**LAB FILE**

**REINFORCEMENT LEARNING AND DEEP LEARNING LAB**

**(ML‐409P)**

Student Name: Lakshay Sharma

Roll No: 02396402721

Semester: 7th

Group: 7-CSE-AIML-II-C

Faculty name: Mr. Ajay Tiwari



Department Of Computer Science & Engineering

Maharaja Agrasen Institute of Technology, PSP area, Sector – 22, Rohini,

New Delhi – 110085

(Affiliated to Guru Gobind Singh Indraprastha University, New Delhi)

**2024**



## MAHARAJA AGRASEN INSTITUTE OF TECHNOLOGY

**VISION**

To nurture young minds in a learning environment of high academic value and imbibe spiritual and ethical values with technological and management competence.

## MISSION

The Institute shall endeavor to incorporate the following basic missions in the teaching methodology:

**Engineering Hardware – Software Symbiosis**

Practical exercises in all Engineering and Management disciplines shall be carried out by Hardware equipment as well as the related software enabling deeper understanding of basic concepts and

encouraging inquisitive nature.

**Life – Long Learning**

The Institute strives to match technological advancements and encourage students to keep updating their knowledge for enhancing their skills and inculcating their habit of continuous learning.

**Liberalization and Globalization**

The Institute endeavors to enhance technical and management skills of students so that they are intellectually capable and competent professionals with Industrial Aptitude to face the challenges of globalization.

**Diversification**

The Engineering, Technology and Management disciplines have diverse fields of studies with different attributes. The aim is to create a synergy of the above attributes by encouraging analytical thinking.

**Digitization of Learning Processes**

The Institute provides seamless opportunities for innovative learning in all Engineering and Management disciplines through digitization of learning processes using analysis, synthesis, simulation, graphics,

tutorials and related tools to create a platform for multi-disciplinary approach.

**Entrepreneurship**

The Institute strives to develop potential Engineers and Managers by enhancing their skills and research capabilities so that they become successful entrepreneurs and responsible citizens.



## MAHARAJA AGRASEN INSTITUTE OF TECHNOLOGY

**COMPUTER SCIENCE & ENGINEERING DEPARTMENT**

# VISION

“To be centre of excellence in education, research and technology transfer in the field of computer engineering and promote entrepreneurship and ethical values.”

# MISSION

“To foster an open, multidisciplinary and highly collaborative research environment to produce world-class engineers capable of providing innovative solutions to real life problems and fulfil societal needs.”

Department of Computer Science and Engineering Rubrics for Lab Assessment

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Rubrics** | | **0** | **1** | **2** | **3** |
| **Missing** | **Inadequate** | **Needs Improvement** | **Adequate** |
| R1 | Is able to identify the problem to be solved and define the objectives of the experiment. | No mention is made of the problem to be solved. | An attempt is made to identify the problem to be solved but it is described in a confusing manner, objectives are not relevant, objectives contain technical/ conceptual errors or objectives are not measurable. | The problem to be solved is described but there are minor omissions or vague details. Objectives are conceptually correct and measurable but may be incomplete in scope or have linguistic errors. | The problem to be solved is clearly stated. Objectives are complete, specific, concise, and measurable. They are written using correct technical terminology and are free from linguistic errors. |
| R2 | Is able to design a reliable experiment that solves the problem. | The experiment does not solve the problem. | The experiment attempts to solve the problem but due to the nature of the design the data will not lead to a reliable solution. | The experiment attempts to solve the problem but due to the nature of the design there is a moderate chance the data will not lead to a reliable  solution. | The experiment solves the problem and has a high likelihood of producing data that will lead to a reliable solution. |
| R3 | Is able to communicate the details of an experimental procedure clearly and completely. | Diagrams are missing and/or experimental procedure is missing or extremely vague. | Diagrams are present but unclear and/or experimental procedure is present but important details are missing. | Diagrams and/or experimental procedure are present but with minor omissions or vague details. | Diagrams and/or experimental procedure are clear and complete. |
| R4 | Is able to record and represent data in a meaningful way. | Data are either absent or incomprehensible. | Some important data are absent or incomprehensible. | All important data are present, but recorded in a way that requires some effort to comprehend. | All important data are present, organized and recorded clearly. |
| R5 | Is able to make a judgment about the results of the experiment. | No discussion is presented about the results of the experiment . | A judgment is made about the results, but it is not reasonable or coherent. | An acceptable judgment is made about the result, but the reasoning is flawed or incomplete. | An acceptable judgment is made about the result, with clear reasoning. The effects of assumptions and experimental uncertainties are considered. |

# PRACTICAL RECORD

**PAPER CODE : ML‐409P**

Name of Student : Lakshay Sharma

University roll : 02396402721

Semester : 7th (4th year)

Group : 7-CSE-AIML-II-C

## PRACTICAL DETAILS

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Exp. no** | **Experiment Name** | **Date of performance** | **Date of checking** | **R1 (3)** | **R2 (3)** | **R3 (3)** | **R4 (3)** | **R5 (3)** | **Total Marks (15)** | **Signature** |
| 1. |  |  |  |  |  |  |  |  |  |  |
| 2. |  |  |  |  |  |  |  |  |  |  |
| 3. |  |  |  |  |  |  |  |  |  |  |
| 4. |  |  |  |  |  |  |  |  |  |  |
| 5. |  |  |  |  |  |  |  |  |  |  |
| 6. |  |  |  |  |  |  |  |  |  |  |

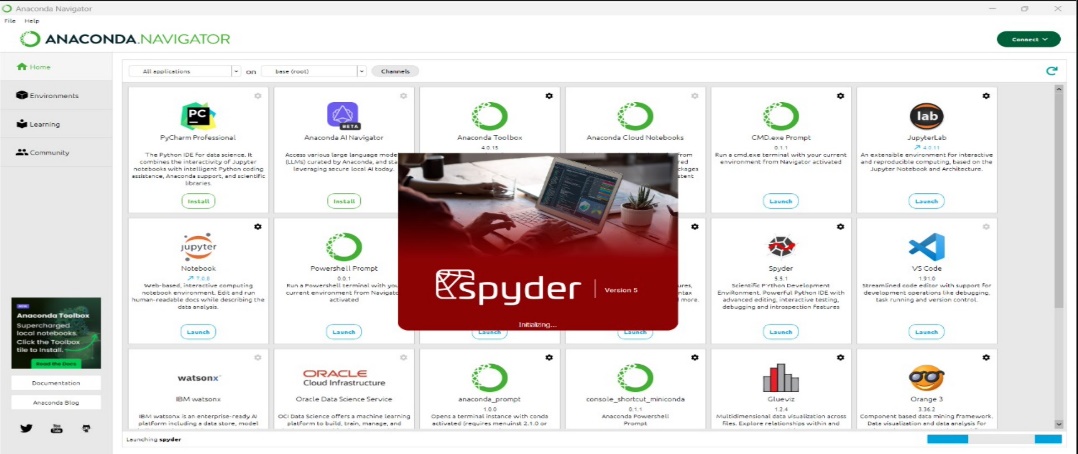
|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Exp. no** | **Experiment Name** | **Date of Performance** | **Date of checking** | **R1 (3)** | **R2 (3)** | **R3 (3)** | **R4 (3)** | **R5 (3)** | **Total Marks (15)** | **Signature** |
| 7. |  |  |  |  |  |  |  |  |  |  |
| 8. |  |  |  |  |  |  |  |  |  |  |
| 9. |  |  |  |  |  |  |  |  |  |  |
| 10. |  |  |  |  |  |  |  |  |  |  |
| 11. |  |  |  |  |  |  |  |  |  |  |
| 12. |  |  |  |  |  |  |  |  |  |  |
| 13. |  |  |  |  |  |  |  |  |  |  |
| 14. |  |  |  |  |  |  |  |  |  |  |
| 15. |  |  |  |  |  |  |  |  |  |  |

