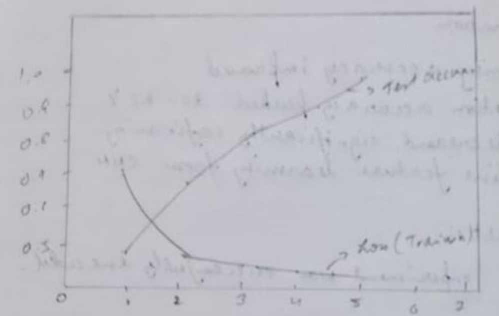
Output : Epoch 1: Loss 0.6832 . Accuracy : 0.4921

Epoch 2 : Loss : 0.0474 Accuracy 0.6821

Epoch 3 : Loss : 0.0019 . Accuracy : 0.7847

Epoch 4 : Loss : 0.0011 . Accuracy : 0.8492

Epoch 5 : Loss : 0.0009 Accuracy : - 0.9531



30-9-25

Lab 9 Build a Recurrent Neural Network.

AIM :

To design, implement and evaluate a RNN model for sequential data, such as text, and analyze its performance.

Pseudo code :

- Load the dataset
- Preprocess data
- convert sequences into input-output pairs
- Define RNN model :
- RNN layer + dense output layer with activation.
- Compile Model
- Select optimizer
- Train Model
- Fit data into RNN for given a batch size.
- Monitor validation Loss.
- Evaluate Model :
- Test data.
- Visualize result :
  Plot accuracy and loss curves.
  Conclude observation and results.

Observation

- The training accuracy increases with epochs, while the loss decreases.
- Overfitting can occur if too many epochs are used without regulaziation (dropout)
- RNN captures seq. dependencies better than feedforward networks
- LSTM variants perform more effictively on long sequences due to vanishing gradient mitigation.
- Validation performance depends on dataset Complexity and preprocessing quality.

Result :

RNN was successfully built and trained on sequential data. "Successfully implemented".

30/9/25

Untitled0.ipynb — Saving failed since 8:13 PM

File Edit View Insert Runtime Tools Help

Commands + Code + Text ▷ Run all

[1]
```python
import torch
import torch.nn as nn
import torch.optim as optim
import matplotlib.pyplot as plt
```

[2]
```python
data = torch.tensor([[0, 1, 2],
                     [1, 2, 3],
                     [2, 3, 4]], dtype=torch.float32)
targets = torch.tensor([[3], [4], [5]], dtype=torch.float32)
```

[3]
```python
data = data.view(3, 1, 3)
```

[4]
```python
class SimpleRNN(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(SimpleRNN, 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, :])
        return out
```

[5]
```python
input_size = 3
hidden_size = 5
output_size = 1
learning_rate = 0.01
epochs = 200
```

[6]
```python
model = SimpleRNN(input_size, hidden_size, output_size)
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=learning_rate)
```

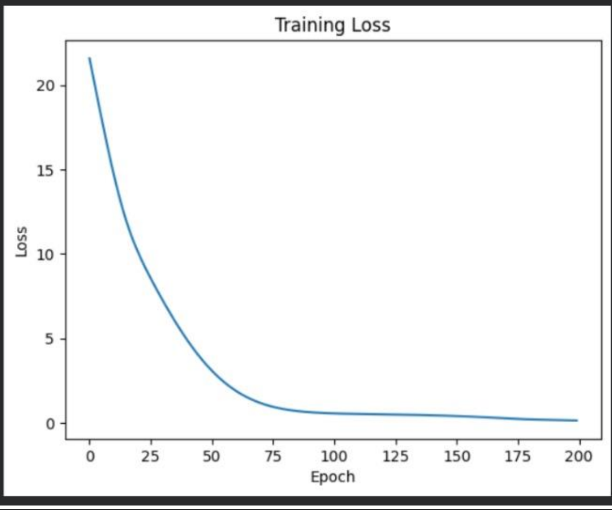
[7]
```python
losses = []
for epoch in range(epochs):
    optimizer.zero_grad()
    output = model(data)
    loss = criterion(output, targets)
    loss.backward()
    optimizer.step()

    losses.append(loss.item())
    if (epoch+1) % 20 == 0:
        print(f'Epoch [{epoch+1}/{epochs}], Loss: {loss.item():.4f}')
```

```
Epoch [20/200], Loss: 10.3701
Epoch [40/200], Loss: 5.0838
Epoch [60/200], Loss: 1.9604
Epoch [80/200], Loss: 0.8091
Epoch [100/200], Loss: 0.5502
Epoch [120/200], Loss: 0.4939
Epoch [140/200], Loss: 0.4378
Epoch [160/200], Loss: 0.3354
Epoch [180/200], Loss: 0.1930
Epoch [200/200], Loss: 0.1241
```

[8]
```python
plt.plot(losses)
plt.xlabel('Epoch')
plt.ylabel('Loss')
```

[8]

```python
plt.plot(losses)
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training Loss')
plt.show()
```



[9]

```python
test_input = torch.tensor([[3, 4, 5]], dtype=torch.float32).view(1, 1, 3)
pred = model(test_input)
print("Prediction for input [3,4,5]:", pred.item())
```

```
Prediction for input [3,4,5]: 4.472318649291992
```