D. Khaja Nawaz

RA2311047010037

Deep Learning Techniques (LAB)

Date	Title	Sign
24/07/2025	1. Exploring the deep learning platform	
31/07/25	2. Implement a classifier using open-source dataset	
31/07/25	3. STudy of the classifiers with respect to statistical parameters	
14/08/25	4. Build a simple feed forword neural network to recognize handwritten character	
22/08/25	5. STudy of Activation functions and their role	
09/09/25	6. Implement gradient descent and backpropagation in deep neural network	
16/09/25	7. Build a CNN model to classify Cat and Dog image	
30/09/2025	8. Experiment of LSTM	
30/09/2025	9. Build a Recurrent Neural Network	

LAB=9

30/09/25

Build a Recurrent Neural Network

*AIM :-

TO build and implement a Recurrent Neural Network (RNN) Model for Sequential data prediction or classifica

*Objective :-

1. TO understand the working of Recurrent Neural Netw
2. TO process sequential input data for model training
3. TO construct, compile, and train an RNN model usin
   Keras / TensorFlow / PyTorch modules.
4. TO evaluate the model's performance on test data

*pseudcode :-

1. Import libraries :-

```
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
```

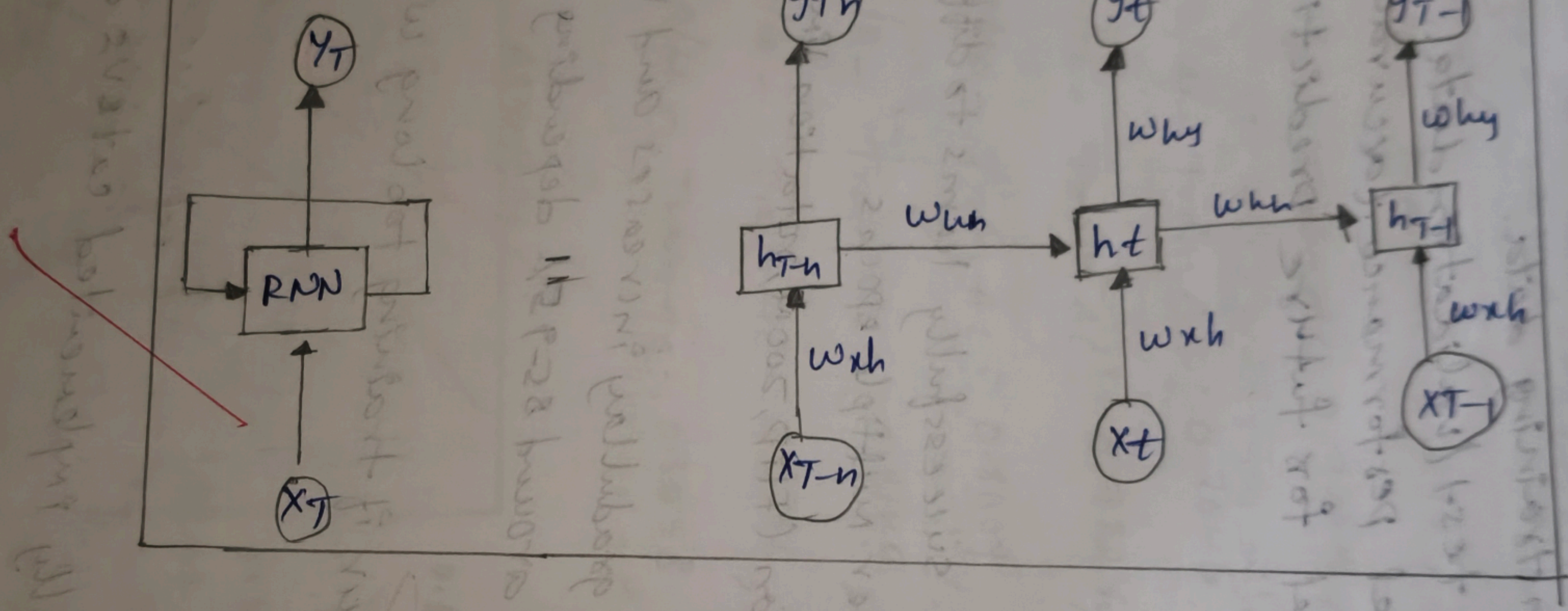
2. prepare the dataset :-

→ create sample sequential data
→ convert data to tensors
→ split into training and testing sets