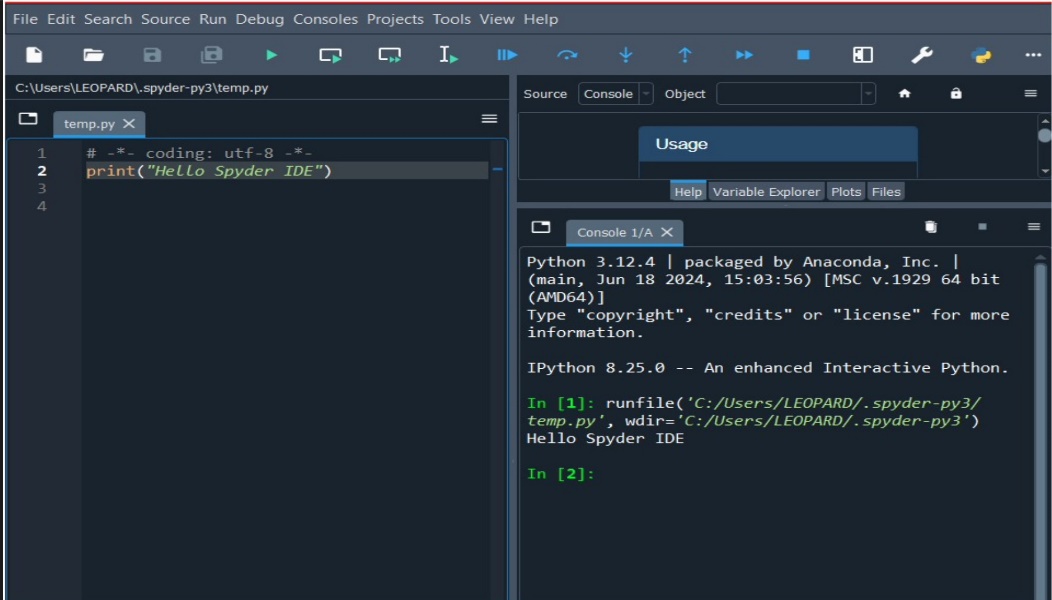
# EXPERIMENT - 1

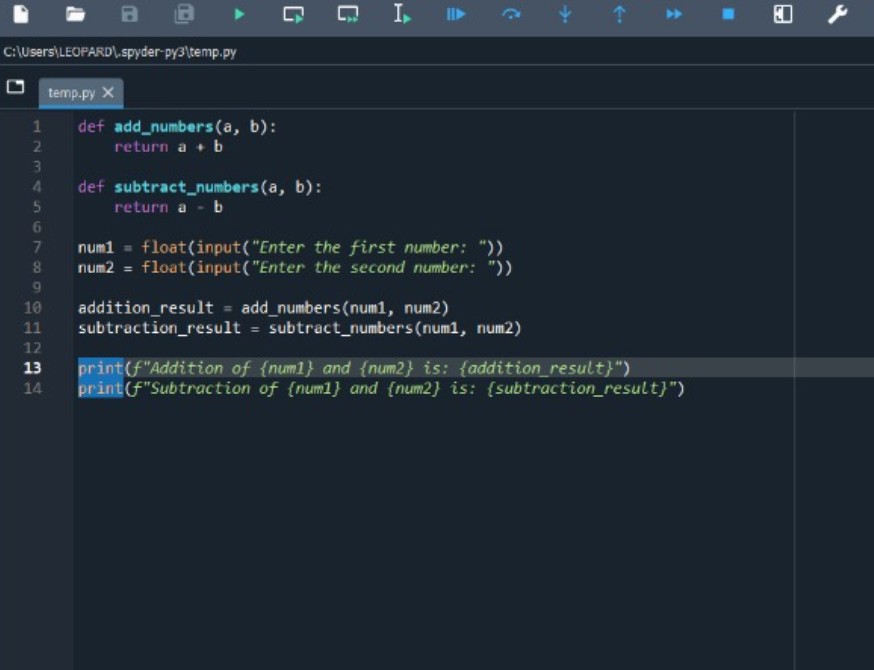
### AIM: To set up the Spyder IDE for executing Python programs and running a basic Python script.

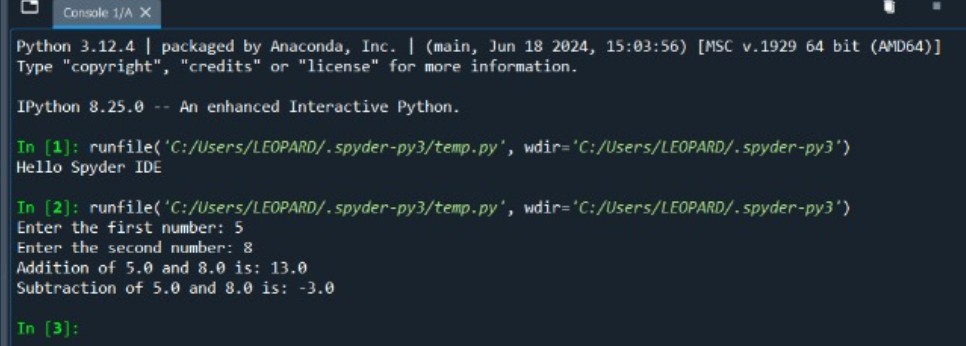
### Theory: Spyder is an open-source integrated development environment (IDE) designed for Python, often used in data science. It allows users to write, debug, and execute Python code. The environment supports powerful libraries and provides a user-friendly interface for beginners and experts alike.

**Output:**







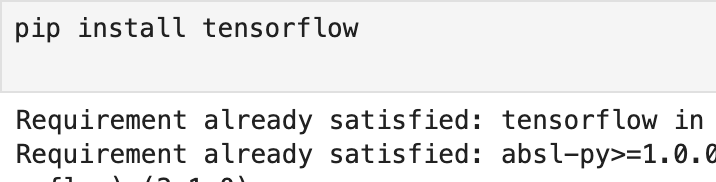


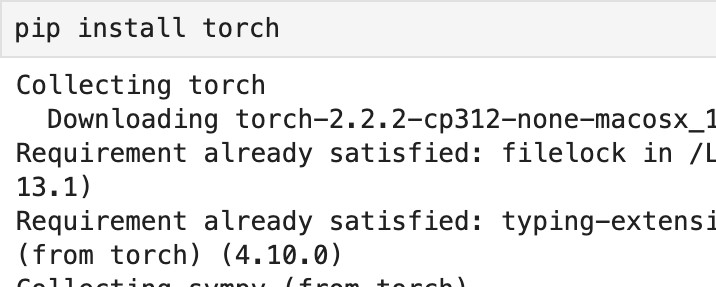
# EXPERIMENT - 2

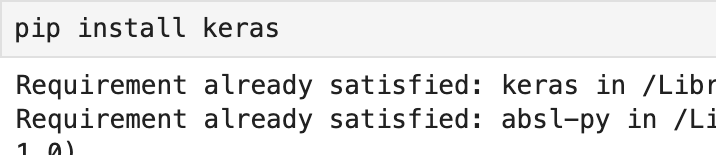
### AIM: To install Keras, TensorFlow, and PyTorch libraries and demonstrate their use in machine learning projects.

### Theory: Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow. TensorFlow is an open-source platform for machine learning, while PyTorch is an open-source machine learning library based on Torch, primarily used for applications such as natural language processing and deep learning.

**Source Code:**



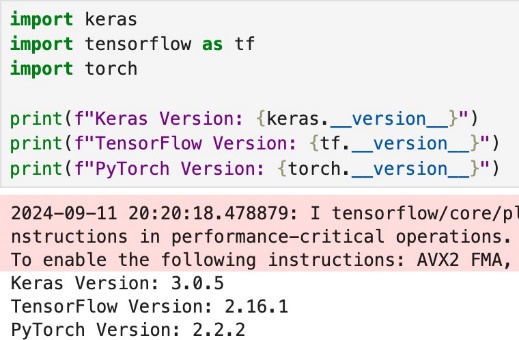




**Implementation Code:**



**Output:**





# EXPERIMENT - 3

### AIM: To implement the Q-learning algorithm in pure Python and train the agent to play a game using the OpenAI Gym environment

### Theory: Q-learning is a model-free reinforcement learning algorithm that seeks to find the best action to take given the current state. It updates a Q-table that stores the cumulative expected reward for each action in each state. OpenAI Gym provides a collection of environments where agents can be trained and tested.

