**MAIN METHOD**

package school;

import java.io.IOException;

import java.util.\*;

import ptolemy.plot.\*;

public class main1

{//784,512,256,128,10 lr=0.0005 ||784,128,64,10 lr=0.0535 ||784,32,16,10 lr=0.1

int ary[]={784,32,16,10};int n=ary.length;Network ob1=new Network(n);int guessy;double input[][],expected[][];Random rgen=new Random();backprop ob3;

main1()

{

ob1.network(ary);

}

static int max(double ary[])

{

double m=ary[0];int pos=0;

for(int a=0;a<ary.length;a++)

{

if(m<ary[a])

{

pos=a;

m=ary[a];

}

}

return(pos);

}

void shuffle()

{

for (int i=0; i<input.length; i++) {

int randomPosition = rgen.nextInt(input.length);

double[] temp = input[i];

input[i] = input[randomPosition];

input[randomPosition] = temp;

double[] temp1 = expected[i];

expected[i] = expected[randomPosition];

expected[randomPosition] = temp1;

}

}

void main(boolean m1)throws IOException

{ output o=new output(); //gui gu=new gui();

int epoch=10;

double learningrate=0.025,beta=0.9;int iterations=42000,startacc=42000;String location[];double c=0;

GrayScale g=new GrayScale(iterations);

if(m1==false) {

g.train();}

input=g.pixelvalues;

expected=g.expected;location=g.s;

Plot plt=new Plot();

plt.setXLabel("epoch");

plt.setYLabel("Cost");

plt.setSize(400,400);

plt.setXRange(0, epoch);

double accuracy=0.0;

int a=0;

for(int a1=0;a1<epoch;a1++) {

shuffle();double sk=0;

for(int b=0;b<iterations;b++)

{

feedforward ob2=new feedforward(ob1.hidden);

ob2.inputs(input[b]);

ob2.main();

ob1.hidden=ob2.hidden;

ob3=new backprop(ob1.hidden,learningrate,expected[b],beta);

ob3.main();

ob1.hidden=ob3.hidden;

System.out.println((a1+1)+"\t"+b);

sk+=ob3.cost;

if(max(ob1.hidden[ob1.hidden.length-1].weightsum)==max(expected[b])&&b>startacc)

accuracy++;

}plt.addPoint(0,a1,sk/iterations, true);

}

plt.fillPlot();

PlotApplication app = new PlotApplication ( plt ) ;

g.test();

input=g.pixelvalues;location=g.s;

expected=g.expected;

input=g.pixelvalues;double accuracy1=0.0;

expected=g.expected;

shuffle();

for(int d=1;d<=600;d++){

feedforward ob2=new feedforward(ob1.hidden);

ob2.inputs(input[d-1]);

ob2.main();

System.out.println("Expected ");

ob1.hidden=ob2.hidden;

for(a=0;a<10;a++)

System.out.print(expected[d-1][a]+"\t");

System.out.println("predicted:");

for( a=0;a<10;a++)

System.out.print(ob1.hidden[ob1.hidden.length-1].weightsum[a]+"\t");

System.out.println();

System.out.println("prediction"+"\t"+max(ob1.hidden[ob1.hidden.length-1].weightsum));

if(max(ob1.hidden[ob1.hidden.length-1].weightsum)==max(expected[d-1]))

accuracy1++;

}

System.out.println();

System.out.println("accuracy test:"+(accuracy1/600)\*100);

System.out.println("accuracy train:"+(accuracy/(iterations-startacc)\*epoch)\*100);

}

void guess(double ary1[])

{

feedforward ob2=new feedforward(ob1.hidden);

ob2.inputs(ary1);

ob2.main();

System.out.println("prediction"+"\t"+max(ob1.hidden[ob1.hidden.length-1].weightsum));

}

}

**Formation of Network**

**package** school;

**import** java.util.\*;

**public** **class** Network

{

**double** weightsum[],weight[],bias,moment,expected[],error[],unactivated[],momentum[],gradient[]; **int** number,innovation[][];List<Double> nodegenes,connectgenes;