3. Define the RNN model :-

```
class RNN model(nn.Module):
def _init_ (self, input_size, hidden_size, ou
super (RNN model, self)._init_()
self.rnn = nn.RNN (input_size, hidden size
                   batch_first=True)
self.fc = nn.Linear (hidden_size, outp
```

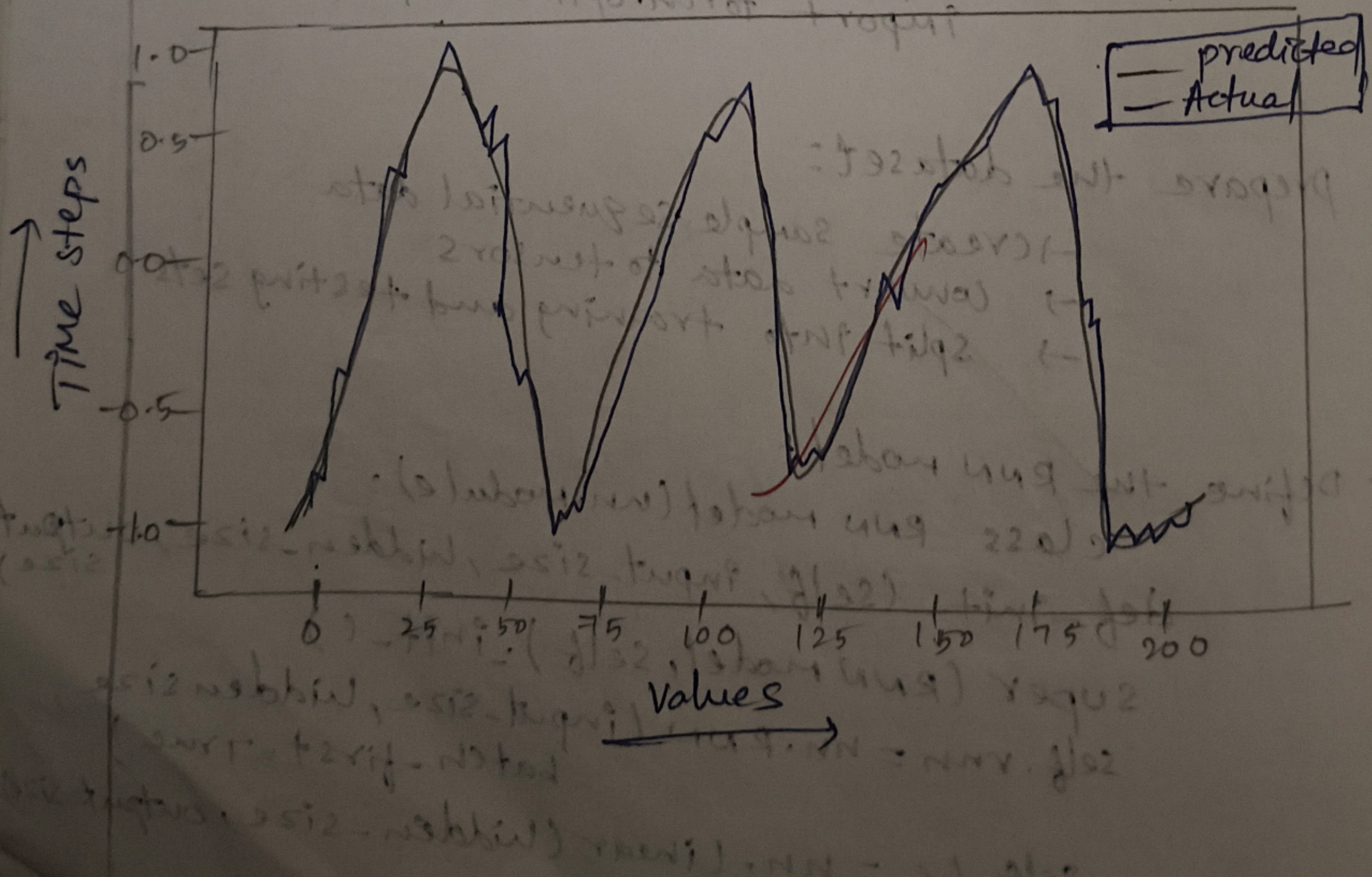
A simple Architecture of RNN model



output :-

Epoch [16/100];

