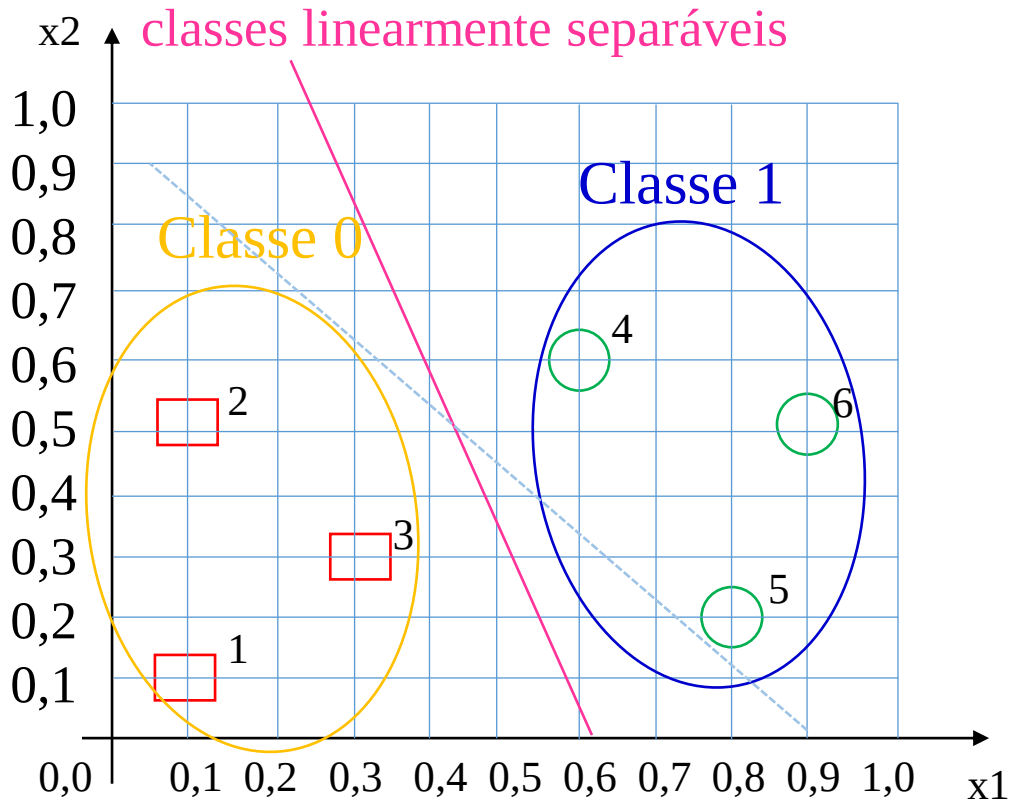
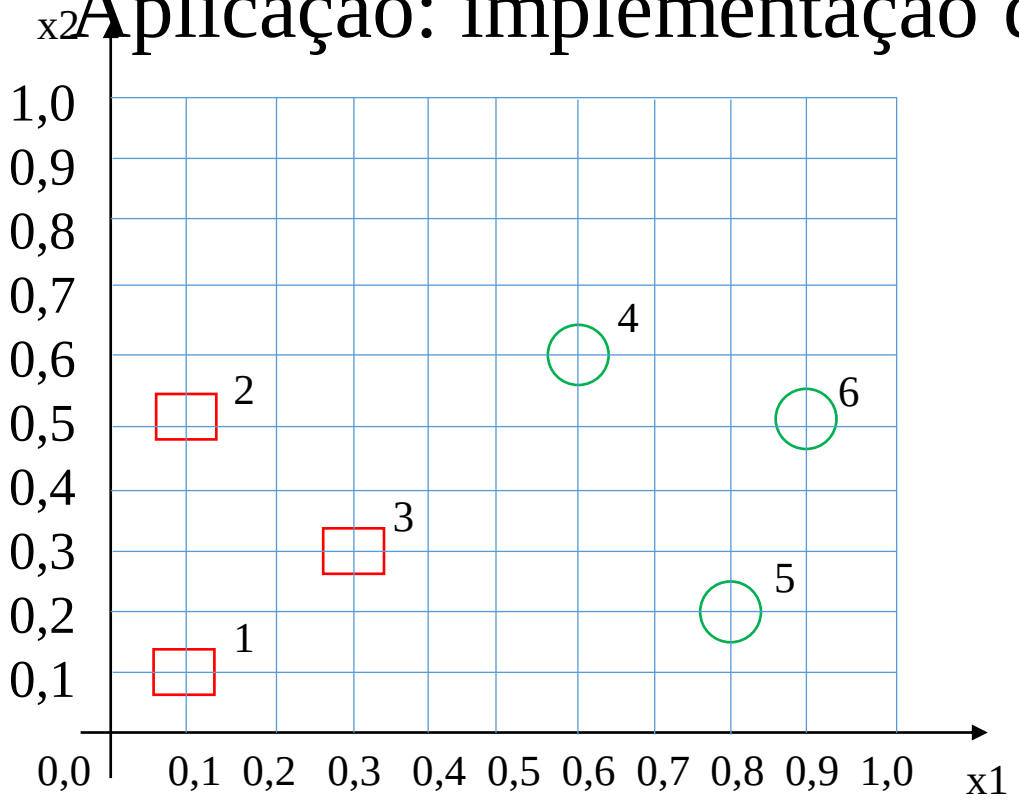
Redes Neurais Artificiais
Aplicação com *Perceptron*

