S. B. JAIN INSTITUTE OF TECHNOLOGY, MANAGEMENT & RESEARCH, NAGPUR

Practical No. 09

Aim: Identify the dataset problem statement and implement Recurrent Neural Network.

Name of Student : Shrutika Pradeep Bagdi

Roll No : CS22130

Semester/Year : VIIth Sem / IVth Year

Academic Session : 2025-2026 [ODD]

Date of Performance : _____

Date of Submission : _____

AIM: Demonstration of the concept of Recurrent Neural Network.

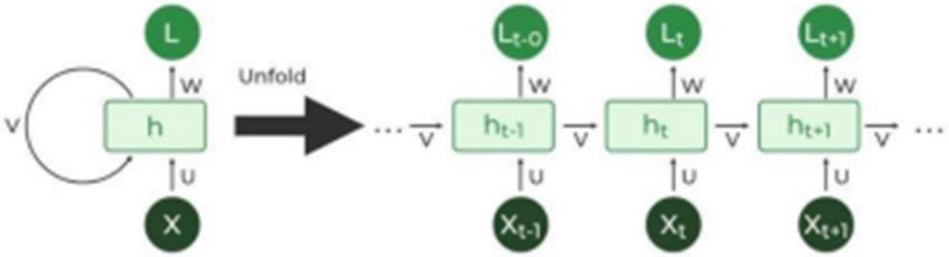
OBJECTIVE/EXPECTED LEARNING OUTCOME:

The objectives and expected learning outcome of this practical are:

- To develop a learning algorithm for Recurrent Neural Network, empowering the networks to be trainedto capture the mapping implicitly.
- To develop learning algorithm for Recurrent Neural Network.
- Efficiently compute the results and loss with respect to the network weights.

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, but 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 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 network.



REcurrent neural network

Architecture Of Recurrent Neural Network

RNNs have the same input and output architecture as any other deep neural architecture. However, differences arise in the way information flows from input to output. Unlike Deep neural networks where we have different weight matrices for each Dense network in RNN, the weight across the network remains

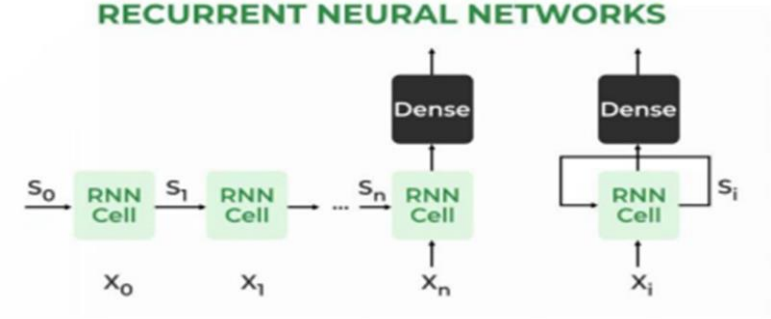
the same. It calculates state hidden state H**i** for every input **Xi .**

By using the following formulas: ***h= σ(UX + Wh-1 + B)***

Y = O(Vh + C) Hence

Y = f (X, h , W, U, V, B, C)

Here S is the State matrix which has element si as the state of the network at timestamp i. The parameters in the network are W, U, V, c, b which are shared across timestamp.



What is Recurrent Neural Network

How RNN works

The Recurrent Neural Network consists of multiple fixed activation function units, one for each time step. Each unit has an internal state which is called the hidden state of the unit. This hidden state signifies the past knowledge that the network currently holds at a given time step. This hidden state is updated at every time step to signify the change in the knowledge of the network about the past. The hidden state is updated using the following recurrence relation:-

