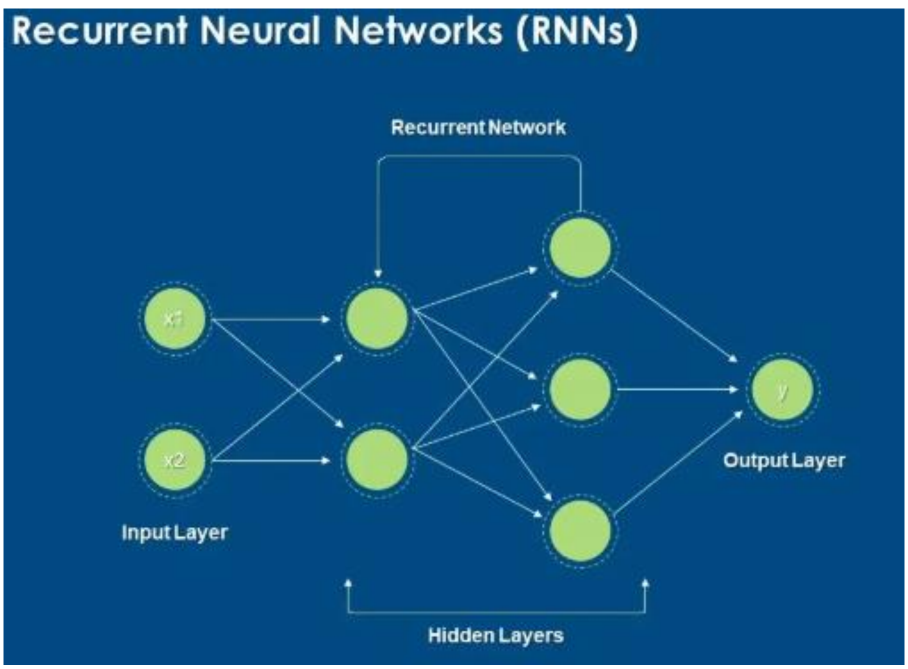
LAB No. 7

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 = 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 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

"""
```

```
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_sequence
s[:, :-1] y =
input_sequence
s[:, -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]
```

```
    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:

```
Input: 'the sun is'
Predicted Next Word: 'very'
```

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,
    812.4,2820.9,2828.6,2835.2,28
```

```
    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']].valu
es
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_sc
aled[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()
= scaler.inverse_transform(y.reshape(-1, 1))
```

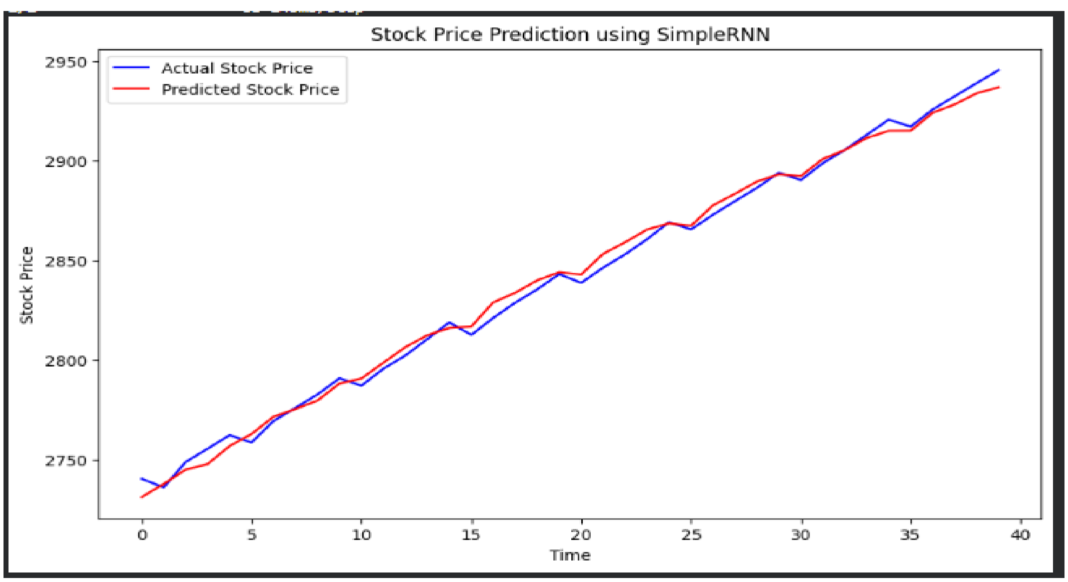
```
plt.figure(figsize=(10,  6))

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.

