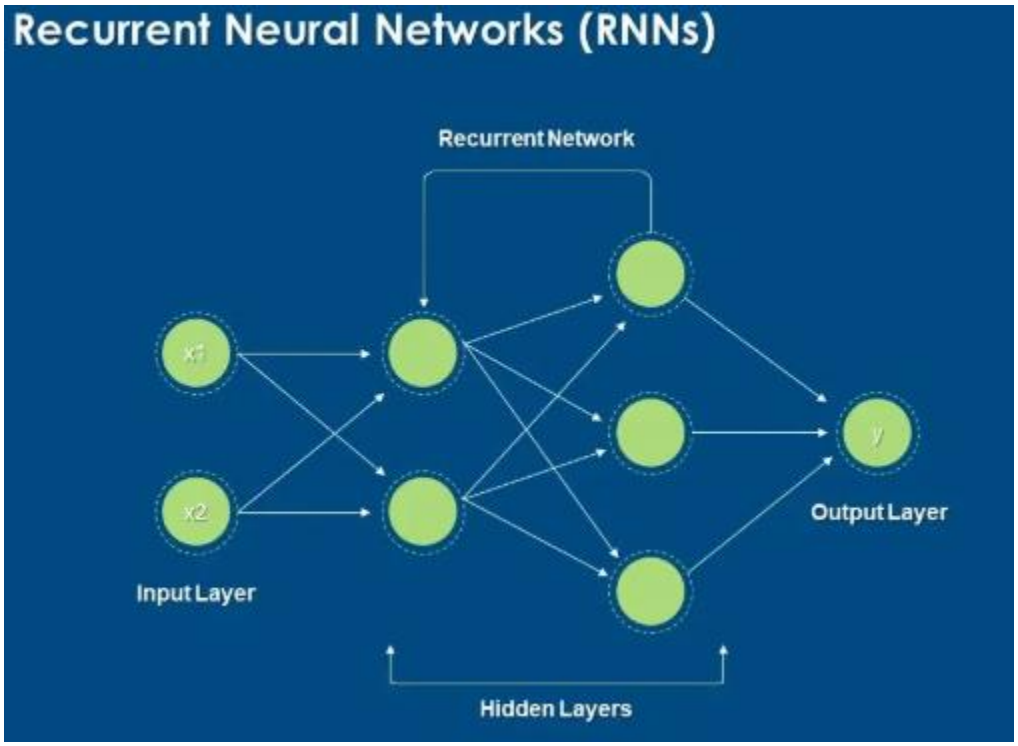
LAB No. 7

Implementation of Recurrent Neural Network (RNN)

In this lab, students will study and implement a Recurrent Neural Network (RNN), a deep learning model designed for sequential and time-dependent data. Unlike feedforward neural networks, RNNs have feedback connections that allow them to retain information from previous time steps. Students will begin with a simple RNN model to understand sequence processing and then apply RNNs to real-world problems such as sequence classification and text processing. Model performance will be evaluated using accuracy metrics.

Introduction

A **Recurrent Neural Network (RNN)** is a class of neural networks specifically designed to process **sequential data**, where the order of inputs matters. RNNs maintain a **hidden state (memory)** that captures information from previous inputs and influences current predictions.



Key Concepts:

- **Sequential Input** – time series or text data
- **Hidden State** – stores past information
- **Recurrent Connection** – connects previous output to current input
- **Activation Functions** – tanh, ReLU
- **Backpropagation Through Time (BPTT)** – training method for RNNs

RNNs are commonly used in **speech recognition, sentiment analysis, time-series forecasting, and natural language processing**. However, basic RNNs may suffer from the **vanishing gradient problem**, which is addressed by advanced variants like **LSTM and GRU**.

Solved Examples:

Example 1:

Simple RNN for Binary Sequence Classification

Build a simple RNN to classify whether a sequence indicates **Pass (1)** or **Fail (0)** based on study performance over time.

Solution:

```
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import SimpleRNN, Dense

# Sample sequential data (5 students, 3 time steps, 1 feature)
X = np.array([
    [[1], [1], [0]],
    [[1], [1], [1]],
    [[0], [0], [1]],
    [[0], [0], [0]],
    [[1], [0], [1]]
])

y = np.array([1, 1, 0, 0, 1])
```

```
# Build RNN model
model = Sequential()
model.add(SimpleRNN(8, activation='tanh', input_shape=(3, 1)))
model.add(Dense(1, activation='sigmoid'))

# Compile and train
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.fit(X, y, epochs=100, verbose=0)

# Prediction
prediction = model.predict([[[1], [1], [1]]])
print("Predicted Result (1=Pass, 0=Fail):", int(prediction[0][0] > 0.5))
```

Explanation

The RNN processes input sequences and uses memory to capture temporal patterns before making a classification.

Example 2:

RNN for Time Series Prediction Predict the next value in a simple numerical sequence using an RNN.

