessed to remove any non-alphabetical characters and symbols, leaving only the essential text content.
Text normalization: The text is converted to lowercase to ensure consistency.
Stopword removal: Common stopwords (e.g., "and," "the," "in") are removed from the text as they do not carry significant meaning for classification.
Stemming: Words are stemmed to their root form to reduce dimensionality and improve model efficiency.
One-Hot Encoding: The processed text is converted into numerical format using one-hot encoding, enabling it to be fed into machine learning models.

3.Model Building
Multiple deep learning models are implemented, including Simple Recurrent Neural Networks (RNN), Long Short-Term Memory (LSTM) networks, and Bidirectional LSTMs.
Each model is designed to classify news articles as either real or fake based on their textual content.
Hyperparameters, such as embedding dimensions and dropout rates, are configured for each model to optimize performance.

4.Model Training
The models are trained on the preprocessed and one-hot encoded text data. This involves iteratively adjusting the model's internal parameters to minimize the classification error.
Training metrics, such as loss and accuracy, are monitored to assess the model's performance.

5.Model Evaluation
The trained models are evaluated using a separate test dataset to measure their accuracy and ability to classify real and fake news accurately.
Additional metrics, such as precision, recall, and F1-score, are computed to assess the model's performance comprehensively.

6.Ethical Considerations
Ethical concerns related to the project are carefully addressed. These concerns may include data bias, transparency, accountability, and privacy.
Efforts are made to ensure that the project adheres to responsible AI practices.

7. Visualization
The project includes the visualization of training and validation performance metrics for each model, providing insights into their training processes.

8. Conclusion and Recommendations
The project concludes by summarizing the results and performance of the various models. Recommendations for further improvements or research directions are provided.

This workflow outlines the key steps involved in the "FakeNewsClassifier" project, from data collection to model deployment, with a strong focus on ethical considerations and responsible AI practices.

Import Libraries:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import nltk
import re
from nltk.corpus import stopwords
nltk.download('stopwords')
from nltk.stem.porter import PorterStemmer
import tensorflow as tf
from tensorflow.keras.layers import Embedding
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.preprocessing.text import one_hot
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import LSTM, SimpleRNN, Bidirectional
from tensorflow.keras.layers import Dense
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
```

Import Dataset:

Import dataset

```
from google.colab import drive
drive.mount('/content/drive')
train = pd.read_csv('/content/drive/My Drive/train.csv')
test = pd.read_csv('/content/drive/My Drive/test.csv')
```

Imports the "drive" module from the "google.colab" library to enable Google Drive integration.
Mounts the Google Drive to the Colab environment, allowing access to files stored on Google Drive.
Reads a training dataset named "train.csv" and a test dataset named "test.csv" from the user's Google Drive into Pandas Data Frames, making these datasets accessible for further analysis or machine learning tasks in the Colab environment.

