deerwalk
DWIT College

LAB 1

Hangman(Simple Intelligent Agent)

Submitted by:	Submitted to:
Nabin Katwal	Deepak Bhatt
Class of 2022	Artificial Intelligence
Roll No: 824	

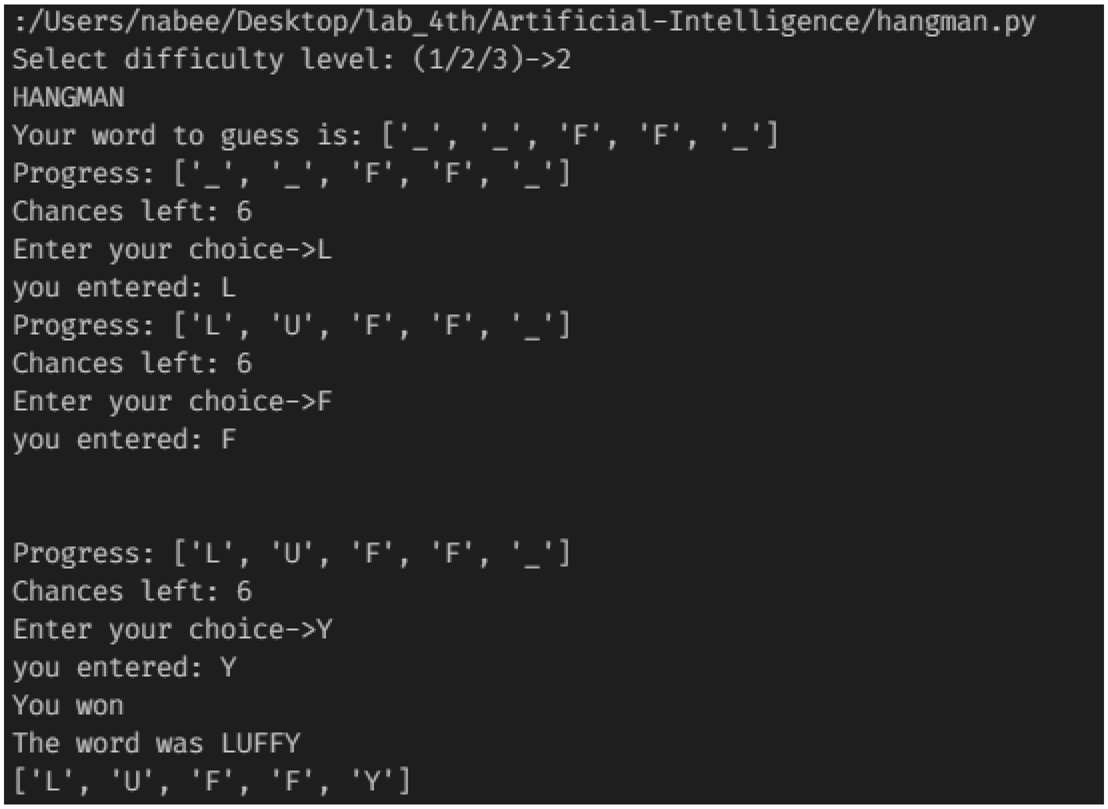
INTRODUCTION

An intelligent agent (IA) is an entity that makes a decision that enables artificial intelligence to be put into action. It can also be described as a software entity that conducts operations in the place of users or programs after sensing the environment. It uses actuators to initiate action in that environment. This agent has some level of autonomy that allows it to perform specific, predictable, and repetitive tasks for users or applications. It’s also termed as ‘intelligent’ because of its ability to learn during the process of performing tasks. The two main functions of intelligent agents include perception and action. Perception is done through sensors while actions are initiated through actuators. Intelligent agents consist of sub-agents that form a hierarchical structure. Lower-level tasks are performed by these sub-agents. The higher-level agents and lower-level agents form a complete system that can solve difficult problems through intelligent behaviors or responses.

PROGRAM CODE

```
import random


class Hangman:
    def __init__(self):
        self.chance = 6;
        self.guessed = []
        self.wrong = []
        self.guess_list = []
        self.letter_list = []
        self.level = 1




    def get_word(self):
        word_list = ['Catfish','Luffy','Artificial']
        return random.choice(word_list).upper()


    def difficulty(self):
        self.level = int(input("Select difficulty level: (1/2/3)->"))
        return self.level


    def logic(self):
        self.difficulty()
        word = self.get_word()
        self.letter_list = [word[i] for i in range(0, len(word))]
        self.guess_list = ['_' for i in range(0,len(word))]
        if self.level == 1:
            word_hint, word_hint1, word_hint2 = random.randint(1, len(self.letter_list)-
1), random.randint(1, len(self.letter_list)-1), random.randint(1, len(self.letter_list)-1)
        if self.level == 2:
            word_hint, word_hint1 = random.randint(1, len(self.letter_list)-
1), random.randint(1, len(self.letter_list)-1)
        if self.level == 3:
            word_hint = random.randint(1, len(self.letter_list)-1)
        for i in range(0, len(self.letter_list)):
            if self.level == 1:
                self.guess_list[word_hint] = self.letter_list[word_hint]
                self.guess_list[word_hint1] = self.letter_list[word_hint1]
                self.guess_list[word_hint2] = self.letter_list[word_hint2]
            if self.level == 2:
                self.guess_list[word_hint] = self.letter_list[word_hint]
                self.guess_list[word_hint1] = self.letter_list[word_hint1]
            if self.level == 3:
                self.guess_list[word_hint] = self.letter_list[word_hint]


    def play(self):
        index_alphabet = ''
        self.logic()
        print("HANGMAN")
        print(f"Your word to guess is: {self.guess_list}")
        word = ''
        while self.chance != 0:
            print(f"Progress: {self.guess_list}")
            print(f"Chances left: {self.chance}")
            word = (input("Enter your choice->")).upper()
            print(f"you entered: {word}")
            if word in self.wrong:
                print("You already guessed this letter. This doesn't count.")
                self.chance += 1
            if word in self.letter_list:
                self.guessed.append(word)
                index_alphabet = self.letter_list.index(word)
                if word in self.guess_list:
                    # self.letter_list.remove(index_alphabet)
                    pass
                else:
                        self.guess_list[index_alphabet] = self.letter_list[index_alphabet]
                    if '_' not in self.guess_list:
                        print("You won")
                        print(f"The word was {self.get_word()}")
                        break
                else:
                    self.wrong.append(word)
                    self.chance -= 1
                if self.chance == 1:
                    print("Last Chance. Exiting at next wrong guess.")
                print("\n")
            print(self.guessed)
            print(self.wrong)


    if __name__ == "__main__":
        hangman = Hangman()
        # hangman.logic()
        hangman.play()
```

OUTPUT



```
:/Users/nabee/Desktop/lab_4th/Artificial-Intelligence/hangman.py
Select difficulty level: (1/2/3)->2
HANGMAN
Your word to guess is: ['_', '_', 'F', 'F', '_']
Progress: ['_', '_', 'F', 'F', '_']
Chances left: 6
Enter your choice->L
you entered: L
Progress: ['L', 'U', 'F', 'F', '_']
Chances left: 6
Enter your choice->F
you entered: F


Progress: ['L', 'U', 'F', 'F', '_']
Chances left: 6
Enter your choice->Y
you entered: Y
You won
The word was LUFFY
['L', 'U', 'F', 'F', 'Y']
```

deerwalk

DWIT College

LAB 2

Depth first search and Breadth first search

Submitted by:

Nabin Katwal

Class of 2022

Roll No: 824

Submitted to:

Deepak Bhatt

Artificial Intelligence

Depth first search

Depth-first search is recursive algorithm for traversing a tree or graph data structure. It is called the depth-first search because it starts from the root node and follows each path to its greatest depth node before moving to the next path. DFS uses a stack data structure for its implementation. The process of the DFS algorithm is similar to the BFS algorithm.

Advantage:

- DFS requires very less memory as it only needs to store a stack of the nodes on the path from root node to the current node.
- It takes less time to reach to the goal node than BFS algorithm (if it traverses in the right path).

Disadvantage:

- There is the possibility that many states keep re-occurring, and there is no guarantee of finding the solution.
- DFS algorithm goes for deep down searching and sometime it may go to the infinite loop.

Breadth first search

Breadth-first search is the most common search strategy for traversing a tree or graph. This algorithm searches breadthwise in a tree or graph, so it is called breadth-first search. BFS algorithm starts searching from the root node of the tree and expands all successor node at the current level before moving to nodes of next level. The breadth-first search algorithm is an example of a general-graph search algorithm. Breadth-first search implemented using FIFO queue data structure.

Advantages:

- BFS will provide a solution if any solution exists.
- If there are more than one solutions for a given problem, then BFS will provide the minimal solution which requires the least number of steps.

Disadvantages:

- It requires lots of memory since each level of the tree must be saved into memory to expand the next level.
- BFS needs lots of time if the solution is far away from the root node.

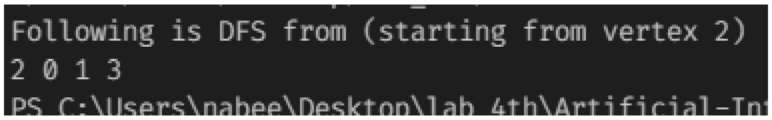
Depth first search

PROGRAM CODE

```
from collections import defaultdict


class Graph:
    def __init__(self):
        self.graph = defaultdict(list)


    def addEdge(self, u, v):
        self.graph[u].append(v)


    def DFSUtil(self, v, visited):
        visited.add(v)
        print(v, end=' ')
        for neighbour in self.graph[v]:
            if neighbour not in visited:
                self.DFSUtil(neighbour, visited)


    def DFS(self, v):
        visited = set()
        self.DFSUtil(v, visited)


g = Graph()
g.addEdge(0, 1)
```

```
g.addEdge(0, 2)
g.addEdge(1, 2)
g.addEdge(2, 0)
g.addEdge(2, 3)
g.addEdge(3, 3)


print("Following is DFS from (starting from vertex 2)")
g.DFS(2)
```

OUTPUT



```
Following is DFS from (starting from vertex 2)
2 0 1 3
PS C:\Users\nabee\Desktop\lab 4th\Artificial-In
```

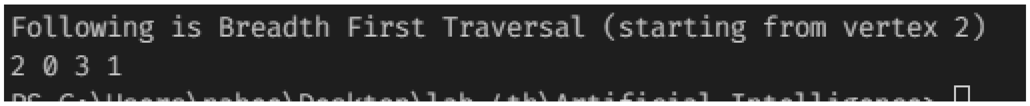
Breadth first search

PROGRAM CODE

```
from collections import defaultdict


class Graph:
    def __init__(self):
        self.graph = defaultdict(list)


    def addEdge(self,u,v):
        self.graph[u].append(v)


    def BFS(self, s):
        visited = [False] * (max(self.graph) + 1)
        queue = []


        queue.append(s)
        visited[s] = True


        while queue:
            s = queue.pop(0)
            print (s, end = " ")
            for i in self.graph[s]:
                if visited[i] == False:
                    queue.append(i)
                    visited[i] = True
```

```
g = Graph()

g.addEdge(0, 1)

g.addEdge(0, 2)

g.addEdge(1, 2)

g.addEdge(2, 0)

g.addEdge(2, 3)

g.addEdge(3, 3)


print ("Following is Breadth First Traversal"

          " (starting from vertex 2)")

g.BFS(2)
```

OUTPUT



```
Following is Breadth First Traversal (starting from vertex 2)
2 0 3 1
PS C:\Users\nahee\Desktop\lab (th)\Artificial Intelligence>
```

deerwalk
DWIT College

LAB 3

Greedy Best First, A*

Submitted by:	Submitted to:
Nabin Katwal	Deepak Bhatt
Class of 2022	Artificial Intelligence
Roll No: 824	

INTRODUCTION

Greedy Best First

Greedy best-first search algorithm always selects the path which appears best at that moment. It is the combination of depth-first search and breadth-first search algorithms. It uses the heuristic function and search. Best-first search allows us to take the advantages of both algorithms. With the help of best-first search, at each step, we can choose the most promising node. In the best first search algorithm, we expand the node which is closest to the goal node and the closest cost is estimated by heuristic function, i.e.

