Week-11 RNN

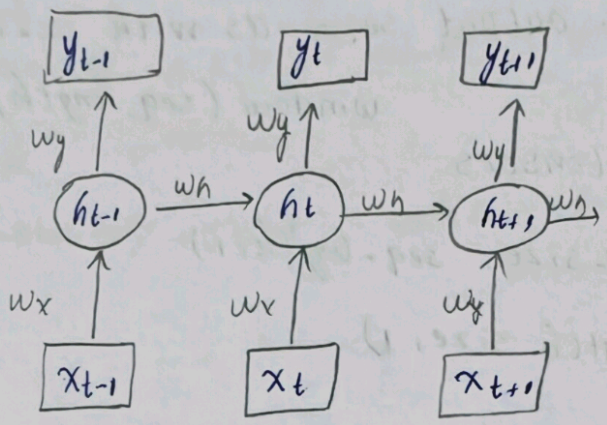
Aim: To implement a Recurrent Neural Network using PyTorch that processes sequential data and makes predictions based on past info

Algorithm:

* Import Libraries: Load PyTorch Modules
* Prepare Dataset: Define a small sequence dataset convert inputs and targets into tensor formats
* Define RNN
  Use nn.Embedding to map indices to dense vectors.
  Add nn.RNN layer to capture sequential dependencies.
  Add nn.Linear layer to map hidden state
* Initialize Hyper Parameters:
  vocabulary size, hidden layer size, learning rate, no of epochs.
  Initialize Optimizer and loss function
* Train The Model, for each epoch
  Initialize hidden state, Forward Pass: input → RNN → output
  compute loss b/w predicted output and target
  BackPropogate gradients
* Evaluate Model:
  Feed the input sequence to trained model
  Decode Predictions into readable formats

Pseudo code

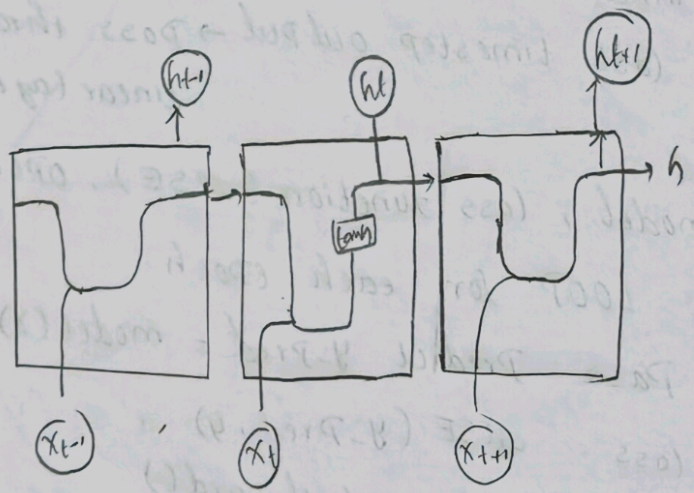
```
BEGIN
  Import  torch, nn, optim

# step-1 : Dataset Preparation
Define vocabulary and mapping (char2idx, idx2char)
Encode input sequence x, y
convert x, y to tensor
# step-2 : Define model
CLASS RNNModel:
      INIT (vocab_size, hidden_size):
        embedding = nn.Embedding (vocab_size, hidden_size)
        rnn = nn.RNN (hidden_size, hidden_size)

      FORWARD (x, hidden)
        x_embedded = embedding (x)
        output, hidden = rnn (x_embedded, hidden)
        Prediction = fc (output)
        RETURN Prediction, hidden

# step:4 - Training Loop
 For epoch in range (num_epochs):
      hidden = zeroes (initial_hidden_state)
      output, hidden = model (x, hidden)
       Loss = loss_fn ( output, y)
       optimizer.zero_grad()
       Loss.backward()

# step:5 - Final evaluation
   hidden = zeroes( initial_hidden_state)
   output, hidden = model (x, hidden)
   Predicted_sequence = decode (output)
    Print( Predicted_sequence)
END
```



