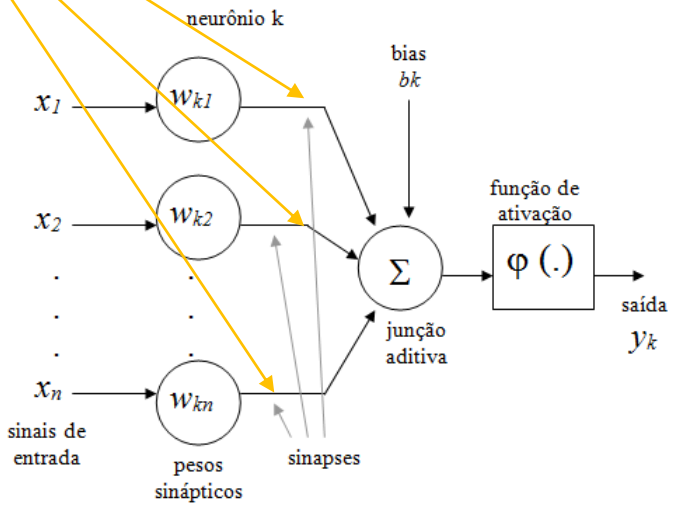
Redes Neurais Artificiais
Aprendizado e algoritmo *Perceptron*

profº Mauricio Conceição Mario

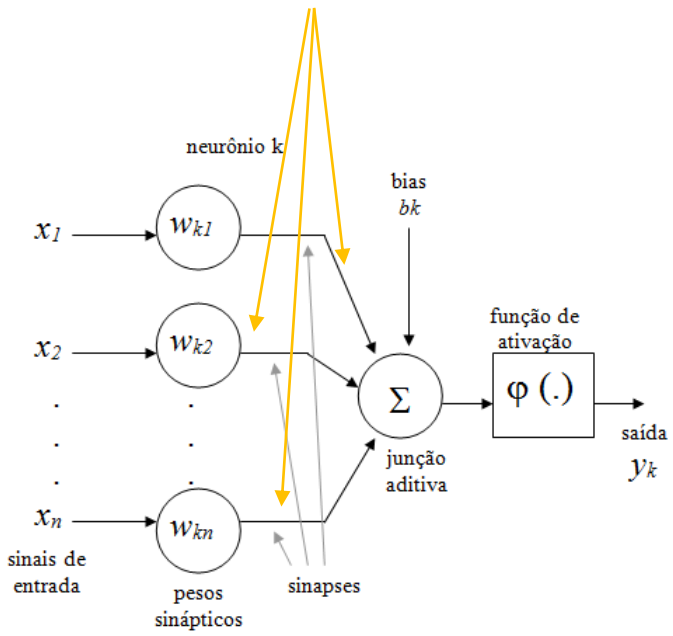
Redes Neurais Artificiais: Aprendizado

- A etapa de aprendizado de uma RNA consiste de um processo iterativo de ajuste de parâmetros da rede, os **pesos das conexões**, que guardam, ao final do processo, o conhecimento que a rede adquiriu do ambiente externo. (Braga, 2007)



Redes Neurais Artificiais: Aprendizado

- **pesos das conexões**



vetor de pesos **w**(t + 1) no instante t + 1;
w(t) = vetor de pesos no instante t;
Δ**w**(t) = ajuste aplicado aos pesos.

$$\mathbf{w}(t+1) = \mathbf{w}(t) + \Delta\mathbf{w}(t)$$

Os algoritmos de aprendizado diferem na forma como Δ**w**(t) é calculado.

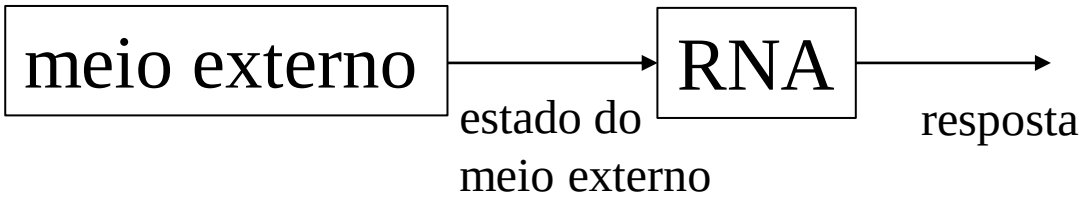
Há algoritmos diferentes para treinamentos de Redes Neurais, podendo os mesmos serem agrupados em dois paradigmas principais:

- *aprendizado supervisionado* e
- *aprendizado não-supervisionado*

(Braga, 2007)

Aprendizado não-supervisionado

- No aprendizado não-supervisionado não há a presença de um componente supervisor. Somente os padrões de entrada estão disponíveis para a rede. Durante o processo de aprendizado os padrões de entrada são apresentados continuamente à rede, e a existência de regularidades nesses dados faz com que o aprendizado seja possível. Se aplica a casos onde há regularidade e redundância nas entradas.



aprendizado não-supervisionado

O aprendizado não-supervisionado se aplica a problemas que visam à descoberta de características estatisticamente relevantes nos dados de entrada, como, por exemplo, a descoberta de agrupamentos, ou de classes.

(Braga, 2007)

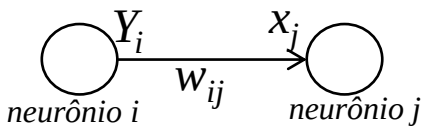
Aprendizado não-supervisionado → aprendizado *Hebbiano*

- A regra de aprendizado de ***Hebb*** (Hebb, 1949) propõe que o peso de uma conexão sináptica deve ser ajustado se houver sincronismo entre os "níveis de atividade" de entrada e saída. Se dois neurônios, em lados distintos de uma sinapse, são ativados sincronamente, tem-se um fortalecimento desta sinapse. Se estes tipos de neurônios citados são ativados assincronamente, a sinapse será enfraquecida ou mesmo eliminada.

$$\Delta w_{ij}(t) = \eta\, Y_i(t)\, x_j(t)$$

A interpretação do postulado de *Hebb* dentro do contexto das RNAs é que o ajuste dos pesos no tempo *t* deve ser proporcional ao produto dos valores de entrada e saída da rede.

$w_{ij}(t)$ = valor do peso *j* do neurônio *i*;
$x_j(t)$ = valor da entrada *j*;
η = constante positiva que determina a taxa de aprendizado;
$Y_i(t)$ = saída do neurônio *i*;



(Braga, 2007)

Aprendizado não-supervisionado → aprendizado por competição

- É um caso particular de aprendizado não-supervisionado. Dado um padrão de entrada, deve-se fazer com que as unidades de saída disputem entre si para serem ativadas. Existe, então, uma competição entre as unidades de saída para qual delas prevalecerá vencedora e, conseqüentemente será ativada. A unidade vencedora terá seus pesos atualizados no treinamento.

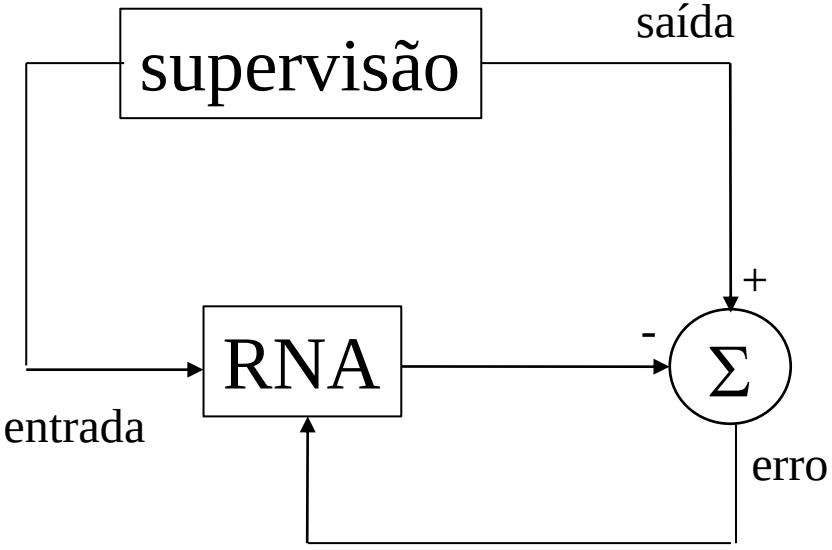
  As unidades de entrada são diretamente conectadas às unidades de saída, e estas últimas também podem estar ligadas entre si via conexões laterais inibitórias ou negativas.