The formula for calculating the current state:

$$h_t = f(h_{t-1}, x_t)$$

where:

ht -> current state

ht-1 -> previous state

xt -> input state

Formula for applying Activation function(tanh):

$$h_t = \tanh (W_{hh}h_{t-1} + W_{xh}x_t)$$

where:

whh -> weight at recurrent neuron

wxh -> weight at input neuron

The formula for calculating output:

$$y_t = W_{hy} h_t$$

Yt -> output

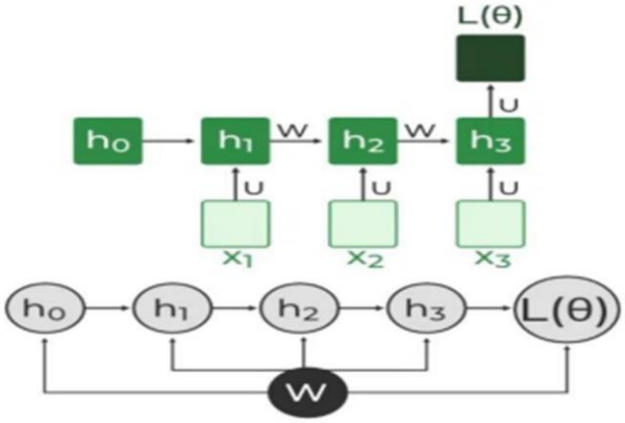
Why -> weight at output layer

These parameters are updated using Backpropagation. However, since RNN works on sequential data here we use an updated backpropagation which is known as Backpropagation through time.

Backpropagation Through Time (BPTT)

In RNN the neural network is in an ordered fashion and since in the ordered network each variable is computed one at a time in a specified order like first h1 then h2 then h3 so on. Hence we will apply backpropagation throughout all these hidden time states

sequentially.



Backpropagation Through Time (BPTT) In RNN

L(θ)(loss function) depends on h3

h3 in turn depends on h2 and W

h2 in turn depends on h1 and W

h1 in turn depends on h0 and W

where h0 is a constant starting state.

Training through RNN

1. A single-time step of the input is provided to the network.
2. Then calculate its current state using a set of current input and the previous state.
3. The current ht becomes ht-1 for the next time step.
4. One can go as many time steps according to the problem and join the information from all the previous states.

5. Once all the time steps are completed the final current state is used to calculate the output.
6. The output is then compared to the actual output i.e the target output and the error is generated.
7. The error is then back-propagated to the network to update the weights and hence the network (RNN) is trained using Backpropagation through time.

Advantages:

1. An RNN remembers each and every piece of information through time. It is useful in time series prediction only because of the feature to remember previous inputs as well. This is called Long Short Term Memory.
2. Recurrent neural networks are even used with convolutional layers to extend the effective pixel neighborhood.

Disadvantages:

1. Gradient vanishing and exploding problems.
2. Training an RNN is a very difficult task.
3. It cannot process very long sequences if using tanh or relu as an activation function.

Applications:

1. Language Modelling and Generating Text
2. Speech Recognition
3. Machine Translation
4. Image Recognition, Face detection
5. Time series Forecasting

Types Of RNN:

There are four types of RNNs based on the number of inputs and outputs in the network.

1. One to One
2. One to Many
3. Many to One
4. Many to Many

Steps to implement:

Loading Data:

```
dataset_train = pd.read_csv('../input/stockprice-train/Stock_Price_Train.csv')
```

Feature Scaling:

```
from sklearn.preprocessing import MinMaxScaler #bununla, 0-1 arasına
scale ettik scaler = MinMaxScaler(feature_range = (0, 1))
```

```
train_scaled = scaler.fit_transform(train)
train_scaled
```

Create Data Structure:

```
        #ilk 1-50 yi alıp X_train'e, 51. data point'i de y_train'e,
#2-51'i alıp X_train'e, 52'yi y_train'e ...olacak şekilde data frame i
oluşturuyoruz: X_train = []
y_train = []
timesteps = 50

for i in range(timesteps, 1250):
   X_train.append(train_scaled[i - timesteps:i, 0])
   y_train.append(train_scaled[i, 0])

X_train, y_train = np.array(X_train), np.array(y_train)
```

Reshape:

```
X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
```

Create RNN Model

```
#import libraries and packages:
from keras.modelsimport Sequential
from keras.layers import Dense
from keras.layersimport SimpleRNN
from keras.layers import Dropout
#Initialize RNN:
regressor = Sequential()

#Adding the first RNN layer and some Dropout regularization
regressor.add(SimpleRNN(units = 50, activation='tanh', return_sequences=True, input_shape=
(X_train.shape[1],1)))  regressor.add(Dropout(0.2))

#Adding the second RNN layer and some Dropout regularization
regressor.add(SimpleRNN(units = 50, activation='tanh', return_sequences=True))
regressor.add(Dropout(0.2))
```

