Experiment 05 : Write a program to implement RNN.

Learning Objective : Write a program to implement RNN.

Tools : Python

Theory :

Recurrent Neural Network(RNN) is a type of Neural Network where the output from the previous step is fed as input to the current step. In traditional neural networks, all the inputs and outputs are independent of each other. Still, in cases when it is required to predict the next word of a sentence, the previous words are required and hence there is a need to remember the previous words. Thus RNN came into existence, which solved this issue with the help of a Hidden Layer. The main and most important feature of RNN is its Hidden state, which remembers some information about a sequence. The state is also referred to as the Memory State since it remembers the previous input to the network. It uses the same parameters for each input as it performs the same task on all the inputs or hidden layers to produce the output. This reduces the complexity of parameters, unlike other neural networks.

The key feature of RNNs is their ability to maintain a state or memory of previous inputs while processing new inputs. This memory enables RNNs to capture context and dependencies within sequential data. Each neuron in an RNN is equipped with a "hidden state" that serves as its memory, and this hidden state is updated at each time step based on the current input and the previous hidden state.

In summary, RNNs are a powerful class of neural networks capable of capturing temporal dynamics in sequential data, with applications ranging from natural language processing to time series analysis and beyond.

Steps to implement RNN :

- **Initialization :** Initialize the parameters of the RNN, including the weights connecting the input layer to the hidden layer, the weights connecting the hidden layer to itself (recurrent weights), and the weights connecting the hidden layer to the output layer. Also, initialize biases for each layer.
- **Forward Pass :**
    - For each time step
    - t in the input sequence:
        - Compute the hidden state at time t using the input at time t and the previous hidden state.
        - Compute the output at time t using the hidden state at time t.
        - Store the hidden states and outputs for each time step.

- **Compute Loss :** Calculate the loss between the predicted outputs and the true labels at each time step using a suitable loss function (e.g., cross-entropy loss for classification tasks, mean squared error for regression tasks).

- **Backpropagation Through Time (BPTT) :**
  - Initialize gradients for all the weights and biases to zero.
  - For each time step t in reverse order :
    - Compute the gradients of the loss with respect to the output at time t.
    - Backpropagate the gradients through the network to compute the gradients of the loss with respect to the hidden state at time t.
    - Update the gradients of the loss with respect to the weights and biases using the gradients computed in the previous steps.
  - Clip gradients if necessary to prevent exploding gradients.

- **Update Parameters :** Update the parameters (weights and biases) of the network using an optimization algorithm such as stochastic gradient descent (SGD), Adam, RMSprop, etc. Adjust the learning rate if necessary.

- **Repeat :** Repeat steps 2-5 for a fixed number of iterations (epochs) or until convergence criteria are met.

Implementation :

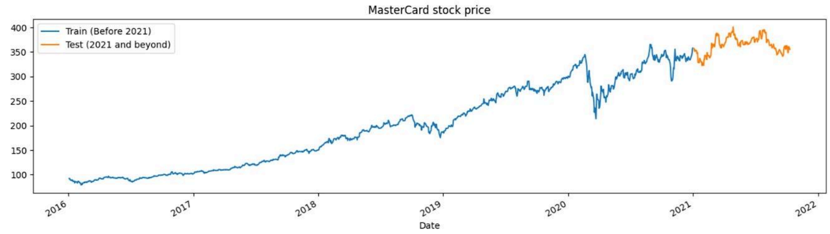
[1]:
```
# Importing the libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, Dropout, GRU, Bidirectional
from tensorflow.keras.optimizers import SGD
from tensorflow.random import set_seed

set_seed(455)
np.random.seed(455)
```

[2]:
```
dataset = pd.read_csv("/content/Mastercard_stock_history.csv",␣
↪index_col="Date", parse_dates=["Date"]).drop(["Dividends", "Stock Splits"],␣
↪axis=1)
print(dataset.head())
```

```
                Open      High       Low     Close     Volume
Date
2006-05-25  3.748967  4.283869  3.739664  4.279217  395343000
2006-05-26  4.307126  4.348058  4.103398  4.179680  103044000
2006-05-30  4.183400  4.184330  3.986184  4.093164   49898000
2006-05-31  4.125723  4.219679  4.125723  4.180608   30002000
2006-06-01  4.179678  4.474572  4.176887  4.419686   62344000
```

[3]:
```
print(dataset.describe())
```

[5]:
```
tstart = 2016
tend = 2020

def train_test_plot(dataset, tstart, tend):
    dataset.loc[f"{tstart}":f"{tend}", "High"].plot(figsize=(16, 4),␣
 ↪legend=True)
    dataset.loc[f"{tend+1}":, "High"].plot(figsize=(16, 4), legend=True)
    plt.legend([f"Train (Before {tend+1})", f"Test ({tend+1} and beyond)"])
    plt.title("MasterCard stock price")
    plt.show()

train_test_plot(dataset,tstart,tend)
```



[6]:
```
def train_test_split(dataset, tstart, tend):
    train = dataset.loc[f"{tstart}":f"{tend}", "High"].values
    test = dataset.loc[f"{tend+1}":, "High"].values
    return train, test
training_set, test_set = train_test_split(dataset, tstart, tend)
```

[7]:
```
sc = MinMaxScaler(feature_range=(0, 1))
training_set = training_set.reshape(-1, 1)
training_set_scaled = sc.fit_transform(training_set)
```

