In [1]:
```
# This Python 3 environment comes with many helpful analytics libraries instal
led
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/d
ocker-python
# For example, here's several helpful packages to load
import warnings
warnings.filterwarnings("ignore")
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_
score
# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will lis
t all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that ge
ts preserved as output when you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved
outside of the current session
```

In [2]:
```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset
import pandas as pd

# Load the Data
url = "https://raw.githubusercontent.com/MuhammadYaseenKhan/Urdu-Sentiment-Cor
pus/master/urdu-sentiment-corpus-v1.tsv"
df = pd.read_csv(url, sep='\t', names=['Text', 'Label'])
```

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

Out[3]:

	Text	Label
0	Tweet	Class
1	میں نے ایٹم بم بنایا ہے ۔۔۔او بھائی ایٹم بمب ...	P
2	چندے سے انقلاب اور عمران خان وزیر اعظم نہیں بن...	N
3	ٹویٹر کا خیال کیسے آیا ؟	O
4	سرچ انجن گوگل کے نائب صدر نے فضا میں ، 130,000...	P

In [5]:
```
# Preprocess the text data
X = df['Text'].values
y = df['Label'].values
```

In [6]:
```
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Dense, SimpleRNN, GRU, LSTM, Bi
directional
from tensorflow.keras.layers import Dropout
```

In [7]:
```
# Tokenize the text data
tokenizer = Tokenizer()
tokenizer.fit_on_texts(X)
X = tokenizer.texts_to_sequences(X)
# Pad sequences to a fixed length
X = pad_sequences(X)
```

In [9]:
```
# Split the Data into Train and Test Sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, rand
om_state=42)
```

In [10]:
```
# Convert 'P' to 1 and 'N' to 0 in y_train and y_test
y_train = np.array([1 if label == 'P' else 0 for label in y_train])
y_test = np.array([1 if label == 'P' else 0 for label in y_test])
```

In [ ]:
```

```

In [11]:
```
# Define the RNN Model
def create_rnn_model(num_layers, dropout_rate):
    vocab_size = len(tokenizer.word_index) + 1
    max_length = X.shape[1]

    model = Sequential()
    model.add(Embedding(input_dim=vocab_size, output_dim=100, input_length=max
_length))

    for _ in range(num_layers - 1):
        model.add(SimpleRNN(units=128, return_sequences=True))
        model.add(Dropout(dropout_rate))

    model.add(SimpleRNN(units=128))
    model.add(Dropout(dropout_rate))
    model.add(Dense(units=1, activation='sigmoid'))

    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accu
racy'])
    return model
```

In [12]:
```
# Define the GRU Model
def create_gru_model(num_layers, dropout_rate):
    vocab_size = len(tokenizer.word_index) + 1
    max_length = X.shape[1]

    model = Sequential()
    model.add(Embedding(input_dim=vocab_size, output_dim=100, input_length=max
_length))

    for _ in range(num_layers - 1):
        model.add(GRU(units=128, return_sequences=True))
        model.add(Dropout(dropout_rate))

    model.add(GRU(units=128))
    model.add(Dropout(dropout_rate))
    model.add(Dense(units=1, activation='sigmoid'))

    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accu
racy'])
    return model
```

In [13]:
```
# Define the LSTM Model
def create_lstm_model(num_layers, dropout_rate):
    vocab_size = len(tokenizer.word_index) + 1
    max_length = X.shape[1]

    model = Sequential()
    model.add(Embedding(input_dim=vocab_size, output_dim=100, input_length=max
_length))

    for _ in range(num_layers - 1):
        model.add(LSTM(units=128, return_sequences=True))
        model.add(Dropout(dropout_rate))

    model.add(LSTM(units=128))
    model.add(Dropout(dropout_rate))
    model.add(Dense(units=1, activation='sigmoid'))

    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accu
racy'])
    return model
```

In [14]:
```
#Define the BiLSTM Model
def create_bilstm_model(num_layers, dropout_rate):
    vocab_size = len(tokenizer.word_index) + 1
    max_length = X.shape[1]

    model = Sequential()
    model.add(Embedding(input_dim=vocab_size, output_dim=100, input_length=max
_length))

    for _ in range(num_layers - 1):
        model.add(Bidirectional(LSTM(units=128, return_sequences=True)))
        model.add(Dropout(dropout_rate))

    model.add(Bidirectional(LSTM(units=128)))
    model.add(Dropout(dropout_rate))
    model.add(Dense(units=1, activation='sigmoid'))

    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accu
racy'])
    return model
```

