21BCP094
OM M PATEL
G3
D2
AI LAB

1. WAP to implement DFS and BFS for traversing a graph from source node (S) to goal node (G), where source node and goal node is given by the user as an input.

Code:
(Implementation of 8-Puzzle Problem using BFS and DFS)

```
goal_state = [1,2,3,4,5,6,7,8,0]

def generate_state(state,m,b):
    temp = state.copy()
    if m=='d':
        temp[b+3],temp[b] = temp[b],temp[b+3]
    if m=='u':
        temp[b-3],temp[b] = temp[b],temp[b-3]
    if m=='l':
        temp[b-1],temp[b] = temp[b],temp[b-1]
    if m=='r':
        temp[b+1],temp[b] = temp[b],temp[b+1]
    return temp

def get_moves(state,visited):
    idx = state.index(0)
    possible_moves = []
    if idx not in [0,1,2]:
        possible_moves.append('u')
    if idx not in [0,3,6]:
        possible_moves.append('l')
    if idx not in [6,7,8]:
        possible_moves.append('d')
    if idx not in [2,5,8]:
        possible_moves.append('r')

    moves = []
    for move in possible_moves:
        moves.append(generate_state(state,move,idx))

    return [m for m in moves if m not in visited]

def bfs(initial_state):
    print("Finding Goal State using Breadth First Search")
    print("Start State: ",initial_state)
    print("-"*45)
    queue = []
    queue.append(initial_state)
    visited = []
    count= 0

    while (len(queue)>0):
        count+=1
        cur_state = queue.pop(0)
        visited.append(cur_state)
        print(cur_state)
```

```
        if cur_state== goal_state:
            print("Successfully found the Goal State")
            print("No. of iterations: ",count)
            return
        moves = get_moves(cur_state,visited)

        for move in moves:
            if move not in queue:
                queue.append(move)

def dfs(initial_state):
    print("Finding Goal State using Depth First Search")
    print("Start State: ",initial_state)
    print("-"*45)
    stack = []
    stack.append(initial_state)
    visited = []
    count=0

    while (len(stack)>0):
        count+=1
        cur_state = stack.pop(-1)
        visited.append(cur_state)
        print(cur_state)
        if cur_state== goal_state:
            print("Successfully found the Goal State")
            print("No. of iterations: ",count)
            return
        moves = get_moves(cur_state,visited)
        for move in moves:
            if move not in stack:
                stack.append(move)
if __name__ == "__main__":
    state = [1,2,3,4,5,0,6,7,8]
    bfs(state)
    dfs(state)
```

Output:

```
Finding Goal State using Breadth First Search
Start State:  [1, 2, 3, 4, 5, 6, 0, 7, 8]
---------------------------------------------
[1, 2, 3, 4, 5, 6, 0, 7, 8]
[1, 2, 3, 0, 5, 6, 4, 7, 8]
[1, 2, 3, 4, 5, 6, 7, 0, 8]
[0, 2, 3, 1, 5, 6, 4, 7, 8]
[1, 2, 3, 5, 0, 6, 4, 7, 8]
[1, 2, 3, 4, 0, 6, 7, 5, 8]
[1, 2, 3, 4, 5, 6, 7, 8, 0]
Successfully found the Goal State
No. of iterations:  7
Finding Goal State using Depth First Search
Start State:  [1, 2, 3, 4, 5, 6, 0, 7, 8]
---------------------------------------------
[1, 2, 3, 4, 5, 6, 0, 7, 8]
[1, 2, 3, 4, 5, 6, 7, 0, 8]
[1, 2, 3, 4, 5, 6, 7, 8, 0]
Successfully found the Goal State
No. of iterations:  3
```

2. Design water jug problem solver

You are given two jugs with m litres and a n litre capacity. Both the jugs are initially empty. The jugs don't have markings to allow measuring smaller quantities. You have to use the jugs to measure d litres of water where d is less than n.

Code:

```
def die_hard_problem(capacity_a, capacity_b, target):
def gcd(a, b):
while b != 0:
a, b = b, a % b
return a

def can_measure_water(jug1, jug2, target):
if jug1 == target or jug2 == target or jug1 + jug2 == target:
return True
if jug1 == 0 or jug2 == 0:
return False
if jug1 > jug2:
jug1, jug2 = jug2, jug1
if target % gcd(jug2, jug1) != 0:
return False
return target <= jug1 + jug2
return can_measure_water(capacity_a, capacity_b, target)

if __name__ == "__main__":
capacity_a = int(input("Enter the capacity of jug A: "))
capacity_b = int(input("Enter the capacity of jug B: "))
target = int(input("Enter the target amount of water: "))

if die_hard_problem(capacity_a, capacity_b, target):
print("Yes, it is possible to measure the target amount of water.")
else:
print("No, it is not possible to measure the target amount of water.")
```