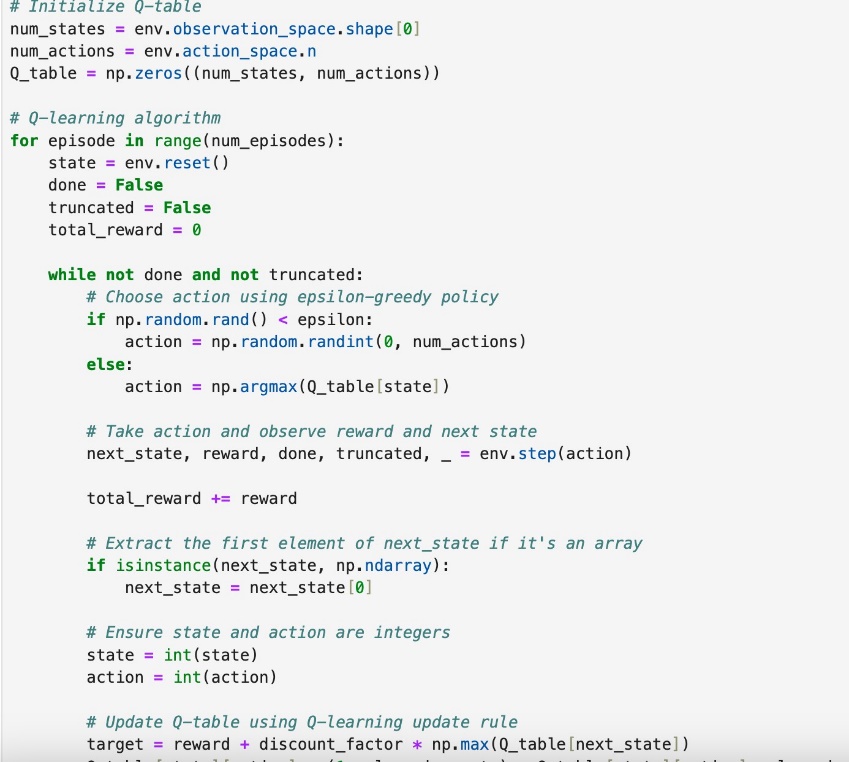
**Source Code:**

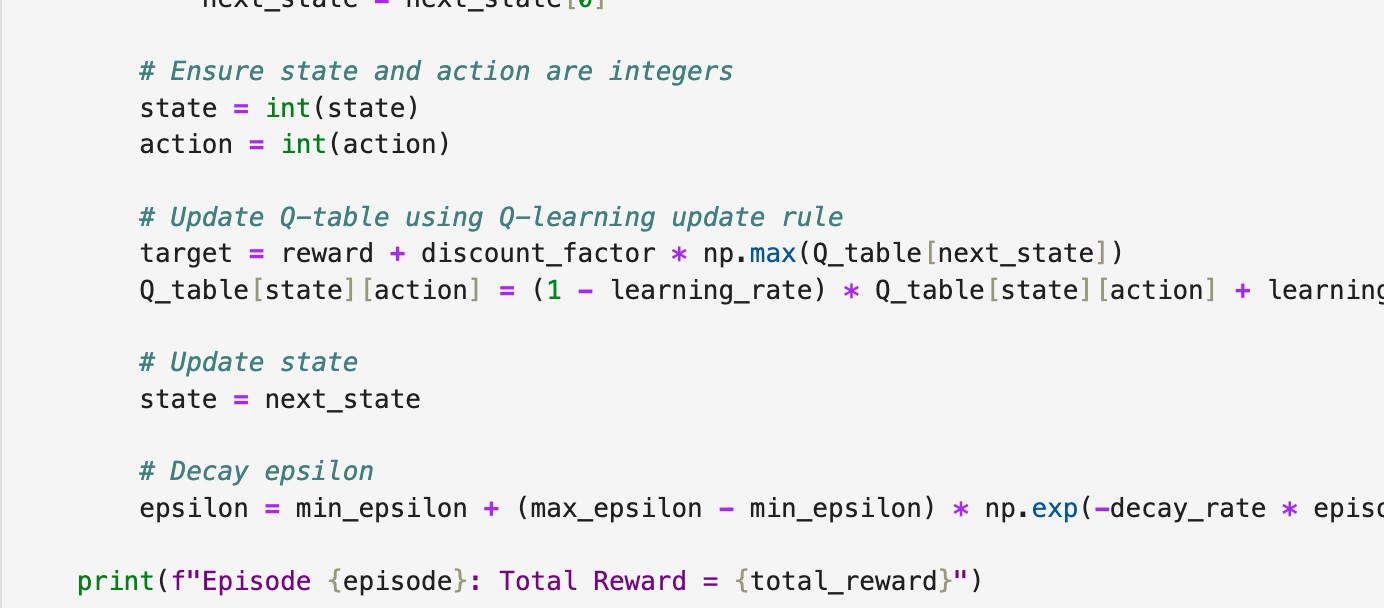
From the code and context, the game being played is FrozenLake-v1 from OpenAI Gym, specifically with is\_slippery=False. This environment simulates an agent trying to navigate across a grid of frozen ground and holes (pits) to reach a goal.

