AD18511 – DEEP LEARNING LABORATORY

DATE:

EX.NO: 14 **RECURRENT NEURAL NETWORK(RNN).**

AIM:

To implement RNN (recurrent neural network) using Tensorflow.

DESCRIPTION:

- Recurrent Neural Networks (RNNs) are a class of neural network architectures designed to process sequential data by maintaining a hidden state that captures information from previous elements in the sequence.
- RNNs are specifically designed for sequential data, such as time series, natural language, speech, and more.
- RNN can handle data of varying lengths and are well-suited for tasks where the order of elements in the sequence is important.
- The core idea of an RNN is the use of recurrent connections. At each step, the RNN takes an input and its hidden state from the previous time step to produce an output and an updated hidden state.
- The hidden state serves as a memory that retains information about previous elements in the sequence, allowing the network to capture dependencies and context.
- In an RNN, the same set of weights and biases are used at each time step. This weight sharing enables the network to apply the same operation across the entire sequence.
- A limitation of traditional RNNs is the vanishing and exploding gradient problem. During training, gradients can become too small (vanishing) or too large (exploding), which hinders the learning process for long sequences.

PROGRAM:

```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt

t= np.linspace(0 , 10, 1000) #time step
data = np.sin(t) + 0.1 * np.random.randn(1000)

data = (data - np.mean(data)) / np.std(data)
split = int(0.8 * len(data))
train_data, test_data = data[:split], data[split:]

def create_sequences(data, seq_length):
    sequences=[]
    labels=[]

    for i in range(len(data)- seq_length):
        sequence=data[i:i+seq_length]
        label = data[i+seq_length]
        sequences.append(sequence)
        labels.append(label)

    return np.array(sequences), np.array(labels)
```

```
seq_length=10
train_sequence, train_labels = create_sequences(train_data,seq_length)

model= tf.keras.Sequential([
   tf.keras.layers.SimpleRNN(64,input_shape=(seq_length,1)),
   tf.keras.layers.Dense(1)
])

model.compile(loss='mean_squared_error',optimizer='adam')
model.fit(train_sequence,train_labels,epochs=100,batch_size=10)
```

OUTPUT:

```
Epoch 1/100
79/79 [==============================] - 1s 1ms/step - loss: 0.0397
Epoch 2/100
79/79 [==============================] - 0s 1ms/step - loss: 0.0295
Epoch 3/100
79/79 [==============================] - 0s 1ms/step - loss: 0.0282
Epoch 4/100
79/79 [==============================] - 0s 1ms/step - loss: 0.0290
Epoch 5/100
79/79 [==============================] - 0s 1ms/step - loss: 0.0280
Epoch 6/100
79/79 [==============================] - 0s 1ms/step - loss: 0.0277
Epoch 7/100
79/79 [==============================] - 0s 1ms/step - loss: 0.0284
Epoch 8/100
79/79 [==============================] - 0s 1ms/step - loss: 0.0295
Epoch 9/100
79/79 [==============================] - 0s 1ms/step - loss: 0.0281
Epoch 10/100
79/79 [==============================] - 0s 1ms/step - loss: 0.0284
----------------------------------------------------------------------------------------------------
Epoch 90/100
79/79 [==============================] - 0s 1ms/step - loss: 0.0259
Epoch 91/100
79/79 [==============================] - 0s 1ms/step - loss: 0.0266
Epoch 92/100
79/79 [==============================] - 0s 1ms/step - loss: 0.0270
Epoch 93/100
79/79 [==============================] - 0s 1ms/step - loss: 0.0259
Epoch 94/100
79/79 [==============================] - 0s 1ms/step - loss: 0.0261
Epoch 95/100
79/79 [==============================] - 0s 1ms/step - loss: 0.0291
Epoch 96/100
79/79 [==============================] - 0s 1ms/step - loss: 0.0275
Epoch 97/100
79/79 [==============================] - 0s 989us/step - loss: 0.0267
Epoch 98/100
79/79 [==============================] - 0s 1ms/step - loss: 0.0265
Epoch 99/100
79/79 [==============================] - 0s 1ms/step - loss: 0.0263
```

```
Epoch 100/100
79/79 [==============================] - 0s 1ms/step - loss: 0.0256

<keras.callbacks.History at 0x7fbc585a8e50>
```

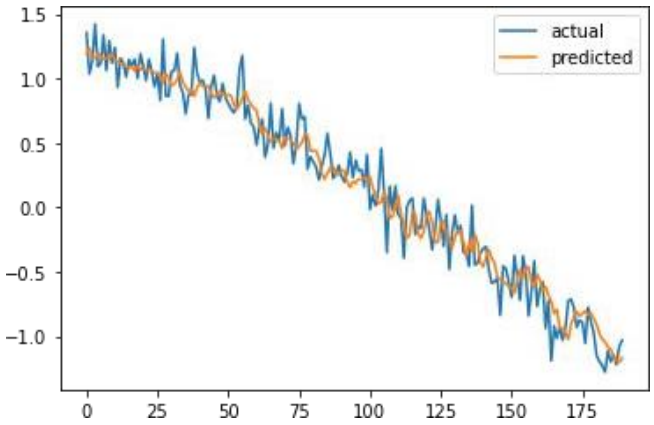
```
test_sequences,test_labels = create_sequences(test_data,seq_length)
predictions=model.predict(test_sequences)
```

OUTPUT:

```
6/6 [==============================] - 0s 903us/step
```

```
plt.plot(test_labels,label='actual')
plt.plot(predictions,label='predicted')
plt.legend()
plt.show()
```

OUTPUT:



RESULT:

Thus the implementation of Recurrent Neural Network (RNN**)** is trained and tested successfully.