Network hidden[],link[];

**int** layers;

Network(**int** no)

{

layers=no;

hidden=**new** Network[layers];

}

Network(){}

**void** node(**int** sz)

{

weightsum=**new** **double**[sz];

unactivated=**new** **double**[sz];

}

**void** weights(**int** a,**int** ary[],**int** c)

{

link=**new** Network[ary[a]];

**for**(**int** a1=0;a1<ary[a];a1++)

link[a1]=**new** Network();

**if**(a<ary.length-1)

{

**if**(a+1<c)

**for**(**int** a1=0;a1<ary[a];a1++)

{

link[a1].momentum=**new** **double**[ary[a+1]];

link[a1].weight=**new** **double**[ary[a+1]];

link[a1].gradient=**new** **double**[ary[a+1]];

**if**(a>0)

link[a1].bias=Math.*random*()\*(2)-1;

}

**for**(**int** a1=0;a1<ary[a];a1++)

**for**(**int** a2=0;a2<ary[a+1];a2++)

link[a1].weight[a2]=Math.*random*()\*(2)-1;

}

**if**(a==ary.length-1)

{

**for**(**int** a1=0;a1<ary[a];a1++)

link[a1].bias=Math.*random*()\*(2)-1;

}

}

**void** number(Network a[])

{

number=1;

**for**(**int** a1=0;a1<a.length;a1++)

{

**for**(**int** b=0;b<a[a1].weightsum.length;b++)

{

a[a1].link[b].number=number++;

}

}

}

**void** links(Network a[],**int** d)

{

**for**(**int** b=0;b<a[d].weightsum.length;b++)

a[d].link[b].innovation=**new** **int**[a[d+1].weightsum.length][4];

**for**(**int** b=0;b<a[d].weightsum.length;b++)

{

**for**(**int** c=0;c<a[d+1].weightsum.length;c++)

a[d].link[b].innovation[c]=**new** **int**[]{d,b,d+1,c};

}

}

**void** network(**int** ary[])

{

**for**(**int** a=0;a<ary.length;a++)

{

hidden[a]=**new** Network();

hidden[a].node(ary[a]);

}

**for**(**int** a=0;a<ary.length;a++)

hidden[a].weights(a,ary,layers);

}

}

**Feedforward**

**package** school;

**import** java.util.\*;

**public** **class** feedforward

{

//int size;

Network hidden[];**double** weightedsum[][],bias[][],weights[][];

feedforward(Network main1[])

{

//size=n;

hidden=main1;

}

**double** sigmoid(**double** n)

{

**return**(1.0/(1+Math.*exp*(-n)));

}

**void** inputs(**double** ary[])

{**try** {

hidden[0].weightsum=ary;}

**catch**(Exception e) {}

}

**void** toArray(**int** a)

{

bias=**new** **double**[1][hidden[a+1].weightsum.length];

**for**(**int** a1=0;a1<hidden[a+1].weightsum.length;a1++)

bias[0][a1]=hidden[a+1].link[a1].bias;

weights=**new** **double**[hidden[a].weightsum.length][hidden[a+1].weightsum.length];

**for**(**int** a1=0;a1<hidden[a].weightsum.length;a1++)

**for**(**int** a2=0;a2<hidden[a+1].weightsum.length;a2++)

weights[a1][a2]=hidden[a].link[a1].weight[a2];

}

**double**[][] calculate(**double** inputs[])

{

**double** in[][]=**new** **double**[1][inputs.length];

**for**(**int** a=0;a<inputs.length;a++)

in[0][a]=inputs[a];

Matrix A=**new** Matrix(bias);

Matrix C=**new** Matrix(in);

Matrix B=**new** Matrix(weights);

Matrix D=A.plus(C.times(B));

