**Nature Inspired Algorithms Neural Algorithms G.SHOBAB 2014064 G.SUPRAJA 2014063**

## java Implementation of PERCEPTRON

## This is an example of the Perceptron algorithm implemented in the Java Programming Language. The algorithm was implemented using an online learning method, meaning the weights are updated after each input pattern is observed. A step transfer function is used to convert the activation into a binary output € {0,1} . Random samples are taken from the domain to train the weights, and similarly, random samples are drawn from the domain to demonstrate what the network has learned. A bias weight is used for stability with a constant input of 1.0.

## Geometric interpretation of a perceptron:

## • input patterns ( x 1,... ,x n ) are points in n-dimensional space

## • points with w 0 + <w,x > = 0 are on a hyperplane defined by w 0 and w

## • points with w 0 + <w,x > >0 are above the hyperplane

## • points with w 0 + <w,x > < 0 are below the hyperplane

## • perceptrons partition the input space into two halfspaces along a hyperplane

## perceptron_linear.png\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## Perceptron learning problem:

## Given:

## • A set of input patterns P ⊆ R n, called the set of positive examples • Another set of input patterns N ⊆ R n, called the set of negative examples task:

## • Generate a perceptron that yields 1 for all patterns from P and 0 for all patterns from N ◮ obviously, there are cases in which the learning task is unsolvable, e.g. P ∩ N 6= ∅

## Program:-

## import java.util.\*;

## import java.lang.\*;

## import java.text.\*;

## class perceptron{

## public static void main(String args[]){

## int maxIterations=100;

## int maxSamples=100;

## int threshold=0;

## int iteration=0,output;

## double Error=0,localError=0,learnRate=0.1;

## neuron a[] = new neuron[100];

## for(int i=0;i<maxSamples;i++){

## a[i]=new neuron();

## }

## for(int i=0;i<maxSamples/2;i++){

## a[i].x=5+Math.random()\*5;

## a[i].y=4+Math.random()\*4;

## a[i].z=2+Math.random()\*7;

## a[i].output=1;

## }

## for(int i=50;i<maxSamples;i++){

## a[i].x=-1+Math.random()\*4;

## a[i].y=-4+Math.random()\*6;

## a[i].z=-3+Math.random()\*8;

## a[i].output=0;

## }

## double weights[] = new double[4];

## for(int i=0;i<4;i++){

## weights[i]=Math.random()\*1;

## System.out.print(round(weights[i])+" ");

## }

## do{

## Error = 0;

## for(int i=0;i<maxSamples;i++){

## output = calculateOutput(threshold,weights,a[i].x,a[i].y,a[i].z);

## localError = a[i].output - output;

## 

## weights[0]+= learnRate\*localError\*a[i].x;

## weights[1]+= learnRate\*localError\*a[i].y;

## weights[2]+= learnRate\*localError\*a[i].z;

## weights[3]+= learnRate\*localError;

## Error += Math.pow(localError,2);

## }

## System.out.println("Iteration : "+iteration+" ||| RMSE : "+round(Math.sqrt(Error/maxSamples)));

## iteration++;

## }while(iteration<=maxIterations&&Error!=0);

## 

## for(int i=0;i<10;i++){

## double x = -10+Math.random()\*20;

## double y = -10+Math.random()\*20;

## double z = -10+Math.random()\*20;

## 

## output=calculateOutput(threshold,weights,x,y,z);

## System.out.println("New random point ");

## System.out.println("x : "+round(x)+" y : "+round(y)+" z : "+round(z)+" output : "+output);

## System.out.println("");

## System.out.println("");

## }

## }

## 

## public static int calculateOutput(int threshold,double weights[],double x,double y,double z){

## double sum = x\*weights[0]+y\*weights[1]+z\*weights[2]+weights[3];

## if(sum>=threshold) return 1;

## return 0;

## }

## public static double round(double n){

## DecimalFormat df = new DecimalFormat("#.####");

## String s = df.format(n);

## return Double.parseDouble(s);

## }

## }

## class neuron{

## double x,y,z;

## int output;

## public neuron(){

## this.x=0;

## this.y=0;

## this.z=0;

## this.output=0;

## }

## }

**Output :-**

0.1249 0.4695 0.522 0.6614 Iteration : 0 ||| RMSE : 0.3162

Iteration : 1 ||| RMSE : 0.2646

Iteration : 2 ||| RMSE : 0.2449

Iteration : 3 ||| RMSE : 0.0

New random point

x : -9.5035 y : -2.5928 z : 5.02 output : 0

New random point

x : 5.9147 y : -5.4534 z : 0.6845 output : 0

New random point

x : -7.7494 y : -3.2196 z : -6.2221 output : 0

New random point

x : 1.6333 y : 8.1724 z : -0.5282 output : 1

New random point

x : -6.9537 y : -4.9738 z : -5.5119 output : 0

New random point

x : 0.4955 y : -5.0758 z : 2.922 output : 0

New random point

x : 8.5512 y : 7.0969 z : 0.1604 output : 1

New random point

x : 3.8219 y : -1.9045 z : 1.9768 output : 0

New random point

x : 2.4703 y : -7.0828 z : -4.8362 output : 0

New random point

x : -9.922 y : 5.8803 z : 8.4343 output : 1

**Analasis:-**

If the given problem is solvable, perceptron learning terminates after at most ( n + 1) 2 2 (n+1) log( n+1) iterations.

But in this implementation, we have set the maximum iterations to 100.