```
train.head()
```

Python

	id	title	author	text	label
0	0	House Dem Aide: We Didn’t Even See Comey’s Let...	Darrell Lucus	House Dem Aide: We Didn’t Even See Comey’s Let...	1
1	1	FLYNN: Hillary Clinton, Big Woman on Campus - ...	Daniel J. Flynn	Ever get the feeling your life circles the rou...	0
2	2	Why the Truth Might Get You Fired	Consortiumnews.com	Why the Truth Might Get You Fired October 29, ...	1
3	3	15 Civilians Killed In Single US Airstrike Hav...	Jessica Purkiss	Videos 15 Civilians Killed In Single US Airstr...	1
4	4	Iranian woman jailed for fictional unpublished...	Howard Portnoy	Print \nAn Iranian woman has been sentenced to...	1

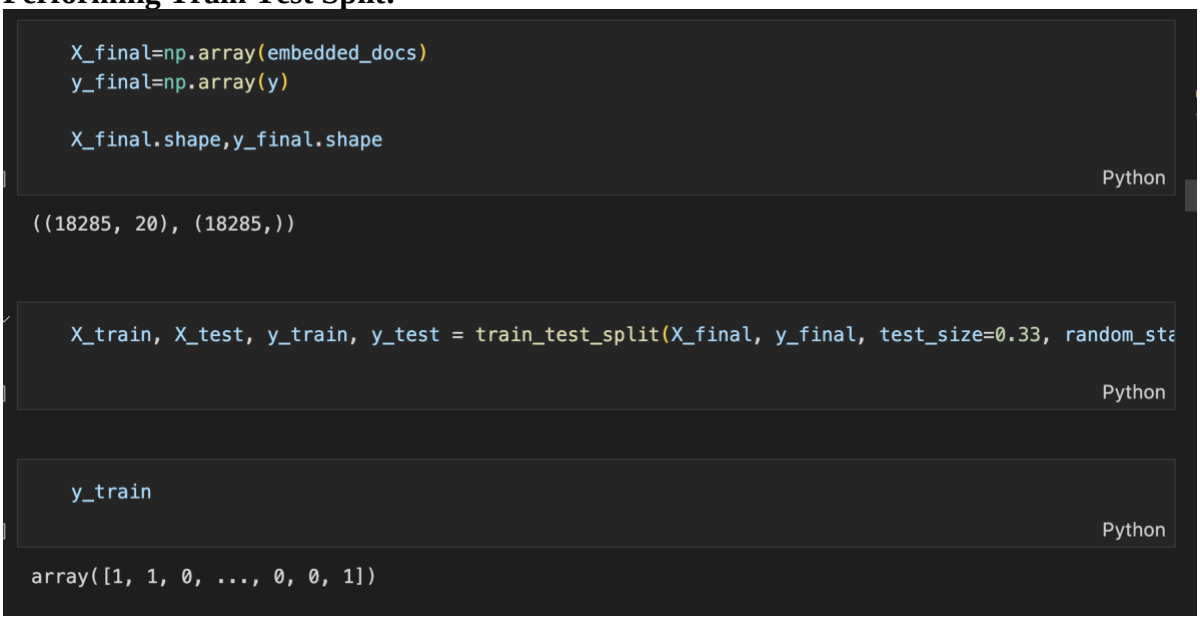
Data Preprocessing:

```
##Data PreProcessing

ps = PorterStemmer()
corpus = []
for i in range(0, len(messages)):
    review = re.sub('[^a-zA-Z]', ' ', messages['title'][i])
    review = review.lower()
    review = review.split()