Output:

```
Enter the capacity of jug A: 3
Enter the capacity of jug B: 5
Enter the target amount of water: 4
Yes, it is possible to measure the target amount of water.
```

3. Solve 8 puzzle problem using A* algorithm where initial state and Goal state will be given by the user

Code:

```
import networkx as nx
import matplotlib.pyplot as plt

# Create a graph
G = nx.Graph()

# Add nodes
num = int(input("Enter the number of nodes: "))
for i in range(num):
    G.add_node(i, heuristic=int(input(f"Enter the heuristic for node {i}: ")))
```

```
# Add edges
edges = int(input("Enter the number of edges: "))
for i in range(edges):
    a, b = map(int, input("Enter the edge (space-separated): ").split())
    weight = int(input("Enter the weight: "))
    G.add_edge(a, b, weight=weight)

# A star algorithm
def aStar(G, start, end):
    closedSet = set()
    openSet = {start}
    cameFrom = {}

    gScore = {node: float('inf') for node in G.nodes()}
    gScore[start] = 0

    fScore = {node: float('inf') for node in G.nodes()}
    fScore[start] = G.nodes[start]['heuristic']

    while openSet:
        current = min(openSet, key=lambda node: fScore[node])
        if current == end:
            path = [current]
            while current in cameFrom:
                current = cameFrom[current]
                path.append(current)
            return path[::-1]

        openSet.remove(current)
        closedSet.add(current)

        for neighbor in G.neighbors(current):
            if neighbor in closedSet:
                continue

            tentative_gScore = gScore[current] + G[current][neighbor]['weight']
            if tentative_gScore < gScore[neighbor]:
                cameFrom[neighbor] = current
                gScore[neighbor] = tentative_gScore
                fScore[neighbor] = gScore[neighbor] + G.nodes[neighbor]['heuristic']
                if neighbor not in openSet:
                    openSet.add(neighbor)

    return None

print("The graph is: ")
print(G.nodes())
print("Weights")
print(G.edges(data=True))

# Draw the graph with heuristic and weights
pos = nx.spring_layout(G)
nx.draw(G, pos, with_labels=True)
labels = nx.get_edge_attributes(G, 'weight')
heuristic = nx.get_node_attributes(G, 'heuristic')
nx.draw_networkx_edge_labels(G, pos, edge_labels=labels)
heuristic_pos = {k: [v[0], v[1] - 0.1] for k, v in pos.items()}  # Adjust heuristic label positions
```

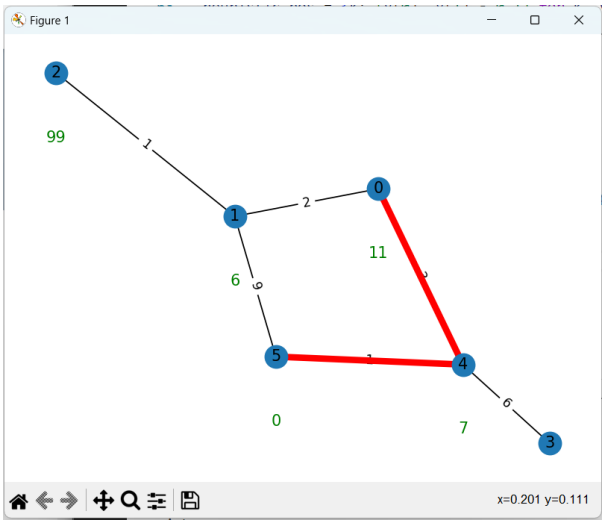
```
nx.draw_networkx_labels(G, heuristic_pos, font_color='g', labels=heuristic)

# Show the path
start_node = int(input("Enter start node: "))
end_node = int(input("Enter end node: "))
path = aStar(G, start_node, end_node)
print(f"The path is: {path}")
if path:
    nx.draw_networkx_edges(G, pos, edgelist=[(path[i], path[i + 1]) for i in range(len(path) - 1)], width=5, edge_color='r')

plt.show()
```

Output:

```
Enter the number of nodes: 6
Enter the heuristic for node 0: 11
Enter the heuristic for node 1: 6
Enter the heuristic for node 2: 99
Enter the heuristic for node 3: 1
Enter the heuristic for node 4: 7
Enter the heuristic for node 5: 0
Enter the number of edges: 6
Enter the edge (space-separated): 0 1
Enter the weight: 2
Enter the edge (space-separated): 1 2
Enter the weight: 1
Enter the edge (space-separated): 0 4
Enter the weight: 3
Enter the edge (space-separated): 3 4
Enter the weight: 6
Enter the edge (space-separated): 4 5
Enter the weight: 1
Enter the edge (space-separated): 1 5
Enter the weight: 9
The graph is:
[0, 1, 2, 3, 4, 5]
Weights
[(0, 1, {'weight': 2}), (0, 4, {'weight': 3}), (1, 2, {'weight': 1}), (1, 5, {'weight': 9}), (3, 4, {'weight': 6}), (4, 5, {'weight': 1})]
Enter start node: 0
Enter end node: 5
The path is: [0, 4, 5]
```