$F(n) = g(n)$

Advantages:

- Best first search can switch between BFS and DFS by gaining the advantages of both the algorithms.
- This algorithm is more efficient than BFS and DFS algorithms.

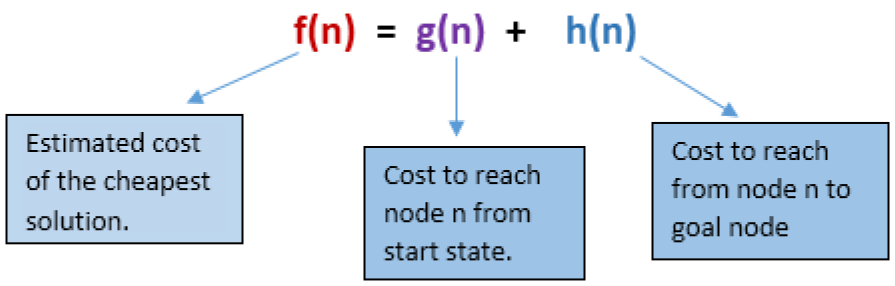
Disadvantages:

- It can behave as an unguided depth-first search in the worst case scenario.
- It can get stuck in a loop as DFS.
- This algorithm is not optimal.

A Star

A* search is the most commonly known form of best-first search. It uses heuristic function h(n), and cost to reach the node n from the start state g(n). It has combined features of UCS and greedy best-first search, by which it solve the problem efficiently. A* search algorithm finds the shortest path through the search space using the heuristic function. This search algorithm expands less search tree and provides optimal result faster. A* algorithm is similar to UCS except that it uses g(n)+h(n) instead of g(n).

In A* search algorithm, we use search heuristic as well as the cost to reach the node. Hence we can combine both costs as following, and this sum is called as a fitness number.


f(n) = g(n) + h(n)
Estimated cost of the cheapest solution.
Cost to reach node n from start state.
Cost to reach from node n to goal node

Advantages:

- A* search algorithm is the best algorithm than other search algorithms.
- A* search algorithm is optimal and complete.
- This algorithm can solve very complex problems.

Disadvantages:

- It does not always produce the shortest path as it mostly based on heuristics and approximation.
- A* search algorithm has some complexity issues.
- The main drawback of A* is memory requirement as it keeps all generated nodes in the memory, so it is not practical for various large-scale problems.

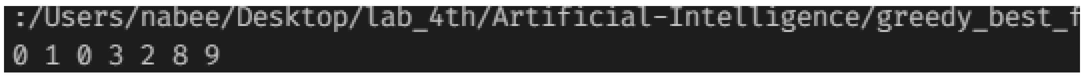
Greedy Best First

PROGRAM CODE

```
from queue import PriorityQueue
v = 14
graph = [[] for i in range(v)]


def best_first(source, target, n):
   Visited = [0]*n
   visited = True
   queue = PriorityQueue()
```

```
    queue.put((0, source))
    while queue.empty() == False:
        u = queue.get()[1]
        print(u, end = " ")
        if u == target:
            break

        for v,c in graph[u]:
            if Visited[v] == False:
                Visited[v] = True
                queue.put((c,v))
    print()

def add_edge(x, y, cost):
    graph[x].append((y, cost))
    graph[y].append((x, cost))

add_edge(0, 1, 3)
add_edge(0, 2, 6)
add_edge(0, 3, 5)
add_edge(1, 4, 9)
add_edge(1, 5, 8)
add_edge(2, 6, 12)
add_edge(2, 7, 14)
add_edge(3, 8, 7)
add_edge(8, 9, 5)

add_edge(8, 10, 6)

add_edge(9, 11, 1)

add_edge(9, 12, 10)

add_edge(9, 13, 2)


source = 0

target = 9

best_first(source, target, v)
```

OUTPUT



```
:/Users/nabee/Desktop/lab_4th/Artificial-Intelligence/greedy_best_f
0 1 0 3 2 8 9
```

A Star

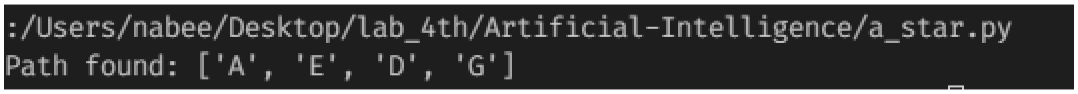
PROGRAM CODE

```
def aStarAlgo(start_node, stop_node):


        open_set = set(start_node)

        closed_set = set()

        g = {}

        parents = {}

        g[start_node] = 0

        parents[start_node] = start_node




        while len(open_set) > 0:

            n = None

            for v in open_set:

                if n == None or g[v] + heuristic(v) < g[n] + heuristic(n):

                    n = v




            if n == stop_node or Graph_nodes[n] == None:

                pass

            else:
```

```
    for (m, weight) in get_neighbors(n):
        if m not in open_set and m not in closed_set:
            open_set.add(m)
            parents[m] = n
            g[m] = g[n] + weight

        else:
            if g[m] > g[n] + weight:
                g[m] = g[n] + weight
                parents[m] = n
                if m in closed_set:
                    closed_set.remove(m)
                    open_set.add(m)

if n == None:
    print('Path does not exist!')
    return None

if n == stop_node:
    path = []

    while parents[n] != n:
        path.append(n)
        n = parents[n]
```

```
                path.append(start_node)

                path.reverse()

                print('Path found: {}'.format(path))
                return path

            open_set.remove(n)
            closed_set.add(n)

        print('Path does not exist!')
        return None

def get_neighbors(v):
    if v in Graph_nodes:
        return Graph_nodes[v]
    else:
        return None

def heuristic(n):
        H_dist = {
            'A': 11,
            'B': 6,
            'C': 99,
            'D': 1,
            'E': 7,

            'G': 0,


        }


        return H_dist[n]


Graph_nodes = {

    'A': [('B', 2), ('E', 3)],

    'B': [('C', 1),('G', 9)],

    'C': None,

    'E': [('D', 6)],

    'D': [('G', 1)],


}

aStarAlgo('A', 'G')
```

OUTPUT



```
:/Users/nabee/Desktop/lab_4th/Artificial-Intelligence/a_star.py
Path found: ['A', 'E', 'D', 'G']
```

LAB 4

Constraint Satisfaction Problem

Submitted by:

Nabin Katwal

Class of 2022

Roll No: 824

Submitted to:

Deepak Bhatt

Artificial Intelligence

INTRODUCTION

Water Jug Problem

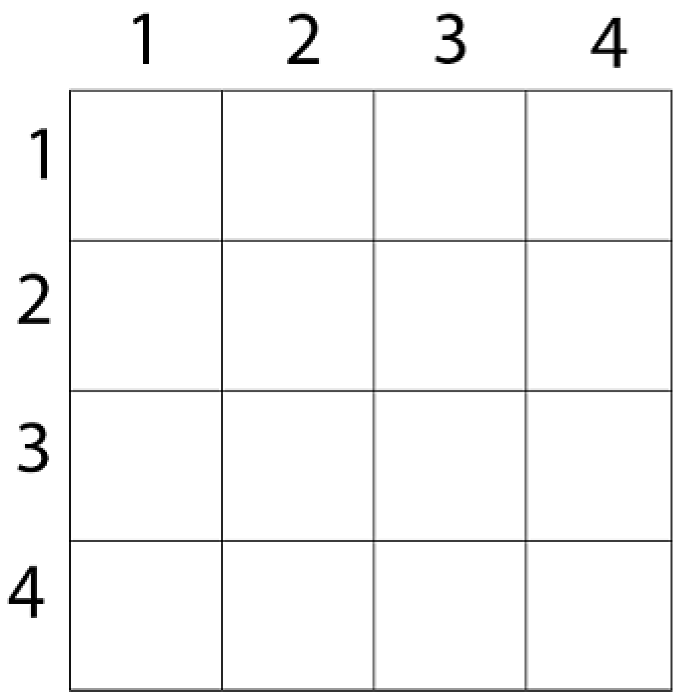
In the water jug problem in Artificial Intelligence, we are provided with two jugs: one having the capacity to hold 3 gallons of water and the other has the capacity to hold 4 gallons of water. There is no other measuring equipment available and the jugs also do not have any kind of marking on them. So, the agent's task here is to fill the 4-gallon jug with 2 gallons of water by using only these two jugs and no other material. Initially, both our jugs are empty.

N Queen problem

N - Queens problem is to place n - queens in such a manner on an n x n chessboard that no queens attack each other by being in the same row, column or diagonal.

It can be seen that for n =1, the problem has a trivial solution, and no solution exists for n =2 and n =3. So first we will consider the 4 queens problem and then generate it to n - queens problem.

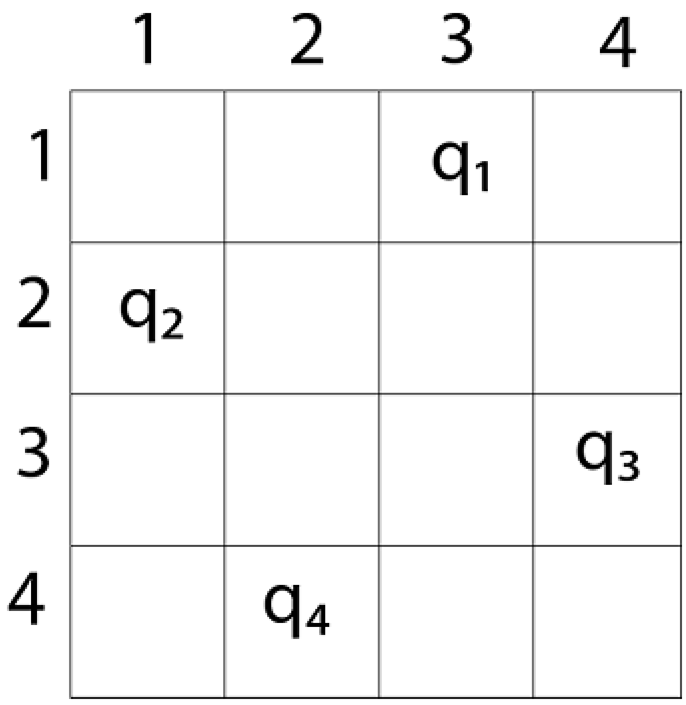
Given a 4 x 4 chessboard and number the rows and column of the chessboard 1 through 4.