Solution:

```
# Generate sequence data
X = np.array([
    [[1], [2], [3]],
    [[2], [3], [4]],
    [[3], [4], [5]],
    [[4], [5], [6]]
])

y = np.array([4, 5, 6, 7])
# Build RNN model
model = Sequential()
model.add(SimpleRNN(10, activation='tanh', input_shape=(3, 1)))
model.add(Dense(1))
```

```
# Compile and train
model.compile(optimizer='adam', loss='mse')
model.fit(X, y, epochs=200, verbose=0)

# Predict next value
prediction = model.predict([[[5], [6], [7]]])
print("Predicted Next Value:", prediction[0][0])
```

Explanation

The RNN learns temporal relationships in numeric sequences and predicts future values.

Example 3:

RNN for Text Classification (Simple Sentiment Analysis)

Build a simple RNN model to classify text sentiment as **positive or negative**.

Solution

```
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

# Sample text data
texts = [
    "I love this course",
    "This lab is very good",
    "I hate this subject",
    "This is boring",
    "Excellent explanation"
]

labels = [1, 1, 0, 0, 1]

# Tokenize text
tokenizer = Tokenizer(num_words=100)
```

```
tokenizer.fit_on_texts(texts)

sequences = tokenizer.texts_to_sequences(texts)
padded_sequences = pad_sequences(sequences, maxlen=5)

# Build RNN model
model = Sequential()
model.add(SimpleRNN(16, activation='tanh', input_shape=(5,)))
model.add(Dense(1, activation='sigmoid'))

# Compile and train
model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
model.fit(padded_sequences, labels, epochs=100, verbose=0)

# Prediction
prediction = model.predict(padded_sequences)
print("Predicted Sentiments:", prediction.round())
```

The RNN processes text sequences word by word, capturing contextual meaning for sentiment classification.

Limitations of Basic RNN

- Vanishing gradient problem
- Difficulty learning long-term dependencies
- Slower training compared to feedforward networks

LAB Assignment No 6

LAB Task 1:

Next Word Prediction using RNN

Objective: Learn how RNNs can predict the next word in a sentence.
Dataset: Any small text corpus — e.g., *Shakespeare.txt* or *Wikipedia sample*.
Tasks:

1. Load and clean the text data.
2. Tokenize and convert text into sequences.
3. Build a simple **RNN model** using keras.layers.SimpleRNN.
4. Train it to predict the next word given previous 3–5 words.
5. Test by entering a custom text prompt and predict the next word.

Output:

Model predicts probable next word, e.g.,
Input: "The sun is" → **Output:** "shining"

Required Solution

- Understanding of dataset
- Understanding of code
- Learning outcomes analysis

LAB Task 2:

Stock Price Prediction using RNN

Objective: Predict future stock prices using time series data.
Dataset: Use *Google Stock Price* dataset (from Kaggle or Yahoo Finance).
Tasks:

1. Import dataset and normalize values.
2. Prepare time-step sequences (e.g., 60 previous days → next day price).
3. Build and train an **RNN model** using SimpleRNN layers.
4. Evaluate predictions vs actual prices (plot graph).

Output:

Line graph showing predicted vs real stock price trend.

Required Solution

- Understanding of dataset
- Understanding of code
- Learning outcomes analysis

LAB Task 3:

Sentiment Analysis using RNN

Objective: Classify movie reviews as positive or negative using RNN.
Dataset: *IMDb Movie Reviews* dataset (available in Keras).
Tasks:

1. Load dataset and preprocess text (tokenize and pad sequences).
2. Build RNN with Embedding + SimpleRNN layers.
3. Train for binary classification (positive/negative).
4. Evaluate accuracy on test data.

Output:

Accuracy score (e.g., 85%) and prediction for custom input text.

Required Solution

- Understanding of dataset
- Understanding of code
- Learning outcomes analysis

LAB Task 4:

Weather Forecasting using RNN

Objective: Predict future temperature based on previous days' readings.
Dataset: *Daily temperature dataset* (e.g., "Jena Climate Dataset" from TensorFlow).
Tasks:

1. Load and visualize temperature over time.

2. Prepare input-output sequences for time series prediction.
3. Build an RNN to predict next day's temperature.
4. Plot actual vs predicted temperature.

Output:

Graph showing predicted vs actual temperature trends.

Required Solution

- Understanding of dataset
- Understanding of code
- Learning outcomes analysis

LAB Task 5:

Music Note Generation using RNN

Objective: Generate new music sequences using RNN.
Dataset: *MIDI music dataset* (short sequences or melodies).
Tasks:

1. Convert MIDI data into integer-encoded notes.

2. Train an RNN on note sequences (input: previous notes → output: next note).
3. Generate a new sequence using the trained model.

Output:

A sequence of generated notes that can be converted to a playable MIDI file

Required Solution

- Understanding of dataset
- Understanding of code
- Learning outcomes analysis

LAB Assessment

Student Name		**LAB Rubrics**	CLO3 , P5, PLO5
		Total Marks	10
Registration No		**Obtained Marks**	
		Teacher Name	Dr. Syed M Hamedoon
Date		**Signature**	