4. Implement the Fixed Increment Perceptron Learning algorithm as presented in the attachment.

Code:

```
import numpy as np
class FixedIncrementPerceptron:
def __init__(self, num_features, learning_rate=0.1):
self.num_features = num_features
self.learning_rate = learning_rate
self.weights = np.zeros(num_features + 1) # Add 1 for bias term

def predict(self, x):
x = np.insert(x, 0, 1) # Insert 1 for bias term
activation = np.dot(self.weights, x)
return 1 if activation >= 0 else 0

def train(self, X, y, max_epochs=1000):
for epoch in range(max_epochs):
error_count = 0
for i in range(len(X)):
prediction = self.predict(X[i])
error = y[i] - prediction
if error != 0:
error_count += 1
self.weights += self.learning_rate * error * np.insert(X[i], 0, 1) # Update weights
print(f"Epoch {epoch + 1}: Error Count = {error_count}")
if error_count == 0:
print(f"Converged after {epoch + 1} epochs.")
break
else:
print("Maximum epochs reached. Training did not converge.")


if __name__ == "__main__":

X = np.array([[0, 0],
[0, 1],
[1, 0],
[1, 1]])
y = np.array([0, 1, 1, 1]) # OR gate
perceptron = FixedIncrementPerceptron(num_features=X.shape[1])
perceptron.train(X, y)
print("Final weights:", perceptron.weights)
```

Output:

```
(base) tanmay@Tanmays-Air AI_Lab % /Users/tanmay/anaconda3/bin/python /Users/tanmay/Documents/AI_Lab/fpm.py
Epoch 1: Error Count = 2
Epoch 2: Error Count = 2
Epoch 3: Error Count = 1
Epoch 4: Error Count = 0
Converged after 4 epochs.
Final weights: [-0.1  0.1  0.1]
(base) tanmay@Tanmays-Air AI_Lab %
```

Q.5) Given a c++ code bnp, identify the algorithm implemented through the code. Also document the code