profº Mauricio Conceição Mario

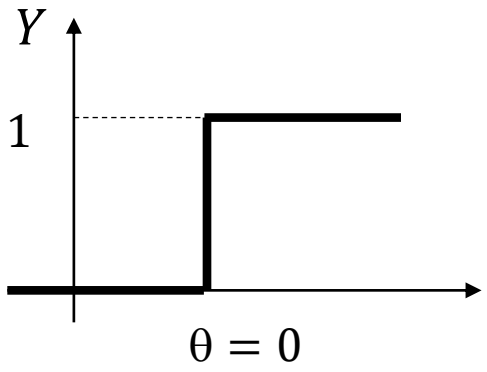
Aplicação: implementação de Perceptron para classificar 2 classes de números



Aplicação: implementação de Perceptron



elemento	x1	x2	
1	0,1	0,1	Classe 0
2	0,1	0,5	
3	0,3	0,3	
4	0,6	0,6	Classe 1
5	0,8	0,2	
6	0,9	0,5	



Função de ativação de limiar

Uso do Perceptron para classificar padrões ou classes

Classe 0

w_1 w_2

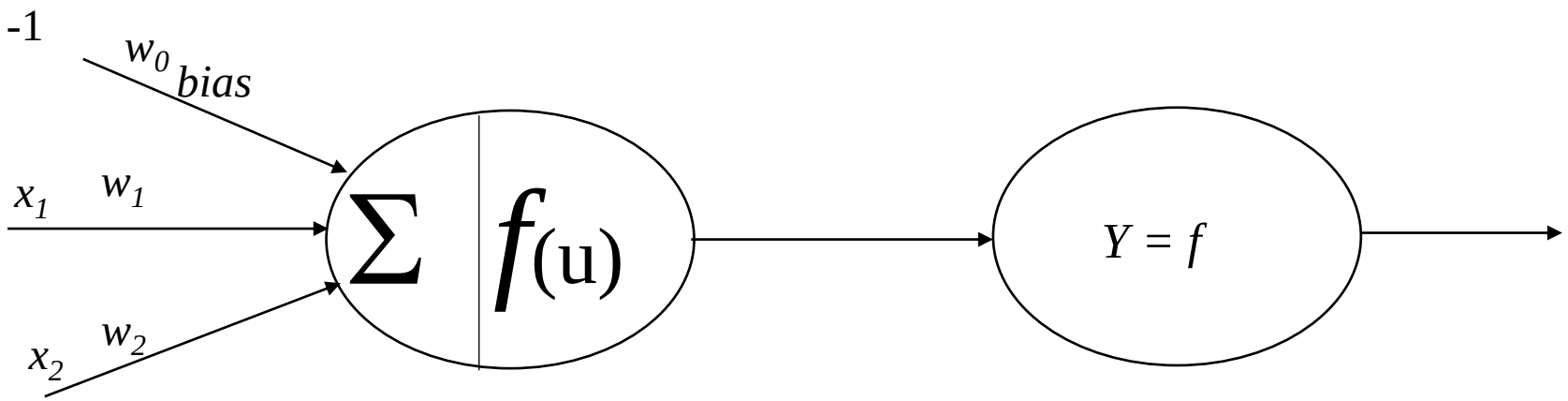
{0.1, 0.1},
{0.1, 0.5},
{0.3, 0.3}$\rightarrow Y = 0$