**return** D.data;

}

**void** main()

{

**for**(**int** a=0;a<hidden.length-1;a++)

{

toArray(a);

weightedsum=calculate(hidden[a].weightsum);

**for**(**int** a1=0;a1<hidden[a+1].weightsum.length;a1++)

{

hidden[a+1].unactivated[a1]=weightedsum[0][a1];

hidden[a+1].weightsum[a1]=sigmoid(weightedsum[0][a1]);

}

}

}

}

**Backpropogation**

**package** school;

**import** java.util.\*;

**public** **class** backprop

//momentum added as optimizer

{

Network hidden[];**double** learningrate; **double** expects[];**double** beta;feedforward k;**double** x;**double** cost;

backprop(Network k[],**double** learn,**double** expected[],**double** beta)

{

beta=**this**.beta;

hidden=k;learningrate=learn;expects=expected;

cost=0;

}

**double** dsigmoid(**double** x)

{

**return**(x\*(1-x));

}

**void** expect()

{

hidden[hidden.length-1].expected=**new** **double**[expects.length];

hidden[hidden.length-1].expected=expects;

}

**void** error(**int** a)

{

hidden[a].error=**new** **double**[hidden[a].weightsum.length];

Matrix A=**new** Matrix(hidden[a].expected,hidden[a].weightsum.length,0);

Matrix B=**new** Matrix(hidden[a].weightsum,hidden[a].weightsum.length,0);

Matrix C=B.minus(A);

hidden[a].error=C.toSDA().data1;

}

**void** hiddenerror(**int** a)

{

hidden[a].error=**new** **double**[hidden[a].weightsum.length];**double** ary[]=**new** **double**[hidden[a+1].weightsum.length];

**for**(**int** a1=0;a1<hidden[a].weightsum.length;a1++)

{

**for**(**int** a2=0;a2<hidden[a+1].weightsum.length;a2++)

hidden[a].error[a1]+=hidden[a].link[a1].weight[a2]\*hidden[a+1].error[a2];

}

}

**void** changeweights(**int** a)

{

**for**(**int** a2=0;a2<hidden[a+1].weightsum.length;a2++)

{

**for**(**int** a1=0;a1<hidden[a].weightsum.length;a1++)

{ hidden[a].link[a1].gradient[a2]=learningrate\*hidden[a+1].error[a2]\*dsigmoid(hidden[a+1].weightsum[a2])\*hidden[a].weightsum[a1];

hidden[a].link[a1].momentum[a2]=beta\*hidden[a].link[a1].momentum[a2]-hidden[a].link[a1].gradient[a2];

hidden[a].link[a1].weight[a2]=hidden[a].link[a1].weight[a2]+hidden[a].link[a1].momentum[a2];

}

}

**for**(**int** a1=0;a1<hidden[a+1].weightsum.length;a1++)

{

**double** db=hidden[a+1].error[a1]\*dsigmoid(hidden[a+1].weightsum[a1]);

hidden[a+1].link[a1].moment=beta\*hidden[a+1].link[a1].moment-learningrate\*hidden[a+1].error[a1]\*dsigmoid(hidden[a+1].weightsum[a1]);

hidden[a+1].link[a1].bias=hidden[a+1].link[a1].bias+hidden[a+1].link[a1].moment;

}

}

**void** main()

{

expect();

error(hidden.length-1);

**for**(**int** a=hidden.length-2;a>0;a--)

hiddenerror(a);

**for**(**int** a=hidden.length-2;a>=0;a--)

changeweights(a);

**for**(**int** a=0;a<hidden[hidden.length-1].error.length;a++)

{cost+=Math.*pow*(hidden[hidden.length-1].error[a],2);

}

cost=cost/hidden[hidden.length-1].error.length;

System.***out***.print(cost+"\t");