Code:

```
#include <math.h>
#include<iostream>
#include <stdlib.h>
#include <stdio.h>
#include <fstream>
#include <string.h>

using namespace std;

// GLOBAL FILE NAME DECLARATIONS
char file_name[9],file_name_inf[14],file_name_wgt[14],file_name_rst[14];
char file_name_out[14],file_name_dat[14];

class matrix
{
int row,col;
public:
float mat[15][15];
matrix() { row = 0; col = 0; }
void set(int,int);
int getrows()
{return row;}
int getcols()
{return col;}
void getdata();
FILE *fgetdata(FILE *);
void displaydata();
void displaydat();
FILE *fputdata(FILE *);
FILE *fputdat(FILE *);
matrix operator +(matrix);
matrix operator -();
matrix operator *(matrix);
matrix operator *(float);
};

void matrix :: set(int i,int j)
{
row=i;col=j;
}
FILE *matrix::fgetdata(FILE *fmat)
{
char line;
int i,j;
fscanf(fmat,"%d%d",&(row),&(col));
for(i=1; i <= row ; i++)
```

```
for(j=1; j <= col; j++)
fscanf(fmat,"%f",&(mat[i][j]));
return(fmat);
}
void matrix::getdata()
{
int i,j;
cout<<"Enter the size of the matrix:";
cin>>row>>col;
for(i=1;i<=row;i++)
for(j=1;j<=col;j++)
{
cout << "element [" << i << "] [ " << j << " ] ";
cin>>mat[i][j];
}
}

void matrix::displaydata()
{
int i,j;
// cout<<"The required matrix is :" << endl;
for(i=1;i<=row;i++,printf("\n\r"))
for(j=1;j<=col;j++,printf("\t")) printf("\t\t%10.2f",mat[i][j]);
}

void matrix::displaydat()
{
int i;
cout<<row;
}

FILE *matrix::fputdata(FILE *fmat)
{
int i,j;
// fprintf(fmat,"\n");
fprintf(fmat,"%d\n%d\n",row,col);
for(i=1;i<=row;i++)
for(j=1;j<=col;j++) fprintf(fmat,"%f\n",mat[i][j]);
return(fmat);
}

FILE *matrix::fputdat(FILE *fmat)
{
int i;
fprintf(fmat,"%d",row);
return(fmat);
}

matrix matrix :: operator + (matrix m)
{
matrix temp;
int i,j;
if((row==m.row)&&(col==m.col))
for(i=1;i<=row;i++)
for(j=1;j<=col;j++)
temp.mat[i][j]=mat[i][j]+m.mat[i][j];
else
```

```
{ cout<< "The addition of the matrices is not possible";exit(1);}
temp.row = row;
temp.col = col;
return(temp);
}


//Matrix Transposition
matrix matrix ::operator-()
{
matrix temp;
int i,j;temp.row=col;temp.col=row;
for(i=1;i<=col;i++)
for(j=1;j<=row;j++)
temp.mat[i][j]=mat[j][i];
return(temp);
}


//Matrix multiplication
matrix matrix :: operator*(matrix m)
{
matrix temp;
int i,j,k;
if(col==m.row)
{
for(i=1;i<=row;i++)
for(j=1;j<=m.col;j++)
{
temp.mat[i][j] = 0;
for(k=1;k<=col;k++)
temp.mat[i][j]=temp.mat[i][j]+(mat[i][k]*m.mat[k][j]);
}
}
else
{
cout<< "The multiplication of the matrices is not possible";
exit(1);
}
temp.row = row;
temp.col = m.col;
return(temp);
}
matrix matrix :: operator*(float svalue)
{
matrix temp;
int i,j;
for(i=1;i<=row;i++)
for(j=1;j<=col;j++)
temp.mat[i][j]=mat[i][j]*svalue;
temp.row = row;
temp.col = col;
return(temp);
}

class training
{
```

```
FILE *fin,*fout,*fwt;
matrix Input[5],Output[5], //array of vectors
Weights[5],dWeights[5],d,e,T;
float alpha, /* Momentum factor */
eta, /* Learning rate */
err, /* Error Constant */
theta, /* Threshold value */
lamda, /* Scaling parameter */
edata,
error;
int TotalLayers, // total # of layers
HiddenLayers, //total number of hidden layers
l[5], // NO of neurons in each layer
ntest,
iterates;
long filepos;
public :
training();
void readinputs();
void printing();
void initweights();
void initdweights();
void train();
void io_values();
void backpropagate();
void errors();
void chgweights();
void newweights();
~training();
};

training::training()
{
if ( (fin = fopen(file_name_dat,"r")) == NULL) exit(1);
if ( (fout=fopen(file_name_out,"w")) == NULL) exit(1);
if ( (fwt=fopen(file_name_wgt,"w") ) == NULL) exit(1);
/*if ( (fin = fopen("SOIL30.DAT","r")) == NULL) exit(1);
if ( (fout=fopen("SOIL30.OUT","w")) == NULL) exit(1);
if ( (fwt=fopen("SOIL30.WGT","w") ) == NULL) exit(1);*/
readinputs();
printing();
initweights();
initdweights();
train();
}

void training :: readinputs()
{
int i;error=0;
char line;
fscanf(fin,"%d",&HiddenLayers); /* get no. of hidden layers */
TotalLayers = HiddenLayers + 1 ; /* total number of layers */
for (i = 0; i <= TotalLayers; i++)
fscanf(fin,"%d",&l[i]);
fscanf(fin,"%f%f%f%f%f",&alpha,&err,&eta,&theta,&lamda);
fscanf(fin,"%d%d",&ntest,&iterates);
filepos=ftell(fin);
```

```
}

void training :: printing()
{
for (int i = 0; i <= TotalLayers; i++)
fprintf(fout,"\nNumber of Neurons in layer[%d]=%d",i+1,l[i]);
fprintf(fout,"\nAlpha value(Momentum factor): %f",alpha);
fprintf(fout,"\nError constant : %f",err);
fprintf(fout,"\nLearning rate : %f", eta);
fprintf(fout,"\nThreshold value : %f",theta);
fprintf(fout,"\nScaling Parameter: %f",lamda);
fprintf(fout,"\nNo of Training data : %d",ntest);
fprintf(fout,"\nMaximum Iteration : %d",iterates);
//
printf("\n\n\n");
for (int i = 0; i <= TotalLayers; i++)
printf("\n\t\tNumber of Neurons in layer[%d]=%d",i+1,l[i]);
printf("\n\n\t\tAlpha value(Momentum factor): %f",alpha);
printf("\n\t\tError constant : %f",err);
printf("\n\t\tLearning rate : %f", eta);
printf("\n\t\tThreshold value : %f",theta);
printf("\n\t\tScaling Parameter : %f",lamda);
printf("\n\t\tNo of Training data : %d",ntest);
printf("\n\t\tMaximum Iteration : %d",iterates);
;
}

void training :: initweights()
{
/* Initialize initial weights randomly */
// srand(2000); randomize();
for (int k=0; k < TotalLayers; k++)
{
Weights[k].set(l[k],l[k+1]);
for ( int i=1;i<=l[k];i++)
for(int j=1;j<=l[k+1];j++)
Weights[k].mat[i][j] =((float) rand() / 32767)- 0.5;
fprintf(fout,"\nWeights[%d]:",k);
Weights[k].fputdata(fout);
}
}

void training :: initdweights()
{
for (int k=0; k < TotalLayers; k++)
{
/* Initial differnce in weights is zero */
dWeights[k].set(l[k],l[k+1]);
for(int i=1;i<=l[k];i++)
for(int j=1;j<=l[k+1];j++)