Code:

```
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,
```

```
  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:
```

Output:

```
1641221/1641221 ━━━━━━━━━━━━━━━━━━━━ 0s
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 179ms/step
Sentiment: Negative Review
```

LAB Task 4:

Weather Forecasting using RNN

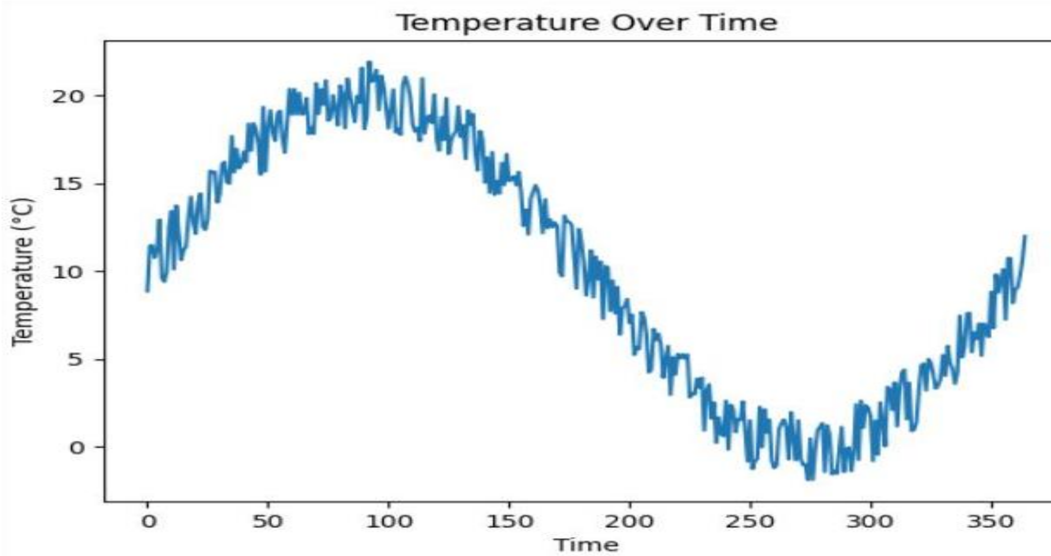
Objective: Predict future temperature based on previous days' readings.
Dataset: *Daily temperature dataset* (e.g., "Jena Climate Dataset" from TensorFlow).
Tasks:

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.

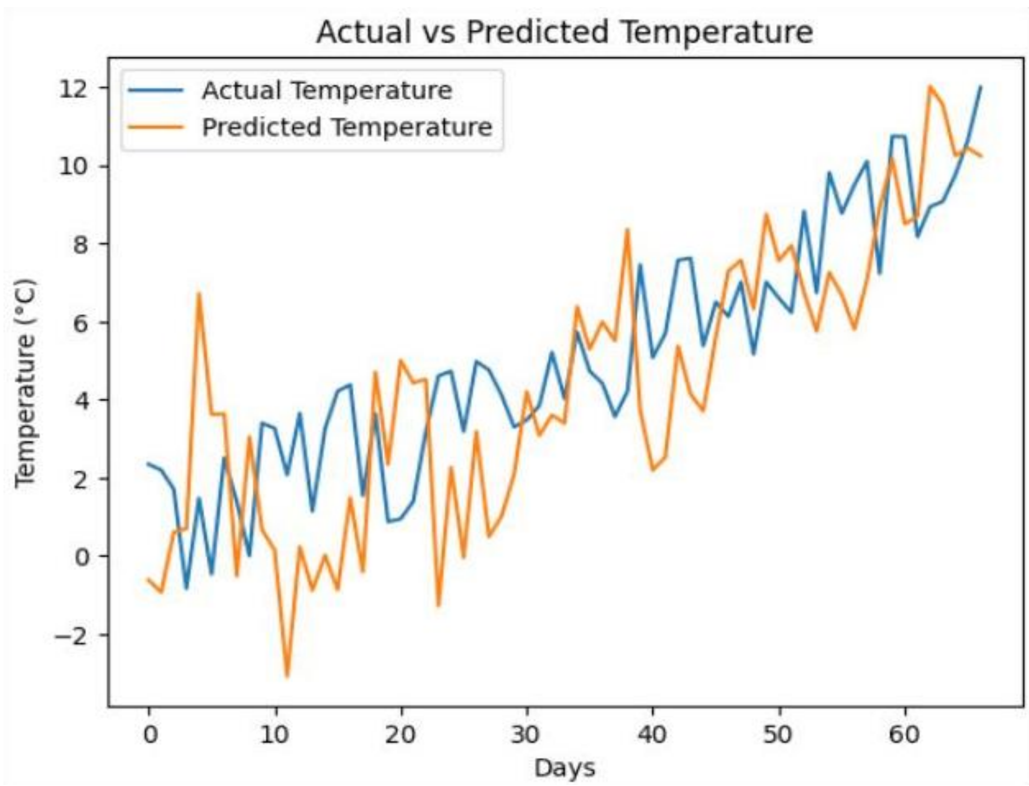
```
import numpy as
np import pandas
as pd
import matplotlib.pyplot as plt

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("Temperat

ure (°C)") plt.legend()
```