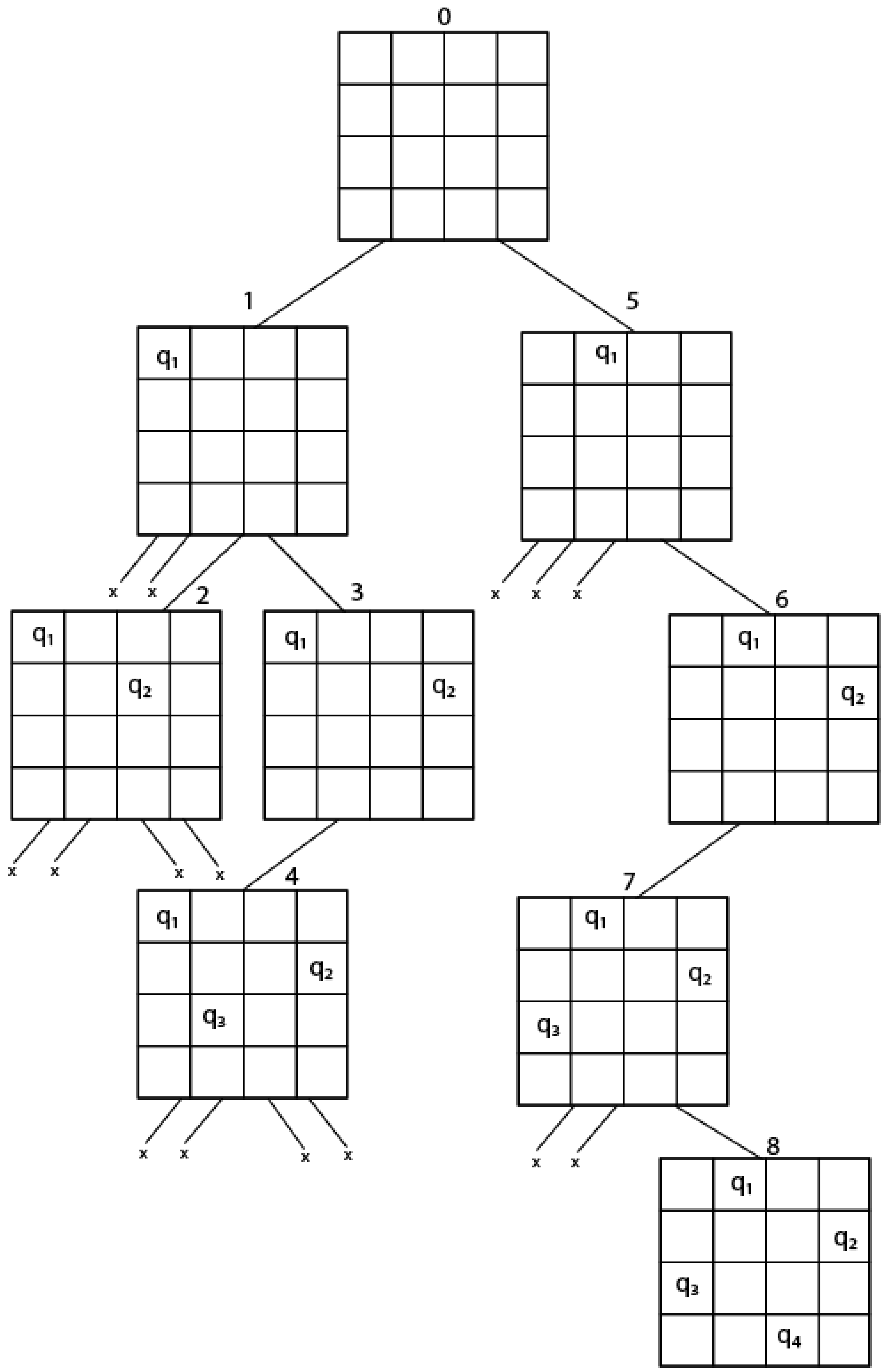
4x4 chessboard

Since, we have to place 4 queens such as q1 q2 q3 and q4 on the chessboard, such that no two queens attack each other. In such a conditional each queen must be placed on a different row, i.e., we put queen "i" on row "i."

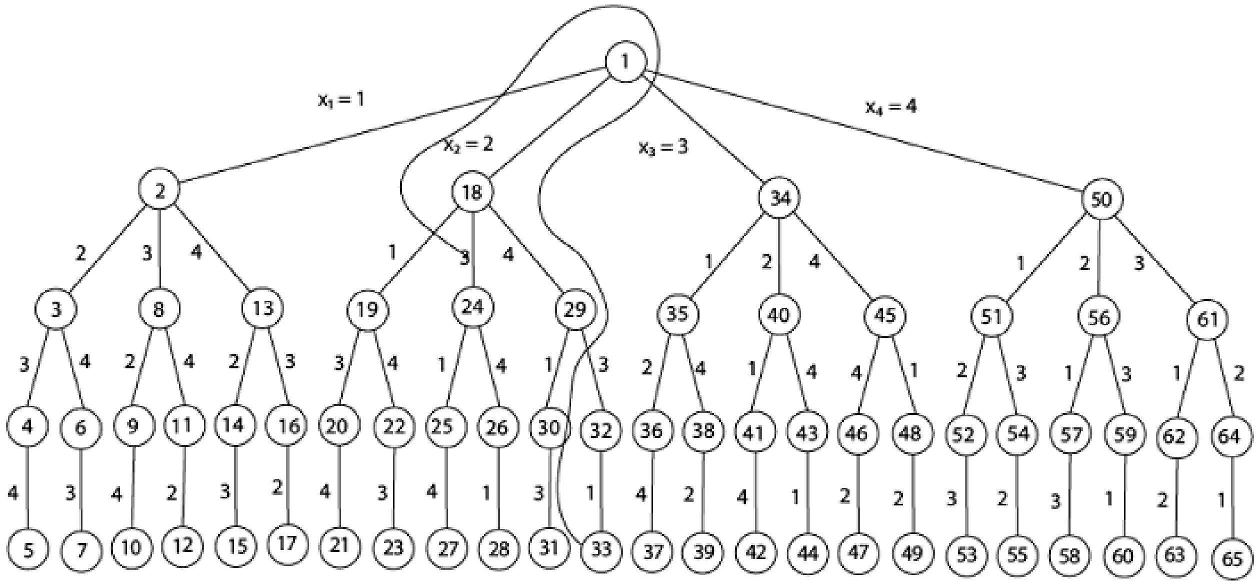
Now, we place queen q1 in the very first acceptable position (1, 1). Next, we put queen q2 so that both these queens do not attack each other. We find that if we place q2 in column 1 and 2, then the dead end is encountered. Thus the first acceptable position for q2 in column 3, i.e. (2, 3) but then no position is left for placing queen 'q3' safely. So we backtrack one step and place the queen 'q2' in (2, 4), the next best possible solution. Then we obtain the position for placing 'q3' which is (3, 2). But later this position also leads to a dead end, and no place is found where 'q4' can be placed safely. Then we have to backtrack till 'q1' and place it to (1, 2) and then all other queens are placed safely by moving q2 to (2, 4), q3 to (3, 1) and q4 to (4, 3). That is, we get the solution (2, 4, 1, 3). This is one possible solution for the 4-queens problem. For another possible solution, the whole method is repeated for all partial solutions. The other solutions for 4 - queens problems is (3, 1, 4, 2) i.e.



The implicit tree for 4 - queen problem for a solution (2, 4, 1, 3) is as follows:

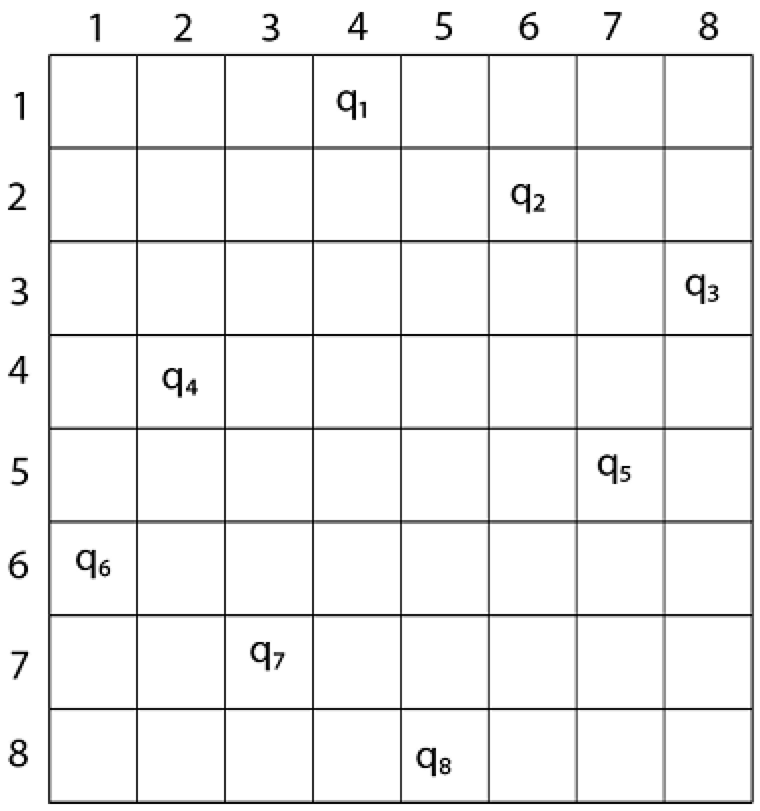

0
1
2
3
4
5
6
7
8
q1
q2
q3
q4
x

But we can use backtracking method to generate the necessary node and stop if the next node violates the rule, i.e., if two queens are attacking.



It can be seen that all the solutions to the 4 queens problem can be represented as 4 - tuples (x1, x2, x3, x4) where xi represents the column on which queen "qi" is placed.

One possible solution for 8 queens problem is shown in fig:



Water jug problem

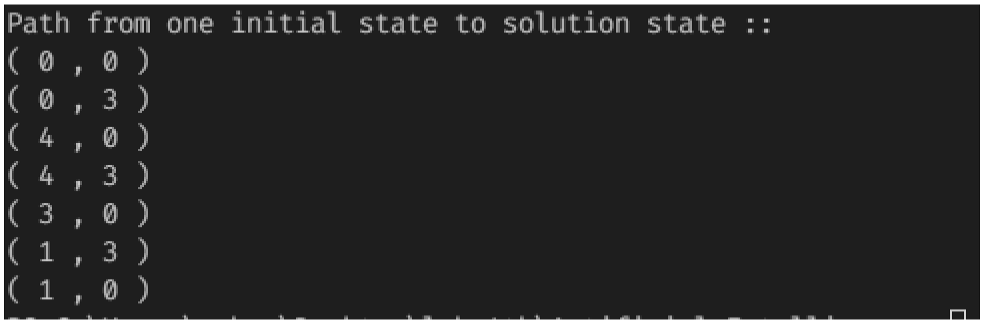
PROGRAM CODE

```
from collections import deque


def BFS(a, b, target):
    m = {}
    isSolvable = False
    path = []
    q = deque()
    q.append((0, 0))

    while (len(q) > 0):
        u = q.popleft()

        if ((u[0], u[1]) in m):
            continue

        if ((u[0] > a or u[1] > b or
             u[0] < 0 or u[1] < 0)):
            continue

        path.append([u[0], u[1]])

        m[(u[0], u[1])] = 1
```

```
if u[0] == target or u[1] == target:
    isSolvable = True


    if (u[0] == target):
        if (u[1] != 0):


            path.append([u[0], 0])
    else:
        if (u[0] != 0):


            path.append([0, u[1]])


    sz = len(path)
    for i in range(sz):
        print("(", path[i][0], ",",
                    path[i][1], ")")
    break


q.append([u[0], b]) # Fill Jug2
q.append([a, u[1]]) # Fill Jug1


for ap in range(max(a, b) + 1):


    # Pour amount ap from Jug2 to Jug1
    c = u[0] + ap
```

```
            d = u[1] - ap

            # Check if this state is possible or not
            if (c == a or (d == 0 and d >= 0)):
                q.append([c, d])

            # Pour amount ap from Jug 1 to Jug2
            c = u[0] - ap
            d = u[1] + ap

            # Check if this state is possible or not
            if ((c == 0 and c >= 0) or d == b):
                q.append([c, d])

        q.append([a, 0])
        q.append([0, b])

    if (not isSolvable):
        print ("No solution")

if __name__ == "__main__":
    a,b,target = 4,3,1
    print("Path from one initial state to solution state ::")
    BFS(a, b, target)
```

