LAB Assignment No 7

Recurrent Neural Network (RNN)

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:

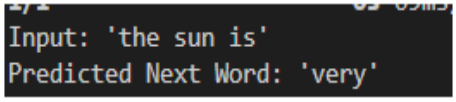
1. Load and clean the text data.
2. Tokenize and convert text into sequences.
3. Build a simple **RNN model** using keras.layers.SimpleRNN.
4. Train it to predict the next word given previous 3–5 words.
5. Test by entering a custom text prompt and predict the next word.

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, SimpleRNN, Dense
text = """
The sun is shining bright
The sun is hot today
The weather is sunny
The sun is very bright
"""
text = text.lower().strip()
tokenizer = Tokenizer()
tokenizer.fit_on_texts([text])
```

```
total_words = len(tokenizer.word_index) + 1
input_sequences = []
for line in text.split("\n"):
  token_list = tokenizer.texts_to_sequences([line])[0]
  for i in range(1, len(token_list)):
    input_sequences.append(token_list[:i+1])
max_seq_len = max(len(seq) for seq in input_sequences)
input_sequences = pad_sequences(input_sequences,
                  maxlen=max_seq_len,
                  padding='pre')
X = input_sequences[:, :-1]
y = input_sequences[:, -1]
y = tf.keras.utils.to_categorical(y, num_classes=total_words)
model = Sequential()
model.add(Embedding(total_words, 50, input_length=max_seq_len-1))
model.add(SimpleRNN(100))
model.add(Dense(total_words, activation='softmax'))
model.compile(loss='categorical_crossentropy',
        optimizer='adam',
        metrics=['accuracy'])
model.summary()
model.fit(X, y, epochs=200, verbose=1)
def predict_next_word(model, tokenizer, text, max_seq_len):
  text = text.lower()
  sequence = tokenizer.texts_to_sequences([text])[0]
  sequence = pad_sequences([sequence],
                maxlen=max_seq_len-1,
                padding='pre')
```

```
    prediction = model.predict(sequence, verbose=0)
    predicted_index = np.argmax(prediction)
    for word, index in tokenizer.word_index.items():
        if index == predicted_index:
            return word
input_text = "the sun is"
predicted_word = predict_next_word(model, tokenizer, input_text, max_seq_len)
print(f"Input: '{input_text}'")
print(f"Predicted Next Word: '{predicted_word}'")
```

Output:



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).

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import SimpleRNN, Dense
data = {
  "Date": [
    "2023-01-02","2023-01-03","2023-01-04","2023-01-05","2023-01-06",
    "2023-01-09","2023-01-10","2023-01-11","2023-01-12","2023-01-13",
    "2023-01-16","2023-01-17","2023-01-18","2023-01-19","2023-01-20",
    "2023-01-23","2023-01-24","2023-01-25","2023-01-26","2023-01-27",
    "2023-01-30","2023-01-31","2023-02-01","2023-02-02","2023-02-03",
    "2023-02-06","2023-02-07","2023-02-08","2023-02-09","2023-02-10",
    "2023-02-13","2023-02-14","2023-02-15","2023-02-16","2023-02-17",
    "2023-02-20","2023-02-21","2023-02-22","2023-02-23","2023-02-24",
    "2023-02-27","2023-02-28","2023-03-01","2023-03-02","2023-03-03",
    "2023-03-06","2023-03-07","2023-03-08","2023-03-09","2023-03-10"
  ],"Close": [
    2685.5,2690.2,2688.75,2701.3,2710.4,
    2705.6,2718.9,2725.1,2719.8,2732.5,
    2740.2,2735.9,2748.6,2755.3,2762.1,
    2812.4,2820.9,2828.6,2835.2,2842.9,
    2838.5,2846.1,2852.8,2860.4,2868.9,
    2865.3,2872.7,2879.5,2886.2,2893.8,
    2890.1,2898.6,2905.2,2912.8,2920.4,
    2916.9,2925.5,2932.1,2938.7,2945.3
```

```
      2786.9,2795.4,2802.1,2810.3,2818.7,
      2812.4,2820.9,2828.6,2835.2,2842.9,
      2838.5,2846.1,2852.8,2860.4,2868.9,
      2865.3,2872.7,2879.5,2886.2,2893.8,
      2890.1,2898.6,2905.2,2912.8,2920.4,
      2916.9,2925.5,2932.1,2938.7,2945.3
   ]
}
dataset = pd.DataFrame(data)
prices = dataset[['Close']].values
scaler = MinMaxScaler(feature_range=(0, 1))
prices_scaled = scaler.fit_transform(prices)
X = []
y = []
for i in range(10, len(prices_scaled)):
   X.append(prices_scaled[i-10:i, 0])
   y.append(prices_scaled[i, 0])
X = np.array(X)
y = np.array(y)
X = np.reshape(X, (X.shape[0], X.shape[1], 1))
model = Sequential()
model.add(SimpleRNN(50, activation='tanh', input_shape=(10, 1)))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mean_squared_error')
model.summary()
model.fit(X, y, epochs=20, batch_size=8)
predicted_prices = model.predict(X)
predicted_prices = scaler.inverse_transform(predicted_prices)
real_prices = scaler.inverse_transform(y.reshape(-1, 1))
```