**Environment Setup and Introduction to OpenAI Gym:**

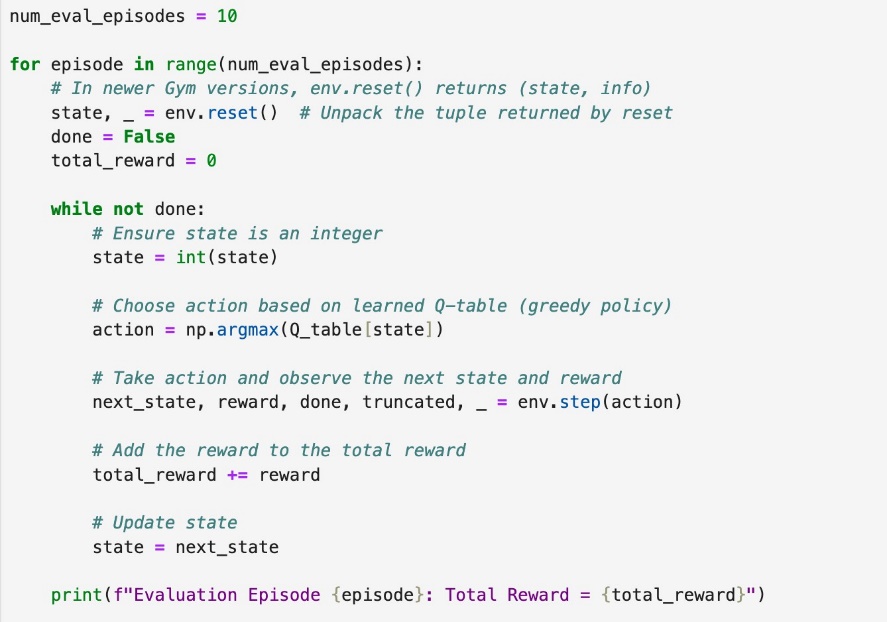


**Write Q learning algo and train the Agent to play the game**

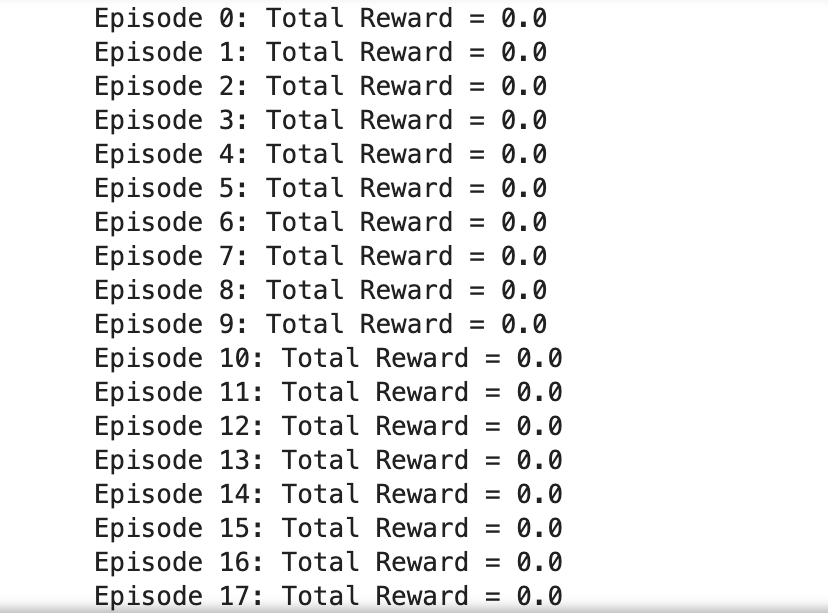


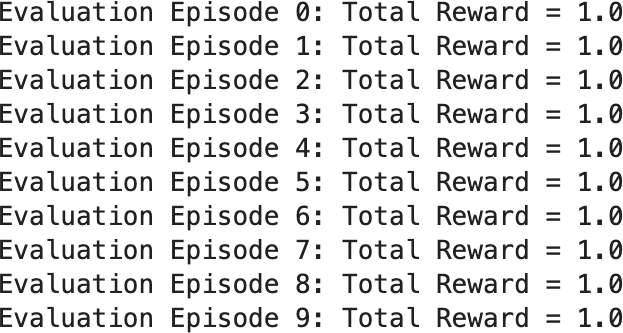


**Watch Trained Agent play the game**



**Output:**



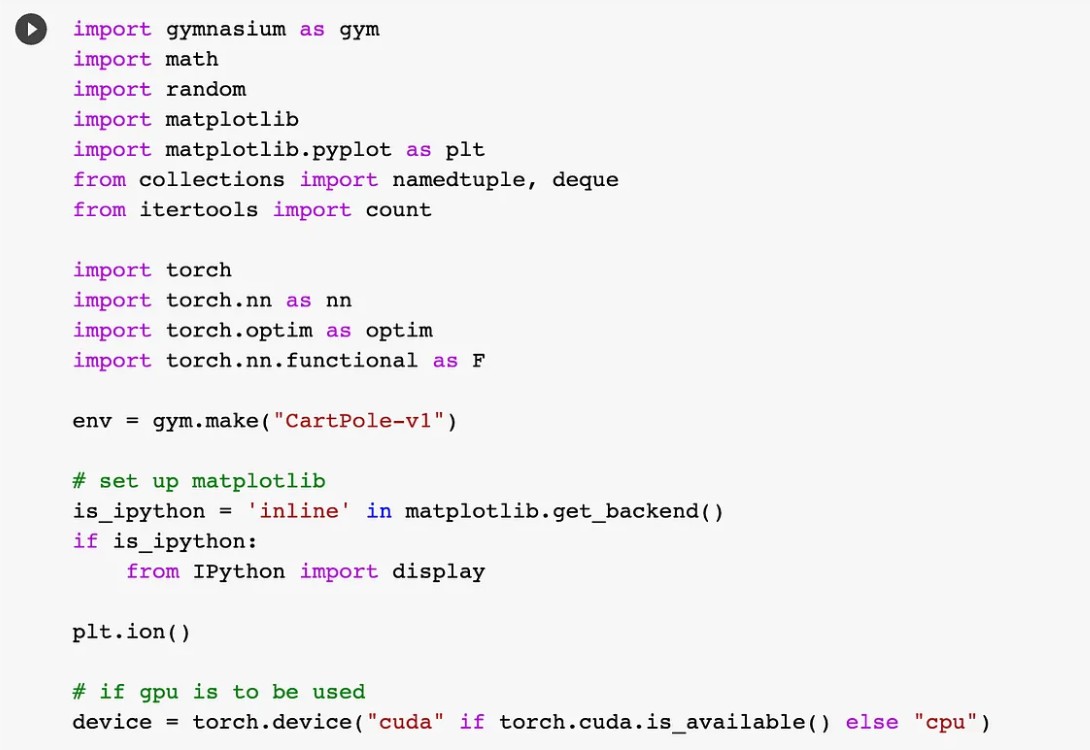
**WATCH THE MODEL PLAY:**

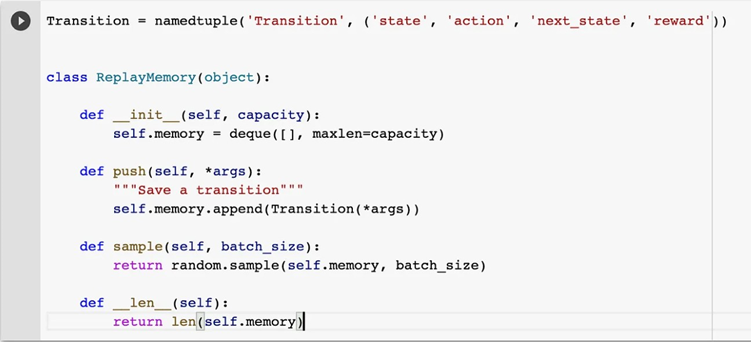
# EXPERIMENT - 4

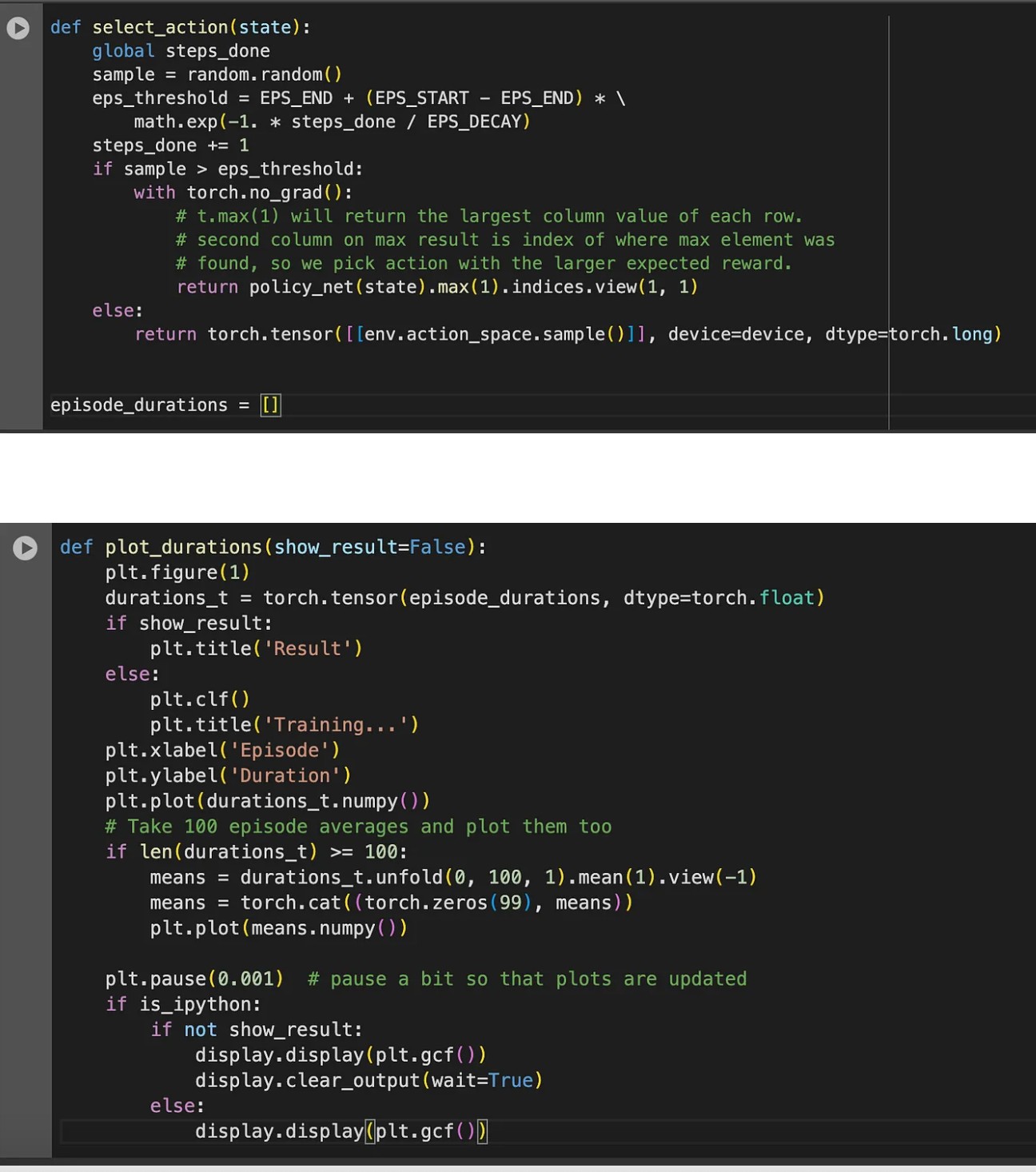
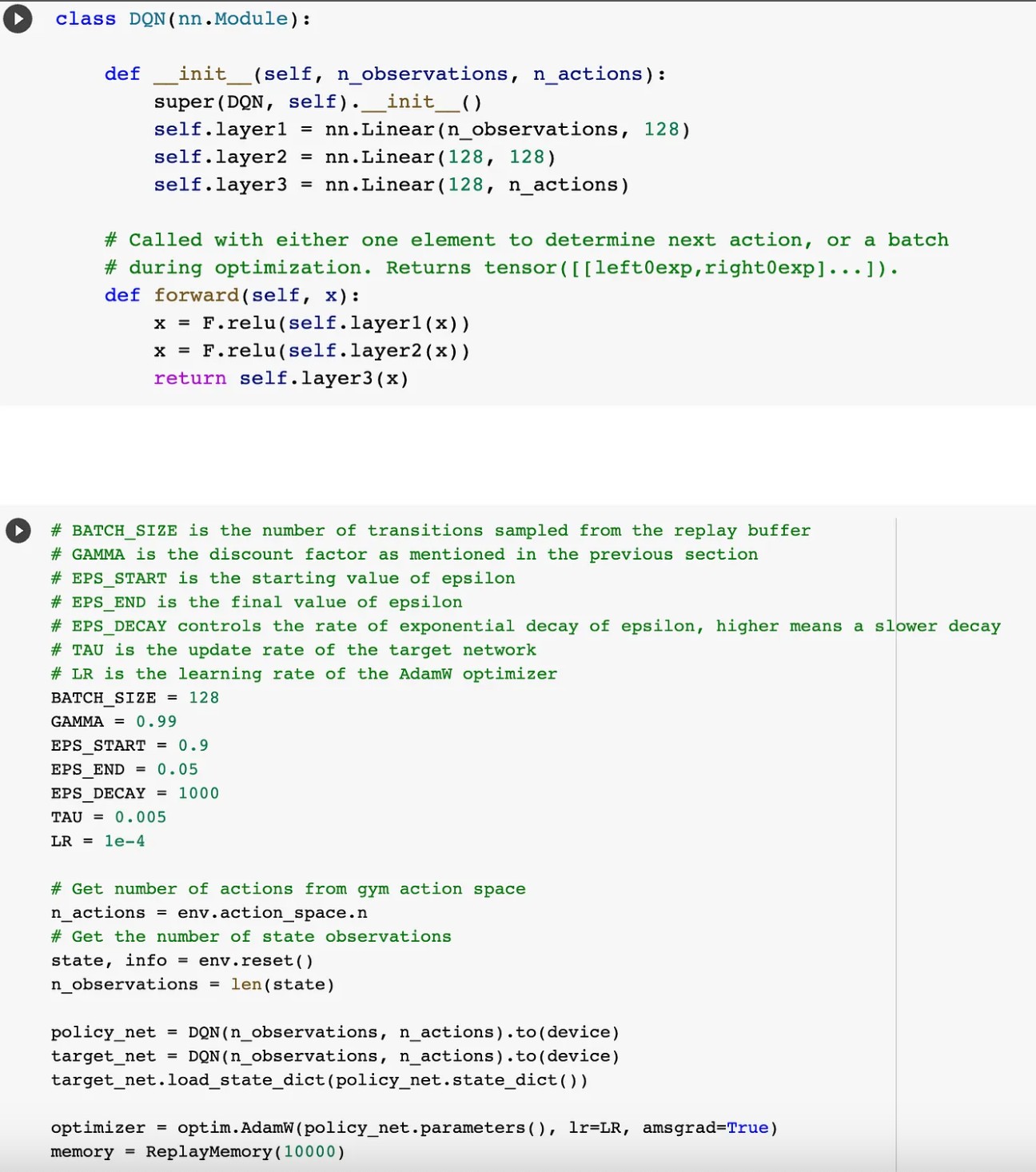
### AIM: To implement a Deep Q-Network (DQN) using PyTorch for reinforcement learning tasks.

