```python
import torch
import torch.nn as nn
import torch.optim as optim
import numpy as np
import matplotlib.pyplot as plt

# Step 1: Prepare the dataset
np.random.seed(42)
x = np.linspace(0, 100, 1000)
data = np.sin(x)

# Convert sequence to supervised data
def create_dataset(seq, look_back=20):
    X, Y = [], []
    for i in range(len(seq) - look_back):
        X.append(seq[i:i+look_back])
        Y.append(seq[i+look_back])
    return np.array(X), np.array(Y)

look_back = 20
X, Y = create_dataset(data, look_back)
```

{} Variables   Terminal   ✓ 5:56 PM

```python
# Split train-test data
train_size = int(len(X) * 0.8)
X_train, X_test = X[:train_size], X[train_size:]
y_train, y_test = Y[:train_size], Y[train_size:]

# Convert to PyTorch tensors
X_train = torch.tensor(X_train, dtype=torch.float32).unsqueeze(-1)  # [batch, time, feature]
y_train = torch.tensor(y_train, dtype=torch.float32).unsqueeze(-1)
X_test = torch.tensor(X_test, dtype=torch.float32).unsqueeze(-1)
y_test = torch.tensor(y_test, dtype=torch.float32).unsqueeze(-1)

# Step 2: Define the RNN Model
class RNNModel(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(RNNModel, self).__init__()
        self.rnn = nn.RNN(input_size, hidden_size, batch_first=True)
        self.fc = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        out, _ = self.rnn(x)
        out = self.fc(out[:, -1, :])  # use output from last time step
```

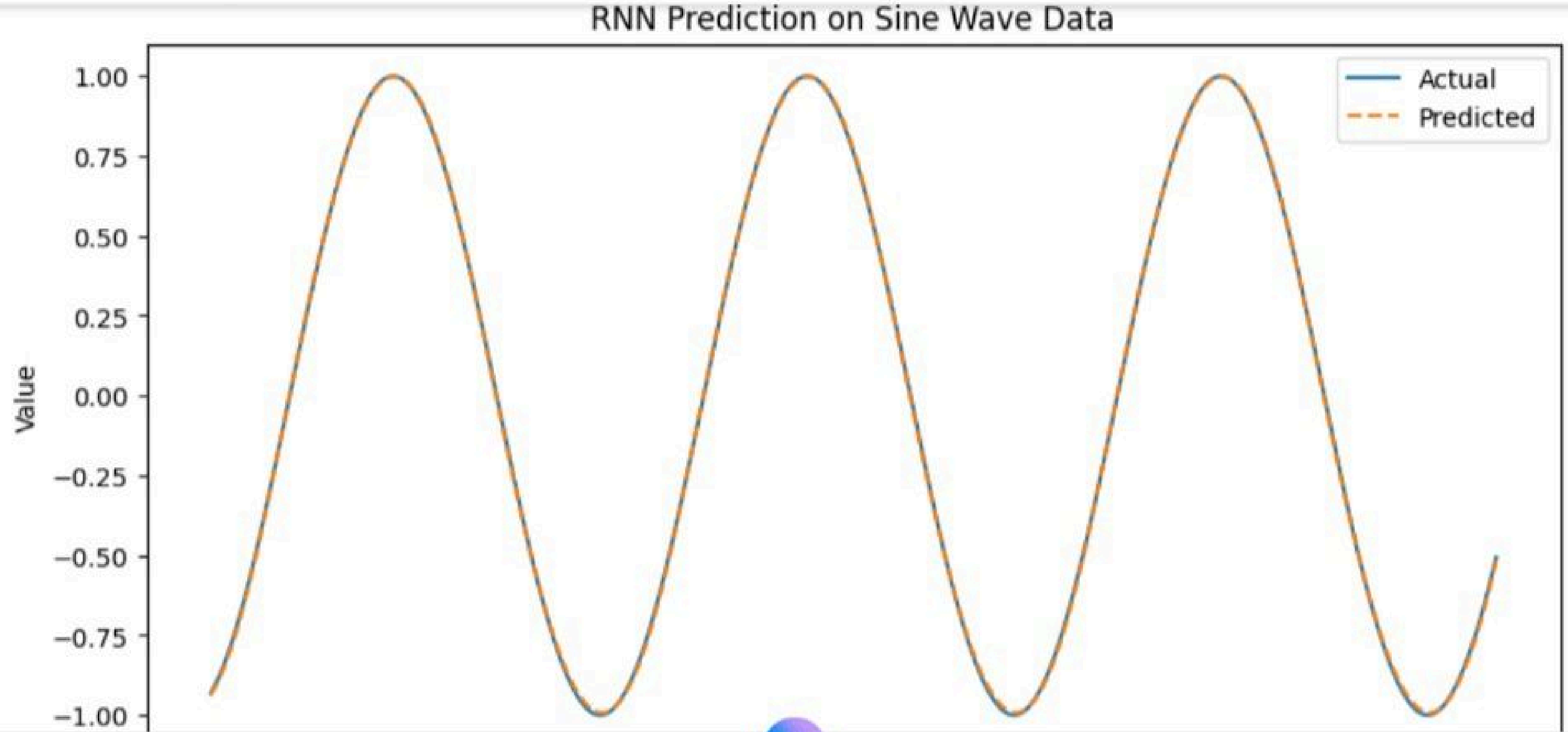
{} Variables   Terminal   ✓ 5:56 PM

```python
        return out

# Step 3: Initialize model, loss, and optimizer
model = RNNModel(input_size=1, hidden_size=32, output_size=1)
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.01)

# Step 4: Train the model
epochs = 100
for epoch in range(epochs):
    model.train()
    output = model(X_train)
    loss = criterion(output, y_train)

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    if (epoch+1) % 10 == 0:
        print(f"Epoch [{epoch+1}/{epochs}], Loss: {loss.item():.6f}")
```

```python
# Step 5: Evaluate the model
model.eval()
predicted = model(X_test).detach().numpy()

# Step 6: Plot actual vs predicted
plt.figure(figsize=(10,5))
plt.plot(range(len(y_test)), y_test, label='Actual')
plt.plot(range(len(predicted)), predicted, label='Predicted', linestyle='--')
plt.title("RNN Prediction on Sine Wave Data")
plt.xlabel("Time Step")
plt.ylabel("Value")
plt.legend()
plt.show()
```

```
Epoch [10/100], Loss: 0.010170
Epoch [20/100], Loss: 0.001869
Epoch [30/100], Loss: 0.001341
Epoch [40/100], Loss: 0.000489
Epoch [50/100], Loss: 0.000267
Epoch [60/100], Loss: 0.000118
Epoch [70/100], Loss: 0.000100
Epoch [80/100], Loss: 0.000077
Epoch [90/100], Loss: 0.000058
Epoch [100/100], Loss: 0.000047
```

