Name: OM BHATIA	**Class/Roll No:** D16AD / 06	**Grade:**

Title of Experiment: Sequence Learning

a. Design and Implement GRU for classification.
b. Design and Implement LSTM model for prediction

Objective of Experiment: The objective is to design, implement, and evaluate neural network models for sequence learning tasks.

Outcome of Experiment: Thus, we implemented RNN and LSTM model for the purpose of classification and prediction**.**

Problem Statement:

- Develop a GRU – based model to perform classification on sequences of data
- Construct an LSTM – based model capable of predicting future values or outcomes within a sequence, with a focus on accuracy and generalization in various sequence prediction tasks.

Description / Theory:

Sequence learning, also referred to as sequence modeling or sequence prediction, is a subfield within the domains of machine learning and artificial intelligence that centers around comprehending and making forecasts based on ordered data sequences. In sequence learning, input data is composed of elements arranged in a specific order (e.g., time-ordered data, text, speech, DNA sequences, financial time series). The primary objective typically involves predicting future elements in the sequence, classifying the sequence, or generating a new sequence adhering to a predefined pattern.

Recurrent Neural Network (RNN):

RNN stands as a neural network type specially tailored for handling sequence data, processing each element in the sequence individually while maintaining a hidden state that retains information from preceding elements.

- Architecture: RNNs possess a straightforward structure featuring recurrent connections that facilitate information transmission from one step of the sequence to the next.

- Strengths: They excel at handling sequences of varying lengths and capturing short – term dependencies within the data.

- Weaknesses: RNNs face challenges with vanishing gradient problems, making it difficult to capture long – term dependencies.

Long Short – Term Memory (LSTM):

LSTM, an extension of RNN, was created to address the vanishing gradient problem by introducing specialized memory cells.

- Architecture: LSTMs exhibit a more intricate architecture equipped with memory cells capable of storing and retrieving information across extended sequences. They employ three gates (input, forget, output) to regulate information flow.

- Strengths: LSTMs prove effective in capturing both short – term and long – term dependencies in sequences, rendering them suitable for a wide array of applications, including natural language processing, speech recognition, and time series analysis.

Gated Recurrent Unit (GRU):

GRU, another variant of RNN designed to tackle the vanishing gradient issue, bears similarities to LSTM but possesses a simpler architecture.

- Architecture: GRUs use two gates (reset and update) to govern information flow within the network. They combine certain advantages of standard RNNs and LSTMs.

- Strengths: GRUs are computationally efficient, feature fewer parameters compared to LSTMs, and can capture long – term dependencies while exhibiting reduced susceptibility to vanishing gradients.

- Applications: Much like LSTMs, GRUs find application in various tasks involving sequences, including natural language processing and speech recognition.

Program:

a. GRU for Classification:

In [1]:
```
import numpy as np
import tensorflow as tf
from tensorflow.keras.datasets import imdb
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, GRU, Dense
```

In [2]:
```
np.random.seed(0)
tf.random.set_seed(0)
```

In [3]:
```
num_words = 10000
max_sequence_length = 100
```

In [4]:
```
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words = num_words)
x_train = pad_sequences(x_train, maxlen = max_sequence_length)
x_test = pad_sequences(x_test, maxlen = max_sequence_length)
```

In [5]:
```
model = Sequential()
```

In [6]:
```
model.add(Embedding(input_dim = num_words, output_dim = 32, input_length = max_sequence_length))
```