OUTPUT



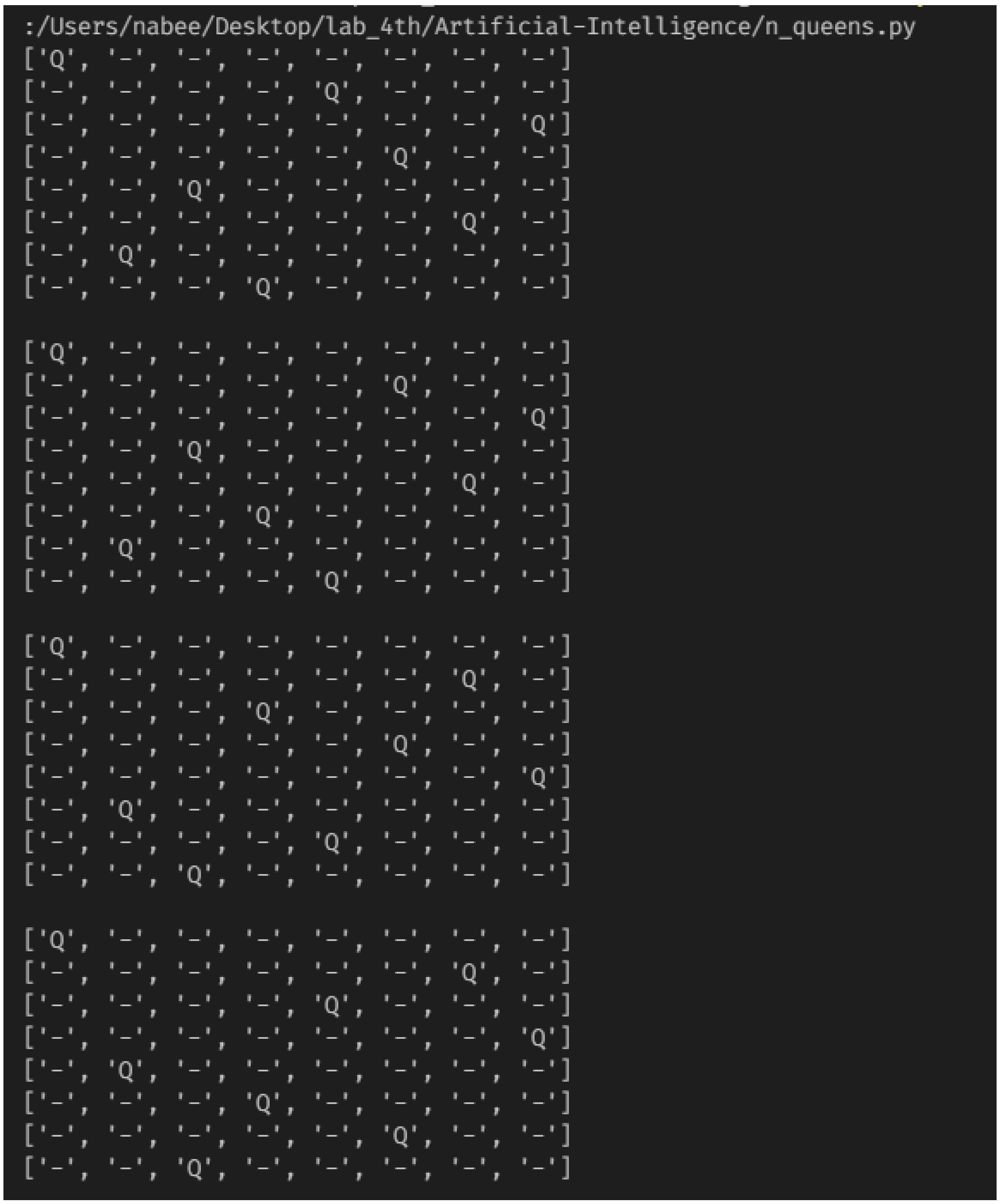
```
Path from one initial state to solution state ::
( 0 , 0 )
( 0 , 3 )
( 4 , 0 )
( 4 , 3 )
( 3 , 0 )
( 1 , 3 )
( 1 , 0 )
```

N Queen Problem

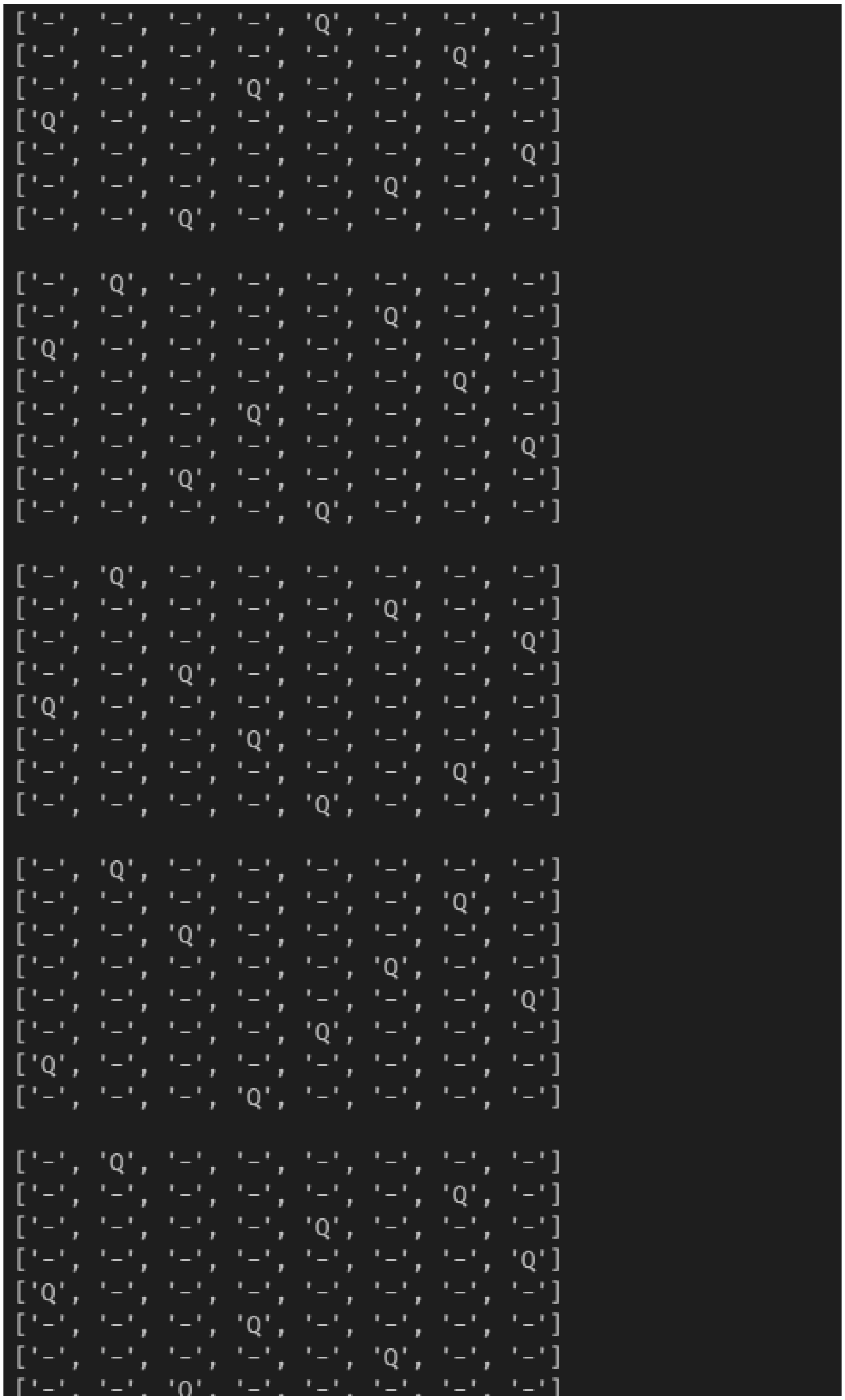
PROGRAM CODE

```
def isSafe(mat, r,c):
    for i in range(r):
        if mat[i][c] == 'Q':
            return False


    (i,j) = (r,c)
    while i >= 0 and j>= 0:
        if mat[i][j] == 'Q':
            return False
        i -= 1
        j -= 1


    (i, j) = (r, c)
    while i >= 0 and j < N:
        if mat[i][j] == 'Q':
            return False
        i -= 1
        j += 1

    return True

def printSolution(mat):
    for i in range(N):
        print(mat[i])
    print()

def nQueen(mat, r):
    if r==N:
        printSolution(mat)
        return

    for i in range(N):
        if isSafe(mat, r, i):
            mat[r][i] = 'Q'
            nQueen(mat, r+1)
            mat[r][i] = '-'

if __name__ == '__main__':
    N = 8
    mat = [['-' for x in range(N)] for y in range(N)]
    nQueen(mat, 0)
```

OUTPUT



```
:/Users/nabee/Desktop/lab_4th/Artificial-Intelligence/n_queens.py
['Q', '-', '-', '-', '-', '-', '-', '-']
['-', '-', '-', '-', 'Q', '-', '-', '-']
['-', '-', '-', '-', '-', '-', '-', 'Q']
['-', '-', '-', '-', '-', 'Q', '-', '-']
['-', '-', 'Q', '-', '-', '-', '-', '-']
['-', '-', '-', '-', '-', '-', 'Q', '-']
['-', 'Q', '-', '-', '-', '-', '-', '-']
['-', '-', '-', 'Q', '-', '-', '-', '-']

['Q', '-', '-', '-', '-', '-', '-', '-']
['-', '-', '-', '-', '-', 'Q', '-', '-']
['-', '-', '-', '-', '-', '-', '-', 'Q']
['-', '-', 'Q', '-', '-', '-', '-', '-']
['-', '-', '-', '-', '-', '-', 'Q', '-']
['-', '-', '-', 'Q', '-', '-', '-', '-']
['-', 'Q', '-', '-', '-', '-', '-', '-']
['-', '-', '-', '-', 'Q', '-', '-', '-']

['Q', '-', '-', '-', '-', '-', '-', '-']
['-', '-', '-', '-', '-', '-', 'Q', '-']
['-', '-', '-', 'Q', '-', '-', '-', '-']
['-', '-', '-', '-', '-', 'Q', '-', '-']
['-', '-', '-', '-', '-', '-', '-', 'Q']
['-', 'Q', '-', '-', '-', '-', '-', '-']
['-', '-', '-', '-', 'Q', '-', '-', '-']
['-', '-', 'Q', '-', '-', '-', '-', '-']

['Q', '-', '-', '-', '-', '-', '-', '-']
['-', '-', '-', '-', '-', '-', 'Q', '-']
['-', '-', '-', '-', 'Q', '-', '-', '-']
['-', '-', '-', '-', '-', '-', '-', 'Q']
['-', 'Q', '-', '-', '-', '-', '-', '-']
['-', '-', '-', 'Q', '-', '-', '-', '-']
['-', '-', '-', '-', '-', 'Q', '-', '-']
['-', '-', 'Q', '-', '-', '-', '-', '-']
```



```
['-', '-', '-', '-', 'Q', '-', '-', '-']
['-', '-', '-', '-', '-', '-', 'Q', '-']
['-', '-', '-', 'Q', '-', '-', '-', '-']
['Q', '-', '-', '-', '-', '-', '-', '-']
['-', '-', '-', '-', '-', '-', '-', 'Q']
['-', '-', '-', '-', '-', 'Q', '-', '-']
['-', '-', 'Q', '-', '-', '-', '-', '-']

['-', 'Q', '-', '-', '-', '-', '-', '-']
['-', '-', '-', '-', '-', 'Q', '-', '-']
['Q', '-', '-', '-', '-', '-', '-', '-']
['-', '-', '-', '-', '-', '-', 'Q', '-']
['-', '-', '-', 'Q', '-', '-', '-', '-']
['-', '-', '-', '-', '-', '-', '-', 'Q']
['-', '-', 'Q', '-', '-', '-', '-', '-']
['-', '-', '-', '-', 'Q', '-', '-', '-']

['-', 'Q', '-', '-', '-', '-', '-', '-']
['-', '-', '-', '-', '-', 'Q', '-', '-']
['-', '-', '-', '-', '-', '-', '-', 'Q']
['-', '-', 'Q', '-', '-', '-', '-', '-']
['Q', '-', '-', '-', '-', '-', '-', '-']
['-', '-', '-', 'Q', '-', '-', '-', '-']
['-', '-', '-', '-', '-', '-', 'Q', '-']
['-', '-', '-', '-', 'Q', '-', '-', '-']

['-', 'Q', '-', '-', '-', '-', '-', '-']
['-', '-', '-', '-', '-', '-', 'Q', '-']
['-', '-', 'Q', '-', '-', '-', '-', '-']
['-', '-', '-', '-', '-', 'Q', '-', '-']
['-', '-', '-', '-', '-', '-', '-', 'Q']
['-', '-', '-', '-', 'Q', '-', '-', '-']
['Q', '-', '-', '-', '-', '-', '-', '-']
['-', '-', '-', 'Q', '-', '-', '-', '-']

['-', 'Q', '-', '-', '-', '-', '-', '-']
['-', '-', '-', '-', '-', '-', 'Q', '-']
['-', '-', '-', '-', 'Q', '-', '-', '-']
['-', '-', '-', '-', '-', '-', '-', 'Q']
['Q', '-', '-', '-', '-', '-', '-', '-']
['-', '-', '-', 'Q', '-', '-', '-', '-']
['-', '-', '-', '-', '-', 'Q', '-', '-']
['-', '-', 'Q', '-', '-', '-', '-', '-']
```

deerwalk
DWIT College

LAB 5

Simple Chat Bot

Submitted by:

Nabin Katwal

Class of 2022

Roll No: 824

Submitted to:

Deepak Bhatt

Artificial Intelligence

INTRODUCTION

Humans claim that how intelligence is achieved-not by purely reflect mechanisms but by process of reasoning that operate on internal representation of knowledge. In AI this techniques for intelligence are present in Knowledge Based Agents.