Epoch	Step_loss	Val_loss
1	0.5478	0.0234
2	0.0259	0.0181
3	0.0166	0.0196
4	0.0157	0.0172
5	0.0150	0.0161
6	0.0150	0.0153
7	0.0151	0.0178
8	0.0165	0.0157
9	0.0144	0.0170
10	0.0152	0.0153



```
    def forward(self, x):
        out, _ = self.rnn(x)
        out = self.fc(out[:, -1, :])
        return out
```

4. Initialize model, loss function, and optimizer:

```
model = RNNModel(input_size=1, hidden_size=32,
                 output_size=1)
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=
                       0.01)
```

5. Train the model:

```
for epoch in range(100):
    output = model(X_train)
    loss = criterion(output, y_train)
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

6. Evaluate:

```
with torch.no_grad():
    predictions = model(X_test)
    test_loss = criterion(prediction, y_test
```

*Observation:-

→ loss decreases gradually with training, but may plateau faster than LSTM

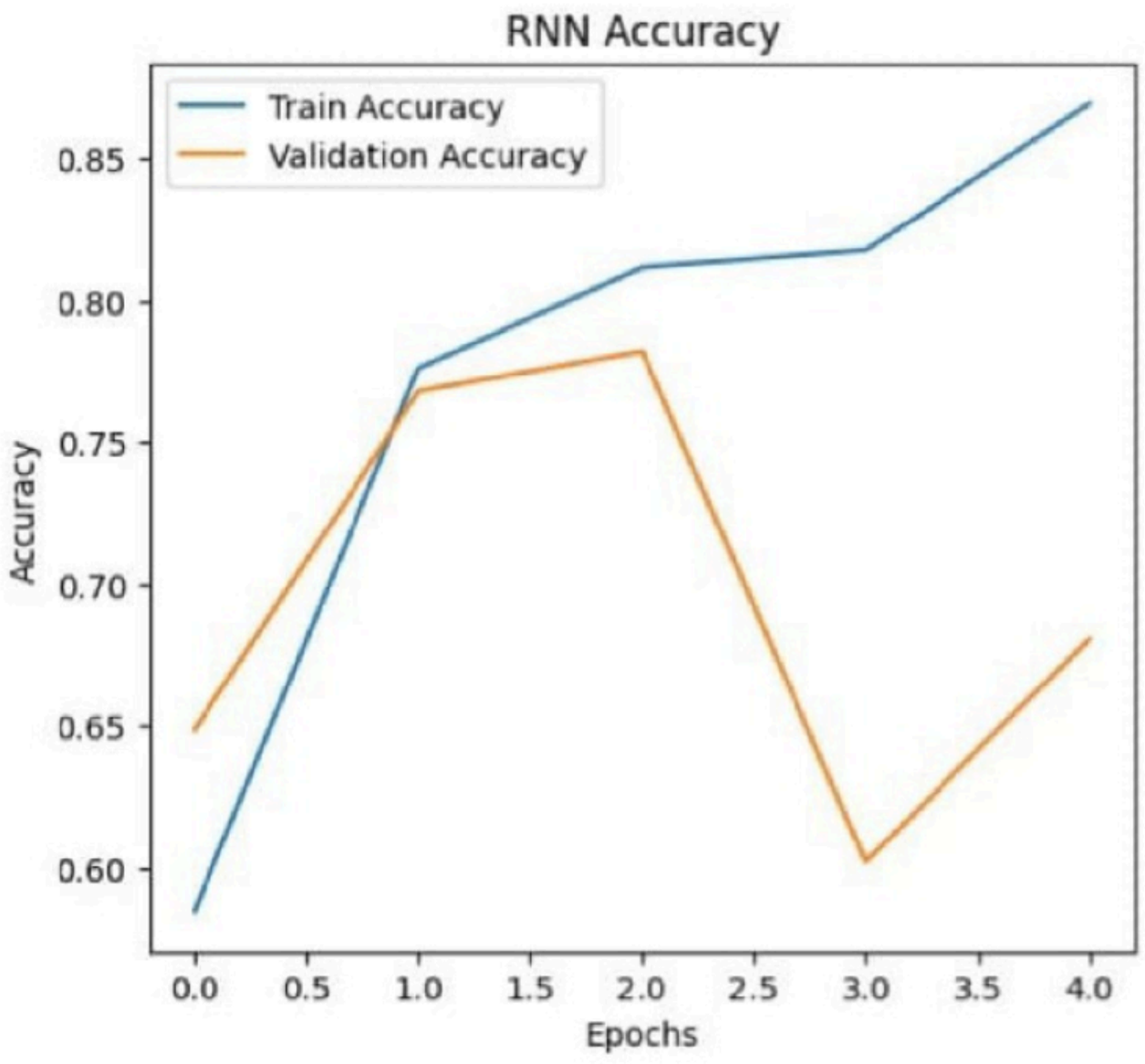
→ RNN captures temporal patterns, but struggles with long-term dependencies

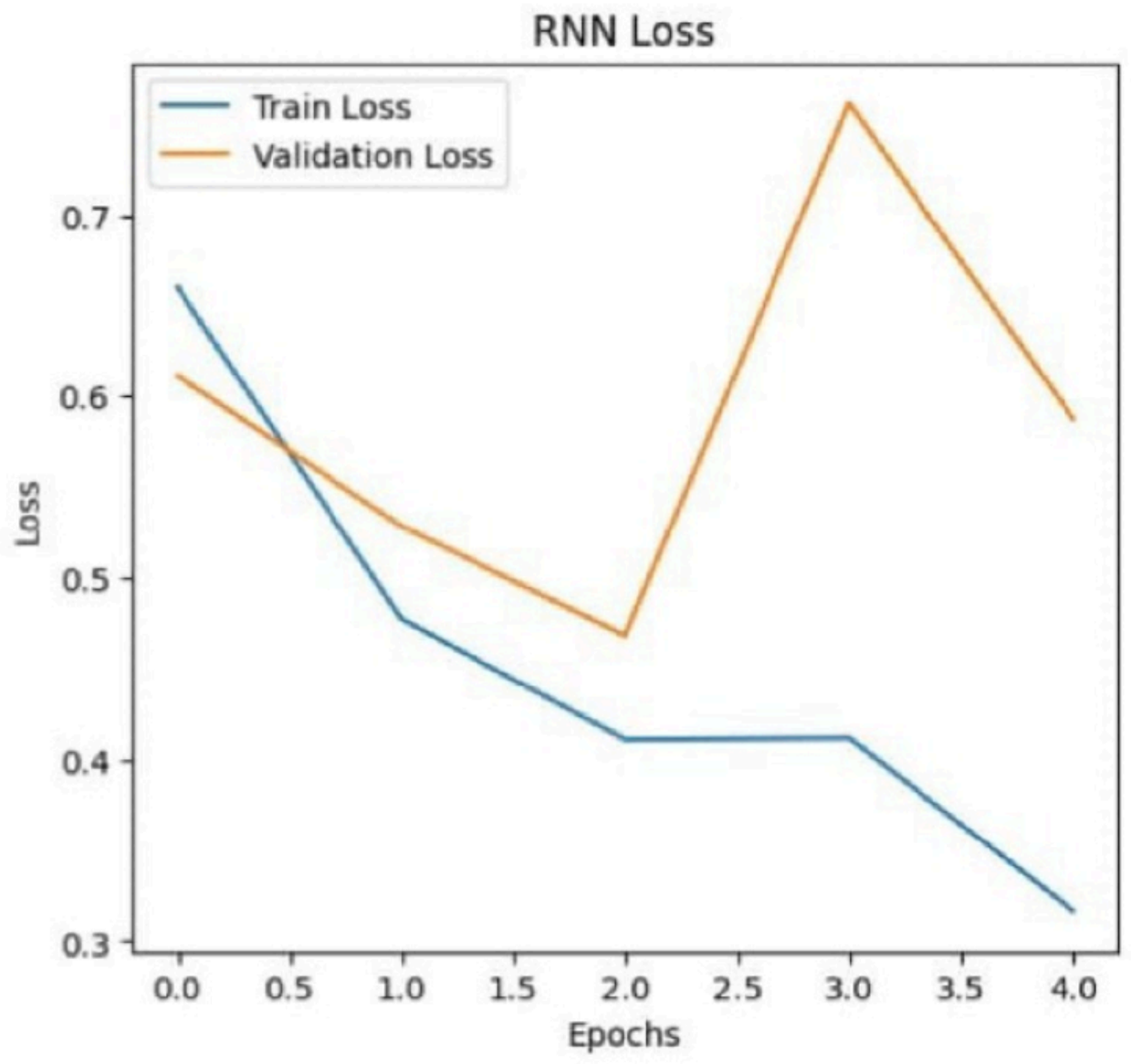
*Result:-

Successfully implemented the Recurrent Neural Network model.