dWeights[k].mat[i][j]=0.0;
}
}

void training :: train()
{
int k;float error1;
```

```
for(int jtr=1; jtr <= iterates; jtr++) //iterates
{
/* ITERATION LOOP */
error= 0.0;
fseek(fin,filepos,SEEK_SET);
cout<<"\nIteration Number: "<< jtr<<endl;
for (int itr=1; itr <= ntest; itr++)
{
/* NTEST LOOP */
Input[0].fgetdata(fin); /* Read Training Data */
T.fgetdata(fin); /* Read Desired Output */
cout<<"\rTraining Data Number: "<< itr;
io_values();
backpropagate();
errors();
chgweights();
newweights();
}
// ntest loop
fprintf(fout," %10.3E\n",error/ntest);
}
;
/* iterates loop */
for(k=0; k < TotalLayers; k++)
fwt=Weights[k].fputdata(fwt);
}
//////////////////////////TRAIN////////////////////////////////

void training :: io_values()
{
/* CALCULATION OF Input/OUT VALUES OF NEURONS */

Output[0] = Input[0];
for (int m=0; m <= TotalLayers-1; m++)
{
Input[m+1]=-Weights[m]*Output[m];
Output[m+1].set(l[m+1],1);
for (int i=1; i <= l[m+1]; i++)
Output[m+1].mat[i][1]=1.0/(1.0+ exp(-lamda*(Input[m+1].mat[i][1]+theta)));
}
}
void training :: backpropagate()
{
/* BACK PROPAGATION */
d.set(l[TotalLayers],1);
for (int i=1; i <= l[TotalLayers]; i++)
d.mat[i][1]=Output[TotalLayers].mat[i][1]*(1-Output[TotalLayers].mat[i][1])*(T.mat[i][1]-Output[TotalLayers].mat[i][1]);
dWeights[TotalLayers-1]=(dWeights[TotalLayers-1]*alpha)+((Output[TotalLayers-1]*-d)*eta);
}
void training :: chgweights()
{
/* CALCULATION OF CHANGE IN WEIGHTS */
int k;
for (int i=0; i <= TotalLayers-2; i ++)
{
k = TotalLayers - i -1;
e=Weights[k]*d;
```

```
d.set(l[k],1);
for (int j=1; j <= l[k]; j++)
{
d.mat[j][1] = Output[k].mat[j][1] * (1 - Output[k].mat[j][1]) * e.mat[j][1];
}
// dWeights[k-1]=(Output[k-1]*-d);
dWeights[k-1]=(dWeights[k-1]*alpha)+((Output[k-1]*-d)*eta);
}
}

void training :: newweights()
{
for(int k=0;k<TotalLayers;k++)
Weights[k]=Weights[k]+dWeights[k];
}

void training :: errors()
{
/* ERROR CALCULATION */
float sum=0.0,x,y1,y2;
for (int j=1; j <=l[TotalLayers]; j++)
{
y1 = T.mat[j][1];y2 = Output[TotalLayers].mat[j][1];
x = fabs(y1 - y2);
x = x*x;
sum= sum + x;
}
sum=sqrt(sum/l[TotalLayers]);
error = error + sum;
// printf(" Error=%10.3f",error);
cout<< "\t\t Error ="<< error;
}

training :: ~training()
{
fclose(fin);
fclose(fout);
fclose(fwt);
}
//////// INFERENCE ////////////

class inference
{
FILE *fin,*fout,*fwt;
matrix Input[5],Output[5], //array of vectors
Weights[5],T,CalculatedErr,NoOfTest;
float alpha, /* Momentum factor */
eta, /* Learning rate */
err, /* Error Constant */
theta, /* Threshold value */
x1,x2,
lamda, /* Scaling Parameter */
Calerror;
int TotalLayers, // total Number of layers
ntest, // Number of test data
l[10]; // Number of neurons in each layer
public :
```

```
inference();
void readinput();
void print();
void infer();
};


inference :: inference()
{
puts("Inference Session");
if ( (fin=fopen(file_name_inf,"r") ) == NULL) exit(1);
if ( (fout=fopen(file_name_rst,"w") ) == NULL) exit(1);
if ( (fwt=fopen(file_name_wgt,"r") ) == NULL) exit(1);
readinput();
print();
infer();
}
void inference :: readinput()
{
fscanf(fin,"%d",&TotalLayers); /* get no. of hidden layers */
TotalLayers = TotalLayers + 1 ; /* total number of layers */
for (int i = 0; i <= TotalLayers; i++)
fscanf(fin,"%d",&l[i]);
fscanf(fin,"%f%f%f%f%f",&alpha,&err,&eta,&theta,&lamda);
fscanf(fin,"%d",&ntest);
}
void inference :: print()
{
//
printf("\n");
for (int i = 0; i <= TotalLayers; i++)
printf("\n\t\tNumber of Neurons in layer[%d]=%d",i+1,l[i]);
printf("\n\n\t\tAlpha value(Momentum factor): %f",alpha);
printf("\n\t\tError constant : %f",err);
printf("\n\t\tLearning rate : %f", eta);
printf("\n\t\tThreshold value : %f",theta);
printf("\n\t\tScaling Parameter : %f",lamda);
printf("\n\t\tNo of Training data : %d",ntest);
;
}
void inference :: infer()
{
float Calerror=0.0;

for(int no=1; no<=ntest ;no++)
{
Input[0].fgetdata(fin);
//Input[0].fputdata(fout);
T.fgetdata(fin);
for(int i=0; i < TotalLayers; i++)
Weights[i].fgetdata(fwt);
Output[0] = Input[0];
for (int m=0; m <= TotalLayers-1; m++)
{
Input[m+1]=-Weights[m]*Output[m];
Output[m+1].set(l[m+1],1);
for(int i=1; i <= l[m+1]; i++)
```

```
Output[m+1].mat[i][1]= 1.0 / (1.0+ exp(-lamda*(Input[m+1].mat[i][1]+theta)));
}
for (int j=1; j <=l[TotalLayers]; j++)
{
x1 = T.mat[j][1];
x2 = Output[TotalLayers].mat[j][1];
Calerror = fabs(x1 - x2);
CalculatedErr.set(j,1);
CalculatedErr.mat[j][1] =Calerror;
}
printf("\n");
fprintf(fout,"\n\tINFERENCE TEST DATA NO :");
NoOfTest.set(no,1);
NoOfTest.fputdat(fout);
fprintf(stdout,"\n\t\tINFERENCE TEST DATA NO:"); NoOfTest.displaydat();
fprintf(fout,"\nCalculated Value:\n"); Output[TotalLayers].fputdata(fout);
fprintf(stdout,"\n\t\tCALCULATED VALUE:\n"); Output[TotalLayers].displaydata();
fprintf(fout,"\nActual Value:\n"); T.fputdata(fout);
fprintf(stdout,"\n\t\tACTUAL VALUE:\n"); T.displaydata();
fprintf(fout,"\nCALCULATED ERROR:\n");CalculatedErr.fputdata(fout);
fprintf(stdout,"\n\t\tCalculated Error:\n");CalculatedErr.displaydata();
printf("\t\tPress any Key to Continue ");
;
}

}
////////// MAIN PROGRAM //////////
int main(void)
{

int ch;
//
cout << endl << "Enter the file name : ";
cin >> file_name;
strcat(strcpy(file_name_dat,file_name),".dat");
strcat(strcpy(file_name_inf,file_name),".inf");
strcat(strcpy(file_name_wgt,file_name),".wgt");
strcat(strcpy(file_name_rst,file_name),".rst");
strcat(strcpy(file_name_out,file_name),".out");


do
{
//
cout << "\n\t\t\t1.Training";
cout << "\n\t\t\t2.Inference";
cout << "\n\t\t\t3.Exit";
cout << "\nEnter your choice : ";
cin >> ch;
if (ch==1)
{training t;
}
else
{ if (ch==2)
{inference i;}
else
exit(0);
```

```
}
}
while (ch!=3);
}
```