}

}

**GUI**

package school;

import java.awt.EventQueue;

import java.awt.Graphics;

import java.awt.Graphics2D;

import java.util.\*;

import javax.swing.JFrame;

import javax.swing.JPanel;

import javax.imageio.ImageIO;

import javax.swing.JButton;

import java.awt.event.ActionListener;

import java.io.File;

import java.io.IOException;

import java.awt.event.ActionEvent;

import java.awt.Panel;

import java.awt.Rectangle;

import java.awt.Robot;

import java.awt.AWTException;

import java.awt.Color;

import java.awt.event.MouseMotionAdapter;

import java.awt.image.BufferedImage;

import java.awt.event.MouseEvent;

import javax.swing.JLabel;

import java.awt.Font;

import java.awt.event.MouseAdapter;

public class gui extends JPanel{

int x,y;BufferedImage img=new BufferedImage(112, 112, BufferedImage.TYPE\_INT\_ARGB);int d=0;

JFrame frame; JPanel panel;boolean c=false;Scanner in=new Scanner(System.in);

private JButton btnGuess;

/\*\*

\* Launch the application.

\*/

public static void main(String[] args) {

EventQueue.invokeLater(new Runnable() {

public void run() {

try {

gui window = new gui();

window.frame.setVisible(true);

} catch (Exception e) {

e.printStackTrace();

}

}

});

}

/\*\*

\* Create the application.

\*/

public gui() {

initialize();

img.createGraphics().setBackground(new Color(0,0,0,0));

img.createGraphics().dispose();

}

public void paintComponent(Graphics g,Graphics g1)

{

super.paintComponent(g1);

g.setColor(Color.WHITE);g1.setColor(Color.WHITE);

g.fillOval(x,y,9,9);g1.fillOval(x,y,10,10);

//g.dispose();

}

public BufferedImage createImage(JPanel panel2) {

int w = panel2.getWidth();

int h = panel2.getHeight();

BufferedImage bi = new BufferedImage(w, h, BufferedImage.TYPE\_INT\_ARGB);

Graphics2D g = bi.createGraphics();

panel2.paint(g);

return bi;

}

/\*\*

\* Initialize the contents of the frame.

\*/

private void initialize() {

main1 ob1=new main1();GrayScale ob=new GrayScale();

frame = new JFrame();

frame.setBounds(100, 100, 429, 273);

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

frame.getContentPane().setLayout(null);

panel = new JPanel();

panel.setBackground(Color.BLACK);

panel.setBounds(156, 27, 112, 112);

frame.getContentPane().add(panel);

panel.setVisible(true);

panel.addMouseMotionListener(new MouseMotionAdapter() {

public void mouseDragged(MouseEvent e) {

Graphics g=panel.getGraphics();

x=e.getX();y=e.getY();

paintComponent(img.createGraphics(),panel.getGraphics());

}

});

JButton btnNewButton = new JButton("Train");

btnNewButton.setBounds(10, 200, 120, 23);

frame.getContentPane().add(btnNewButton);

JButton btnClear = new JButton("Clear");

btnClear.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

img.createGraphics().setBackground(new Color(0,0,0,0));

img.createGraphics().clearRect(0,0,112,112);

panel.revalidate();

panel.repaint();

}

});

btnClear.setBounds(148, 200, 120, 23);

frame.getContentPane().add(btnClear);

btnGuess = new JButton("Guess");

btnGuess.setBounds(283, 200, 120, 23);

frame.getContentPane().add(btnGuess);

btnNewButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

try {

ob1.main(c);

c=true;

} catch (IOException e1) {

// TODO Auto-generated catch block

e1.printStackTrace();

}

}

});

btnGuess.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

BufferedImage image=output.main(img,28,28);

BufferedImage image1=new BufferedImage(

image.getWidth(),

image.getHeight(),

BufferedImage.TYPE\_BYTE\_BINARY);

Graphics2D graphic = image1.createGraphics();

graphic.drawImage(image, 0, 0, Color.BLACK, null);

graphic.dispose();

double []ary = ob.Pixelvalues(image1);

ob1.guess(ary);