### Theory: Deep Q-Network is a reinforcement learning algorithm that uses a deep neural network to approximate the optimal action-value function. It is commonly used in environments where an agent interacts with a dynamic environment to maximize cumulative rewards.

**Source Code:**



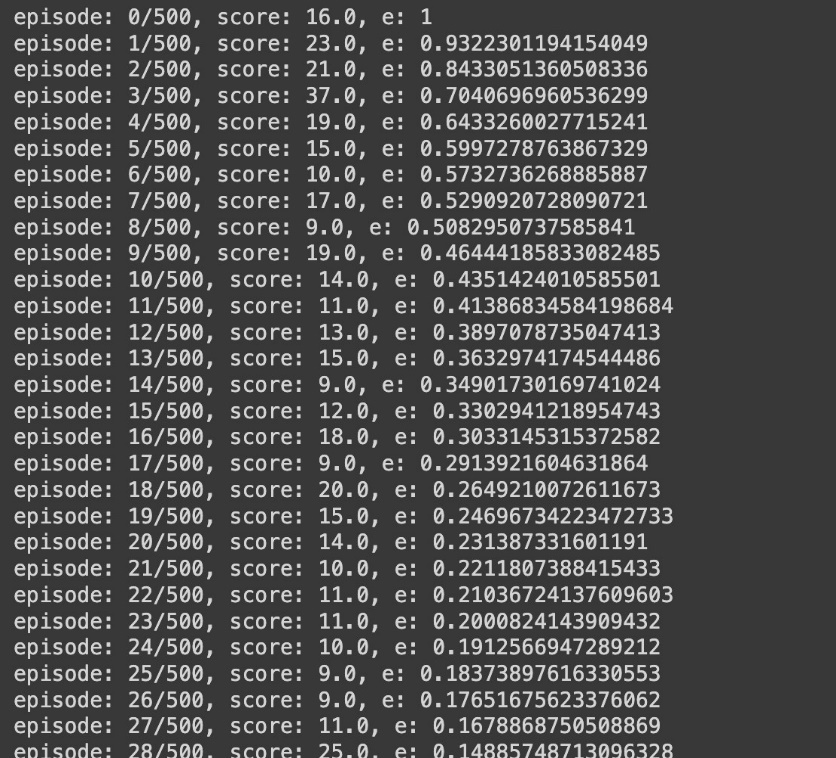
****

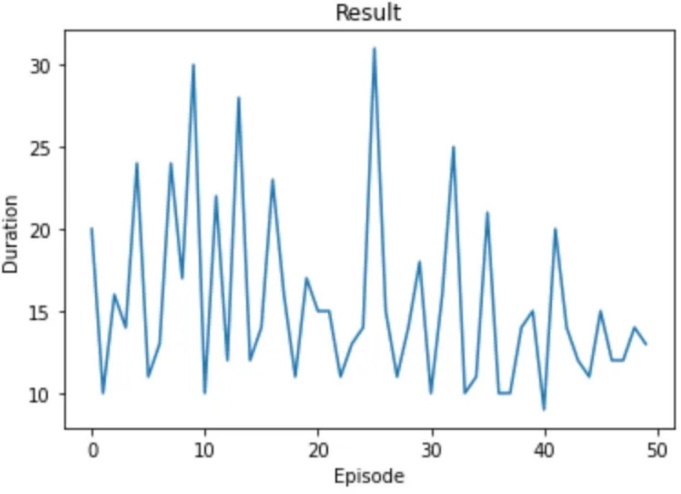






**Output:**



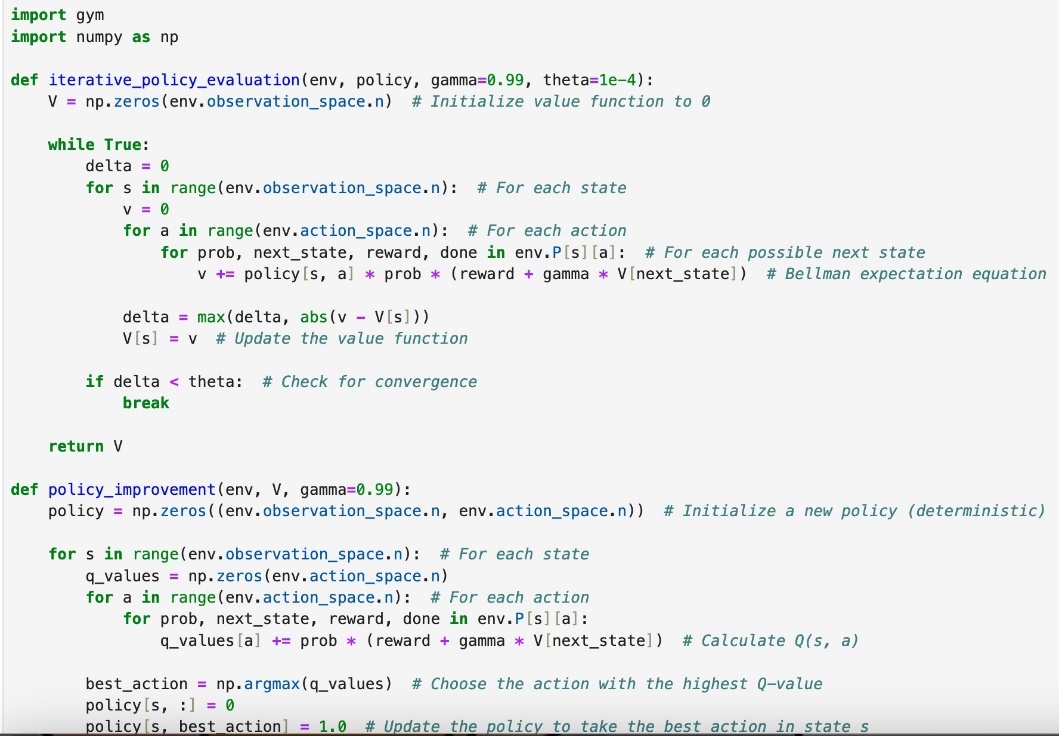


# EXPERIMENT - 5

### AIM: To implement iterative policy evaluation and update using Python.

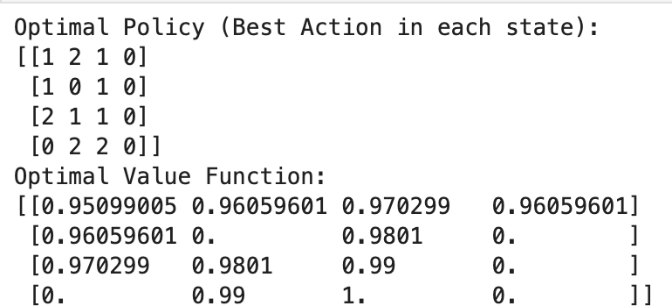
### Theory: Iterative policy evaluation is a technique in reinforcement learning where the value function is updated iteratively for a given policy, allowing for the improvement and optimization of the policy based on value function approximations.

**Source Code:**





**Output:**



# EXPERIMENT - 6

### AIM: Chatbot using bi-directional LSTMs

### Theory: LSTM is a type of Recurrent Neural Network (RNN) that is used to process sequential data. Bi-directional LSTMs can capture dependencies in both forward and backward directions, making them useful in applications like chatbots where the meaning of a sentence can depend on the entire context.