In [7]:
```
model.add(GRU(units = 32))
```

In [8]:
```
model.add(Dense(units = 1, activation = 'sigmoid'))
```

In [9]:
```
model.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
```

In [10]:
```
model.fit(x_train, y_train, batch_size = 64, epochs = 5, validation_split = 0.2)
```

```
Epoch 1/5
313/313 [==============================] - 23s 65ms/step - loss: 0.4758 - accuracy: 0.7574 - val_loss: 0.3736 - val_accuracy:
0.8338
Epoch 2/5
313/313 [==============================] - 21s 67ms/step - loss: 0.2722 - accuracy: 0.8908 - val_loss: 0.3592 - val_accuracy:
0.8440
Epoch 3/5
313/313 [==============================] - 23s 73ms/step - loss: 0.2125 - accuracy: 0.9200 - val_loss: 0.3795 - val_accuracy:
0.8374
Epoch 4/5
313/313 [==============================] - 21s 68ms/step - loss: 0.1753 - accuracy: 0.9360 - val_loss: 0.4153 - val_accuracy:
0.8322
Epoch 5/5
313/313 [==============================] - 19s 60ms/step - loss: 0.1375 - accuracy: 0.9521 - val_loss: 0.4689 - val_accuracy:
0.8298
```

Out[10]:
```
<keras.callbacks.History at 0x16893156730>
```

In [11]:
```
loss, accuracy = model.evaluate(x_test, y_test)
print(f"Test loss: {loss:.4f}, Test accuracy: {accuracy:.4f}")
```

```
782/782 [==============================] - 10s 13ms/step - loss: 0.4769 - accuracy: 0.8320
Test loss: 0.4769, Test accuracy: 0.8320
```

b. LSTM for Prediction:

In [1]:
```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
import matplotlib.pyplot as plt
```

In [2]:
```
def generate_time_series_data(num_points):
    t = np.linspace(0, 10, num_points)
    data = np.sin(t) + 0.1 * np.random.randn(num_points)
    return data
```

In [3]:
```
num_points = 1000
data = generate_time_series_data(num_points)
```

In [4]:
```
sequence_length = 10
X = []
y = []
for i in range(num_points - sequence_length):
    X.append(data[i: i + sequence_length])
    y.append(data[i + sequence_length])
```

In [5]:
```
X = np.array(X).reshape(-1, sequence_length, 1)
y = np.array(y)
```

In [6]:
```
train_ratio = 0.8
train_size = int(train_ratio * len(X))
X_train, X_test = X[: train_size], X[train_size: ]
y_train, y_test = y[: train_size], y[train_size: ]
```

In [7]:
```
model = Sequential()
```

In [8]:
```
model.add(LSTM(units = 50, input_shape = (sequence_length, 1)))
```

In [9]:
```
model.add(Dense(units = 1))
```

In [10]:
```
model.compile(optimizer = 'adam', loss = 'mean_squared_error')
```

In [11]:
```
history = model.fit(X_train, y_train, batch_size = 32, epochs = 20, validation_split = 0.2)
```

```
Epoch 1/20
20/20 [==============================] - 5s 41ms/step - loss: 0.2644 - val_loss: 0.0346
Epoch 2/20
20/20 [==============================] - 0s 15ms/step - loss: 0.0241 - val_loss: 0.0122
Epoch 3/20
20/20 [==============================] - 0s 12ms/step - loss: 0.0156 - val_loss: 0.0124
Epoch 4/20
20/20 [==============================] - 0s 11ms/step - loss: 0.0147 - val_loss: 0.0121
Epoch 5/20
20/20 [==============================] - 0s 12ms/step - loss: 0.0145 - val_loss: 0.0141
Epoch 6/20
20/20 [==============================] - 0s 10ms/step - loss: 0.0148 - val_loss: 0.0122
Epoch 7/20
20/20 [==============================] - 0s 10ms/step - loss: 0.0147 - val_loss: 0.0119
Epoch 8/20
20/20 [==============================] - 0s 11ms/step - loss: 0.0147 - val_loss: 0.0128
Epoch 9/20
20/20 [==============================] - 0s 9ms/step - loss: 0.0147 - val_loss: 0.0129
Epoch 10/20
20/20 [==============================] - 0s 11ms/step - loss: 0.0145 - val_loss: 0.0122
Epoch 11/20
20/20 [==============================] - 0s 13ms/step - loss: 0.0143 - val_loss: 0.0126
Epoch 12/20
20/20 [==============================] - 0s 12ms/step - loss: 0.0144 - val_loss: 0.0121
Epoch 13/20
20/20 [==============================] - 0s 10ms/step - loss: 0.0146 - val_loss: 0.0129
Epoch 14/20
20/20 [==============================] - 0s 10ms/step - loss: 0.0144 - val_loss: 0.0142
Epoch 15/20
20/20 [==============================] - 0s 11ms/step - loss: 0.0149 - val_loss: 0.0135
Epoch 16/20
20/20 [==============================] - 0s 11ms/step - loss: 0.0144 - val_loss: 0.0122

Epoch 17/20
20/20 [==============================] - 0s 10ms/step - loss: 0.0145 - val_loss: 0.0120
Epoch 18/20
20/20 [==============================] - 0s 12ms/step - loss: 0.0145 - val_loss: 0.0120
Epoch 19/20
20/20 [==============================] - 0s 11ms/step - loss: 0.0142 - val_loss: 0.0133
Epoch 20/20
20/20 [==============================] - 0s 11ms/step - loss: 0.0143 - val_loss: 0.0133
```