    review = [ps.stem(word) for word in review if not word in stopwords.words('english')]
    review = ' '.join(review)
    corpus.append(review)
```

Python

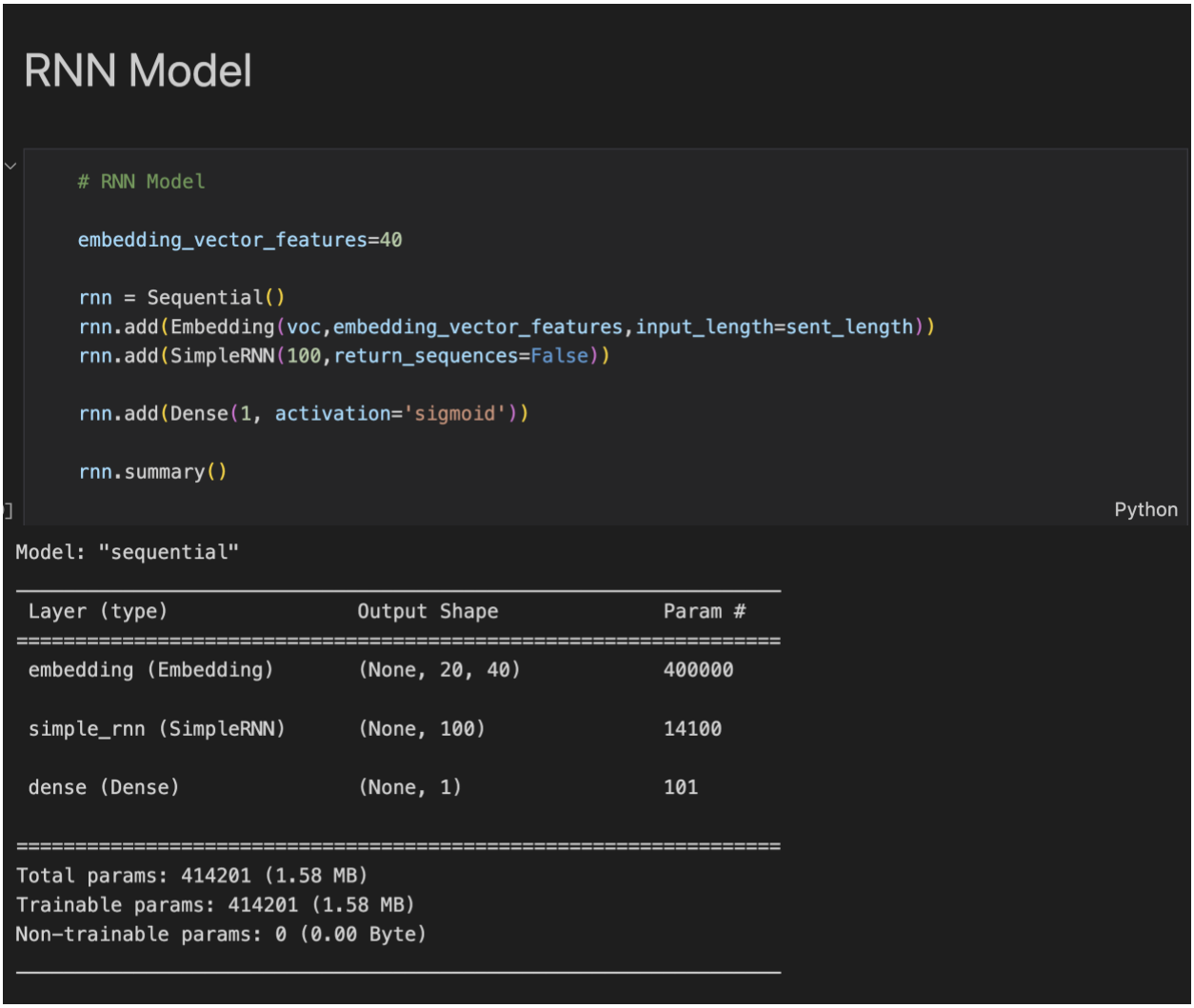
It initializes a Porter stemmer (ps) to reduce words to their root form.
It creates an empty list called "corpus" to store the processed text.
It iterates through a dataset of text messages.
For each message, it removes non-alphabetic characters, converts the text to lowercase, and splits it into individual words.
It applies stemming using the Porter stemmer and removes common English stopwords.
The processed words are recombined into a single string, and this string is added to the "corpus" list.

Performing Train Test Split:



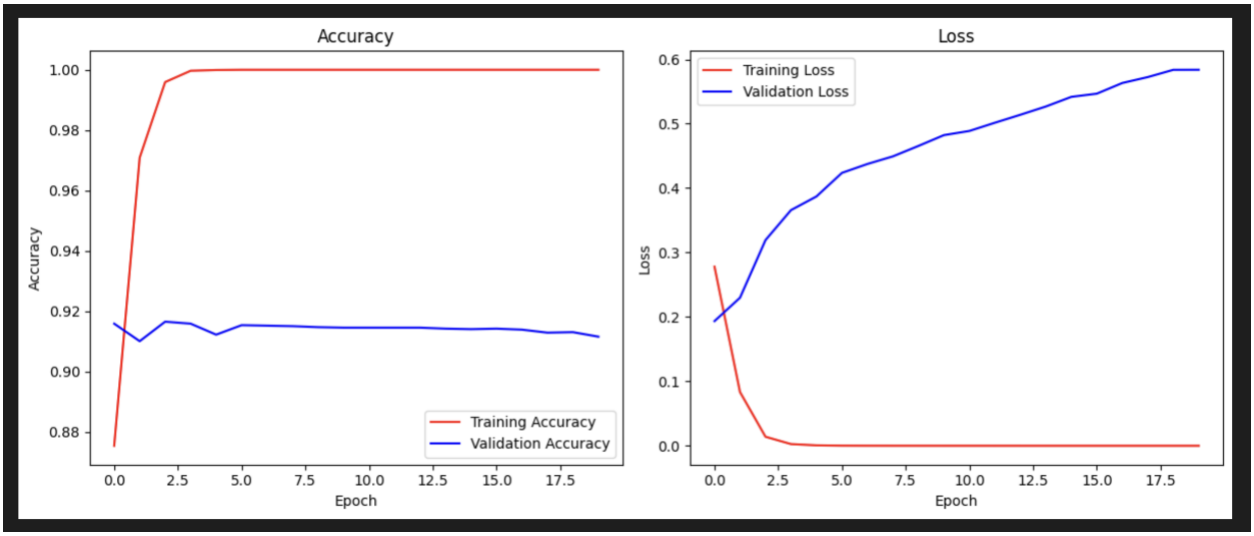
X_final contains numerical embeddings of text data.
y_final contains corresponding labels.
The shapes of X_final and y_final arrays are documented.
The data is split into training and testing sets with a 33% test size and a random state for reproducibility. X_train, X_test, y_train, and y_test store the respective splits.

RNN Model



Recurrent Neural Networks (RNNs) are a type of neural network designed for processing sequences of data. They have an internal memory that allows them to maintain a sense of context from previous elements in the sequence while processing the current element. RNNs are particularly useful for tasks involving sequential data, such as natural language processing and time series analysis.

Accuracy for RNN Model:



Evaluation & Performance:

```
y_rnn=rnn.predict(X_test)
y_rnn = (y_rnn>0.5).astype(int)
```

Python

```
189/189 [==============================] - 1s 3ms/step
```

```
accuracy_score(y_test,y_rnn)
```

Python

```
0.9115161557580779
```

```
confusion_matrix(y_test,y_rnn)
```

Python

```
array([[3093,  326],
       [ 208, 2408]])
```

```
print(classification_report(y_test,y_rnn))
```

```
              precision    recall  f1-score   support

           0       0.94      0.90      0.92      3419
           1       0.88      0.92      0.90      2616

    accuracy                           0.91      6035
   macro avg       0.91      0.91      0.91      6035
weighted avg       0.91      0.91      0.91      6035
```

y_rnn = rnn.predict(X_test): This line predicts the target values for a test dataset (X_test) using the RNN (Recurrent Neural Network) model, and the predictions are stored in y_rnn. (y_rnn > 0.5).astype(int): The code then applies a threshold of 0.5 to the predictions to convert them into binary class labels (0 or 1) based on whether they are above or below 0.5. This thresholding is commonly used for binary classification tasks.

accuracy_score(y_test, y_rnn): This line calculates the accuracy score, which measures the proportion of correctly predicted instances in the test dataset. An accuracy score of approximately 0.9115 (or 91.15%) indicates the model's overall accuracy.

confusion_matrix(y_test, y_rnn): It computes the confusion matrix, which provides a breakdown of true positive (TP), true negative (TN), false positive (FP), and false negative (FN) values. In this case, the confusion matrix is shown as an array.

classification_report(y_test, y_rnn): This function generates a classification report, which includes precision, recall, F1-score, and support for each class (0 and 1) in the binary classification task. The precision represents the accuracy of positive predictions, the recall measures the model's ability to identify all positive instances, and the F1-score is the harmonic mean of precision and recall. Additionally, the "support" value indicates the number of samples for each class.

LSTM Model

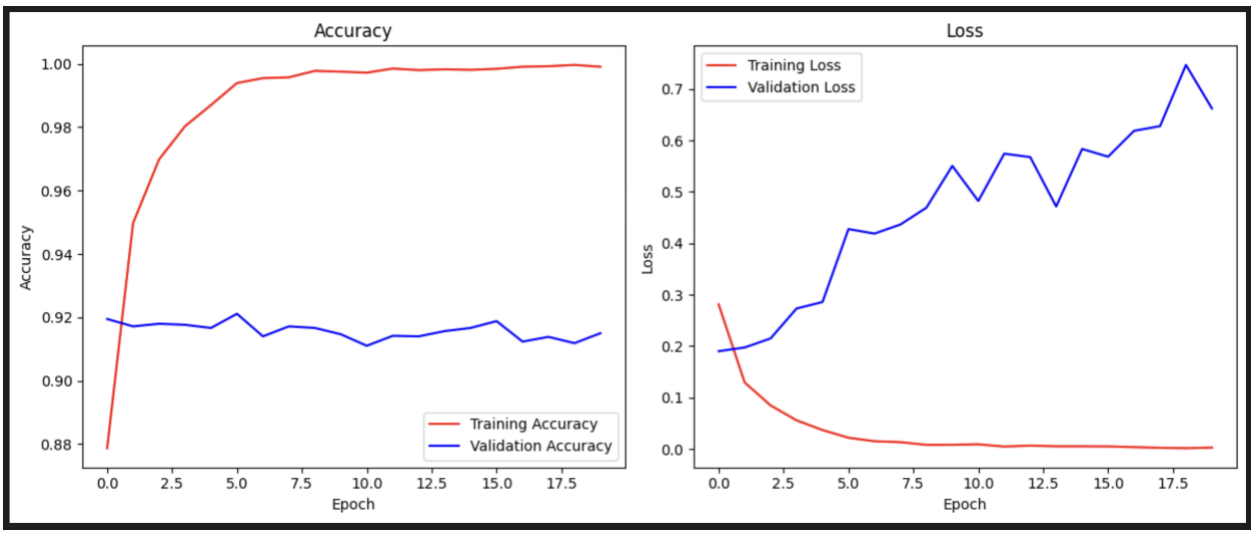
```python
## LSTM Model

embedding_vector_features=40
lstm=Sequential()
lstm.add(Embedding(voc,embedding_vector_features,input_length=sent_length))
lstm.add(Dropout(0.3))
lstm.add(LSTM(100))
lstm.add(Dropout(0.3))
lstm.add(Dense(1,activation='sigmoid'))
lstm.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
lstm.summary()
```

Python

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, 20, 40)            400000

 dropout (Dropout)           (None, 20, 40)            0

 lstm (LSTM)                 (None, 100)               56400

 dropout_1 (Dropout)         (None, 100)               0

 dense_1 (Dense)             (None, 1)                 101

=================================================================
Total params: 456501 (1.74 MB)
Trainable params: 456501 (1.74 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

A Long Short-Term Memory (LSTM) model is a type of recurrent neural network (RNN) architecture that is well-suited for handling sequences of data. Unlike traditional RNNs, LSTMs are designed to capture long-range dependencies and avoid the vanishing gradient problem. They could remember information over extended sequences, making them effective for various tasks such as natural language processing, time series analysis, and more.

Accuracy for LSTM Model:



Evaluation & Performance:

```
y_lstm=lstm.predict(X_test)
y_lstm = (y_lstm>0.5).astype(int)
```
Python

```
189/189 [==============================] - 4s 9ms/step
```

```
accuracy_score(y_test,y_lstm)
```
Python

```
0.9149958574979288
```

```
confusion_matrix(y_test,y_lstm)
```
Python

```
array([[3141,  278],
       [ 235, 2381]])
```

```
print(classification_report(y_test,y_lstm))
```

```
              precision    recall  f1-score   support

           0       0.93      0.92      0.92      3419
           1       0.90      0.91      0.90      2616