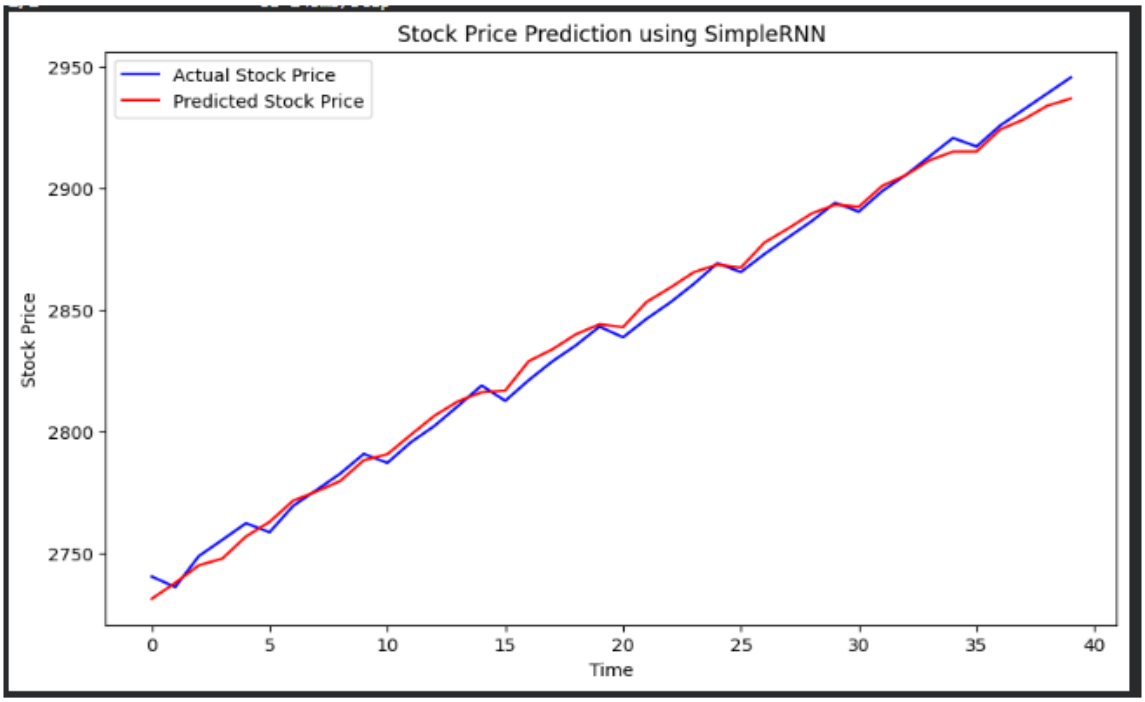
```
plt.figure(figsize=(10, 6))

plt.plot(real_prices, color='blue', label='Actual Stock Price')

plt.plot(predicted_prices, color='red', label='Predicted Stock Price')

plt.title('Stock Price Prediction using SimpleRNN')

plt.xlabel('Time')

plt.ylabel('Stock Price')

plt.legend()

plt.show()
```

Output:

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
simple_rnn (SimpleRNN)	(None, 50)	2,600
dense (Dense)	(None, 1)	51

```
Total params: 2,651 (10.36 KB)
Trainable params: 2,651 (10.36 KB)
Non-trainable params: 0 (0.00 B)
```