Output:

```
Enter  the  file  name  :  TRAIN

                      1.Training
                      2.Inference
                      3.Exit
Enter your choice : 1



               Number of Neurons in layer[1]=7
               Number of Neurons in layer[2]=10
               Number of Neurons in layer[3]=5

               Alpha value(Momentum factor): 0.900000
               Error constant              : 0.001000
               Learning rate               : 0.600000
               Threshold value             : 0.000000
               Scaling Parameter           : 1.000000
               No of Training data         : 32
               Maximum Iteration           : 500
Iteration Number: 1
Training Data Number: 32         Error = Error =22.0818
Iteration Number: 2
Training Data Number: 32         Error = Error =22.0818
Iteration Number: 3
Training Data Number: 32         Error = Error =22.0818
Iteration Number: 4
```

```
                      1.Training
                      2.Inference
                      3.Exit
Enter your choice : 2
Inference Session


               Number of Neurons in layer[1]=7
               Number of Neurons in layer[2]=10
               Number of Neurons in layer[3]=5

               Alpha value(Momentum factor): 0.900000
               Error constant              : 0.001000
               Learning rate               : 0.600000
               Threshold value             : 0.000000
               Scaling Parameter           : 1.000000
               No of Training data         : 16

               INFERENCE TEST DATA NO:1
               CALCULATED VALUE:
                     1.00
                     1.00
                     1.00
                     1.00
                     1.00

               ACTUAL VALUE:
                     0.24
                     0.60
                     0.09
                     0.42
                     0.17

               Calculated Error:
                     0.76
                     0.40
                     0.91
```

```
                Press any Key to Continue
                      1.Training
                      2.Inference
                      3.Exit
Enter your choice : 3
```