Knowledge-based agents have explicit representation of knowledge that can be reasoned. They maintain internal state of knowledge, reason over it, update it and perform actions accordingly. These agents act intelligently according to requirements.

Knowledge based agents give the current situation in the form of sentences. They have complete knowledge of current situation of mini-world and its surroundings. These agents manipulate knowledge to infer new things at “Knowledge level”.

A knowledge-based system has following features:

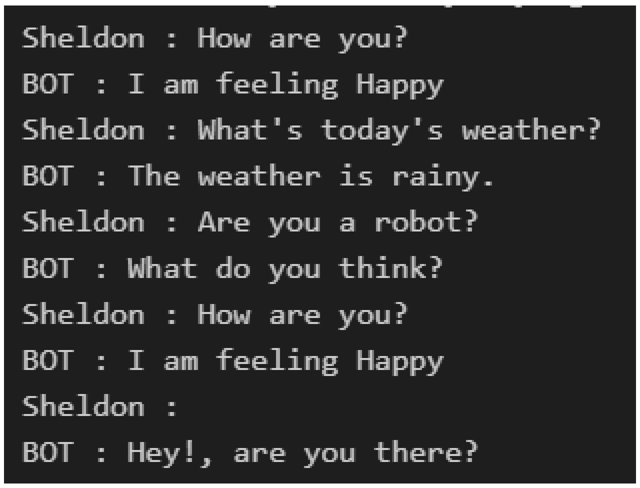
Knowledge base (KB): It is the key component of a knowledge-based agent. These deal with real facts of world. It is a mixture of sentences which are explained in knowledge representation language.

Inference Engine (IE): It is knowledge-based system engine used to infer new knowledge in the system.

PROGRAM CODE

```
import random


def respond(message):
    if message in responses:
        bot_message = random.choice(responses[message])
    else:
        bot_message = random.choice(responses["default"])

    return bot_message


def related(x_text):
    if "name" in x_text:
        y_text = "What's your name?"
    elif "weather" in x_text:
        y_text = "What's today's weather?"
    elif "robot" in x_text:
        y_text = "Are you a robot?"
    elif "how are" in x_text:
        y_text = "How are you?"
    else:
        y_text = ""

    return y_text

def send_message(message):
    print(user_template.format(message))
    response = respond(message)
    print(bot_template.format(response))

def start_bot():
    while 1:
        my_input = input()
        my_input = my_input.lower()
        related_text = related(my_input)
        send_message(related_text)
        if my_input == "exit" or my_input == "stop":
            break


if __name__ == "__main__":
    print("BOT: What do you want me to call you?")
    user_name = input()


    bot_template = "BOT : {0}"
    user_template = user_name + " : {0}"


    name = "Bot"
    weather = "rainy"
    mood = "Happy"


    responses = {
        "What's your name?":[
            "They call me {0}".format(name),
            "I usually go by {0}".format(name),
            "My name is the {0}".format(name)],
        "What's today's weather?":[
            f"The weather is {weather}.",
            f"It's {weather} today.",
            f"Let me check, it looks {weather} today."
        ],
        "Are you a robot?":[
            "What do you think?",
            "Maybe yes, Maybe no!",
            "Yes, I am a robot with human feelings."
        ],
        "How are you?":[
            f"I am feeling {mood}",
            f"{mood}! How about you?",
            f"I am {mood}! How about yourself?"
        ],
        "":[
            "Hey!, are you there?",
            "What do you mean by saying nothing?",
            "Sometimes saying nothing tells a lot :)"
        ],
        "default":[
            "This is a default message"
        ]
    }

    start_bot()
```

OUTPUT



```
Sheldon : How are you?
BOT : I am feeling Happy
Sheldon : What's today's weather?
BOT : The weather is rainy.
Sheldon : Are you a robot?
BOT : What do you think?
Sheldon : How are you?
BOT : I am feeling Happy
Sheldon :
BOT : Hey!, are you there?
```

deerwalk
DWIT College

LAB 6

Naïve Bayes Implementation

Submitted by:

Nabin Katwal

Class of 2022

Roll No: 824

Submitted to:

Deepak Bhatt

Artificial Intelligence

INTRODUCTION

A naive Bayes classifier uses probability theory to classify data. Naive Bayes classifier algorithms make use of Bayes' theorem. The key insight of Bayes' theorem is that the probability of an event can be adjusted as new data is introduced.

What makes a naive Bayes classifier naive is its assumption that all attributes of a data point under consideration are independent of each other. A classifier sorting fruits into apples and oranges would know that apples are red, round and are a certain size, but would not assume all these things at once. Oranges are round too, after all.

A naive Bayes classifier is not a single algorithm, but a family of machine learning algorithms that make uses of statistical independence. These algorithms are relatively easy to write and run more efficiently than more complex Bayes algorithms.

The most popular application is spam filters. A spam filter looks at email messages for certain key words and puts them in a spam folder if they match.

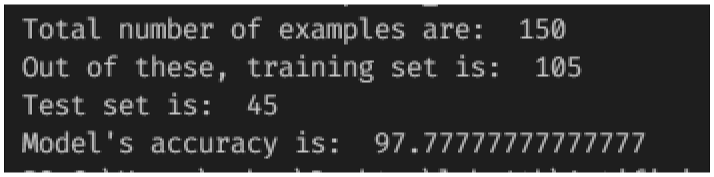
Despite the name, the more data it gets, the more accurate a naive Bayes classifier becomes, such as from a user flagging email messages in an inbox for spam.

PROGRAM CODE

Note: The following code uses Iris dataset from Scikit library

```
import math

import random

import csv


def encode_class(mydata):

    classes = []

    for i in range(len(mydata)):

        if mydata[i][-1] not in classes:

            classes.append(mydata[i][-1])


    for i in range(len(classes)):
        for j in range(len(mydata)):
            if mydata[j][-1] == classes[i]:
                mydata[j][-1] = i
    return mydata


def splitting(mydata, ratio):
    train_num = int(len(mydata) * ratio)
    train = []
    test = list(mydata)
    while len(train) < train_num:
        index = random.randrange(len(test))
        train.append(test.pop(index))
    return train, test


def groupUnderClass(mydata):
    dict = {}
    for i in range(len(mydata)):
        if mydata[i][-1] not in dict:
            dict[mydata[i][-1]] = []
        dict[mydata[i][-1]].append(mydata[i])
    return dict


def mean(numbers):
    return sum(numbers) / float(len(numbers))
```

```
def std_dev(numbers):
    avg = mean(numbers)
    variance = sum([pow(x-avg,2) for x in numbers])/float(len(numbers)-1)
    return math.sqrt(variance)

def meanAndStdDev(mydata):
    info = [(mean(attribute), std_dev(attribute)) for attribute in zip(*mydata)]
    del info[-1]
    return info

def meanAndStdDevForClass(mydata):
    info = {}
    dict = groupUnderClass(mydata)
    for classValue, instances in dict.items():
        info[classValue] = meanAndStdDev(instances)
    return info

def calculateGaussianProbability(x, mean, stdev):
    expo = math.exp(-(math.pow(x-mean, 2)/(2*math.pow(stdev,2))))
    return (1/(math.sqrt(2*math.pi)*stdev))*expo

def calculateClassProbabilites(info, test):
    probabilites = {}
    for classValue, classSummaries in info.items():
        probabilites[classValue] = 1
        for i in range(len(classSummaries)):
            mean, std_dev = classSummaries[i]
            x = test[i]
            probabilites[classValue] *= calculateGaussianProbability(x, mean, std_dev)
    return probabilites

def predict(info, test):
    probabilites = calculateClassProbabilites(info, test)
    bestLabel, bestProb = None, -1
    for classValue, probability in probabilites.items():
        if bestLabel is None or probability > bestProb:
            bestProb = probability
            bestLabel = classValue
    return bestLabel

def getPredictions(info, test):
    predictions = []
    for i in range(len(test)):
        result = predict(info, test[i])
        predictions.append(result)
    return predictions

def accuracy_rate(test, predictions):
    correct = 0
    for i in range(len(test)):
        if test[i][-1] == predictions[i]:
            correct += 1
    return (correct / float(len(test))) * 100.0


if __name__ == '__main__':
    filename = r'D:\Fourth Sem\AI\Artificial-Intelligence\naive_bayes\Iris.csv'
    mydata = csv.reader(open(filename, "rt"))
    mydata = list(mydata)
    mydata = encode_class(mydata)
    for i in range(len(mydata)):
        mydata[i] = [float(x) for x in mydata[i]]


    ratio = 0.7
    train_data, test_data = splitting(mydata, ratio)
    print('Total number of examples are: ', len(mydata))
    print('Out of these, training set is: ', len(train_data))
    print('Test set is: ', len(test_data))


    info = meanAndStdDevForClass(train_data)


    predictions = getPredictions(info, test_data)
    accuracy = accuracy_rate(test_data, predictions)
    print("Model's accuracy is: ",accuracy)
```
OUTPUT



```
Total number of examples are:  150
Out of these, training set is:  105
Test set is:  45
Model's accuracy is:  97.77777777777777
```

LAB 7

Neural Network

Submitted by:

Nabin Katwal

Class of 2022

Roll No: 824

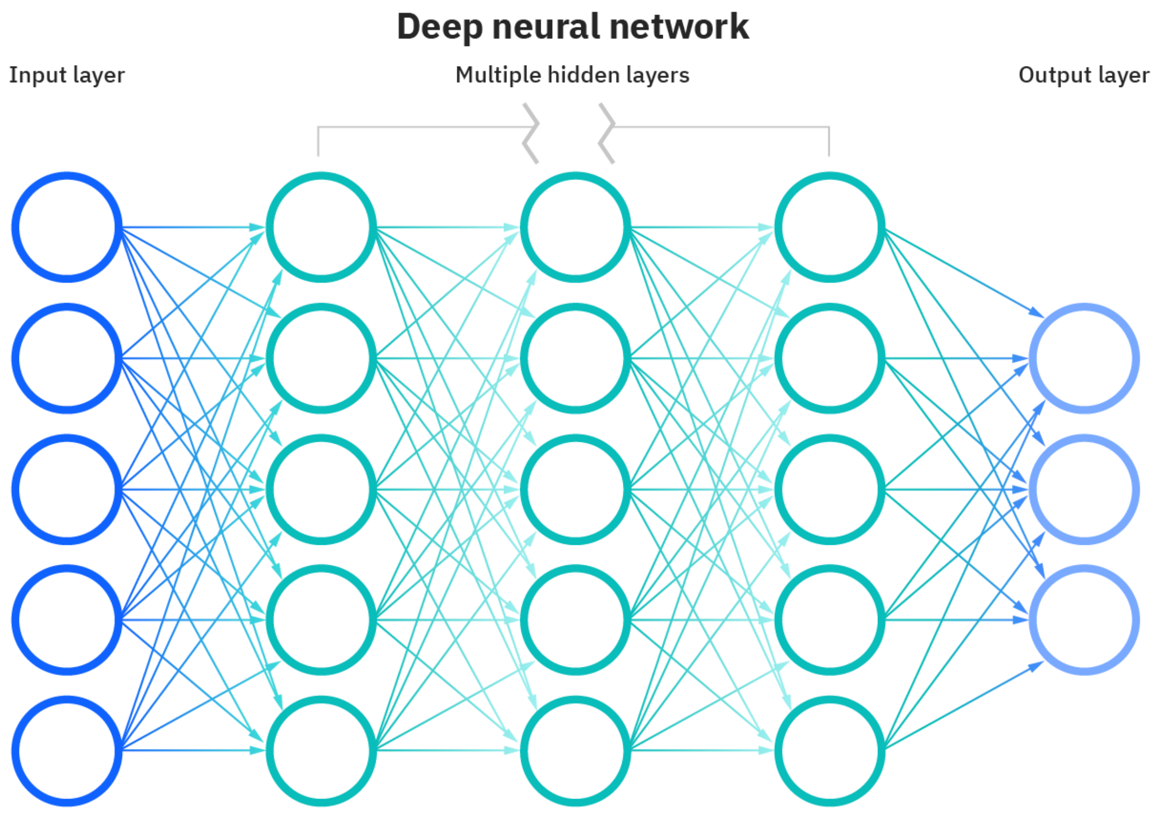
Submitted to:

Deepak Bhatt

Artificial Intelligence

INTRODUCTION

A neural network is a series of algorithms that endeavors to recognize underlying relationships in a set of data through a process that mimics the way the human brain operates. In this sense, neural networks refer to systems of neurons, either organic or artificial in nature. Neural networks can adapt to changing input; so the network generates the best possible result without needing to redesign the output criteria. The concept of neural networks, which has its roots in artificial intelligence, is swiftly gaining popularity in the development of trading systems.