Output:



Temperature Over Time
Temperature (°C)
Time



Actual vs Predicted Temperature
Actual Temperature
Predicted Temperature
Temperature (°C)
Days

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_lengt
  h] 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]
generate
d_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
```

LAB Assessment

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

Laboratory Work Assessment Rubrics

Sr. No.	Performance Indicator	Excellent (5)	Good (4)	Average (3)	Fair (2)	Poor (1)
1	**Theoretical knowledge 10%**	Student knows all the related concepts about the theoretical background of the experiment and rephrase those concepts in written and oral assessments	Student knows most of the related concepts about the theoretical background of the experiment and partially rephrase those concepts in written and oral assessments	Student knows few of the related concepts about the theoretical background of the experiment and partially rephrase those concepts in written and oral assessments	Student knows very little about the related concepts about the theoretical background of the experiment and poorly rephrase those concepts in written and oral assessments	Student has poor understanding of the related concepts about the theoretical background of the experiment and unable to rephrase those concepts in written and oral assessments
2	**Application Functionality 10%**	Application runs smoothly and operation of the application runs efficiently	Application compiles with no warnings. Robust operation of the application, with good recovery.	Application compiles with few or no warnings. Consideration given to unusual conditions with reasonable	Application compiles and runs without crashing. Some attempt at detecting and correcting errors.	Application does not compile or compiles but crashes. Confusing. Little or no error detection or correction.
3	**Specifications 10%**	The program works very efficiently and meets all of the required specifications.	The program works and meets some of the specifications.	The program works and produces the correct results and displays them correctly. It also meets most of the other specifications.	The program produces correct results but does not display them correctly.	The program is producing incorrect results.
4	**Level of understanding of the learned skill 10%**	Provide complete and logical answers based upon accurate technical content to the questions asked by examiner	Provide complete and logical answers based upon accurate technical content to the questions asked by examiner with few errors	Provide partially correct and logical answers based upon minimum technical content to the questions asked by examiner	Provide very few and illogical answers to the questions asked by examiner.	Provide no answer to the questions asked by examiner.
5	**Readability and Reusability 10%**	The code is exceptionally well organized and very easy to follow and reused	The code is fairly easy to read. The code could be reused as a whole or each class could be reused.	Most of the code could be reused in other programs.	Some parts of the code require change before they could be reused in other programs.	The code is poorly organized and very difficult to read and not organized for reusability.

6	**AI System Design 10%**	Well-designed AI models. Code is highly maintainable	Good designed AI models and Little code duplications	Some attempt to make AI models. Code can be maintained with significant effort	Little attempt to design AI models and less understanding of code	Very poor attempt to design AI models and its code
7	**Responsiveness to Questions/ Accuracy 10%**	1. Responds well, quick and very accurate all the time. 2. Effectively uses eye contact, speaks clearly, effectively and confidently using suitable volume	1. Generally Responsive and accurate most of the times. 2. Maintains eye contact, speaks clearly with suitable volume and pace.	1. Generally Responsive and accurate few times. 2. Some eye contact, speaks clearly and unclearly in different portions.	1. Not much Responsive and accurate most of the times. 2. Uses eye contact ineffectively and fails to speak clearly and audibly	. 1. Non Responsive and inaccurate all the times. 2. No eye contact and unable to speak 3. Dresses inappropriately
8	**Efficiency 10%**	The code is extremely efficient without sacrificing readability and understanding	The code is fairly efficient without sacrificing readability and understanding	Some part of the code is efficient and other part of the code is not understandable and work properly	The code is brute force and unnecessarily long	The code is huge and appears to be patched together
9	**Delivery** **10%**	The program was delivered in time during lab.	The program was delivered in Lab before the end time.	The program was delivered within the due date.	The code was delivered within a day after the due date.	The code was delivered more than 2 days overdue.
10	**Awareness of Safety Guidelines 10%**	Student has sufficient knowledge of the laboratory safety SOPs and protocol and is fully compliant to the guidelines	Student has sufficient knowledge of the laboratory safety SOPs and protocol and is Partially compliant to the guidelines	Student has little knowledge of the laboratory safety SOPs and protocol and is Partially compliant to the guidelines	Student has little knowledge of the laboratory safety SOPs and protocol and is non-compliant to the guidelines	Student has no knowledge of the laboratory safety SOPs and protocol and is non-compliant to the guidelines