Commands + Code + Text Run all Reconnect



Variables Terminal 5:56 PM

9/10/25 9. BUILD A RECURRENT NEURAL NETWORK

Aim :

TO implement and train a Recurrent neural network

Objective :

1. To understand the structure and working of RNNs.
2. To implement an RNN model using PyTorch framework.
3. To train and test the RNN model on sequential data
4. To observe how RNN learns temporal dependencies across time steps.

Pseudo Code :

step 1 : Import required libraries

step 2 : Prepare the dataset

- Create or load sequential data
- Normalize and Convert to tensors
- split into training & testing sets
- Reshape data as [batch_size, time_steps, features]

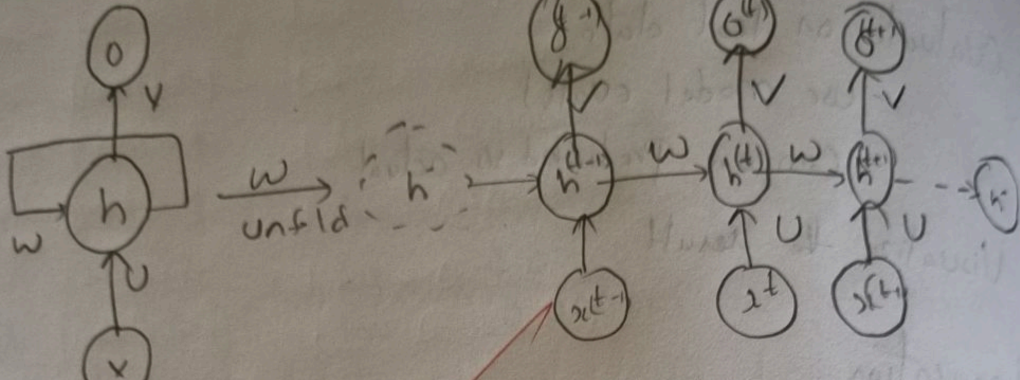
step 3 : Define RNN model

step 4 : Initialize model, define loss and optimizer

step 5 : Train the model

step 6 : Evaluate the model

step 7 : Display results



9. BUILD

Aim :

To in

network

Objective :

1. To unde

2. To impl

3. To trai

4. To o

time steps.

