04-10-25 Exp: 8

Experiment Using LSTM

Aim:

To design, implement and train a Long Short-Term Memory (LSTM) model to learn temporal dependencies in sequential data and to predict future sequence values.

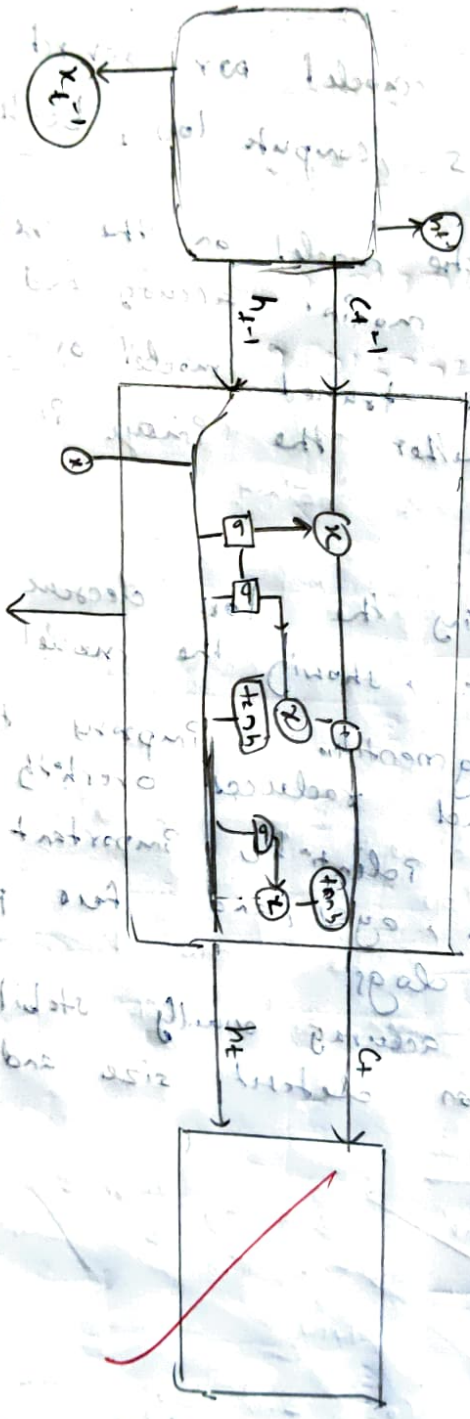
Objectives:

- To understand the architecture and functions of LSTM networks and their role in solving vanishing gradient problems
- To preprocess sequential or time-series data into supervised learning format for LSTM input
- To construct and train an LSTM model using torch.nn.LSTM
- To measure training performance using Mean Squared Error.
- To visualize the loss curve and interpret results.

Pseudo Code:

- Import pytorch and supporting libraries
- Create sequential input-output pairs from time series data
- Define LSTM model using nn.LSTM and nn.Linear
- Specify optimizer (Adam) and loss function (MSE)
- Train the model across epochs, updating weights
- Plot loss vs epochs graph and analyze result

LSTM Architecture



Training ~~RNN~~ LSTM

Epoch [5/30], Loss: 0.045311

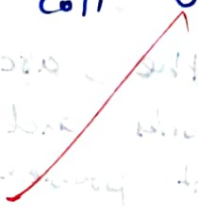
Epoch [10/30], Loss: 0.029799

Epoch [15/30], Loss: 0.020167

Epoch [20/30], Loss: 0.18411

Epoch [25/30], Loss: 0.019032

Epoch [30, 30], Loss: 0.016702.



Observation

- The LSTM model converged faster and with lower final loss compared to RNN
- Training curve was smoother, showing stable learning behaviour
- LSTM effectively - some extend longer sequence dependencies
- The model generalized well without oscillation in loss
- Increasing hidden size could further improve accuracy.

Result

Succesfully implemented LSTM.