$$h_t = \tanh(w_h h_{t-1} + x_t w_x + b_h)$$

$$y_t = w_y h_t + b_y$$

RNN Internal Architecture



Epoch 1

accuracy: 0.5433 - loss: 0.6817

EPOC 2

accuracy: 0.7579 - loss: 0.5022

Epoch 3

Accuracy: 0.8231 - loss: 0.4031

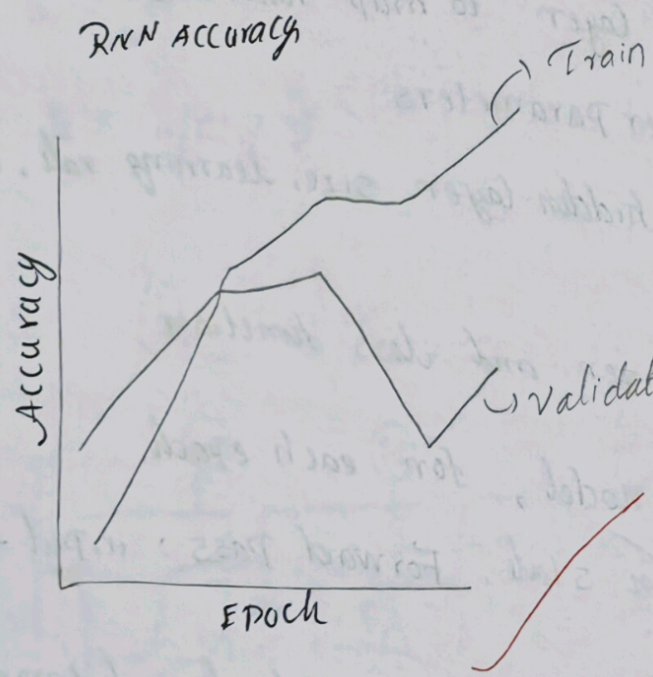
EPoch 4

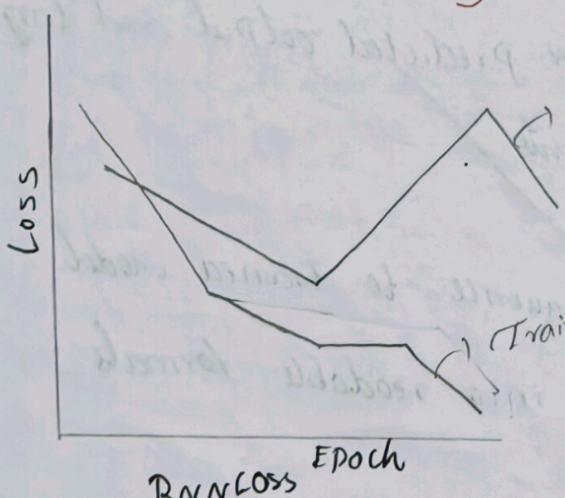
Accuracy: 0.8235 - loss: 0.4016

EPoch 5

Accuracy: 0.8476 - loss: 0.3546

RNN Test Accuracy: 68.15%, Loss: 0.5839





Result:
Implementation of RNN using IMDB was done successfully

Date
Exp 3 30/09/25

8. Implement a LSTM

Aim:
To implement a Long short Term memory model using Pytorch for time series forecasting.

Algorithm:

* Data Preparation - Generate a sine wave dataset as the time series input.
* Divide the sequence into input - output pairs
* Model Design - Define an LSTM network
* Use MSE as the loss function
* Use Adam optimizer for parameter updates
* Training Phase - initialize hidden and all states
* compute loss between Predicted and Actual values
* BackPropogate the error and update weights
* Feed test input into the trained model
* Plot Actual data vs Predicted data to evaluate Performance.

*Pseudocode:

BEGIN

Import libraries (torch, numpy, matplotlib)

Generate sine wave dataset

- Define timestamps
- Create input-output sequences with lookback window (seq_length)

Convert dataset to tensors

- X shape = (batch_size, seq_length, 1)
- y shape = (batch_size, 1)

Define LSTM Model class

- LSTM layer (input_size, hidden_size, num_layers)
- Fully connected layer for output
- Forward Pass:
  - initialize h0, c0
  - Pass input through LSTM
  - Take last timestep output → Pass through linear layer

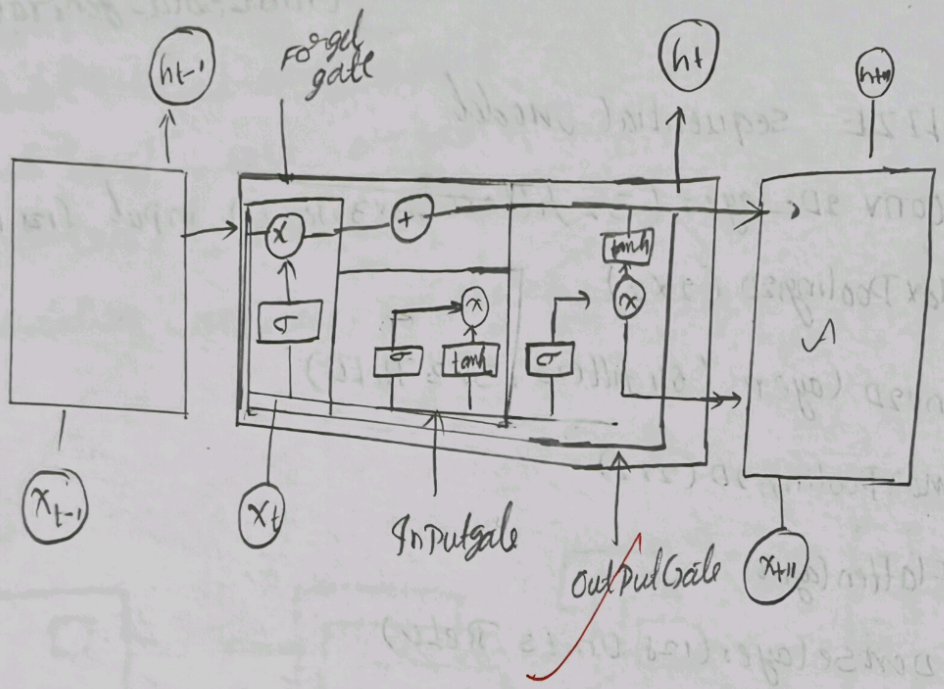
- Initialize model, loss function (MSE), optimizer
- TRAINING LOOP for each epoch:
  - Forward Pass: Predict y_pred = model(X)
  - compute loss = MSE(y_pred, y)
  - BackPropogation: loss.backward()