Pseudo

Step 1 :

step 2

step 3

step 4

step

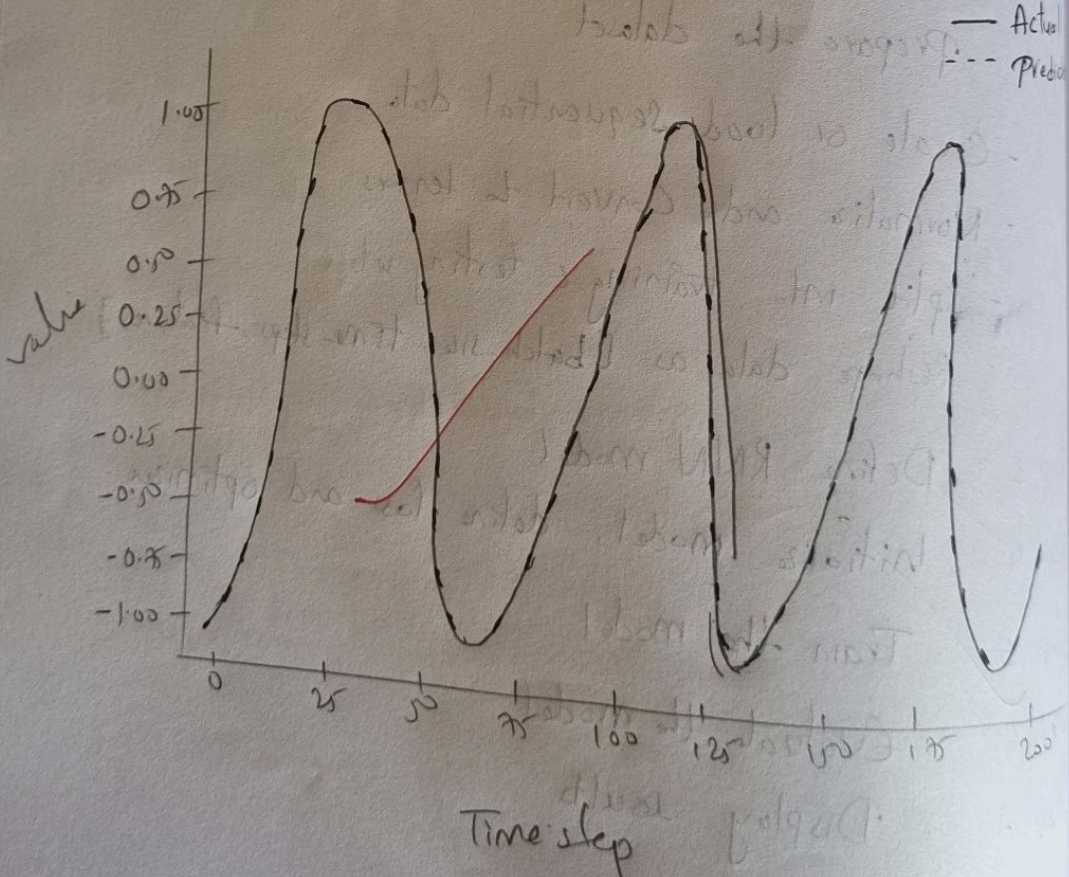
step

step

O/P

Epoch [10/100], loss : 0.010170
Epoch [20/100], loss : 0.001869
Epoch [30/100], loss : 0.001341
Epoch [40/100], loss : 0.000489
Epoch [50/100], loss : 0.000267
Epoch [60/100], loss : 0.000118
Epoch [70/100], loss : 0.000100
Epoch [80/100], loss : 0.000077
Epoch [90/100], loss : 0.000058
Epoch [100/100], loss : 0.000047

RNN Prediction on sine wave data



Observation :

1. The RNN mo
that it learne
2. The network
3. With more
slightly, but

Result :

Succe
network mo

Observation :

1. The RNN model's training loss decreased steadily, showing that it learned from sequential input

2. The network was able to predict future sequence values

3. With more hidden units or layers, accuracy improved slightly, but training time also increased.

Result :

Successfully implemented an Recurrent Neural network model.