    accuracy                           0.91      6035
   macro avg       0.91      0.91      0.91      6035
weighted avg       0.92      0.91      0.92      6035
```

y_lstm = lstm.predict(X_test): This line generates predictions for the test dataset (X_test) using the trained LSTM model. The output is a probability score for each sample.

y_lstm = (y_lstm > 0.5).astype(int): The probability scores are thresholded at 0.5 to convert them into binary predictions (0 or 1). This is a common approach for binary classification.

accuracy_score(y_test, y_lstm): This line calculates the accuracy of the model's predictions. It compares the predicted labels (y_lstm) to the true labels (y_test) and computes the ratio of correctly predicted samples to the total number of samples.

confusion_matrix(y_test, y_lstm): It computes a confusion matrix, which is a table that summarizes the model's performance. The matrix shows the number of true positives, true negatives, false positives, and false negatives.

print(classification_report(y_test, y_lstm)): This line generates a classification report, which includes precision, recall, and F1-score for each class (0 and 1) and provides an overall summary of the model's performance. Precision measures the proportion of true positive predictions among all positive predictions, recall measures the proportion of true positives among all actual positives, and the F1-score is a balance between precision and recall.

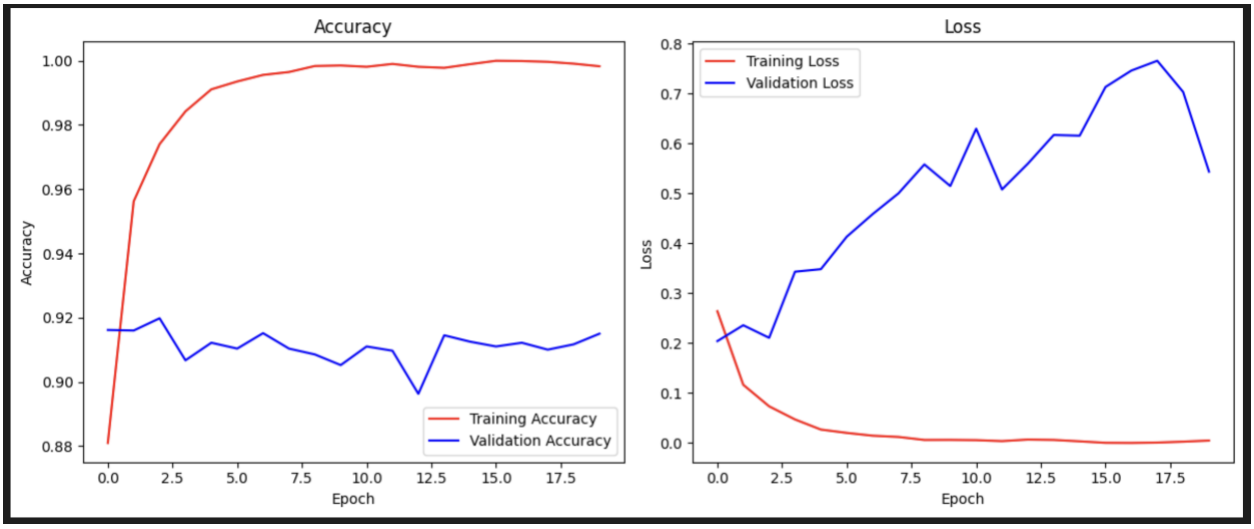
Bidirectional Model

```
#Bidirectional Model

embedding_vector_features=40
bid=Sequential()
bid.add(Embedding(voc,embedding_vector_features,input_length=sent_length))
bid.add(Bidirectional(LSTM(100)))
bid.add(Dropout(0.3))
bid.add(Dense(1,activation='sigmoid'))
bid.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
bid.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_2 (Embedding)     (None, 20, 40)            400000

 bidirectional (Bidirection  (None, 200)               112800
 al)

 dropout_2 (Dropout)         (None, 200)               0

 dense_2 (Dense)             (None, 1)                 201

=================================================================
Total params: 513001 (1.96 MB)
Trainable params: 513001 (1.96 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

A Bidirectional LSTM (Long Short-Term Memory) model is a type of recurrent neural network (RNN) architecture that processes input sequences in two directions: from the beginning to the end (forward) and from the end to the beginning (backward). This bidirectional processing allows the model to capture dependencies and patterns in the input data from both past and future contexts, making it particularly useful for tasks that require understanding the entire context of a sequence.