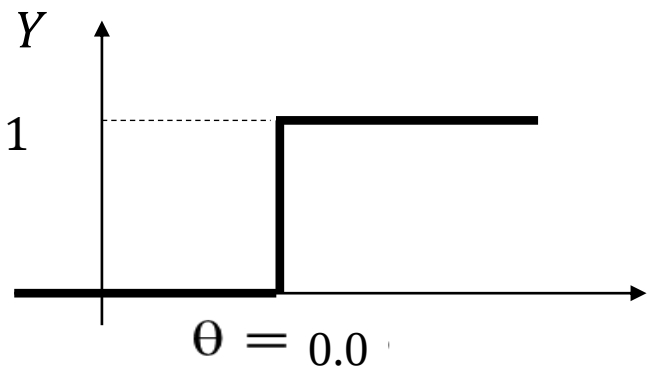
Classe 1

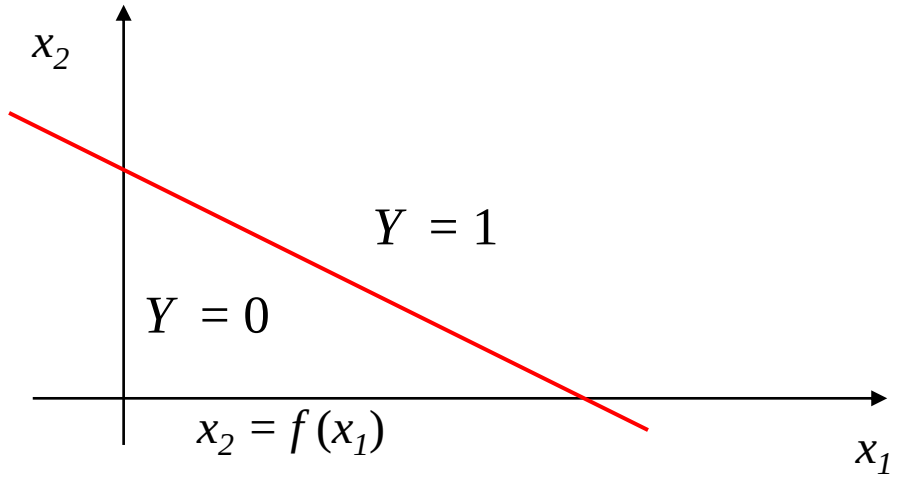
w_1 w_2

{0.6, 0.6},
{0.8, 0.2},
{0.9, 0.5}$\rightarrow Y = 1$

modelo da rede:







(Braga, 2007)