LAB Task 3:

Sentiment Analysis using RNN

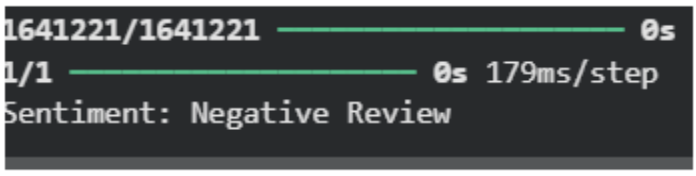
Objective: Classify movie reviews as positive or negative using RNN.
Dataset: *IMDb Movie Reviews* dataset (available in Keras).
Tasks:

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.

```
import numpy as np
from tensorflow.keras.datasets import imdb
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, SimpleRNN, Dense
from tensorflow.keras.preprocessing.sequence import pad_sequences
vocab_size = 10000
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=vocab_size)
max_len = 200
x_train = pad_sequences(x_train, maxlen=max_len)
x_test = pad_sequences(x_test, maxlen=max_len)
model = Sequential([
  Embedding(input_dim=vocab_size, output_dim=128, input_length=max_len),
  SimpleRNN(64, activation='tanh'),
  Dense(1, activation='sigmoid')])
model.compile(
  loss='binary_crossentropy',
  optimizer='adam',
  metrics=['accuracy'])
model.fit(
  x_train,
```

```
    y_train,
    epochs=5,
    batch_size=64,
    validation_split=0.2)
loss, accuracy = model.evaluate(x_test, y_test)
print(f"Test Accuracy: {accuracy * 100:.2f}%")
word_index = imdb.get_word_index()
def encode_review(text):
    encoded = []
    for word in text.lower().split():
        index = word_index.get(word)
        if index is not None and index < vocab_size:
            encoded.append(index + 3)
    return pad_sequences([encoded], maxlen=max_len)
custom_text = "This movie was amazing and full of emotions"
encoded_review = encode_review(custom_text)
prediction = model.predict(encoded_review)
if prediction[0][0] > 0.5:
    print("Sentiment: Positive Review")
else:
    print("Sentiment: Negative Review")
```

Output:



LAB Task 4:

Weather Forecasting using RNN

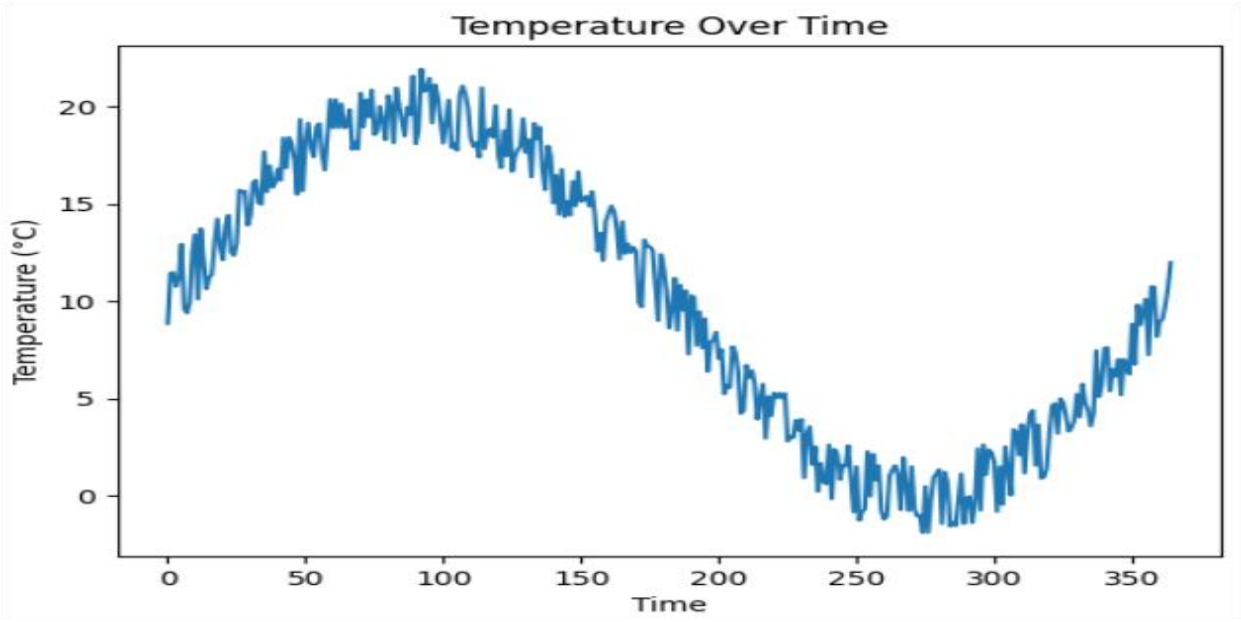
Objective: Predict future temperature based on previous days' readings.
Dataset: *Daily temperature dataset* (e.g., "Jena Climate Dataset" from TensorFlow).
Tasks:

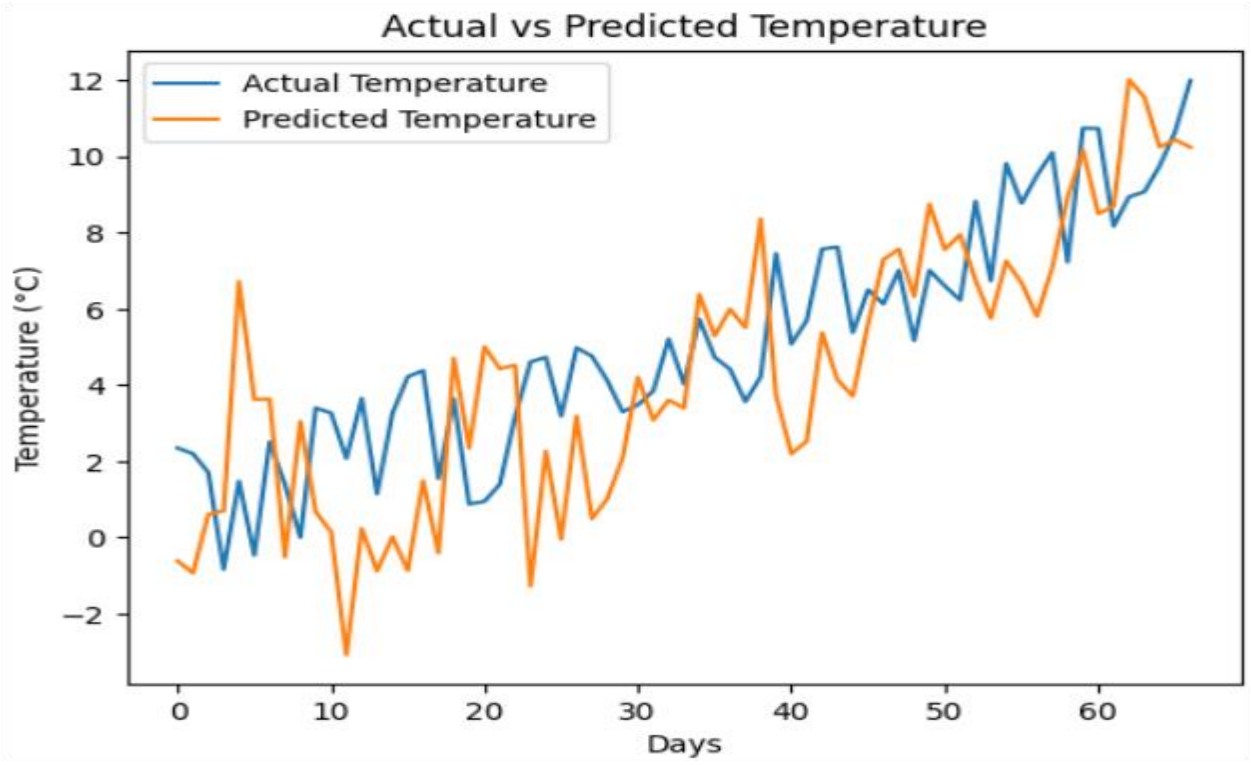
1. Load and visualize temperature over time.
2. Prepare input-output sequences for time series prediction.
3. Build an RNN to predict next day's temperature.
4. Plot actual vs predicted temperature.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import SimpleRNN, Dense
data = pd.read_csv("daily_temperature_dataset.csv")
temperature = data["Temperature (degC)"].values
plt.figure()
plt.plot(temperature[:2000])
plt.title("Temperature Over Time")
plt.xlabel("Time")
plt.ylabel("Temperature (°C)")
plt.show()
scaler = MinMaxScaler()
temperature_scaled = scaler.fit_transform(temperature.reshape(-1, 1))
def create_sequences(data, seq_length):
  X, y = [], []
```