```python
# RNN Implementation
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, SimpleRNN, Dense, Dropout
import matplotlib.pyplot as plt

# Load IMDB dataset
vocab_size = 10000
maxlen = 200

(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=vocab_size)
x_train = pad_sequences(x_train, maxlen=maxlen)
x_test = pad_sequences(x_test, maxlen=maxlen)

# Build RNN model
rnn_model = Sequential([
    Embedding(input_dim=vocab_size, output_dim=128, input_length=maxlen),
    SimpleRNN(128, dropout=0.2),
    Dense(1, activation='sigmoid')
])

rnn_model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# Train model
rnn_history = rnn_model.fit(
    x_train, y_train,
    epochs=5,
    batch_size=64,
    validation_split=0.2
)

# Evaluate model
rnn_score = rnn_model.evaluate(x_test, y_test)
print(f"RNN Test Accuracy: {rnn_score[1]*100:.2f}% | Loss: {rnn_score[0]:.4f}")

# Plot Accuracy and Loss
plt.figure(figsize=(12,5))

# Accuracy
plt.subplot(1,2,1)
plt.plot(rnn_history.history['accuracy'], label='Train Accuracy')
plt.plot(rnn_history.history['val_accuracy'], label='Validation Accuracy')
plt.title('RNN Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

# Loss
plt.subplot(1,2,2)
plt.plot(rnn_history.history['loss'], label='Train Loss')
plt.plot(rnn_history.history['val_loss'], label='Validation Loss')
plt.title('RNN Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz
17464789/17464789 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
Epoch 1/5
/usr/local/lib/python3.12/dist-packages/keras/src/layers/core/embedding.py:97: UserWarning: Argument `input_length` is deprecated. Just remove it.
  warnings.warn(
313/313 ━━━━━━━━━━━━━━━━━━━━ 48s 147ms/step - accuracy: 0.5433 - loss: 0.6817 - val_accuracy: 0.6488 - val_loss: 0.6114
Epoch 2/5
313/313 ━━━━━━━━━━━━━━━━━━━━ 80s 141ms/step - accuracy: 0.7572 - loss: 0.5022 - val_accuracy: 0.7680 - val_loss: 0.5285
Epoch 3/5
313/313 ━━━━━━━━━━━━━━━━━━━━ 44s 140ms/step - accuracy: 0.8231 - loss: 0.4031 - val_accuracy: 0.7822 - val_loss: 0.4681
Epoch 4/5
313/313 ━━━━━━━━━━━━━━━━━━━━ 45s 143ms/step - accuracy: 0.8235 - loss: 0.4016 - val_accuracy: 0.6022 - val_loss: 0.7613
Epoch 5/5
313/313 ━━━━━━━━━━━━━━━━━━━━ 81s 140ms/step - accuracy: 0.8476 - loss: 0.3546 - val_accuracy: 0.6810 - val_loss: 0.5877
782/782 ━━━━━━━━━━━━━━━━━━━━ 18s 23ms/step - accuracy: 0.6824 - loss: 0.5871
RNN Test Accuracy: 68.15% | Loss: 0.5839
```


RNN Accuracy
Train Accuracy
Validation Accuracy
Accuracy
0.85
0.80
0.75
0.70
0.65
0.60
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Epochs


RNN Loss
Train Loss
Validation Loss
Loss
0.7
0.6
0.5
0.4
0.3
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Epochs