Documentation:

Understanding the Code:

1. Header Files and Namespace:
   - The code includes several standard C++ and C header files such as `<iostream>`, `<stdio.h>`, `<stdlib.h>`, etc.
   - It uses the `std` namespace.

2. Global File Name Declarations:
   - Arrays to hold file names like `file_name_dat`, `file_name_inf`, `file_name_wgt`, `file_name_rst`, and `file_name_out` are declared.

3. Matrix Class:
   - Represents a mathematical matrix.
   - Includes methods for initializing, getting data, displaying data, and performing matrix operations like addition, multiplication, and transposition.

4. Training Class:
   - Handles the training process.
   - Reads input data from files, initializes weights, performs training iterations, backpropagation, error calculation, weight adjustment, etc.

5. Inference Class:
   - Handles the inference process.
   - Reads input data and weights from files, calculates output values, and compares them with actual values to calculate errors.

6. Main Function:
  - Allows the user to choose between training and inference.
  - Initializes instances of the `training` or `inference` class based on user input.

Key Functionalities:
- Training:
  - Reads training data, desired outputs, and network parameters from files.
  - Initializes weights randomly.
  - Performs forward pass, error calculation, backpropagation, weight adjustment, and iteration over training data.
  - Updates weights based on calculated errors.

- Inference:
  - Reads test data and weights from files.
  - Performs forward pass to calculate output values.
  - Compares calculated output with actual values to compute errors.

User Interaction:
- The main function provides a simple console-based menu for the user to choose between training and inference or exit the program.

Suggestions for Improvement:
- Encapsulation: Consider encapsulating data members and methods within classes to improve modularity and encapsulation.
- Error Handling: Implement robust error handling mechanisms, especially when dealing with file I/O operations.
- Comments: Add more comments to improve code readability and maintainability.
- Code Structure: Organize the code into smaller functions or methods for better readability and maintainability.

Q.6) Understand the project available on following link
https://github.com/aharley/nn_vis

Now populate the table given below:

Layer	Task	Rationale
Input Layer	Receive Input Images	This layer receives the 28x28 pixel grayscale images of handwritten digits as input for classification.
Convolutional	Feature extraction	Convolutional layers apply filters to the input images to extract features such as edges and textures.
Pooling	Dimension reduction	Pooling layers reduce the dimensionality of the feature maps, making the subsequent computations faster.
Convolutional	Feature extraction	Additional convolutional layers extract higher-level features from the lower-level ones.
Pooling	Dimension reduction	Further dimensionality reduction to capture the most relevant features.
Fully Connected	Classification	Fully connected layers perform classification based on the extracted features.

Output Layer	Output prediction	The output layer predicts the probability distribution of the input image belonging to each digit class.

How does the following hyper-parameters affect the network performance

Hyper Parameter	**One Line Definition**	**Effects on CNN**
Stride	Step size of filter	Affects the spatial dimensions of the output feature maps.
Dilation Rate	Spacing between filter elements	Controls the receptive field and feature map size.
Type of pooling	Method of downsampling	Determines feature map reduction
Layer	(e.g., MaxPooling)	Reduces spatial dimensions, controls overfitting
Kernel size	Size of convolutional filter	Defines receptive field for each neuron
Padding	Addition of zeros at input boundaries	Controls output spatial dimensions, prevents reduction in feature map size

Q.12) WAP to design Tic Tac Toe games from O (Opponent) and X (Player) by using minimax algorithm.