**Source Code:**

**Output:**

import numpy as np

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Embedding, LSTM, Dense, Bidirectional

# Dummy data (replace with real chatbot data)

input\_data = np.random.randint(1000, size=(100, 10))

output\_data = np.random.randint(2, size=(100, 1))

# Model

model = Sequential() model.add(Embedding(input\_dim=1000, output\_dim=64)) model.add(Bidirectional(LSTM(64)))

model.add(Dense(1, activation='sigmoid'))

# Compile

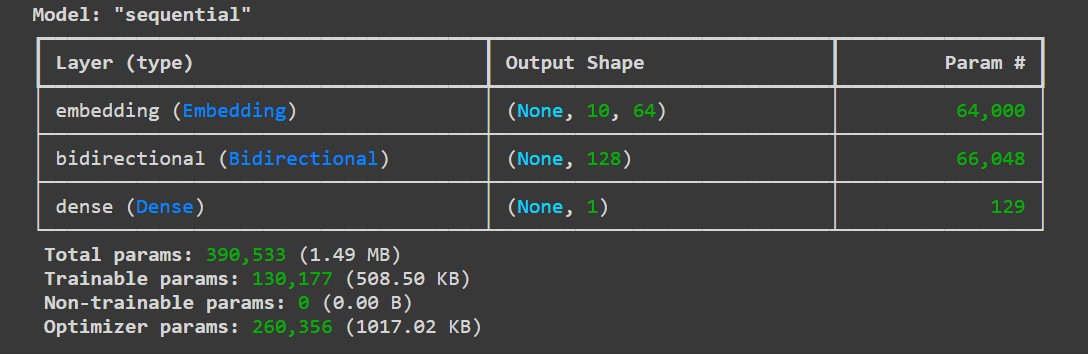
model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

# Train

model.fit(input\_data, output\_data, epochs=10, batch\_size=32)

# Output (Model Summary)

model.summary()



# EXPERIMENT - 7

### AIM: Image classification on MNIST dataset (CNN model with fully connected layer).

### Theory: CNNs are deep learning models typically used for image processing tasks. They use convolutional layers to capture spatial patterns in images. The fully connected layer at the end maps the extracted features to the output labels (digits 0-9 in MNIST).

**Source Code:**

import tensorflow as tf

from tensorflow.keras.datasets

import mnist

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers

import Conv2D, MaxPooling2D, Flatten, Dense

# Load MNIST dataset

(x\_train, y\_train), (x\_test, y\_test) = mnist.load\_data()

# Preprocess data

x\_train = x\_train.reshape(-1, 28, 28, 1) / 255.0

x\_test = x\_test.reshape(-1, 28, 28, 1) / 255.0

# Build CNN model model = Sequential()

model.add(Conv2D(32, (3, 3), activation='relu', input\_shape=(28, 28, 1)))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Flatten())

model.add(Dense(128, activation='relu'))

model.add(Dense(10, activation='softmax'))

# Compile and train the model

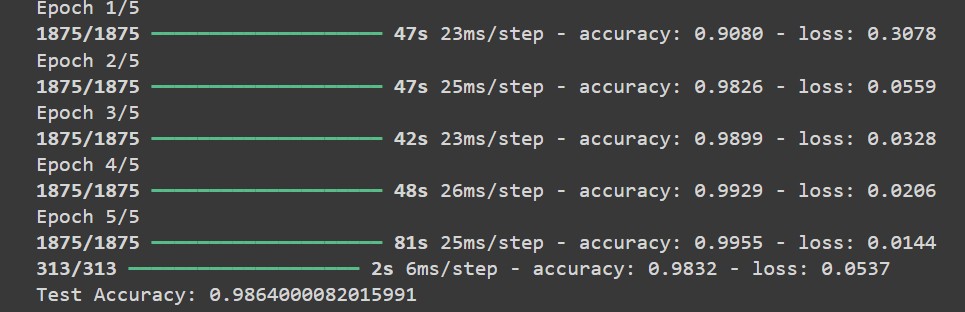
model.compile(optimizer='adam', loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])

model.fit(x\_train, y\_train, epochs=5, batch\_size=32)

# Evaluate

loss, accuracy = model.evaluate(x\_test, y\_test) print(f"Test Accuracy: {accuracy}")

**Output:**



# EXPERIMENT - 8

### AIM: Train a sentiment analysis model on IMDB dataset, use RNN layers with LSTM/GRU

### Theory: Sentiment analysis involves classifying text data into positive or negative sentiment. LSTM and GRU are variants of RNNs that can capture long-term dependencies in text sequences, making them suitable for this task.

**Source Code:**

import tensorflow as tf

from tensorflow.keras.datasets

import imdb

from tensorflow.keras.preprocessing import sequence

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Embedding, LSTM, Dense

# Load IMDB dataset

(x\_train, y\_train), (x\_test, y\_test) = imdb.load\_data(num\_words=10000)

# Preprocess data

x\_train = sequence.pad\_sequences(x\_train, maxlen=500)

x\_test = sequence.pad\_sequences(x\_test, maxlen=500)

# Build LSTM model model = Sequential()

model.add(Embedding(input\_dim=10000, output\_dim=64, input\_length=500))

model.add(LSTM(64))

model.add(Dense(1, activation='sigmoid'))

# Compile and train the model

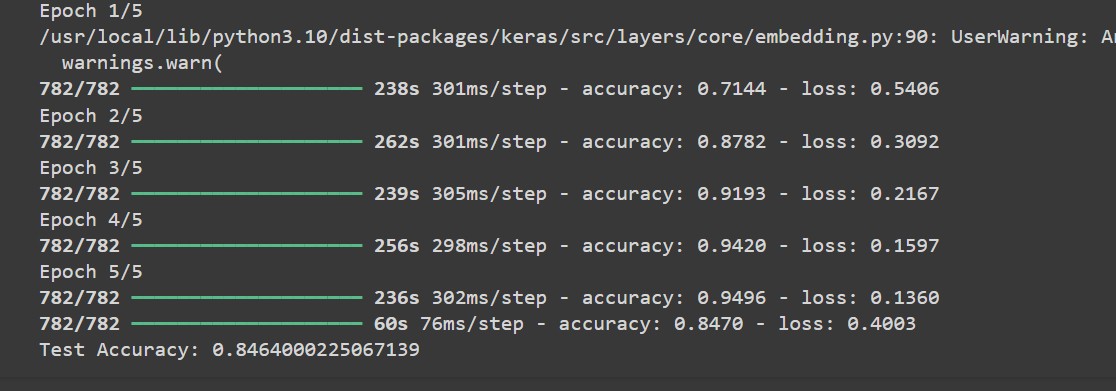
model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy']) model.fit(x\_train, y\_train, epochs=5, batch\_size=32)

# Evaluate

loss, accuracy = model.evaluate(x\_test, y\_test)

print(f"Test Accuracy: {accuracy}")

**Output:**



# EXPERIMENT - 9

### AIM: Applying the Deep Learning Models in the field of Natural Language Processing

### Theory:

### NLP involves processing and understanding human language. Deep learning models such as LSTMs and GRUs are used for tasks like text generation, translation, and sentiment analysis, as they handle sequential data efficiently.

**Source Code:**

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, LSTM, Dense

import numpy as np

# Sample text data

texts = ["Deep learning is amazing", "Natural Language Processing is fun"]

tokenizer = Tokenizer(num\_words=1000)

tokenizer.fit\_on\_texts(texts)

sequences = tokenizer.texts\_to\_sequences(texts)

# Pad sequences

data = pad\_sequences(sequences, maxlen=10)

# Build LSTM model for NLP

model = Sequential()

model.add(Embedding(input\_dim=1000, output\_dim=64, input\_length=10))

model.add(LSTM(64))

model.add(Dense(1, activation='sigmoid'))

# Compile and train the model

model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

model.summary()

# Sample training data (replace with your own data)

X\_train = np.array(data)

y\_train = np.array([1, 0])  # Example labels

# Train the model

model.fit(X\_train, y\_train, epochs=10)

# Sample evaluation data (replace with your own data)

X\_eval = np.array(data)

y\_eval = np.array([1, 0])  # Example labels

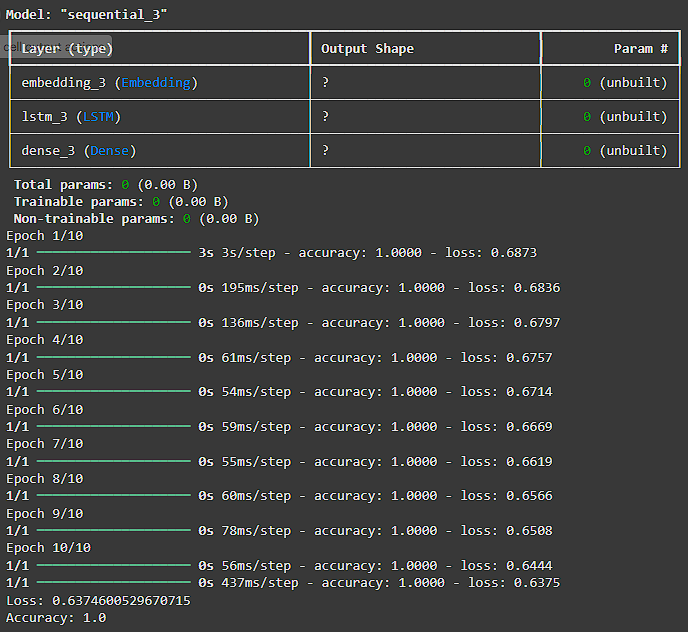
# Evaluate the model

loss, accuracy = model.evaluate(X\_eval, y\_eval)

print("Loss:", loss)

print("Accuracy:", accuracy)

**Output:**



# EXPERIMENT - 10

### AIM: Applying the Convolution Neural Network on computer vision problems

### Theory:

### CNNs have revolutionized computer vision by enabling accurate detection and classification of objects in images. They use convolutional layers to detect patterns like edges and shapes, followed by pooling and fully connected layers for decision-making.