}});}}

**Rescale images**

package school;

import javax.imageio.ImageIO;

import java.awt.\*;

import java.awt.image.BufferedImage;

import java.io.File;

import java.io.IOException;

class output{

BufferedImage main(String s,int height, int width) throws IOException {

File input = new File(s);

BufferedImage img = ImageIO.read(input);

Image tmp = img.getScaledInstance(width, height, Image.SCALE\_SMOOTH);

BufferedImage resized = new BufferedImage(width, height, BufferedImage.TYPE\_INT\_ARGB);

Graphics2D g2d = resized.createGraphics();

g2d.setRenderingHint(RenderingHints.KEY\_INTERPOLATION,RenderingHints.VALUE\_INTERPOLATION\_BILINEAR);

g2d.setRenderingHint(RenderingHints.KEY\_RENDERING,RenderingHints.VALUE\_RENDER\_QUALITY);

g2d.setRenderingHint(RenderingHints.KEY\_ANTIALIASING,RenderingHints.VALUE\_ANTIALIAS\_ON);

g2d.drawImage(tmp, 0, 0, null);

g2d.dispose();

return resized;

}

static BufferedImage main(BufferedImage img,int height, int width) {

Image tmp = img.getScaledInstance(width, height, Image.SCALE\_SMOOTH);

BufferedImage resized = new BufferedImage(width, height, BufferedImage.TYPE\_INT\_ARGB);

Graphics2D g2d = resized.createGraphics();

g2d.setRenderingHint(RenderingHints.KEY\_INTERPOLATION,RenderingHints.VALUE\_INTERPOLATION\_BILINEAR);

g2d.setRenderingHint(RenderingHints.KEY\_RENDERING,RenderingHints.VALUE\_RENDER\_QUALITY);

g2d.setRenderingHint(RenderingHints.KEY\_ANTIALIASING,RenderingHints.VALUE\_ANTIALIAS\_ON);

g2d.drawImage(tmp, 0, 0, null);

g2d.dispose();

return resized;

}

}

**Get pixel values of train and test**

package school;

import java.awt.\*;

import java.awt.image.\*;

import java.io.\*;

import java.util.Arrays;

import javax.imageio.ImageIO;

import javax.swing.JFrame;

public class GrayScale {

int noimages;

BufferedImage image;

int width;String testloc="E:\\St pauls 11&12\\images\\trainingSample\\trainingSample\\",trainloc="E:\\St pauls 11&12\\images\\trainingSet\\trainingSet\\";

int height; double pixelvalues[][];

double expected[][];File input;String[] s;

String[] location;

GrayScale(int n)

{

noimages=n;

}

GrayScale(){};

double[] Pixelvalues(BufferedImage img)

{ int k1=0;

width = img.getWidth();

height = img.getHeight();

double []pixelval=new double[width\*height];

for(int i=0; i<height; i++) {

for(int j=0; j<width; j++) {

Color c = new Color(img.getRGB(j, i));

int red = (int)(c.getRed());

int green = (int)(c.getGreen());

int blue = (int)(c.getBlue());

pixelval[k1++]=(double)((red+blue+green)/3)/255.0;

}

}

return(pixelval);

}

void test() {

try {

pixelvalues=new double[600][784];

expected=new double[600][10];s=new String[600];

int k2=0;

for(int a=0;a<=9;a++)

{

File input1 = new File(testloc+Integer.toString(a));

String file[]=input1.list();

for(int b=0;b<file.length;b++)

{

for(int c=0;c<10;c++)

{

if(c!=a)

expected[k2][c]=0;

else

expected[k2][c]=1;

}

int k1=0;s[k2]=testloc+Integer.toString(a)+"\\"+file[b];

input = new File(s[k2]);

image = ImageIO.read(input);

pixelvalues[k2]=Pixelvalues(image);

k2++;

}

}

}

catch (Exception e) {}

}

void train() {

double per=(double)noimages/42000.0;

try {

pixelvalues=new double[noimages][784];

expected=new double[noimages][10];s=new String[noimages];

int k2=0;

for(int a=0;a<=9;a++)