Plot actual data vs Predicted data

Observation? Result?

END

LSTM Architecture



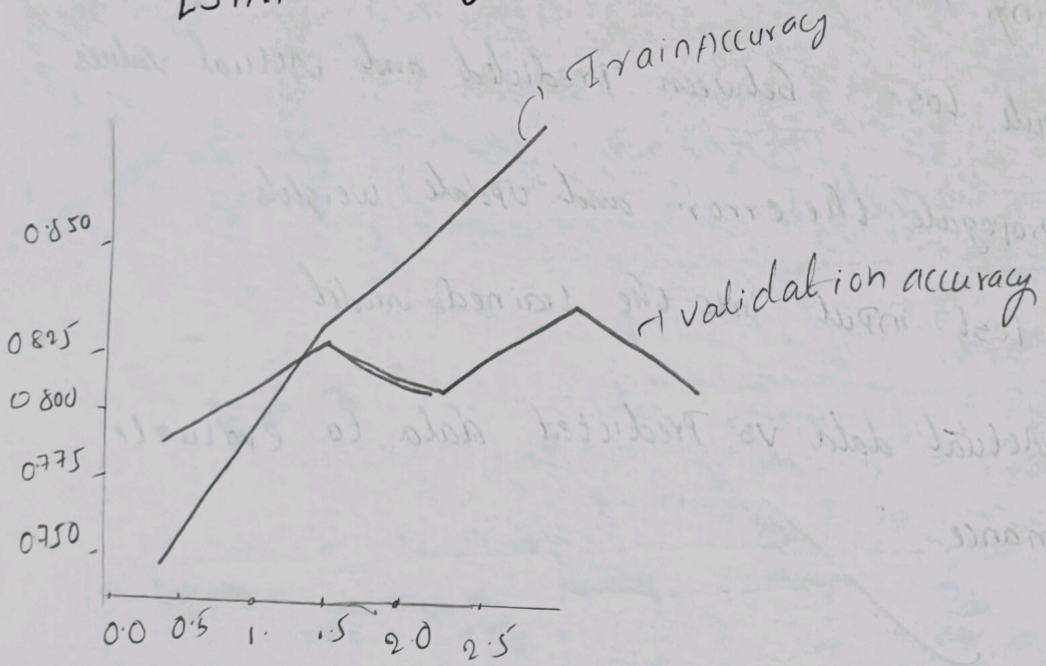
Epoch 1
val_accuracy : 0.6676 - val_loss : 0.4151

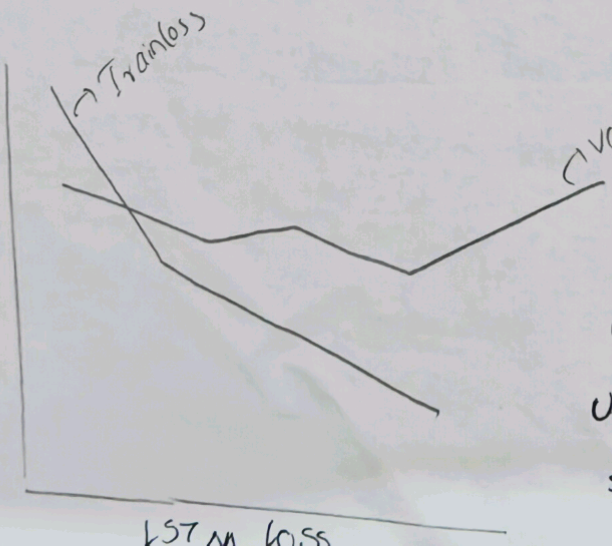
Epoch 2
val_accuracy : 0.8358 - val_loss : 0.3951