Realization of AND Gate

PROGRAM CODE

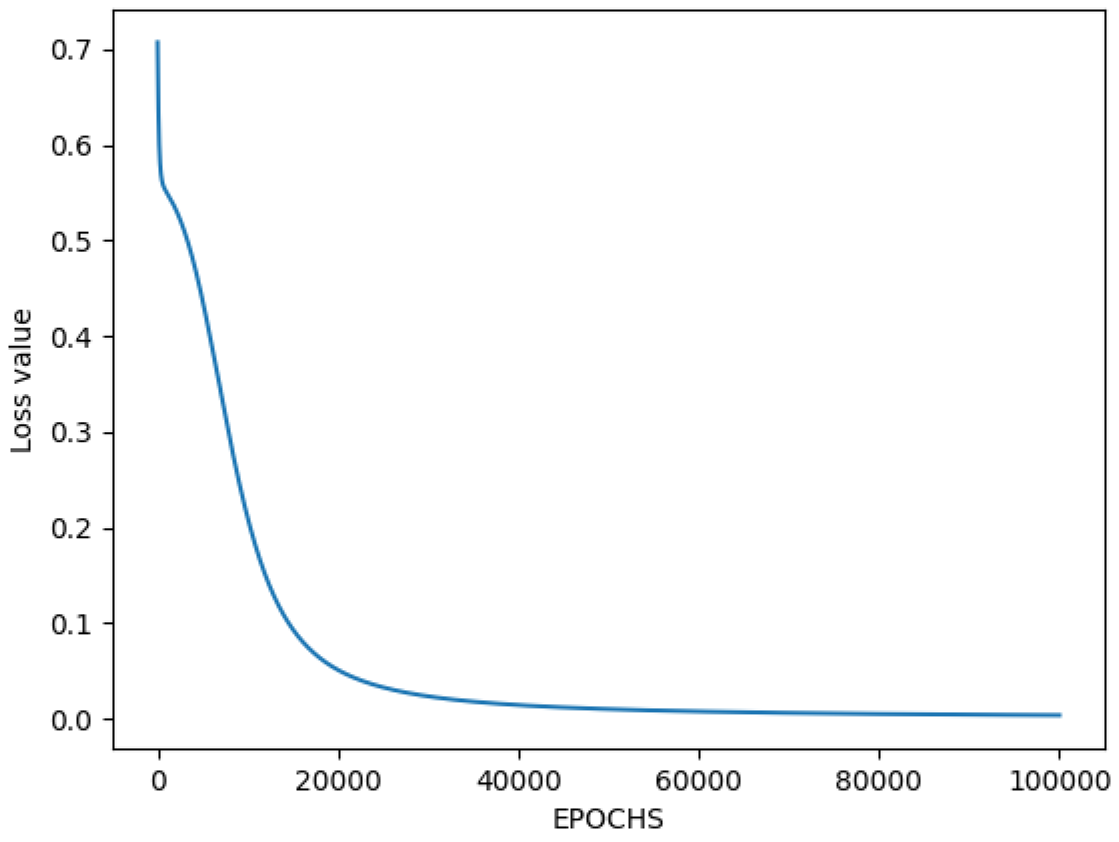
```
import numpy as np

from matplotlib import pyplot as plt

import seaborn as sns
```

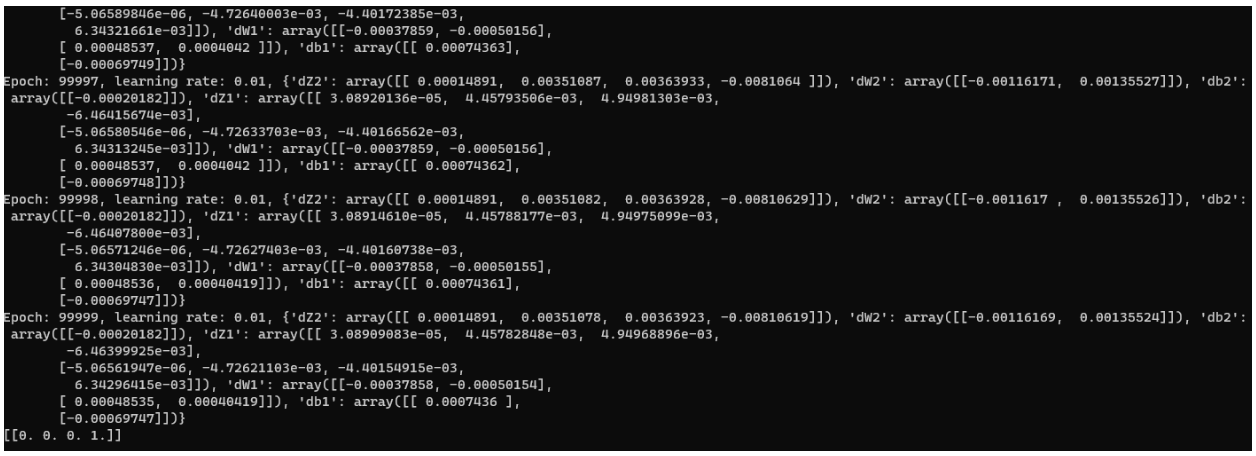
```
def sigmoid(z):
    return 1/(1+np.exp(-z))


def initializeParameters(inputFeatures, neuronsInHiddenLayers, outputFeatures):
    W1 = np.random.randn(neuronsInHiddenLayers, inputFeatures)
    W2 = np.random.randn(outputFeatures, neuronsInHiddenLayers)
    b1 = np.zeros((neuronsInHiddenLayers, 1))
    b2 = np.zeros((outputFeatures, 1))

    parameters = {
        "W1":W1,
        "b1":b1,
        "W2":W2,
        "b2":b2
    }

    return parameters


def forwardPropagation(X, Y, parameters):
    m = X.shape[1]
    W1 = parameters["W1"]
    W2 = parameters["W2"]
    b1 = parameters["b1"]
    b2 = parameters["b2"]
    Z1 = np.dot(W1, X) + b1
    A1 = sigmoid(Z1)
    Z2 = np.dot(W2, A1) + b2
    A2 = sigmoid(Z2)

    cache = (Z1, A1, W1, b1, Z2, A2, W2, b2)
    logprobs = np.multiply(np.log(A2), Y) + np.multiply(np.log(1-A2), (1-Y))
    cost = -np.sum(logprobs) / m
    return cost, cache, A2

def backwardPropagation(X, Y, cache):
    m = X.shape[1]
    (Z1, A1, W1, b1, Z2, A2, W2, b2) = cache

    dZ2 = A2-Y
    dW2 = np.dot(dZ2, A1.T)/m
    db2 = np.sum(dZ2, axis=1, keepdims = True) / m

    dA1 = np.dot(W2.T, dZ2)
    dZ1 = np.multiply(dA1, A1*(1-A1))
    dW1 = np.dot(dZ1, X.T)/m
    db1 = np.sum(dZ1, axis = 1, keepdims=True) / m

    gradients = {
        "dZ2":dZ2,
        "dW2":dW2,
        "db2":db2,
        "dZ1":dZ1,
        "dW1":dW1,
        "db1":db1
    }

    return gradients

def updateParameters(parameters, gradients, learningRate):
    parameters["W1"] = parameters["W1"]-learningRate * gradients["dW1"]
    parameters["W2"]=parameters["W2"]-learningRate*gradients["dW2"]
    parameters["b1"] = parameters["b1"] - learningRate * gradients["db1"]
    parameters["b2"] = parameters["b2"] - learningRate * gradients["db2"]
    return parameters

X = np.array([[0,0,1,1],[0,1,0,1]])
Y = np.array([[0,0,0,1]])

neuronsInHiddenLayers = 2
inputFeatures = X.shape[0]
outputFeatures = Y.shape[0]
parameters = initializeParameters(inputFeatures, neuronsInHiddenLayers, outputFeatures)
epoch = 100000
learningRate = 0.01
```

```
losses = np.zeros((epoch, 1))


for i in range(epoch):
    losses[i,0], cache, A2 = forwardPropagation(X, Y, parameters)
    gradients = backwardPropagation(X, Y, cache)
    parameters = updateParameters(parameters, gradients, learningRate)
    print(f"Epoch: {i}, learning rate: {learningRate}, {gradients}")


plt.figure()
plt.plot(losses)
plt.xlabel("EPOCHS")
plt.ylabel("Loss value")
plt.show()


X = np.array([[0,0,1,1],[0,1,0,1]])
cost, _, A2 = forwardPropagation(X, Y, parameters)
prediction = (A2 > 0.5) * 1.0
print(prediction)
```

OUTPUT

After training for 10, 000 episodes. The following results were obtained





```
       [-5.06589846e-06, -4.72640003e-03, -4.40172385e-03,
         6.34321661e-03]]), 'dW1': array([[-0.00037859, -0.00050156],
       [ 0.00048537,  0.0004042 ]]), 'db1': array([[ 0.00074363],
       [-0.00069749]])}
Epoch: 99997, learning rate: 0.01, {'dZ2': array([[ 0.00014891,  0.00351087,  0.00363933, -0.0081064 ]]), 'dW2': array([[-0.00116171,  0.00135527]]), 'db2':
 array([[-0.00020182]]), 'dZ1': array([[ 3.08920136e-05,  4.45793506e-03,  4.94981303e-03,
        -6.46415674e-03],
       [-5.06580546e-06, -4.72633703e-03, -4.40166562e-03,
         6.34313245e-03]]), 'dW1': array([[-0.00037859, -0.00050156],
       [ 0.00048537,  0.0004042 ]]), 'db1': array([[ 0.00074362],
       [-0.00069748]])}
Epoch: 99998, learning rate: 0.01, {'dZ2': array([[ 0.00014891,  0.00351082,  0.00363928, -0.00810629]]), 'dW2': array([[-0.0011617 ,  0.00135526]]), 'db2':
 array([[-0.00020182]]), 'dZ1': array([[ 3.08914610e-05,  4.45788177e-03,  4.94975099e-03,
        -6.46407800e-03],
       [-5.06571246e-06, -4.72627403e-03, -4.40160738e-03,
         6.34304830e-03]]), 'dW1': array([[-0.00037858, -0.00050155],
       [ 0.00048536,  0.00040419]]), 'db1': array([[ 0.00074361],
       [-0.00069747]])}
Epoch: 99999, learning rate: 0.01, {'dZ2': array([[ 0.00014891,  0.00351078,  0.00363923, -0.00810619]]), 'dW2': array([[-0.00116169,  0.00135524]]), 'db2':
 array([[-0.00020182]]), 'dZ1': array([[ 3.08909083e-05,  4.45782848e-03,  4.94968896e-03,
        -6.46399925e-03],
       [-5.06561947e-06, -4.72621103e-03, -4.40154915e-03,
         6.34296415e-03]]), 'dW1': array([[-0.00037858, -0.00050154],
       [ 0.00048535,  0.00040419]]), 'db1': array([[ 0.0007436 ],
       [-0.00069747]])}
[[0. 0. 0. 1.]]
```