[8]:
```
def split_sequence(sequence, n_steps):
    X, y = list(), list()
    for i in range(len(sequence)):
        end_ix = i + n_steps
        if end_ix > len(sequence) - 1:
            break
        seq_x, seq_y = sequence[i:end_ix], sequence[end_ix]
        X.append(seq_x)
        y.append(seq_y)
    return np.array(X), np.array(y)


n_steps = 60
features = 1
# split into samples
X_train, y_train = split_sequence(training_set_scaled, n_steps)
```

[9]:
```
# Reshaping X_train for model
X_train = X_train.reshape(X_train.shape[0],X_train.shape[1],features)
```

[10]:
```
# The LSTM architecture
model_lstm = Sequential()
model_lstm.add(LSTM(units=125, activation="tanh", input_shape=(n_steps,␣
 ↪features)))
model_lstm.add(Dense(units=1))
# Compiling the model
model_lstm.compile(optimizer="RMSprop", loss="mse")

model_lstm.summary()
```

```
Model: "sequential"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 125)               63500

 dense (Dense)               (None, 1)                 126

=================================================================
Total params: 63626 (248.54 KB)
Trainable params: 63626 (248.54 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

[11]:
```python
model_lstm.fit(X_train, y_train, epochs=50, batch_size=32)
```

```
Epoch 1/50

38/38 [==============================] - 2s 56ms/step - loss: 4.7505e-04
Epoch 50/50
38/38 [==============================] - 3s 78ms/step - loss: 3.7731e-04
```

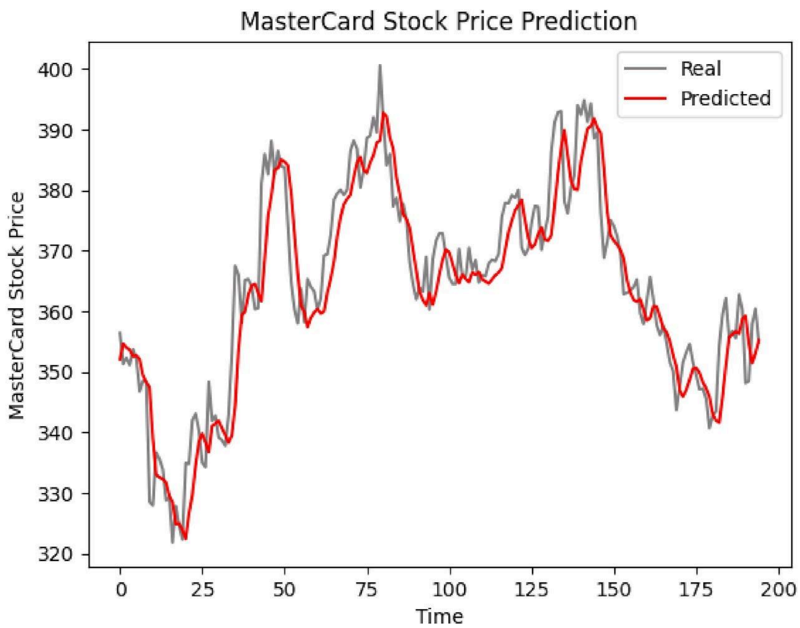
[11]: <keras.src.callbacks.History at 0x7a351a126ef0>

[12]:
```python
dataset_total = dataset.loc[:,"High"]
inputs = dataset_total[len(dataset_total) - len(test_set) - n_steps :].values
inputs = inputs.reshape(-1, 1)
#scaling
inputs = sc.transform(inputs)

# Split into samples
X_test, y_test = split_sequence(inputs, n_steps)
# reshape
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], features)
#prediction
predicted_stock_price = model_lstm.predict(X_test)
#inverse transform the values
predicted_stock_price = sc.inverse_transform(predicted_stock_price)
```

```
7/7 [==============================] - 1s 63ms/step
```

[13]:
```python
def plot_predictions(test, predicted):
    plt.plot(test, color="gray", label="Real")
    plt.plot(predicted, color="red", label="Predicted")
    plt.title("MasterCard Stock Price Prediction")
    plt.xlabel("Time")
    plt.ylabel("MasterCard Stock Price")
    plt.legend()
    plt.show()


def return_rmse(test, predicted):
    rmse = np.sqrt(mean_squared_error(test, predicted))
    print("The root mean squared error is {:.2f}.".format(rmse))
```

[14]:
```python
plot_predictions(test_set,predicted_stock_price)
```



[16]:
```
return_rmse(test_set,predicted_stock_price)
```

```
The root mean squared error is 6.46.
```

Result and Discussion :

Learning Outcomes : Students should have the ability to

LO 4.1: Ability to understand the fundamental concepts of Recurrent Neural Networks, including the role of hidden states, memory cells, and sequential data processing.
LO 4.2: Ability to Identify and describe real-world applications where RNNs.

Course Outcomes :

CO : Understand and apply Recurrent Neural Networks.

Conclusion :

Viva Questions :

Q1. How do RNNs differ from traditional feedforward neural networks?
Q2. What is the role of the hidden state in an RNN?

For Faculty Use

Correction Parameters	Formative Assessment [40%]	Timely completion of Practical [40%]	Attendance / Learning Attitude [20%]	Total
Marks Obtained				