**Source Code:**

import tensorflow as tf

from tensorflow.keras.datasets import cifar10

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

# Load CIFAR-10 dataset (for computer vision tasks)

(x\_train, y\_train), (x\_test, y\_test) = cifar10.load\_data()

# Preprocess data

x\_train = x\_train.astype('float32') / 255.0

x\_test = x\_test.astype('float32') / 255.0

# Build CNN model

model = Sequential()

model.add(Conv2D(32, (3, 3), activation='relu', input\_shape=(32, 32, 3)))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Flatten())

model.add(Dense(128, activation='relu'))

model.add(Dense(10, activation='softmax'))

# Compile and train the model

model.compile(optimizer='adam', loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])

model.summary()

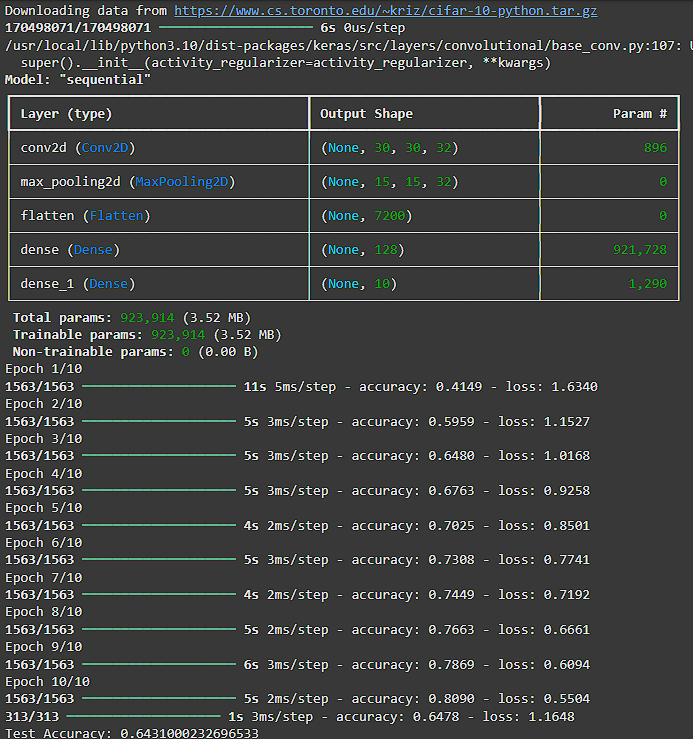
model.fit(x\_train, y\_train, epochs=10, batch\_size=32)

# Evaluate

loss, accuracy = model.evaluate(x\_test, y\_test)

print(f"Test Accuracy: {accuracy}")

**Output:**



# EXPERIMENT - 11

### AIM: Implement Deep Q Networks for CartPole problem where the agent has to balance a pole on a cart.

### Theory: Deep Q-Networks combine Q-learning with deep neural networks to handle environments with large state spaces. In the CartPole problem, the goal is to balance a pole on a cart by applying forces to the left or right. The network learns the Q-value function to predict the best action for each state.

**Source Code:**

import gym

import numpy as np

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

import random

from collections import deque

# Initialize environment

env = gym.make('CartPole-v1')

state\_size = env.observation\_space.shape[0]

action\_size = env.action\_space.n

# Build a simpler Deep Q-Network

model = Sequential()

model.add(Dense(16, input\_dim=state\_size, activation='relu'))

model.add(Dense(16, activation='relu'))

model.add(Dense(action\_size, activation='linear'))

model.compile(loss='mse', optimizer=tf.keras.optimizers.Adam(learning\_rate=0.001))

# Parameters for Deep Q-Learning

gamma = 0.95

epsilon = 1.0

epsilon\_min = 0.01

epsilon\_decay = 0.995

batch\_size = 64

memory = deque(maxlen=500)

def remember(state, action, reward, next\_state, done):

    memory.append((state, action, reward, next\_state, done))

def act(state):

    if np.random.rand() <= epsilon:

        return env.action\_space.sample()

    q\_values = model.predict(state, verbose=0)

    return np.argmax(q\_values[0])

def replay():

    global epsilon

    if len(memory) < batch\_size:

        return

    minibatch = random.sample(memory, batch\_size)

    for state, action, reward, next\_state, done in minibatch:

        target = reward if done else reward + gamma \* np.amax(model.predict(next\_state, verbose=0)[0])

        target\_f = model.predict(state, verbose=0)

        target\_f[0][action] = target

        model.fit(state, target\_f, epochs=1, verbose=0)

    if epsilon > epsilon\_min:

        epsilon \*= epsilon\_decay

# Training the agent with fewer episodes and steps

num\_episodes = 10

for e in range(num\_episodes):

    state = env.reset()

    state = np.reshape(state, [1, state\_size])

    for time in range(200):

        action = act(state)

        next\_state, reward, done, \_ = env.step(action)

        reward = reward if not done else -10

        next\_state = np.reshape(next\_state, [1, state\_size])

        remember(state, action, reward, next\_state, done)

        state = next\_state

        if done:

            print(f"Episode: {e+1}/{num\_episodes}, Score: {time}, Epsilon: {epsilon:.2f}")

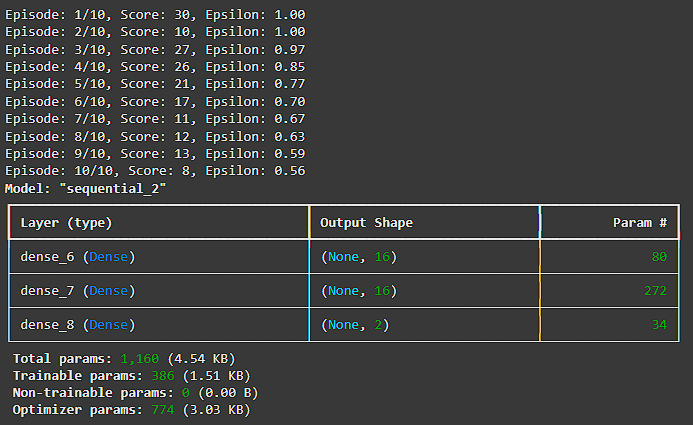
            break

        replay()

# Output: Model Summary

model.summary()

**Output:**

****

# EXPERIMENT - 12

### AIM: Demonstrate the application of transfer learning using Cartpole dataset and MountainCar dataset.

### Theory: Transfer learning allows a model trained on one problem to be used in another related problem by fine-tuning. In this case, a model trained on CartPole can be adapted for the MountainCar problem, transferring knowledge about control systems between tasks.

**Source Code:**

import gym

import torch

import torch.nn as nn

import torch.optim as optim

from torch.distributions import Categorical

# Set up Cartpole environment

cartpole\_env = gym.make('CartPole-v0')

cartpole\_state\_size = cartpole\_env.observation\_space.shape[0]

cartpole\_action\_size = cartpole\_env.action\_space.n

# Set up MountainCar environment

mountaincar\_env = gym.make('MountainCar-v0')

mountaincar\_state\_size = mountaincar\_env.observation\_space.shape[0]

mountaincar\_action\_size = mountaincar\_env.action\_space.n

# Define the base model

class BaseModel(nn.Module):

    def \_\_init\_\_(self, input\_size, hidden\_size):

        super(BaseModel, self).\_\_init\_\_()

        self.fc1 = nn.Linear(input\_size, hidden\_size)

        self.fc2 = nn.Linear(hidden\_size, hidden\_size)

        self.fc3 = nn.Linear(hidden\_size, hidden\_size)

        self.fc4 = nn.Linear(hidden\_size, hidden\_size)

        self.relu = nn.ReLU()

    def forward(self, x):

        x = self.fc1(x)

        x = self.relu(x)

        x = self.fc2(x)

        x = self.relu(x)

        x = self.fc3(x)

        x = self.relu(x)

        x = self.fc4(x)

        x = self.relu(x)

        return x

# Create the Cartpole model

class CartpoleModel(nn.Module):

    def \_\_init\_\_(self, base\_model):

        super(CartpoleModel, self).\_\_init\_\_()

        self.base\_model = base\_model

        self.fc3 = nn.Linear(base\_model.fc2.out\_features, cartpole\_action\_size)

    def forward(self, x):

        x = self.base\_model(x)

        x = self.fc3(x)

        return x

cartpole\_model = CartpoleModel(BaseModel(cartpole\_state\_size, 128))