Code:

```
import copy

# Function to print the Tic-Tac-Toe board
def print_board(board):
print("-------------")
for row in board:
print("| " + " | ".join(row) + " |")
print("-------------")

# Function to check if the game is over
def game_over(board):
# Check rows, columns, and diagonals for a win
for i in range(3):
if board[i][0] == board[i][1] == board[i][2] != " ":
return True
if board[0][i] == board[1][i] == board[2][i] != " ":
return True
if board[0][0] == board[1][1] == board[2][2] != " ":
return True
if board[0][2] == board[1][1] == board[2][0] != " ":
return True
# Check for a tie
if all([cell != " " for row in board for cell in row]):
return True
return False

# Function to evaluate the score of a board state
```

```
def evaluate(board):
# Check rows, columns, and diagonals for wins
for i in range(3):
if board[i][0] == board[i][1] == board[i][2] == "X":
return 1
if board[0][i] == board[1][i] == board[2][i] == "X":
return 1
if board[i][0] == board[i][1] == board[i][2] == "O":
return -1
if board[0][i] == board[1][i] == board[2][i] == "O":
return -1
if board[0][0] == board[1][1] == board[2][2] == "X":
return 1
if board[0][2] == board[1][1] == board[2][0] == "X":
return 1
if board[0][0] == board[1][1] == board[2][2] == "O":
return -1
if board[0][2] == board[1][1] == board[2][0] == "O":
return -1
return 0 # Return 0 for a tie

# Function for Minimax algorithm
def minimax(board, depth, maximizing_player):
if game_over(board) or depth == 0:
return evaluate(board)

if maximizing_player:
max_eval = float("-inf")
for i in range(3):
for j in range(3):
if board[i][j] == " ":
board[i][j] = "X"
eval = minimax(board, depth - 1, False)
board[i][j] = " "
max_eval = max(max_eval, eval)
return max_eval
else:
min_eval = float("inf")
for i in range(3):
for j in range(3):
if board[i][j] == " ":
board[i][j] = "O"
eval = minimax(board, depth - 1, True)
board[i][j] = " "
min_eval = min(min_eval, eval)
return min_eval

# Function to find the best move using Minimax
def find_best_move(board):
best_move = None
best_eval = float("-inf")
for i in range(3):
for j in range(3):
if board[i][j] == " ":
board[i][j] = "X"
eval = minimax(board, 5, False) # You can adjust the depth here for more or less difficulty
board[i][j] = " "
```

```
if eval > best_eval:
best_eval = eval
best_move = (i, j)
return best_move

# Function to play the game
def play_game():
board = [[" " for _ in range(3)] for _ in range(3)]
while not game_over(board):
print_board(board)
player_move = tuple(int(x) for x in input("Enter your move (row col): ").split())
if board[player_move[0]][player_move[1]] == " ":
board[player_move[0]][player_move[1]] = "O"
else:
print("Invalid move, try again.")
continue
print("Your move:")
print_board(board)
if game_over(board):
break
print("AI's move:")
ai_move = find_best_move(board)
board[ai_move[0]][ai_move[1]] = "X"
print_board(board)
if evaluate(board) == 1:
print("You lose!")
elif evaluate(board) == -1:
print("You win!")
else:
print("It's a tie!")

# Start the game
play_game()
```

Output:

```
-------------
|   |   |   |
-------------
|   |   |   |
-------------
|   |   |   |
-------------
Enter your move (row col): 1 1
Your move:
-------------
|   |   |   |
-------------
|   | O |   |
-------------
|   |   |   |
-------------
AI's move:
-------------
| X |   |   |
-------------
|   | O |   |
-------------
|   |   |   |
-------------
Enter your move (row col): 0 2
Your move:
-------------
| X |   | O |
-------------
|   | O |   |
-------------
|   |   |   |
-------------
AI's move:
-------------
| X |   | O |
-------------
|   | O |   |
-------------
| X |   |   |
-------------
Enter your move (row col): 1 0
Your move:
-------------
| X |   | O |
-------------
| O | O |   |
-------------
| X |   |   |
-------------
AI's move:
-------------
| X |   | O |
-------------
| O | O | X |
-------------
| X |   |   |
-------------
Enter your move (row col): 2 1
Your move:
-------------
| X |   | O |
-------------
| O | O | X |
-------------
| X | O |   |
-------------
AI's move:
-------------
| X | X | O |
-------------
| O | O | X |
-------------
| X | O |   |
-------------
Enter your move (row col): 2 2
Your move:
-------------
| X | X | O |
-------------
| O | O | X |
-------------
| X | O | O |
-------------
-------------
| X | X | O |
-------------
| O | O | X |
-------------
| X | O | O |
-------------
It's a tie!
```