Realization of OR Gate

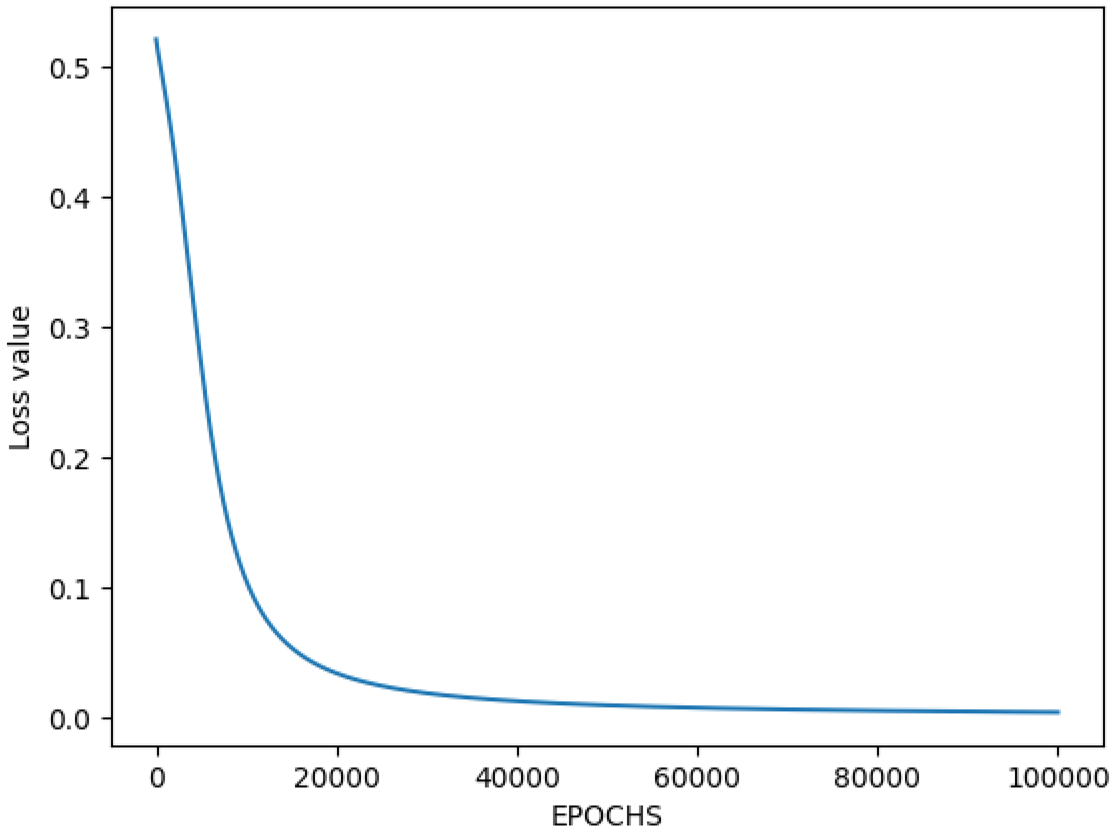
PROGRAM CODE

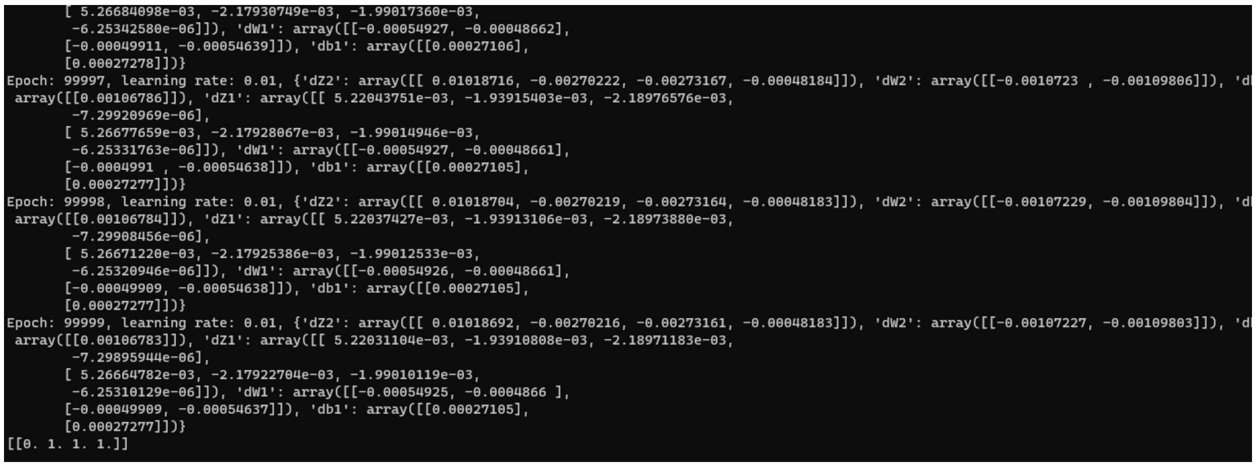
```
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns


def sigmoid(z):
    return 1/(1+np.exp(-z))


def initializeParameters(inputFeatures, neuronsInHiddenLayers, outputFeatures):
    W1 = np.random.randn(neuronsInHiddenLayers, inputFeatures)
    W2 = np.random.randn(outputFeatures, neuronsInHiddenLayers)
    b1 = np.zeros((neuronsInHiddenLayers, 1))
    b2 = np.zeros((outputFeatures, 1))

    parameters = {
        "W1":W1,
        "b1":b1,
        "W2":W2,
        "b2":b2
    }

    return parameters


def forwardPropagation(X, Y, parameters):
    m = X.shape[1]
    W1 = parameters["W1"]
    W2 = parameters["W2"]
    b1 = parameters["b1"]
    b2 = parameters["b2"]

    Z1 = np.dot(W1, X) + b1
    A1 = sigmoid(Z1)
    Z2 = np.dot(W2, A1) + b2
    A2 = sigmoid(Z2)

    cache = (Z1, A1, W1, b1, Z2, A2, W2, b2)
    logprobs = np.multiply(np.log(A2), Y) + np.multiply(np.log(1-A2), (1-Y))
    cost = -np.sum(logprobs) / m
    return cost, cache, A2

def backwardPropagation(X, Y, cache):
    m = X.shape[1]
    (Z1, A1, W1, b1, Z2, A2, W2, b2) = cache

    dZ2 = A2-Y
    dW2 = np.dot(dZ2, A1.T)/m
    db2 = np.sum(dZ2, axis=1, keepdims = True) / m

    dA1 = np.dot(W2.T, dZ2)
    dZ1 = np.multiply(dA1, A1*(1-A1))
    dW1 = np.dot(dZ1, X.T)/m
    db1 = np.sum(dZ1, axis = 1, keepdims=True) / m

    gradients = {
        "dZ2":dZ2,
        "dW2":dW2,
        "db2":db2,
        "dZ1":dZ1,
        "dW1":dW1,
        "db1":db1
    }

    return gradients

def updateParameters(parameters, gradients, learningRate):
    parameters["W1"] = parameters["W1"]-learningRate * gradients["dW1"]
    parameters["W2"]=parameters["W2"]-learningRate*gradients["dW2"]
    parameters["b1"] = parameters["b1"] - learningRate * gradients["db1"]
    parameters["b2"] = parameters["b2"] - learningRate * gradients["db2"]
    return parameters

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

```
neuronsInHiddenLayers = 2
inputFeatures = X.shape[0]
outputFeatures = Y.shape[0]
parameters = initializeParameters(inputFeatures, neuronsInHiddenLayers, outputFeatures)
epoch = 100000
learningRate = 0.01
losses = np.zeros((epoch, 1))

for i in range(epoch):
    losses[i,0], cache, A2 = forwardPropagation(X, Y, parameters)
    gradients = backwardPropagation(X, Y, cache)
    parameters = updateParameters(parameters, gradients, learningRate)
    print(f"Epoch: {i}, learning rate: {learningRate}, {gradients}")

plt.figure()
plt.plot(losses)
plt.xlabel("EPOCHS")
plt.ylabel("Loss value")
plt.show()

X = np.array([[0, 0, 1, 1], [0, 1, 0, 1]])
cost, _, A2 = forwardPropagation(X, Y, parameters)
prediction = (A2 > 0.5) * 1.0
print(prediction)
```

OUTPUT

After training for 10,000 episodes, following results were obtained.