# Create the MountainCar model

class MountainCarModel(nn.Module):

    def \_\_init\_\_(self, base\_model):

        super(MountainCarModel, self).\_\_init\_\_()

        self.base\_model = base\_model

        self.fc3 = nn.Linear(base\_model.fc2.out\_features, mountaincar\_action\_size)

    def forward(self, x):

        x = self.base\_model(x)

        x = self.fc3(x)

        return x

mountaincar\_model = MountainCarModel(BaseModel(mountaincar\_state\_size, 128))

# Train and evaluate the models

def train\_model(model, env, num\_episodes=500, gamma=0.99):

    optimizer = optim.Adam(model.parameters(), lr=0.001)

    for episode in range(num\_episodes):

        state = env.reset()

        done = False

        rewards = []

        log\_probs = []

        while not done:

            state = torch.from\_numpy(state).float().unsqueeze(0)

            output = model(state)

            dist = Categorical(logits=output)

            action = dist.sample().item()

            next\_state, reward, done, \_ = env.step(action)

            log\_prob = dist.log\_prob(torch.tensor(action))

            rewards.append(reward)

            log\_probs.append(log\_prob)

            state = next\_state

        returns = []

        R = 0

        for r in rewards[::-1]:

            R = r + gamma \* R

            returns.insert(0, R)

        loss = 0

        for log\_prob, R in zip(log\_probs, returns):

            loss -= log\_prob \* R

        optimizer.zero\_grad()

        loss.backward()

        optimizer.step()

def evaluate\_model(model, env, num\_episodes=10):

    total\_reward = 0

    for \_ in range(num\_episodes):

        state = env.reset()

        done = False

        while not done:

            state = torch.from\_numpy(state).float().unsqueeze(0)

            action = model(state).argmax().item()

            state, reward, done, \_ = env.step(action)

            total\_reward += reward

    return total\_reward / num\_episodes

# Show Model

def print\_model\_summary(model):

    # Print each layer of the model

    print(f"Model Summary:")

    for name, layer in model.named\_children():

        print(f"{name}: {layer}")

# Print the summary of the Cartpole model

print("\nCartpole Model Summary:")

print\_model\_summary(cartpole\_model)

# Print the summary of the MountainCar model

print("\nMountainCar Model Summary:")

print\_model\_summary(mountaincar\_model)

# Train and evaluate the Cartpole model

train\_model(cartpole\_model, cartpole\_env, num\_episodes=500)

cartpole\_score = evaluate\_model(cartpole\_model, cartpole\_env)

print(f"Cartpole score: {cartpole\_score}")

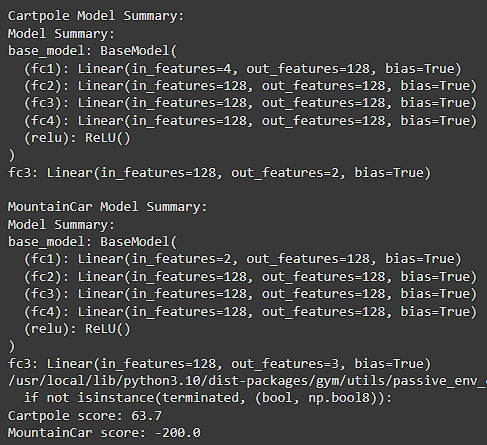
# Train and evaluate the MountainCar model

train\_model(mountaincar\_model, mountaincar\_env, num\_episodes=1000)

mountaincar\_score = evaluate\_model(mountaincar\_model, mountaincar\_env)

print(f"MountainCar score: {mountaincar\_score}")

**Output:**

****

# EXPERIMENT - 13

### AIM: Choose any corpus available on the internet freely. For the corpus, for each document, count how many times each stop word occurs and find out which are the most frequently occurring stop words. Further, calculate the term frequency and inverse document frequency as the motivation behind this is basically to find out how important a document is to a given query.

### Theory: Stop words are common words like "the", "is", "in", which are filtered out in text processing. TF-IDF measures the importance of a word in a document relative to a corpus. It reduces the weight of common words (high frequency) and highlights significant ones (low frequency across documents).

**Source Code:**

import nltk

nltk.download('stopwords')

from sklearn.feature\_extraction.text import TfidfVectorizer

from nltk.corpus import stopwords

# Sample corpus corpus = [

"The brown fox jumps over the lazy dog.",

"The quick brown fox jumped over the lazy dog."

]

# Stop words

stop\_words = list(stopwords.words('english')) # Count stop word occurrences

stop\_word\_count = {word: 0 for word in stop\_words}

for doc in corpus:

for word in doc.lower().split():

if word in stop\_word\_count: stop\_word\_count[word] += 1

# TF-IDF calculation

vectorizer = TfidfVectorizer(stop\_words=stop\_words)

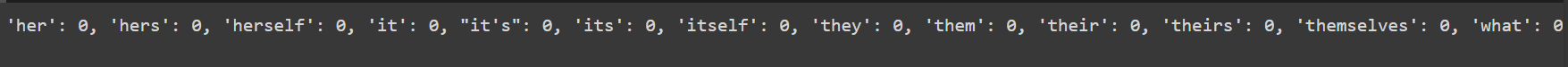
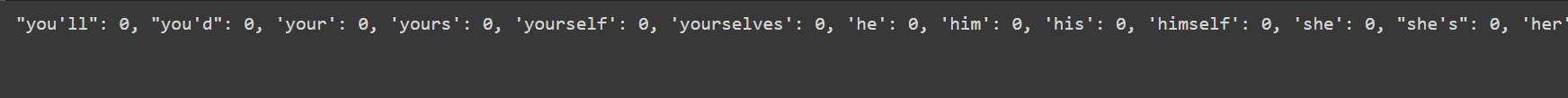
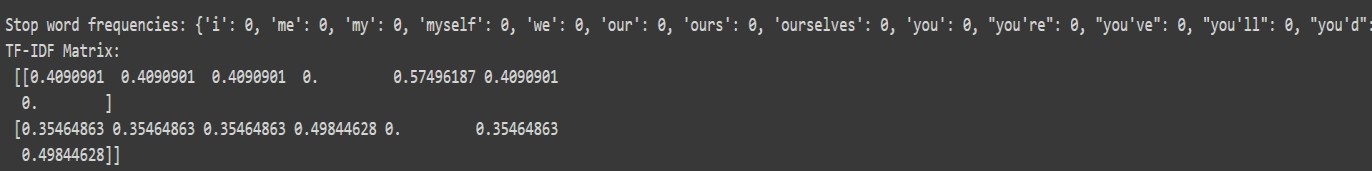
tfidf\_matrix = vectorizer.fit\_transform(corpus)

# Output stop word frequencies and TF-IDF scores

print("Stop word frequencies:", stop\_word\_count)

print("TF-IDF Matrix:\n", tfidf\_matrix.toarray())

**Output:**



# EXPERIMENT - 14

### AIM: Write the python code to develop Spam Filter using NLP.

### Theory: A spam filter classifies emails or text messages as either spam or not spam. NLP techniques such as tokenization, vectorization, and machine learning models (e.g., Naive Bayes, SVM) are used to distinguish between normal and spam content based on text patterns.

**Source Code:**

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.naive\_bayes import MultinomialNB

from sklearn.model\_selection import train\_test\_split

# Sample dataset

emails = ["Hey, wanna grab lunch?", "Win $1000 now!", "Your package has shipped.", "Limited time offer, claim your prize!"]

labels = [0, 1, 0, 1] # 0 = Not Spam, 1 = Spam

# Vectorize emails

vectorizer = CountVectorizer()

X = vectorizer.fit\_transform(emails)

# Split into train and test

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, labels, test\_size=0.2)

# Train Naive Bayes classifier

model = MultinomialNB()

model.fit(X\_train, y\_train)

# Evaluate the model

accuracy = model.score(X\_test, y\_test)

print(f"Spam Filter Accuracy: {accuracy}")

**Output:**



# EXPERIMENT - 15

### AIM: Demonstrate any one application of generative adversarial network (GAN).

### Theory: GANs consist of two neural networks, a generator and a discriminator, which compete in a zero-sum game. The generator creates synthetic data, while the discriminator distinguishes between real and generated data. GANs are used in image generation, style transfer, and more.

**Source Code:**

import tensorflow as tf

from tensorflow.keras.layers import Dense, LeakyReLU, Reshape, Flatten

from tensorflow.keras.models import Sequential

import numpy as np

# Generator

def build\_generator():

model = Sequential()

model.add(Dense(128, input\_dim=100))

model.add(LeakyReLU(alpha=0.01))

model.add(Dense(784, activation='tanh'))

model.add(Reshape((28, 28, 1)))

return model

# Discriminator

def build\_discriminator():

model = Sequential()

model.add(Flatten(input\_shape=(28, 28, 1)))

model.add(Dense(128))

model.add(LeakyReLU(alpha=0.01))

model.add(Dense(1, activation='sigmoid'))

return model

# Create GAN

generator = build\_generator()

discriminator = build\_discriminator()

# Compile GAN (placeholder for real training process)

print("GAN Model Built.")

**Output:**