{

File input1 = new File(trainloc+Integer.toString(a));

String file[]=input1.list();

for(int b=0;b<per\*file.length;b++)

{

for(int c=0;c<10;c++)

{

if(c!=a)

expected[k2][c]=0;

else

expected[k2][c]=1;

}

int k1=0;s[k2]=trainloc+Integer.toString(a)+"\\"+file[b];

input = new File(s[k2]);

System.out.println(s[k2]);

image = ImageIO.read(input);

pixelvalues[k2]=Pixelvalues(image);

k2++;

}

}

}

catch (Exception e) {}

}

}

**Matrix operations(taken from internet)**

**package** school;

**final** **public** **class** Matrix {

**int** M; // number of rows

**int** N; // number of columns

**double**[][] data;

**double** [] data1;// M-by-N array

// create M-by-N matrix of 0's

**public** Matrix(**int** M, **int** N) {

**this**.M = M;

**this**.N = N;

data = **new** **double**[**this**.M][**this**.N];

}

**public** Matrix(**int** M) {

**this**.M = M;

**this**.N = 0;

data1 = **new** **double**[M];

}

**public** Matrix (**double**[] data12,**int** rows,**int** col) {

**int** c=0;

**if**(rows==0&&col==data12.length)

{

M=1;N=col;

**this**.data=**new** **double**[1][col];

**for**(**int** a=0;a<data12.length;a++)

**this**.data[0][a]=data12[c++];

}

**else** **if**(col==0&&rows==data12.length)

{

M=rows;N=0;

**this**.data=**new** **double**[rows][1];

**for**(**int** a=0;a<data12.length;a++)

**this**.data[a][0]=data12[c++];

}

**else** **if**((rows)\*(col)==data12.length)

{

M=rows;N=col;

**this**.data = **new** **double**[rows][col];

**for** (**int** i = 0; i < col; i++)

**for** (**int** j = 0; j <rows; j++)

**this**.data[i][j] = data12[c++];

}

**else**

**throw** **new** RuntimeException("size of sda and dda dont match");

}

// create matrix based on 2d array

**public** Matrix(**double**[][] data) {

M = data.length;

N = data[0].length;

**this**.data = **new** **double**[M][N];

**for** (**int** i = 0; i < M; i++)

**for** (**int** j = 0; j < N; j++)

**this**.data[i][j] = data[i][j];

}

// copy constructor

**private** Matrix(Matrix A) { **this**(A.data); }

// create and return a random M-by-N matrix with values between 0 and 1

**public** **static** Matrix random(**int** M, **int** N) {

Matrix A = **new** Matrix(M, N);

**for** (**int** i = 0; i < M; i++)

**for** (**int** j = 0; j < N; j++)

A.data[i][j] = Math.*random*();

**return** A;

}

// create and return the N-by-N identity matrix

**public** **static** Matrix identity(**int** N) {

Matrix I = **new** Matrix(N, N);

**for** (**int** i = 0; i < N; i++)

I.data[i][i] = 1;

**return** I;

}

// swap rows i and j

**private** **void** swap(**int** i, **int** j) {

**double**[] temp = data[i];

data[i] = data[j];

data[j] = temp;

}

// create and return the transpose of the invoking matrix

**public** Matrix transpose() {

Matrix A = **new** Matrix(N, M);

**for** (**int** i = 0; i < M; i++)

**for** (**int** j = 0; j < N; j++)

A.data[j][i] = **this**.data[i][j];

**return** A;

}

// return C = A + B

**public** Matrix plus(Matrix B) {

Matrix A = **this**;

**if** (B.M != A.M || B.N != A.N) **throw** **new** RuntimeException("Illegal matrix dimensions.");

**if**(M==0)M=1;

**if**(N==0)N=1;

Matrix C = **new** Matrix(M, N);

**for** (**int** i = 0; i < M; i++)

**for** (**int** j = 0; j < N; j++)

C.data[i][j] = A.data[i][j] + B.data[i][j];