#Adding the third RNN layer and some Dropout regularization

regressor.add(SimpleRNN(units = 50, activation='tanh', return_sequences=True))

regressor.add(Dropout(0.2))

#Adding the fourth RNN layer and some Dropout regularization

regressor.add(SimpleRNN(units = 50))

regressor.add(Dropout(0.2))

#Adding the output layer

regressor.add(Dense(units = 1))

#Compile the RNN

regressor.compile(optimizer='adam', loss='mean_squared_error')

#Fitting the RNN to the Training set

regressor.fit(X_train, y_train, epochs=100, batch_size=32)

Prediction and Visualization of RNN Model

dataset_test = pd.read_csv('../input/stockprice-test/Stock_Price_Test.csv')

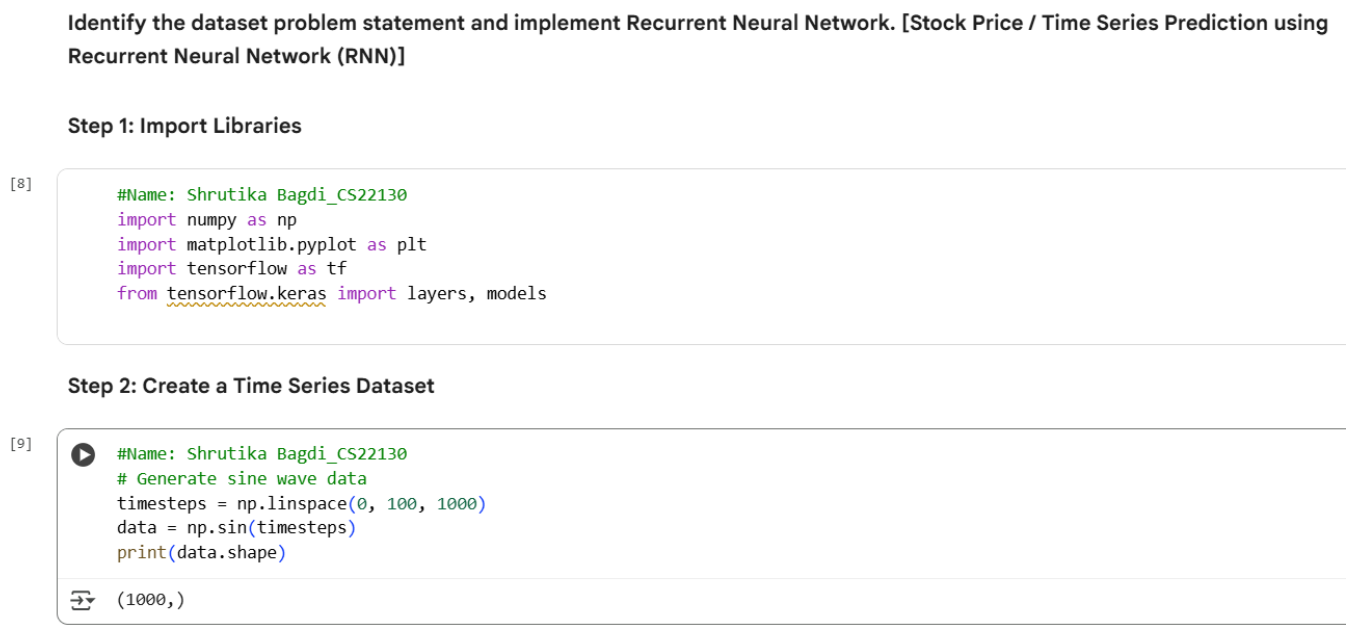
dataset_test.head()