AD18511 – DEEP LEARNING LABORATORY

DATE:

EX.NO: 15 **LONG SHORT TERM MEMORY (LSTM).**

AIM:

To implement LSTM (long short term memory) using Tensorflow.

DESCRIPTION:

- Long Short-Term Memory (LSTM) is a specialized variant of recurrent neural networks (RNNs) designed to address the vanishing gradient problem and better capture long-term dependencies in sequential data.
- LSTMs are used for processing sequential data, where the order of elements in the sequence is important, such as time series, natural language text, and speech.
- LSTMs have two primary components: the cell state and the hidden state.
- The cell state acts as a memory unit and runs along the entire sequence, allowing LSTMs to capture long-term dependencies.
- The hidden state, which is updated at each time step, serves as a working memory and can selectively pass information to the cell state.
- LSTMs utilize three types of gates to control the flow of information: the input gate, the forget gate, and the output gate.
- The input gate regulates how much new information should be added to the cell state.
- The forget gate controls which information should be discarded from the cell state.
- The output gate determines how the updated cell state should influence the hidden state.

PROGRAM:

```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt

t= np.linspace(0 , 10, 1000) #time step
data = np.sin(t) + 0.1 * np.random.randn(1000)

data = (data - np.mean(data)) / np.std(data)
split = int(0.8 * len(data))
train_data, test_data = data[:split], data[split:]

def create_sequences(data, seq_length):
    sequences=[]
    labels=[]

    for i in range(len(data)- seq_length):
        sequence=data[i:i+seq_length]
        label = data[i+seq_length]
        sequences.append(sequence)
        labels.append(label)
```

```
    return np.array(sequences), np.array(labels)

seq_length=10
train_sequence, train_labels = create_sequences(train_data,seq_length)

model= tf.keras.Sequential([
    tf.keras.layers.LSTM(64, return_sequences=True, input_shape=(seq_length,1)),
    tf.keras.layers.LSTM(64),
    tf.keras.layers.Dense(1),
])

model.compile(loss='mean_squared_error',optimizer='adam')
model.fit(train_sequence,train_labels,epochs=100,batch_size=10)
```

OUTPUT:

```
Epoch 1/100
79/79 [==============================] - 2s 3ms/step - loss: 0.1407
Epoch 2/100
79/79 [==============================] - 0s 3ms/step - loss: 0.0318
Epoch 3/100
79/79 [==============================] - 0s 3ms/step - loss: 0.0295
Epoch 4/100
79/79 [==============================] - 0s 3ms/step - loss: 0.0298
Epoch 5/100
79/79 [==============================] - 0s 3ms/step - loss: 0.0288
Epoch 6/100
79/79 [==============================] - 0s 3ms/step - loss: 0.0292
Epoch 7/100
79/79 [==============================] - 0s 3ms/step - loss: 0.0286
Epoch 8/100
79/79 [==============================] - 0s 3ms/step - loss: 0.0295
Epoch 9/100
79/79 [==============================] - 0s 3ms/step - loss: 0.0312
Epoch 10/100
79/79 [==============================] - 0s 3ms/step - loss: 0.0290
--------------------------------------------------------------------------------------------------------
Epoch 91/100
79/79 [==============================] - 0s 3ms/step - loss: 0.0269
Epoch 92/100
79/79 [==============================] - 0s 3ms/step - loss: 0.0277
Epoch 93/100
79/79 [==============================] - 0s 3ms/step - loss: 0.0276
Epoch 94/100
79/79 [==============================] - 0s 3ms/step - loss: 0.0275
Epoch 95/100
79/79 [==============================] - 0s 3ms/step - loss: 0.0276
Epoch 96/100
79/79 [==============================] - 0s 3ms/step - loss: 0.0278
Epoch 97/100
79/79 [==============================] - 0s 3ms/step - loss: 0.0273
Epoch 98/100
79/79 [==============================] - 0s 3ms/step - loss: 0.0280
Epoch 99/100
```

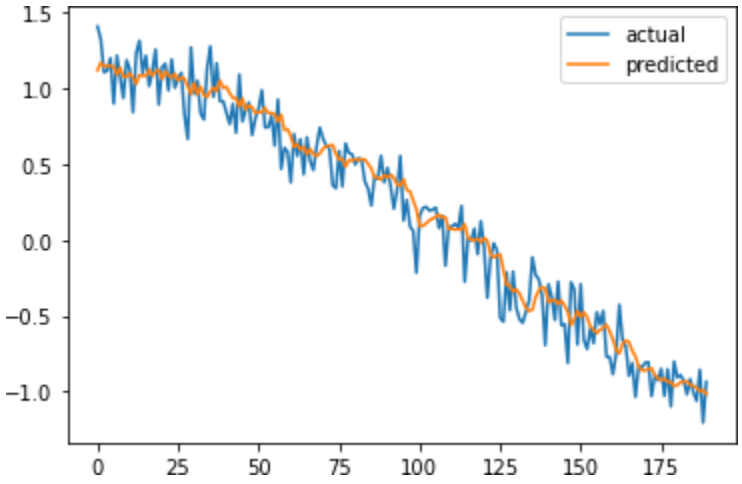
```
79/79 [==============================] - 0s 3ms/step - loss: 0.0283
Epoch 100/100
79/79 [==============================] - 0s 3ms/step - loss: 0.0278

<keras.callbacks.History at 0x7ff4503680a0>

test_sequences,test_labels = create_sequences(test_data,seq_length)
predictions=model.predict(test_sequences)

plt.plot(test_labels,label='actual')
plt.plot(predictions,label='predicted')
plt.legend()
plt.show()
```

OUTPUT:



RESULT:

Thus the implementation of long short term memory (LSTM) model is is trained and tested successfully.