perceptron_III.java

```
12  public class perceptron_III {
13      double w []= {0.22, -0.33, 0.44};
14      double x1[][] = {
15       {-1, -1},
16       {0.1, 0.1},
17       {0.1, 0.5},
18       {0.3, 0.3}}; //  Classe 0
19   double x2[][] = {
20       {-1, -1},
21       {0.6, 0.6},
22       {0.8, 0.2},
23       {0.9, 0.5}}; //  Classe 1
24   double func_ativacao;
25      double limiar = 0.0;
26      double u0;
27      int f;  int y2 = 1; int y1 = 0;
28      double taxa_aprendizado = 0.1;
29
30  public void iteração_perceptron() {
31          int n = 0;
32          while (n < 10){
33
34              for (int v = 1; v < 4; v++){
35       w = entradax1(w, v);
36       w = entradax2(w, v);
37        }
38  n = n + 1;
39       System.out.println("número de treinamentos " + n + "\n");
40      }
41  }
42
43          public double[] entradax1(double[] w, int u){
```

```
    public double[] entradax1(double[] w, int u){
   System.out.println("entrada x1 ");

   u0 = 0;
        u0 = w[0]*x1[0][0] + w[1]*x1[u][0]+ w[2]*x1[u][1];
        System.out.println("u0 = " + u0 );

if (u0 > limiar)
   f = 1;
else f = 0; System.out.println("valor de f = \t" + f);

    w[0]= w[0]+ taxa_aprendizado*(y1-f)*x1[0][0];
    System.out.println("pesos w = \t" + w[0]);
    w[1]= w[1]+ taxa_aprendizado*(y1-f)*x1[u][0];
    System.out.println("pesos w = \t" + w[1]);
    w[2]= w[2]+ taxa_aprendizado*(y1-f)*x1[u][1];
    System.out.println("pesos w = \t" + w[2]);
  return w;}

   public double[] entradax2(double[] w, int u){
     System.out.println("entrada x2 ");

     u0 = 0;
          u0 = w[0]*x2[0][0] + w[1]*x2[u][0]+ w[2]*x2[u][1];
          System.out.println("u0 = " + u0 );

  if (u0 > limiar)
     f = 1;
  else f = 0;System.out.println("valor de f = \t" + f);

     w[0]= w[0]+ taxa_aprendizado*(y2-f)*x2[0][0];
     System.out.println("pesos w = \t" + w[0]);
     w[1]= w[1]+ taxa_aprendizado*(y2-f)*x2[u][0];
     System.out.println("pesos w = \t" + w[1]);
     w[2]= w[2]+ taxa_aprendizado*(y2-f)*x2[u][1];
     System.out.println("pesos w = \t" + w[2]);
   return w;}
```

```
 public void verifica_perceptron(double[] w, double [][]x1, double [][]x2 ){
         System.out.println("TESTE DA REDE NEURAL \n "  );
         this.w = w;
         for (int i = 0; i < w.length; i++){
           System.out.println("pesos resultante do treinamento  "  );
           System.out.println("w[" + i + "] = " + w[i] );
         }

         u0 = 0;
      for(int p = 1; p < 4; p++ ){
          u0 = w[0]*x1[0][0] + w[1]*x1[p][0]+ w[2]*x1[p][1];
          System.out.println("u0 = " + u0 );
 if (u0 > limiar)
    f = 1;
 else f = 0;
System.out.println("teste da entrada x1 saída y = " + f );
      }

        u0 = 0;
 for(int p = 1; p < 4; p++ ){
          u0 = w[0]*x2[0][0] + w[1]*x2[p][0]+ w[2]*x2[p][1];
          System.out.println("u0 = " + u0 );
 if (u0 > limiar)
    f = 1;
 else f = 0;
System.out.println("teste da entrada x2 saída y = " + f );
    }
  }
 }
```

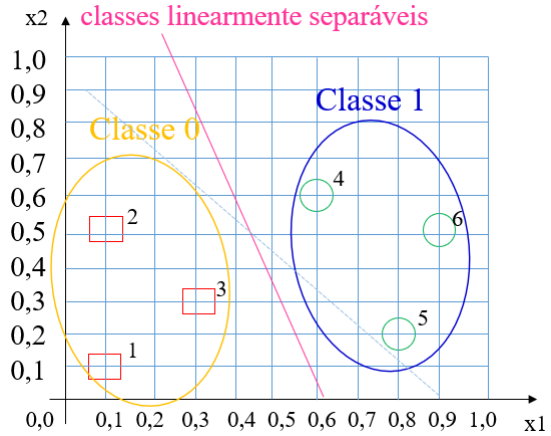
teste_perceptron_III.java | Saída - Redes_Neurais (run)

```
11  */
12
13
14  public class teste_perceptron_III {
15
16      public static void main(String args[]){
17
18       perceptron_III d = new perceptron_III ();
19
20       double x3[][] = {
21        {-1, -1},
22        {0.0, 0.0},
23        {0.15, 0.2},
24        {0.33, 0.1}}; //entradas x1
25       double x4[][] = {
26        {-1, -1},
27        {0.77, 0.55},
28        {0.8, 0.1},
29        {0.95, 0.75}};
30
31      d.iteração_perceptron();
32      d.verifica_perceptron(d.w, x3, x4);
33
34      }
35  }
```

entrada x1
u0 = -0.209
valor de f = 0
pesos w = 0.22
pesos w = -0.33
pesos w = 0.44
entrada x2
u0 = -0.15400000000000003
valor de f = 0
pesos w = 0.12
pesos w = -0.27
pesos w = 0.5
entrada x1
u0 = 0.10300000000000001
valor de f = 1
pesos w = 0.22
pesos w = -0.28
pesos w = 0.45
entrada x2
u0 = -0.35400000000000004
valor de f = 0
pesos w = 0.12
pesos w = -0.2
pesos w = 0.47000000000000003
entrada x1
u0 = -0.03899999999999998
valor de f = 0
pesos w = 0.12
pesos w = -0.2
pesos w = 0.47000000000000003
entrada x2
u0 = -0.06500000000000003
valor de f = 0
pesos w = 0.01999999999999999
pesos w = -0.11
pesos w = 0.52
número de treinamentos 1