Epoch 3
val_accuracy : 0.8328 - val_loss : 0.3862

Epoch 4
val_accuracy : 0.8528 - val_loss : 0.3639

LSTM ACCURACY



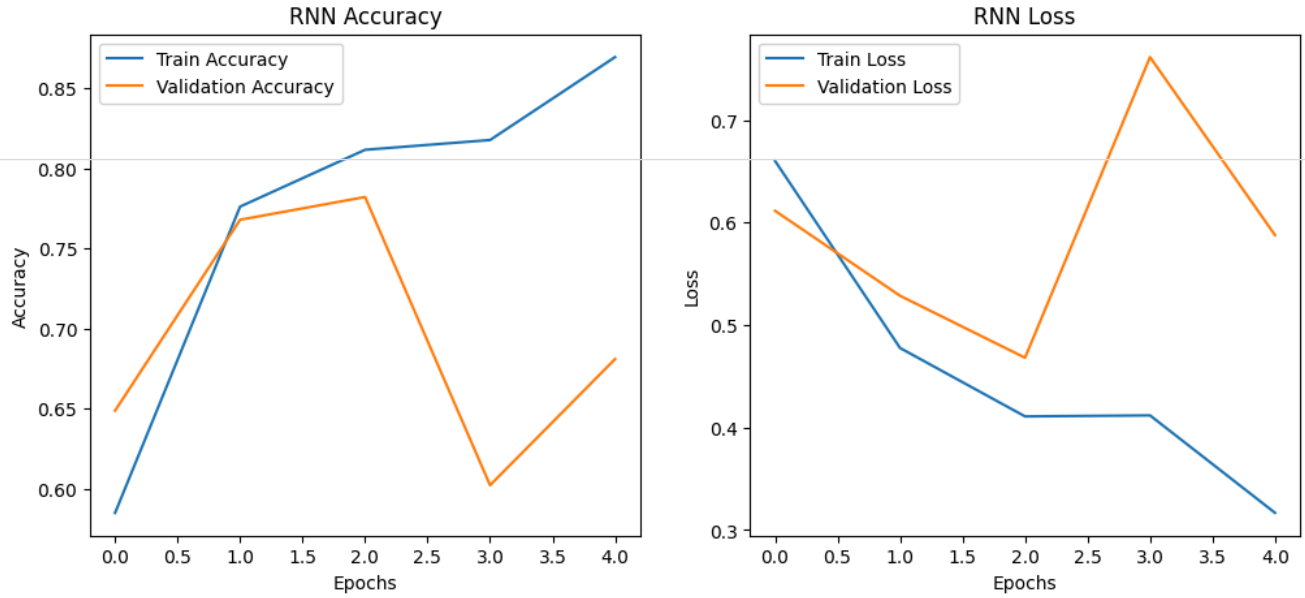


LSTM LOSS

Result:
Implementation of LSTM using IMDB Dataset was done successfully ✓

```
# 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 deprecate
  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
```



```
# LSTM 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, LSTM, 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 LSTM model
lstm_model = Sequential([
    Embedding(input_dim=vocab_size, output_dim=128, input_length=maxlen),
    LSTM(128, dropout=0.2, recurrent_dropout=0.2),
    Dense(1, activation='sigmoid')
])

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

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

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

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

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

# Loss
plt.subplot(1,2,2)
plt.plot(lstm_history.history['loss'], label='Train Loss')
plt.plot(lstm_history.history['val_loss'], label='Validation Loss')
plt.title('LSTM 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 `inp
  warnings.warn(
313/313 ━━━━━━━━━━━━━━━━━━━━ 193s 603ms/step - accuracy: 0.6676 - loss: 0.5828 - val_accuracy: 0.8122 - v
Epoch 2/5
313/313 ━━━━━━━━━━━━━━━━━━━━ 198s 593ms/step - accuracy: 0.8517 - loss: 0.3529 - val_accuracy: 0.8358 - v
Epoch 3/5
313/313 ━━━━━━━━━━━━━━━━━━━━ 206s 606ms/step - accuracy: 0.8840 - loss: 0.2872 - val_accuracy: 0.8328 - v
Epoch 4/5
313/313 ━━━━━━━━━━━━━━━━━━━━ 204s 611ms/step - accuracy: 0.9019 - loss: 0.2525 - val_accuracy: 0.8528 - v
Epoch 5/5
313/313 ━━━━━━━━━━━━━━━━━━━━ 198s 598ms/step - accuracy: 0.9328 - loss: 0.1832 - val_accuracy: 0.8338 - v
782/782 ━━━━━━━━━━━━━━━━━━━━ 62s 79ms/step - accuracy: 0.8309 - loss: 0.4013
LSTM Test Accuracy: 83.48% | Loss: 0.3946
```