PROGRAM CODE:

```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras import layers, models

# Generate sine wave data
timesteps = np.linspace(0, 100, 1000)
data = np.sin(timesteps)
print(data.shape)

# Prepare sequences (look back = 10 time steps)
look_back = 10
x, y = [], []
for i in range(len(data) - look_back):
    x.append(data[i:i+look_back])
    y.append(data[i+look_back])
x, y = np.array(x), np.array(y)
print("x:", x.shape)
```

```
print("y:", y.shape)

# Reshape for RNN (samples, timesteps, features)
# RNN expects input shape = (samples, time steps, features).
x = x.reshape(x.shape[0], x.shape[1], 1)
print("x shape:", x.shape) # (990, 10, 1)
print("y shape:", y.shape)

model = models.Sequential([
    layers.SimpleRNN(50, activation="tanh", input_shape=(look_back, 1)),
    layers.Dense(1) # Regression output
])

model.compile(optimizer="adam", loss="mse")
model.summary()

history = model.fit(x, y, epochs=20, batch_size=32, verbose=1)

# Predict using the same dataset
y_pred =  model.predict(x)
# Plot results
plt.figure(figsize=(10,6))
plt.plot(timesteps [look_back:], y, label="True Values")
plt.plot(timesteps [look_back:], y_pred, label="Predicted Values")
plt.legend()
plt.show()
```

OUTPUT (SCREENSHOT):



Identify the dataset problem statement and implement Recurrent Neural Network. [Stock Price / Time Series Prediction using Recurrent Neural Network (RNN)]

Step 1: Import Libraries

[8]

```
#Name: Shrutika Bagdi_CS22130
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras import layers, models
```

Step 2: Create a Time Series Dataset

[9]

```
#Name: Shrutika Bagdi_CS22130
# Generate sine wave data
timesteps = np.linspace(0, 100, 1000)
data = np.sin(timesteps)
print(data.shape)
```

```
(1000,)
```

Step 3: Build RNN Model

[12]

```
#Name: Shrutika Bagdi_CS22130
model = models.Sequential([
    layers.SimpleRNN(50, activation="tanh", input_shape=(look_back, 1)),
    layers.Dense(1) # Regression output
])
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/rnn/rnn.py:199: UserWarning: Do not pass an `input_shape`
  super().__init__(**kwargs)
```

[13]

```
#Name: Shrutika Bagdi_CS22130
model.compile(optimizer="adam", loss="mse")
model.summary()
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
simple_rnn_1 (SimpleRNN)	(None, 50)	2,600
dense_1 (Dense)	(None, 1)	51

Total params: 2,651 (10.36 KB)
Trainable params: 2,651 (10.36 KB)
Non-trainable params: 0 (0.00 B)

[10]

```
#Name: Shrutika Bagdi_CS22130
# Prepare sequences (look back = 10 time steps)
look_back = 10
x, y = [], []
for i in range(len(data) - look_back):
    x.append(data[i:i+look_back])
    y.append(data[i+look_back])
x, y = np.array(x), np.array(y)
print("x:", x.shape)
print("y:", y.shape)
```

```
x: (990, 10)
y: (990,)
```

[11]

```
#Name: Shrutika Bagdi_CS22130
# Reshape for RNN (samples, timesteps, features)
# RNN expects input shape = (samples, time steps, features).
x = x.reshape(x.shape[0], x.shape[1], 1)
print("x shape:", x.shape) # (990, 10, 1)
print("y shape:", y.shape)
```