```
       [ 5.26684098e-03, -2.17930749e-03, -1.99017360e-03,
        -6.25342580e-06]]), 'dW1': array([[-0.00054927, -0.00048662],
       [-0.00049911, -0.00054639]]), 'db1': array([[0.00027106],
       [0.00027278]])}
Epoch: 99997, learning rate: 0.01, {'dZ2': array([[ 0.01018716, -0.00270222, -0.00273167, -0.00048184]]), 'dW2': array([[-0.0010723 , -0.00109806]]), 'd
 array([[0.00106786]]), 'dZ1': array([[ 5.22043751e-03, -1.93915403e-03, -2.18976576e-03,
        -7.29920969e-06],
       [ 5.26677659e-03, -2.17928067e-03, -1.99014946e-03,
        -6.25331763e-06]]), 'dW1': array([[-0.00054927, -0.00048661],
       [-0.0004991 , -0.00054638]]), 'db1': array([[0.00027105],
       [0.00027277]])}
Epoch: 99998, learning rate: 0.01, {'dZ2': array([[ 0.01018704, -0.00270219, -0.00273164, -0.00048183]]), 'dW2': array([[-0.00107229, -0.00109804]]), 'd
 array([[0.00106784]]), 'dZ1': array([[ 5.22037427e-03, -1.93913106e-03, -2.18973880e-03,
        -7.29908456e-06],
       [ 5.26671220e-03, -2.17925386e-03, -1.99012533e-03,
        -6.25320946e-06]]), 'dW1': array([[-0.00054926, -0.00048661],
       [-0.00049909, -0.00054638]]), 'db1': array([[0.00027105],
       [0.00027277]])}
Epoch: 99999, learning rate: 0.01, {'dZ2': array([[ 0.01018692, -0.00270216, -0.00273161, -0.00048183]]), 'dW2': array([[-0.00107227, -0.00109803]]), 'd
 array([[0.00106783]]), 'dZ1': array([[ 5.22031104e-03, -1.93910808e-03, -2.18971183e-03,
        -7.29895944e-06],
       [ 5.26664782e-03, -2.17922704e-03, -1.99010119e-03,
        -6.25310129e-06]]), 'dW1': array([[-0.00054925, -0.0004866 ],
       [-0.00049909, -0.00054637]]), 'db1': array([[0.00027105],
       [0.00027277]])}
[[0. 1. 1. 1.]]
```

deerwalk DWIT College

LAB 8

Backpropagation Learning

Submitted by:

Nabin Katwal

Class of 2022

Roll No: 824

Submitted to:

Deepak Bhatt

Artificial Intelligence

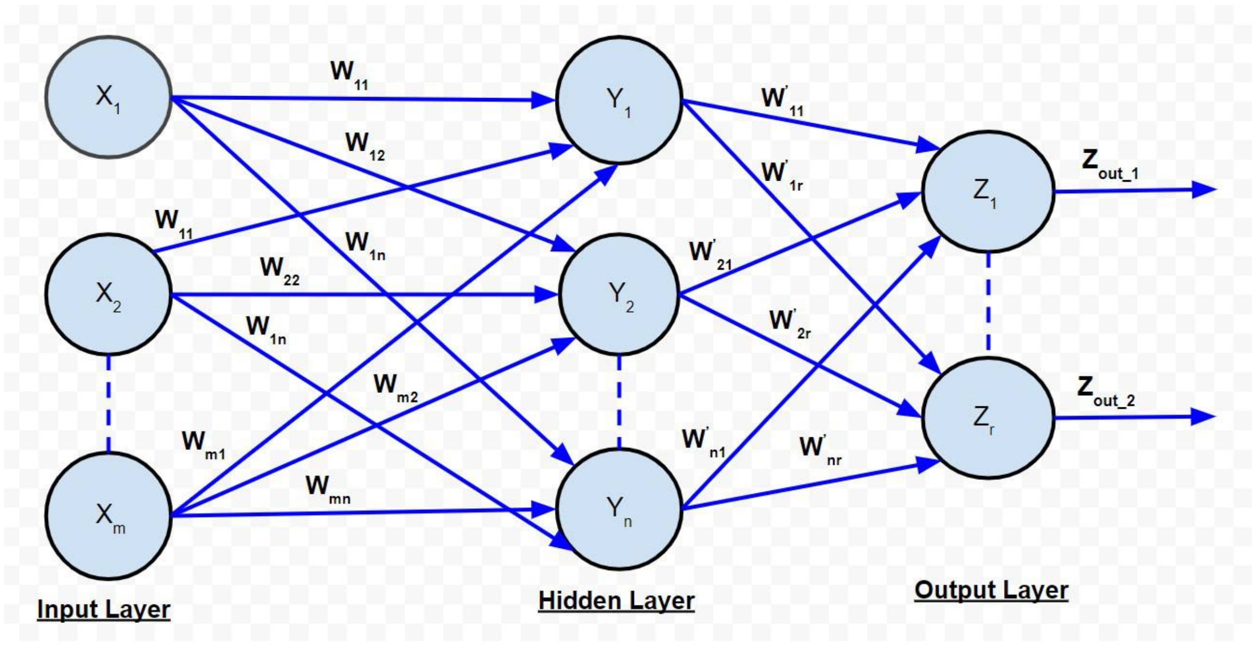
INTRODUCTION

Backpropagation (backward propagation) is an important mathematical tool for improving the accuracy of predictions in data mining and machine learning. Essentially, backpropagation is an algorithm used to calculate derivatives quickly.

Artificial neural networks use backpropagation as a learning algorithm to compute a gradient descent with respect to weights. Desired outputs are compared to achieve system outputs, and then the systems are tuned by adjusting connection weights to narrow the difference between the two as much as possible. The algorithm gets its name because the weights are updated backwards, from output towards input.

The difficulty of understanding exactly how changing weights and biases affects the overall behavior of an artificial neural network was one factor that held back wider application of neural network applications, arguably until the early 2000s when computers provided the necessary insight. Today, backpropagation algorithms have practical applications in many areas of artificial intelligence (AI), including optical character recognition (OCR), natural language processing (NLP) and image processing.

Because backpropagation requires a known, desired output for each input value in order to calculate the loss function gradient, it is usually classified as a type of supervised machine learning. Along with classifiers such as Naïve Bayesian filters and decision trees, the backpropagation algorithm has emerged as an important part of machine learning applications that involve predictive analytics.

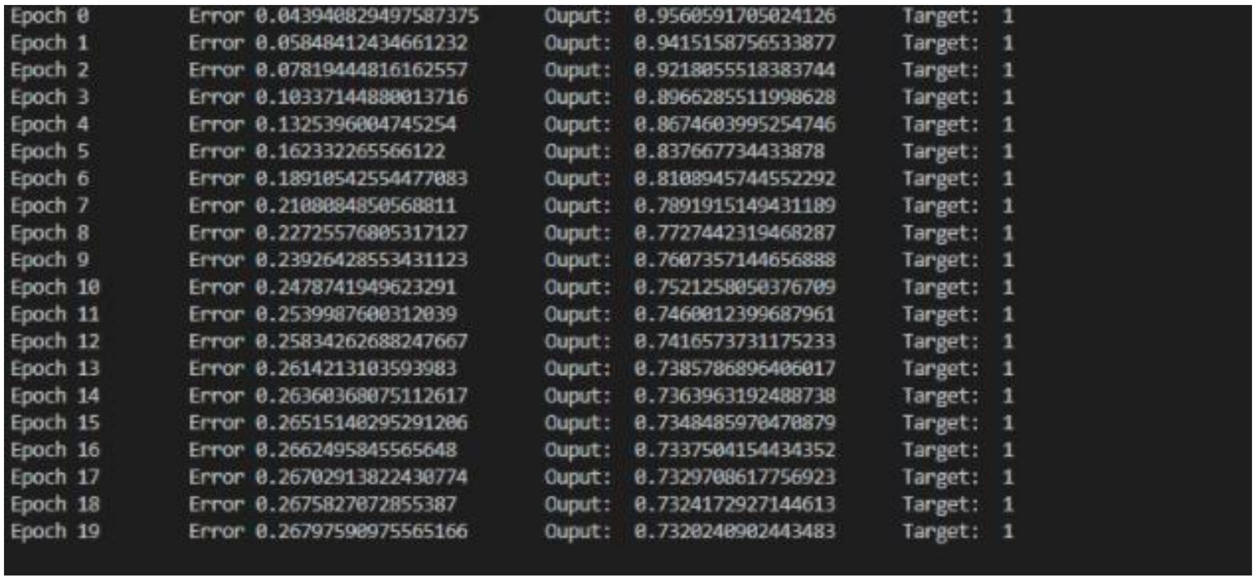


PROGRAM CODE

```
import numpy as np


class NeuralNetwork(object):
    def __init__(self, layers = [2,10,1], activations = ['sigmoid', 'sigmoid']):
        assert(len(layers) == len(activations)+1)
        self.layers = layers
        self.activations = activations
        self.weights = []
        self.biases = []
        for i in range(len(layers)-1):
            self.weights.append(np.random.randn(layers[i+1], layers[i]))
            self.biases.append(np.random.randn(layers[i+1],1))


    def feedforward(self, x):
        a = np.copy(x)
        z_s = []
        a_s = [a]
        for i in range(len(self.weights)):
            activation_function = self.getActivationFunction(self.activations[i])
            z_s.append(self.weights[i].dot(a)+self.biases[i])
            a = activation_function(z_s[-1])
            a_s.append(a)
        return (z_s, a_s)
```

```
@staticmethod
def getActivationFunction(name):
    if name == 'sigmoid':
        return lambda x:np.exp(x)/(1+np.exp(x))
    elif name == 'linear':
        return lambda x:x
    elif name == 'relu':
        def relu(x):
            y = np.copy(x)
            y[y<0] = 0
            return y
        return relu
    else:
        print('Unknown activation function, linear is used')
        return lambda x:x

@staticmethod
def getDerivitiveActivationFunction(name):
    if name == 'sigmoid':
        sig = lambda x: np.exp(x)/(1+np.exp(x))
        return lambda x:sig(x)*(1-sig(x))
    elif name == 'linear':
        return lambda x:1
    elif name == 'relu':
        def relu_diff(x):
            y = np.copy(x)
            y[y>=0]=1
            y[y<0] = 0
            return y
        return relu_diff
    else:
        print('Unknown activation function. linear is used')

def backpropagation(self, y, z_s,a_s):
    dw =[]
    db = []
    deltas = [None] * len(self.weights)
    deltas[-1] =((y-a_s[-1])*(self.getDerivitiveActivationFunction(self.activations[-1]))(z_s[-1]))
    for i in reversed(range(len(deltas)-1)):
        deltas[i] = self.weights[i+1].T.dot(deltas[i+1])*(self.getDerivitiveActivationFunction(self.activations[i])(z_s[i]))
        batch_size = y.shape[1]
        db = [d.dot(np.ones((batch_size,1)))/float(batch_size) for d in deltas]
        dw = [d.dot(a_s[i].T)/float(batch_size) for i,d in enumerate(deltas)]

def train(self, x, y, batch_size = 10, epochs=100, lr=0.01):
    for e in range(epochs):
        i=0
        while(i<len(y)):
            x_batch = x[i:i+batch_size]
            y_batch = y[i:i+batch_size]
            i = i+batch_size
            z_s, a_s = self.feedforward(x_batch)
            dw, db = self.backpropagation(y_batch, z_s, a_s)
            self.weights = [w+lr*dweight for w,dweight in zip(self.weights, dw)]
            self.biases = [w+lr*dbias for w,dbias in zip(self.biases, db) ]
            print("loss = {}".format(np.linalg.norm(a_s[-1]-y_batch)))



if __name__ == '__main__':
    import matplotlib.pyplot as plt
    nn = NeuralNetwork([1,100,1], activations=['sigmoid', 'sigmoid'])
    X = 2*np.pi*np.random.rand(1000).reshape(1,-1)
    y = np.sin(X)

    nn.train(X, y, epochs = 1000, batch_size = 64, lr=0.1)
    _,a_s = nn.feedforward(X)

    plt.scatter(X.flatten(), y.flatten())
    plt.scatter(X.flatten(), a_s[-1].flatten())
    plt.show()
```

Output



```
Epoch 0         Error 0.043940829497587375    Ouput:  0.9560591705024126     Target:  1
Epoch 1         Error 0.05848412434661232     Ouput:  0.9415158756533877     Target:  1
Epoch 2         Error 0.07819444816162557     Ouput:  0.9218055518383744     Target:  1
Epoch 3         Error 0.10337144880013716     Ouput:  0.8966285511998628     Target:  1
Epoch 4         Error 0.1325396004745254      Ouput:  0.8674603995254746     Target:  1
Epoch 5         Error 0.162332265566122       Ouput:  0.837667734433878      Target:  1
Epoch 6         Error 0.18910542554477083     Ouput:  0.8108945744552292     Target:  1
Epoch 7         Error 0.2108084850568811      Ouput:  0.7891915149431189     Target:  1
Epoch 8         Error 0.22725576805317127     Ouput:  0.7727442319468287     Target:  1
Epoch 9         Error 0.23926428553431123     Ouput:  0.7607357144656888     Target:  1
Epoch 10        Error 0.2478741949623291      Ouput:  0.7521258050376709     Target:  1
Epoch 11        Error 0.2539987600312039      Ouput:  0.7460012399687961     Target:  1
Epoch 12        Error 0.25834262688247667     Ouput:  0.7416573731175233     Target:  1
Epoch 13        Error 0.2614213103593983      Ouput:  0.7385786896406017     Target:  1
Epoch 14        Error 0.26360368075112617     Ouput:  0.7363963192488738     Target:  1
Epoch 15        Error 0.26515140295291206     Ouput:  0.7348485970470879     Target:  1
Epoch 16        Error 0.2662495845565648      Ouput:  0.7337504154434352     Target:  1
Epoch 17        Error 0.26702913822430774     Ouput:  0.7329708617756923     Target:  1
Epoch 18        Error 0.2675827072855387      Ouput:  0.7324172927144613     Target:  1
Epoch 19        Error 0.26797590975565166     Ouput:  0.7320240902443483     Target:  1
```

deerwalk
DWIT College

LAB 10

NLTK(Splitting Words from a sentence)

Submitted by:

Nabin Katwal

Class of 2022

Roll No: 824

Submitted to:

Deepak Bhatt

Artificial Intelligence

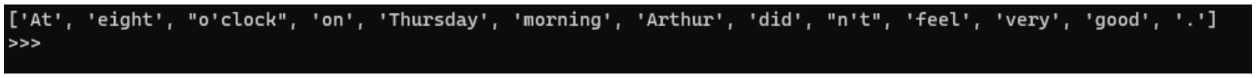
INTRODUCTION

NLTK is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-strength NLP libraries

PROGRAM CODE

```
import nltk


sentence = "At eight o'clock on Thursday morning Arthur didn't feel very good"

tokens = nltk.word_tokenize(sentence)

print(tokens)

tagged = nltk.pos_tag(tokens)

print(tagged)
```

OUTPUT



```
['At', 'eight', "o'clock", 'on', 'Thursday', 'morning', 'Arthur', 'did', "n't", 'feel', 'very', 'good', '.']
>>>
```

References:

1. https://www.nltk.org/
2. https://searchenterpriseai.techtarget.com/definition/backpropagation-algorithm