11/9/25

09-10-25

Exp: 9

Build a Recurrent Neural Network

Aim:

To implement and train a simple Recurrent Neural Network (RNN)

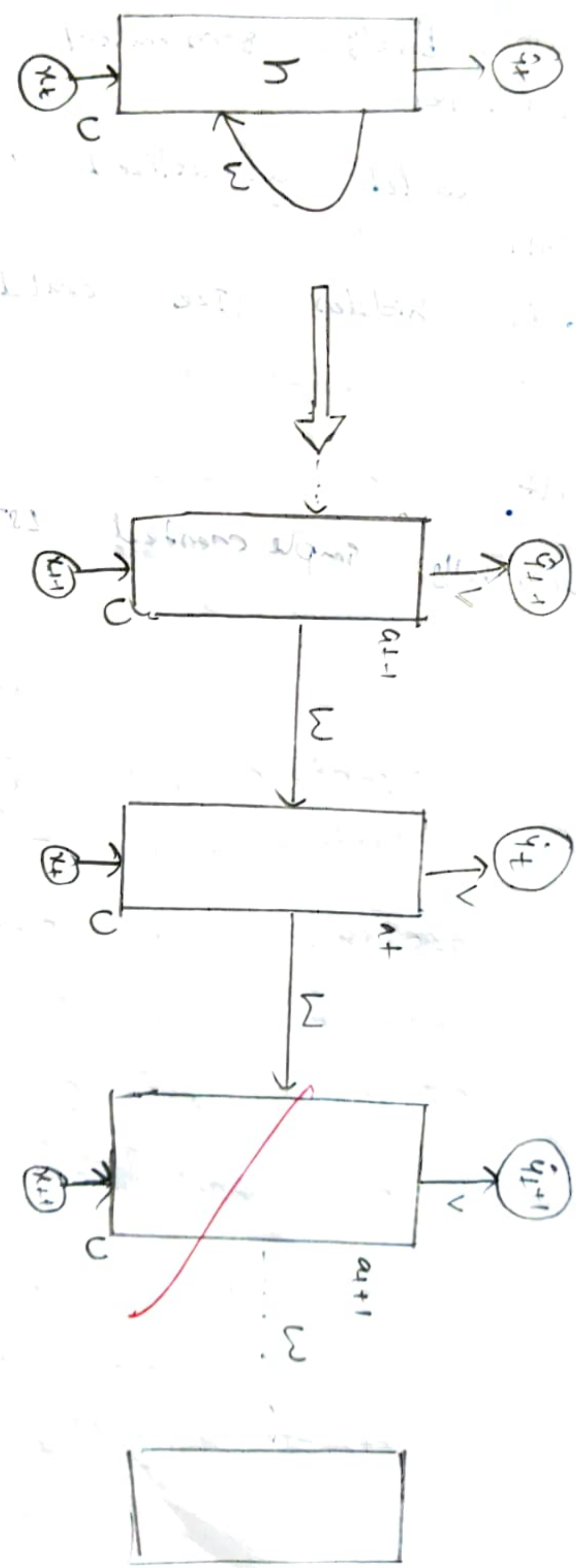
Objectives:

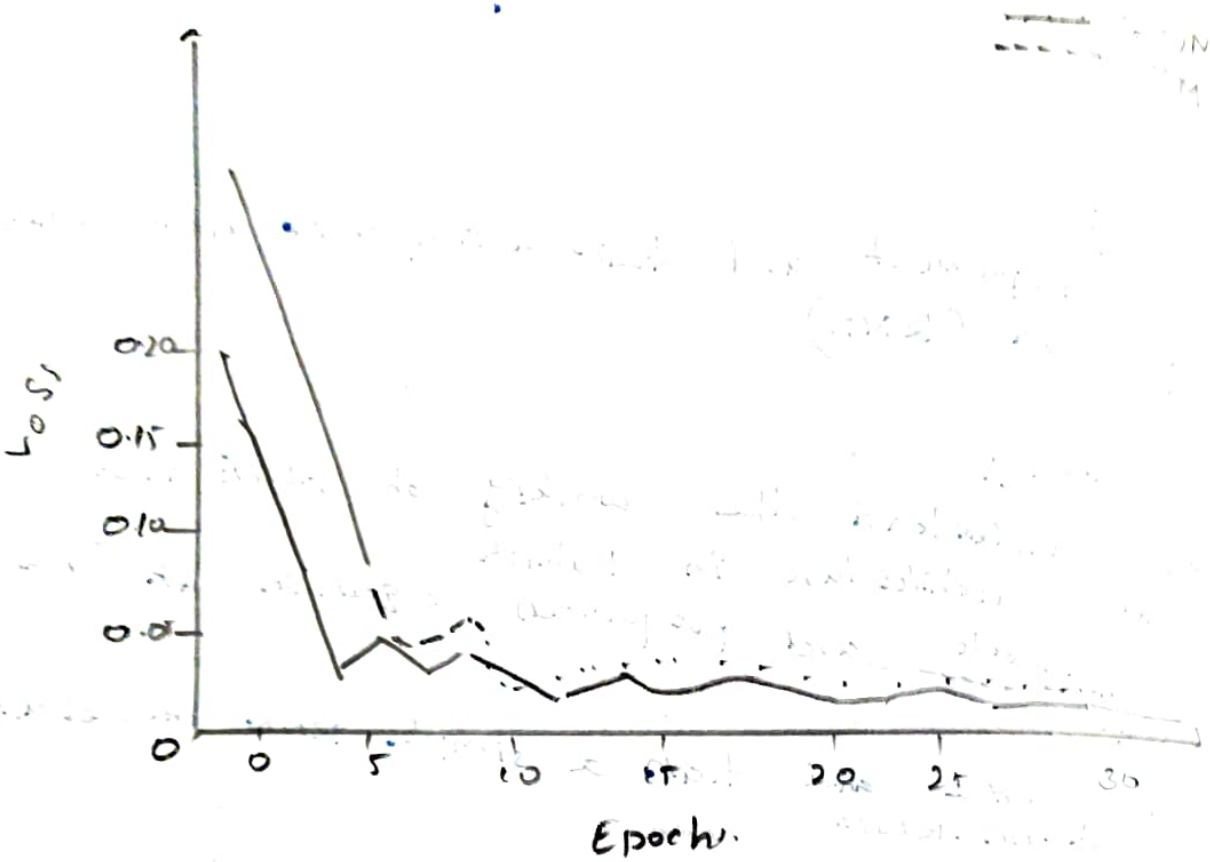
1. To understand the working of RNNs and their architecture in Pytorch
2. To generate and pre process sequential data for model training
3. To define and train a simple RNN model using torch.nn.RNN.
4. To evaluate the network performance based on training loss
5. To visualize the learning curve using matplotlib

Pseudocode:

- Import required libraries (torch, numpy, matplotlib)
- Create sequential data and prepare input-output pairs
- Define the RNN model class using nn.RNN and nn.Linear
- Set loss function and optimizer
- Train the model over several epochs and record training loss
- Plot the training loss curve and observe model performance

RNN Architecture





Training RNN

Epoch [5/30], Loss: 0.038023

Epoch [10/30], Loss: 0.017734

Epoch [15/30], Loss: 0.014965

Epoch [20/30], Loss: 0.012491

Epoch [25/30], Loss: 0.012011

Epoch [30/30], Loss: 0.010986

Observation:

1. The RNN learned sequential dependencies in the data effectively.
2. Training loss gradually decreased, indicating convergence.
3. The network captured short term sequence length and hidden unit patterns accurately.
4. Performance depended on sequence length and hidden units
5. Model learning was stable after about 10 epochs.

Result.

successfully Implemented ~~LSTM and~~ RNN