```
x shape: (990, 10, 1)
y shape: (990,)
```

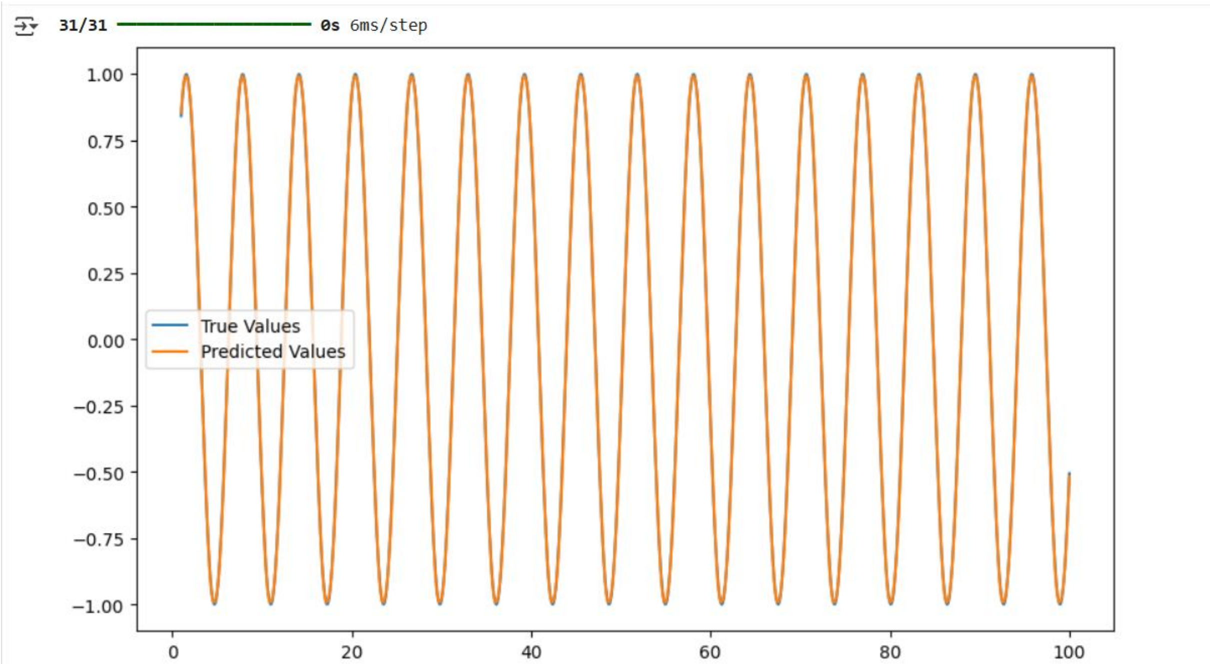
Step 4: Train the Model

```
#Name: Shrutika Bagdi_CS22130
history = model.fit(x, y, epochs=20, batch_size=32, verbose=1)
```

```
Epoch 1/20
31/31 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 0.1901
Epoch 2/20
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0064
Epoch 3/20
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0030
Epoch 4/20
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.0018
Epoch 5/20
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.0012
Epoch 6/20
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 7.8632e-04
Epoch 7/20
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 6.2725e-04
Epoch 8/20
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 3.7427e-04
Epoch 9/20
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 2.7858e-04
Epoch 10/20
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 2.1893e-04
Epoch 11/20
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 1.8638e-04
Epoch 12/20
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 1.5749e-04
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 1.1370e-04
Epoch 16/20
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 9.1576e-05
Epoch 17/20
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 8.6803e-05
Epoch 18/20
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 8.5408e-05
Epoch 19/20
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 7.9376e-05
Epoch 20/20
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 9.7266e-05
```

Step 5: Prediction

```
#Name: Shrutika Bagdi_CS22130
# Predict using the same dataset
y_pred =  model.predict(x)
# Plot results
plt.figure(figsize=(10,6))
plt.plot(timesteps [look_back:], y, label="True Values")
plt.plot(timesteps [look_back:], y_pred, label="Predicted Values")
plt.legend()
plt.show()
```



CONCLUSION:

____

____

____

DISCUSSION AND VIVA VOCE:

1. What is a Recurrent Neural Network (RNN), and how is it different from a feedforward neural network?
2. Why do we use RNNs for sequential data? Give some real-world applications.
3. Define vanishing and exploding gradient problems in RNNs.
4. What are the differences between RNN, LSTM, and GRU?
5. How can RNNs be applied in natural language processing?
6. Give an example of how RNNs are used in speech recognition.
7. How can you use an RNN for sentiment analysis?
8. Why are LSTMs preferred over vanilla RNNs?

REFERENCE:

- 8 - Recurrent Neural Network (RNN) Tutorial | Kaggle
- https://www.analyticsvidhya.com/blog/2022/03/a-brief-overview-of-recurrent-neural-networks-rnn/
- Introduction to Recurrent Neural Network - GeeksforGeeks