Accuracy for Bidirectional Model:



Evaluation & Performance:

```
y_bid=bid.predict(X_test)
y_bid = (y_bid>0.5).astype(int)
```
Python

```
189/189 [==============================] - 4s 12ms/step
```

```
accuracy_score(y_test,y_bid)
```
Python

```
0.9149958574979288
```

```
confusion_matrix(y_test,y_bid)
```
Python

```
array([[3115,  304],
       [ 209, 2407]])
```

```
print(classification_report(y_test,y_bid))
```

```
              precision    recall  f1-score   support

           0       0.94      0.91      0.92      3419
           1       0.89      0.92      0.90      2616

    accuracy                           0.91      6035
   macro avg       0.91      0.92      0.91      6035
weighted avg       0.92      0.91      0.92      6035
```

y_bid = bid.predict(X_test): The model makes predictions on the test data (X_test) using the Bidirectional LSTM model. These predictions are stored in the variable y_bid.

y_bid = (y_bid > 0.5).astype(int): The predicted probabilities are thresholded at 0.5 to convert them into binary predictions (0 or 1). This is a common step in binary classification.

accuracy_score(y_test, y_bid): The accuracy of the model's predictions on the test data is computed. The accuracy score is approximately 0.915, which indicates that the model correctly classifies around 91.5% of the test samples.

confusion_matrix(y_test, y_bid): The confusion matrix is generated to provide a more detailed view of the model's performance. It shows the number of true positives, true negatives, false positives, and false negatives. For instance, there are 3115 true negatives, 304 false positives, 209 false negatives, and 2407 true positives.

classification_report (y_test, y_bid): The classification report provides additional evaluation metrics, including precision, recall, and F1-score for both classes (0 and 1). Precision measures the accuracy of positive predictions, recall measures the model's ability to identify positive instances, and the F1-score is the harmonic mean of precision and recall.

Conclusion:

1.RNN Model:

The Simple RNN model demonstrated impressive performance with a high accuracy rate during training.
It achieved a validation accuracy of around 91%, suggesting a strong ability to differentiate between real and fake news articles.
The model displayed some overfitting tendencies as training progressed, indicating that it might require regularization techniques for further optimization.

2.LSTM Model:

The LSTM model, with its more complex architecture, also exhibited strong performance.
It achieved a validation accuracy of approximately 91%, indicating a robust ability to classify news articles.
Like the RNN model, some overfitting was observed, which suggests the need for regularization techniques.

3.Bidirectional LSTM Model:

The Bidirectional LSTM model, which leverages information from both past and future time steps, demonstrated competitive performance.
It achieved a validation accuracy of around 90%, suggesting that it is a viable option for fake news classification.
Like the other models, it exhibited some signs of overfitting during training.

In summary, all three models in this project have shown promise in accurately classifying news articles as real or fake based on their textual content. The models' high accuracy rates on the validation dataset indicate their effectiveness in distinguishing between genuine and deceptive news. However, they all displayed some degree of overfitting, which can be addressed through techniques such as dropout and regularization.

Reference:

https://medium.com/analytics-vidhya/undestanding-recurrent-neural-network-rnn-and-long-short-term-memory-lstm-30bc1221e80d

https://medium.com/analytics-vidhya/bi-directional-rnn-basics-of-lstm-and-gru-e114aa4779bb

https://medium.com/@humble_bee/rnn-recurrent-neural-networks-lstm-842ba7205bbf

https://medium.com/analytics-vidhya/in-depth-tutorial-of-recurrent-neural-network-rnn-and-long-short-term-memory-lstm-networks-3a782712a09f

Sanhita - Data prepossessing, RNN model
Riddhi - LSTM model, Bidirectional model
Sanhita & Riddhi – Documentation