...

entrada x1
u0 = -0.15899999999999995
valor de f = 0
pesos w = 0.22
pesos w = 0.2200000000000001
pesos w = 0.39000000000000024
entrada x2
u0 = 0.14600000000000019
valor de f = 1
pesos w = 0.22
pesos w = 0.2200000000000001
pesos w = 0.39000000000000024
entrada x1
u0 = -0.002999999999999864
valor de f = 0
pesos w = 0.22
pesos w = 0.2200000000000001
pesos w = 0.39000000000000024
entrada x2
u0 = 0.034000000000000155
valor de f = 1
pesos w = 0.22
pesos w = 0.2200000000000001
pesos w = 0.39000000000000024
entrada x1
u0 = -0.03699999999999991
valor de f = 0
pesos w = 0.22
pesos w = 0.2200000000000001
pesos w = 0.39000000000000024
entrada x2
u0 = 0.1730000000000002
valor de f = 1
pesos w = 0.22
pesos w = 0.2200000000000001
pesos w = 0.39000000000000024
número de treinamentos 10



TESTE DA REDE NEURAL

pesos resultante do treinamento
w[0] = 0.22
pesos resultante do treinamento
w[1] = 0.2200000000000001
pesos resultante do treinamento
w[2] = 0.39000000000000024
u0 = -0.22
teste da entrada x1 saída y = 0
u0 = -0.10899999999999994
teste da entrada x1 saída y = 0
u0 = -0.10839999999999994
teste da entrada x1 saída y = 0
u0 = 0.1639000000000002
teste da entrada x2 saída y = 1
u0 = -0.004999999999999873
teste da entrada x2 saída y = 0
u0 = 0.2815000000000003
teste da entrada x2 saída y = 1

testes
teste utilizando
a generalização

{0.0, 0.0},
{0.15, 0.2},
{0.33, 0.1}}

{0.77, 0.55},
{0.8, 0.1},
{0.95, 0.75}

Referências Bibliográficas

- Braga AP, Carvalho APLF, Ludermir TB. *Redes Neurais Artificiais: teoria e aplicações.* Livros Técnicos e Científicos, Rio de Janeiro – RJ; 2007.

- Haykin S. *Neural Networks – A Comprehensive Foundation.* Prentice-Hall; 1994.

- Haykin S. *Redes Neurais – Princípios e prática.* 2a ed.. Porto Alegre: Bookman; 2001.

- Hebb DO. *The Organization of Behavior.* John Wiley & Sons; 1949.

- Heckerman D. *Probabilistic Similarity Networks.* MIT Press, Cambridge, Massachussets; 1991.

- Hopfield JJ. *Neurons with graded response have collective computational properties like those of two-state neurons.* Proceedings of the National Academy of Sciences of the United States of America, 79, 2554-2558; 1982.

Referências Bibliográficas

- Mario MC. *Proposta de Aplicação das Redes Neurais Artificiais Paraconsistentes como Classificador de Sinais Utilizando Aproximação Funcional*. Univ. Federal de Uberlândia, Dissertação de Mestrado, Uberlândia; 2003.

- McCarthy J. *Programs with commom sense.* In Proceedings of the Symposium on Mechanisation of Thought Processes, Vol. 1, pp. 77-84, London. Her Majesty's Stationery Office; 1958.

- McCulloch W, Pitts W. *A logical calculus of the ideas immanent in nervous activity.* Bulletin of Mathematical Biophysics 5, 115-133; 1943.

- Rosenblatt F. *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms.* Spartan, Chicago; 1962.

- Rumelhart DE, McClelland JL. *Parallel Distributed Processing.* MIT Press, Cambridge, Massachussets; 1986.

- Turing A. *Computing machinery and intelligence.* Mind, 59, 433-460; 1950.