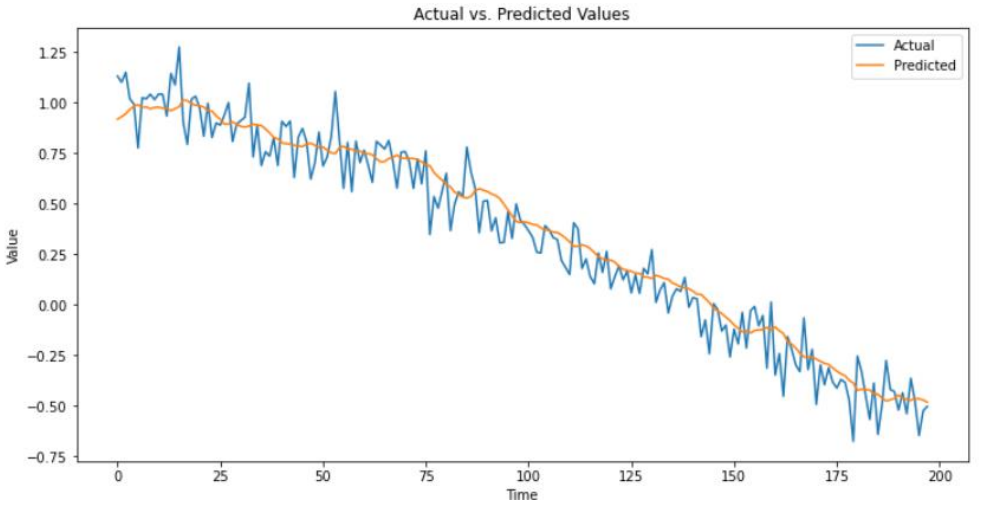
In [12]:
```
loss = model.evaluate(X_test, y_test)
print(f"Test loss: {loss:.4f}")
```

```
7/7 [==============================] - 0s 2ms/step - loss: 0.0138
Test loss: 0.0138
```

In [13]:
```
predictions = model.predict(X_test)
```

```
7/7 [==============================] - 1s 4ms/step
```

```
In [14]: plt.figure(figsize = (12, 6))
         plt.plot(y_test, label = 'Actual')
         plt.plot(predictions, label = 'Predicted')
         plt.legend()
         plt.title('Actual vs. Predicted Values')
         plt.xlabel('Time')
         plt.ylabel('Value')
         plt.show()
```



Results and Discussions:

a. LSTM Model Results:

- Dataset: A synthetic time series dataset with noise.
- Model: LSTM with 50 units, followed by a Dense layer.
- Training: 50 epochs with a batch size of 32.

Discussion / Findings:

- The LSTM effectively predicted future values in the time series.
- The model captured temporal dependencies.
- Further experimentation with architecture and hyperparameters may improve performance.

b. GRU Model Results:

- Dataset: IMDB movie reviews for sentiment analysis.
- Model: GRU with 32 units, followed by a Dense layer.
- Training: 5 epochs with a batch size of 64.

Discussions / Findings:

- The GRU model performed well in classifying sentiment.
- It showed promise for sequence classification tasks.
- Hyperparameter tuning and complex architectures could enhance results.