```
 for i in range(len(data) - seq_length):
      X.append(data[i:i + seq_length])
      y.append(data[i + seq_length])
   return np.array(X), np.array(y)
sequence_length = 30
X, y = create_sequences(temperature_scaled, sequence_length)
split = int(0.8 * len(X))
X_train, X_test = X[:split], X[split:]
y_train, y_test = y[:split], y[split:]
model = Sequential([
   SimpleRNN(50, activation='tanh', input_shape=(sequence_length, 1)),
   Dense(1)])
model.compile(  optimizer='adam',  loss='mse')
model.fit(  X_train,    y_train,
 epochs=10,
   batch_size=32,
   validation_split=0.2)
predicted = model.predict(X_test)
predicted_temp = scaler.inverse_transform(predicted)
actual_temp = scaler.inverse_transform(y_test)
plt.figure()
plt.plot(actual_temp, label="Actual Temperature")
plt.plot(predicted_temp, label="Predicted Temperature")
plt.title("Actual vs Predicted Temperature")
plt.xlabel("Days")
plt.ylabel("Temperature (°C)")
plt.legend()
plt.show()
```

Output:


Temperature Over Time
Temperature (°C)
Time
0
5
10
15
20
0
50
100
150
200
250
300
350


Actual vs Predicted Temperature
Actual Temperature
Predicted Temperature
Temperature (°C)
Days
−2
0
2
4
6
8
10
12
0
10
20
30
40
50
60

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).

```
import numpy as np
from music21 import note, chord, stream
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import SimpleRNN, Dense
notes = [
  "C4","D4","E4","F4","G4","A4","B4","C5",
  "C5","B4","A4","G4","F4","E4","D4","C4",
  "C4.E4.G4","D4.F4.A4","E4.G4.B4","F4.A4.C5"
]
unique_notes = sorted(set(notes))
note_to_int = {n:i for i,n in enumerate(unique_notes)}
int_to_note = {i:n for i,n in enumerate(unique_notes)}
sequence_length = 4
X, y = [], []
for i in range(len(notes)-sequence_length):
  seq_in = notes[i:i+sequence_length]
  seq_out = notes[i+sequence_length]
  X.append([note_to_int[n] for n in seq_in])
  y.append(note_to_int[seq_out])
```

```
X = np.reshape(X, (len(X), sequence_length, 1))
X = X / float(len(unique_notes))
y = np.array(y)
model = Sequential()
model.add(SimpleRNN(128, input_shape=(X.shape[1], X.shape[2])))
model.add(Dense(len(unique_notes), activation='softmax'))
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam')
model.fit(X, y, epochs=100, batch_size=1, verbose=0)
start = np.random.randint(0, len(X)-1)
pattern = X[start]
generated_notes = []
for _ in range(10):
  prediction = model.predict(pattern.reshape(1, sequence_length, 1), verbose=0)
  index = np.argmax(prediction)
  result = int_to_note[index]
  generated_notes.append(result)
  next_input = index / float(len(unique_notes))
  pattern = np.append(pattern, [[next_input]], axis=0)
  pattern = pattern[1:]
print("Generated Notes:", generated_notes)
output_notes = []
offset = 0
for pattern in generated_notes:
if '.' in pattern:
    chord_notes = pattern.split('.')
    chord_objects = [note.Note(n) for n in chord_notes]
new_chord = chord.Chord(chord_objects)
    new_chord.offset = offset
```

```
 output_notes.append(new_chord)
   else:
      new_note = note.Note(pattern)
      new_note.offset = offset
      output_notes.append(new_note)
   offset += 0.5
midi_stream = stream.Stream(output_notes)
midi_stream.write('midi', fp='generated_music_demo.mid')
```

Output:

```
Generated Notes: ['F4', 'E4', 'D4', 'C4', 'C4.E4.G4', 'D4.F4.A4', 'E4.G4.B4', 'F4.A4.C5', 'G4', 'A4']
 MIDI file generated: generated_music_demo.mid
```