**return** C;

}

// return C = A - B

**public** Matrix minus(Matrix B) {

Matrix A = **this**;

**if** (B.M != A.M || B.N != A.N) **throw** **new** RuntimeException("Illegal matrix dimensions.");

**if**(M==0)M=1;

**if**(N==0)N=1;

Matrix C = **new** Matrix(M,N);

**for** (**int** i = 0; i < M; i++)

**for** (**int** j = 0; j < N; j++)

C.data[i][j] = A.data[i][j] - B.data[i][j];

**return** C;

}

// does A = B exactly?

**public** **boolean** eq(Matrix B) {

Matrix A = **this**;

**if** (B.M != A.M || B.N != A.N) **throw** **new** RuntimeException("Illegal matrix dimensions.");

**for** (**int** i = 0; i < M; i++)

**for** (**int** j = 0; j < N; j++)

**if** (A.data[i][j] != B.data[i][j]) **return** **false**;

**return** **true**;

}

// return C = A \* B

**public** Matrix times(Matrix B) {

Matrix A = **this**;

**if** (A.N != B.M) **throw** **new** RuntimeException("Illegal matrix dimensions.");

Matrix C = **new** Matrix(A.M, B.N);

**for** (**int** i = 0; i < C.M; i++)

**for** (**int** j = 0; j < C.N; j++)

**for** (**int** k = 0; k < A.N; k++)

C.data[i][j] += (A.data[i][k] \* B.data[k][j]);

**return** C;

}

**public** Matrix toSDA()

{

**if**(M==1)

{

Matrix C=**new** Matrix(N);

**for**(**int** a=0;a<N;a++)

C.data1[a]=**this**.data[0][a];

**return** C;

}

**else** **if**(N==1)

{

Matrix C=**new** Matrix(M);

**for**(**int** a=0;a<M;a++)

C.data1[a]=**this**.data[a][0];

**return** C;

}

**else**

**return**(**null**);

}

// return x = A^-1 b, assuming A is square and has full rank

**public** Matrix solve(Matrix rhs) {

**if** (M != N || rhs.M != N || rhs.N != 1)

**throw** **new** RuntimeException("Illegal matrix dimensions.");

// create copies of the data

Matrix A = **new** Matrix(**this**);

Matrix b = **new** Matrix(rhs);

// Gaussian elimination with partial pivoting

**for** (**int** i = 0; i < N; i++) {

// find pivot row and swap

**int** max = i;

**for** (**int** j = i + 1; j < N; j++)

**if** (Math.*abs*(A.data[j][i]) > Math.*abs*(A.data[max][i]))

max = j;

A.swap(i, max);

b.swap(i, max);

// singular

**if** (A.data[i][i] == 0.0) **throw** **new** RuntimeException("Matrix is singular.");

// pivot within b

**for** (**int** j = i + 1; j < N; j++)

b.data[j][0] -= b.data[i][0] \* A.data[j][i] / A.data[i][i];

// pivot within A

**for** (**int** j = i + 1; j < N; j++) {

**double** m = A.data[j][i] / A.data[i][i];

**for** (**int** k = i+1; k < N; k++) {

A.data[j][k] -= A.data[i][k] \* m;

}

A.data[j][i] = 0.0;

}

}

// back substitution

Matrix x = **new** Matrix(N, 1);

**for** (**int** j = N - 1; j >= 0; j--) {

**double** t = 0.0;

**for** (**int** k = j + 1; k < N; k++)

t += A.data[j][k] \* x.data[k][0];

x.data[j][0] = (b.data[j][0] - t) / A.data[j][j];

}

**return** x;

}

// print matrix to standard output

**public** **void** show() {

**for** (**int** i = 0; i < M; i++) {

**for** (**int** j = 0; j < N; j++)

System.***out***.printf("%9.4f ", data[i][j]);

}

System.***out***.println();

}

}