In [15]:
```
# Hyperparameters to try
num_layers_list = [2, 3]
dropout_rate_list = [0.3, 0.7]

# Results table to store the performance metrics
results_table = []

# Models dictionary
models_dict = {
    "RNN": create_rnn_model,
    "GRU": create_gru_model,
    "LSTM": create_lstm_model,
    "BiLSTM": create_bilstm_model
}
```

In [16]:
```
# Iterate over all models and hyperparameter combinations
for model_name, create_model_func in models_dict.items():
    for num_layers in num_layers_list:
        for dropout_rate in dropout_rate_list:
            # Create and train the model
            model = create_model_func(num_layers=num_layers, dropout_rate=drop
out_rate)
            model.fit(X_train, y_train, epochs=5, batch_size=32, validation_da
ta=(X_test, y_test), verbose=0)

            # Evaluate the model on the test set
            y_pred = model.predict(X_test)
            y_pred = (y_pred > 0.5).astype(int)

            accuracy = accuracy_score(y_test, y_pred)
            precision = precision_score(y_test, y_pred)
            recall = recall_score(y_test, y_pred)
            f1 = f1_score(y_test, y_pred)

            # Store the results in the table
            results_table.append({
                "Model": model_name,
                "Number of Layers": num_layers,
                "Dropout Rate": dropout_rate,
                "Accuracy": accuracy,
                "Precision": precision,
                "Recall": recall,
                "F-Score": f1
            })
```

```
8/8 [==============================] - 0s 9ms/step
8/8 [==============================] - 0s 9ms/step
8/8 [==============================] - 0s 12ms/step
8/8 [==============================] - 0s 12ms/step
8/8 [==============================] - 1s 17ms/step
8/8 [==============================] - 1s 19ms/step
8/8 [==============================] - 1s 25ms/step
8/8 [==============================] - 1s 27ms/step
8/8 [==============================] - 1s 38ms/step
8/8 [==============================] - 1s 34ms/step
8/8 [==============================] - 2s 54ms/step
8/8 [==============================] - 2s 56ms/step
8/8 [==============================] - 2s 48ms/step
8/8 [==============================] - 2s 54ms/step
8/8 [==============================] - 3s 75ms/step
8/8 [==============================] - 3s 77ms/step
```

In [17]:
```
# Create a DataFrame from the results_table
results_df = pd.DataFrame(results_table)

# Print the results table
print(results_df)
```

```
     Model  Number of Layers  Dropout Rate  Accuracy  Precision    Recall  \
0      RNN                 2           0.3  0.498008   0.486667  0.598361
1      RNN                 2           0.7  0.498008   0.487805  0.655738
2      RNN                 3           0.3  0.458167   0.464646  0.754098
3      RNN                 3           0.7  0.513944   0.000000  0.000000
4      GRU                 2           0.3  0.597610   0.584000  0.598361
5      GRU                 2           0.7  0.581673   0.549708  0.770492
6      GRU                 3           0.3  0.553785   0.539062  0.565574
7      GRU                 3           0.7  0.577689   0.556338  0.647541
8     LSTM                 2           0.3  0.581673   0.555556  0.696721
9     LSTM                 2           0.7  0.577689   0.557971  0.631148
10    LSTM                 3           0.3  0.625498   0.616667  0.606557
11    LSTM                 3           0.7  0.577689   0.545977  0.778689
12  BiLSTM                 2           0.3  0.565737   0.543624  0.663934
13  BiLSTM                 2           0.7  0.561753   0.541667  0.639344
14  BiLSTM                 3           0.3  0.589641   0.551351  0.836066
15  BiLSTM                 3           0.7  0.581673   0.561151  0.639344

     F-Score
0   0.536765
1   0.559441
2   0.575000
3   0.000000
4   0.591093
5   0.641638
6   0.552000
7   0.598485
8   0.618182
9   0.592308
10  0.611570
11  0.641892
12  0.597786
13  0.586466
14  0.664495
15  0.597701
```

In [ ]:
```
```