  Este modelo, em que haverá um único neurônio vencedor, é conhecido como *Winner Takes All* (WTA).

(Braga, 2007)

Redes Neurais Artificiais: Aprendizado Supervisionado

- Um supervisor é responsável por estimular as entradas da rede por meio de padrões de entrada e observar a saída calculada pela mesma, comparando com a saída desejada. Como a resposta da rede é função dos valores atuais do seu conjunto de pesos, estes são ajustados de forma a aproximar a saída da rede da saída desejada.



(Braga, 2007)

O aprendizado supervisionado se aplica a problemas em que se deseja obter um mapeamento entre padrões de entrada e saída.

Os exemplos mais conhecidos de algoritmos para aprendizado são a *regra delta* e a sua generalização para redes de múltiplas camadas, o algoritmo *Backpropagation*.

Implementação ***off-line*** do aprendizado supervisionado: os dados do conjunto de treinamento não mudam, e, uma vez obtida uma solução para a rede, esta deve permanecer fixa.

Implementação ***on-line***: o conjunto de dados muda continuamente, e a rede deve estar em processo contínuo de adaptação.

Aprendizado Supervisionado: correção de erros

- No aprendizado supervisionado por correção de erros, procura-se minimizar o erro da resposta atual da rede em relação à saída desejada. A expressão genérica para o erro *e*(t) no instante de tempo *t* pode ser escrita como:

$$\boldsymbol{e}(t) = \boldsymbol{Y_d}(t) - \boldsymbol{Y}(t)$$

$\boldsymbol{w_i}$ (t) = peso de entrada *i*;

$\boldsymbol{x_i}$ (t) = entrada *i* do neurônio;

$\boldsymbol{\eta}$ = taxa de aprendizado;

$\boldsymbol{e}$(t) = medida do erro;

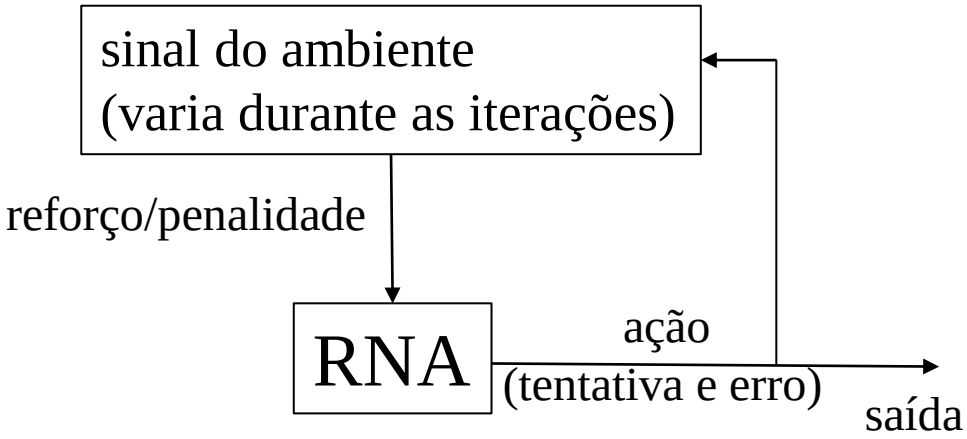
$$\boldsymbol{w_i}(t + 1) = \boldsymbol{w_i}(t) + \boldsymbol{\eta}\, \boldsymbol{e}(t)\, \boldsymbol{x_i}(t)$$

De acordo com a equação o ajuste de pesos deve ser proporcional ao produto do erro pelo valor da entrada da sinapse naquele instante de tempo.

(Braga, 2007)

Aprendizado Supervisionado → aprendizado por reforço

- O aprendizado por reforço pode ser considerado como um caso particular do aprendizado supervisionado. No aprendizado por reforço há a presença de um componente externo do ambiente, o ***crítico***, que procura maximizar o reforço das "boas" ações executadas pela rede. É um processo de tentativa e erro que visa maximizar o índice de desempenho escalar chamado de *sinal de reforço*.



aprendizado por reforço

O aprendizado por reforço se aplica principalmente a problemas de aprendizado envolvendo tarefas de controle nas quais é permitido à rede errar durante o processo de interação com o sistema a ser controlado.

(Braga, 2007)

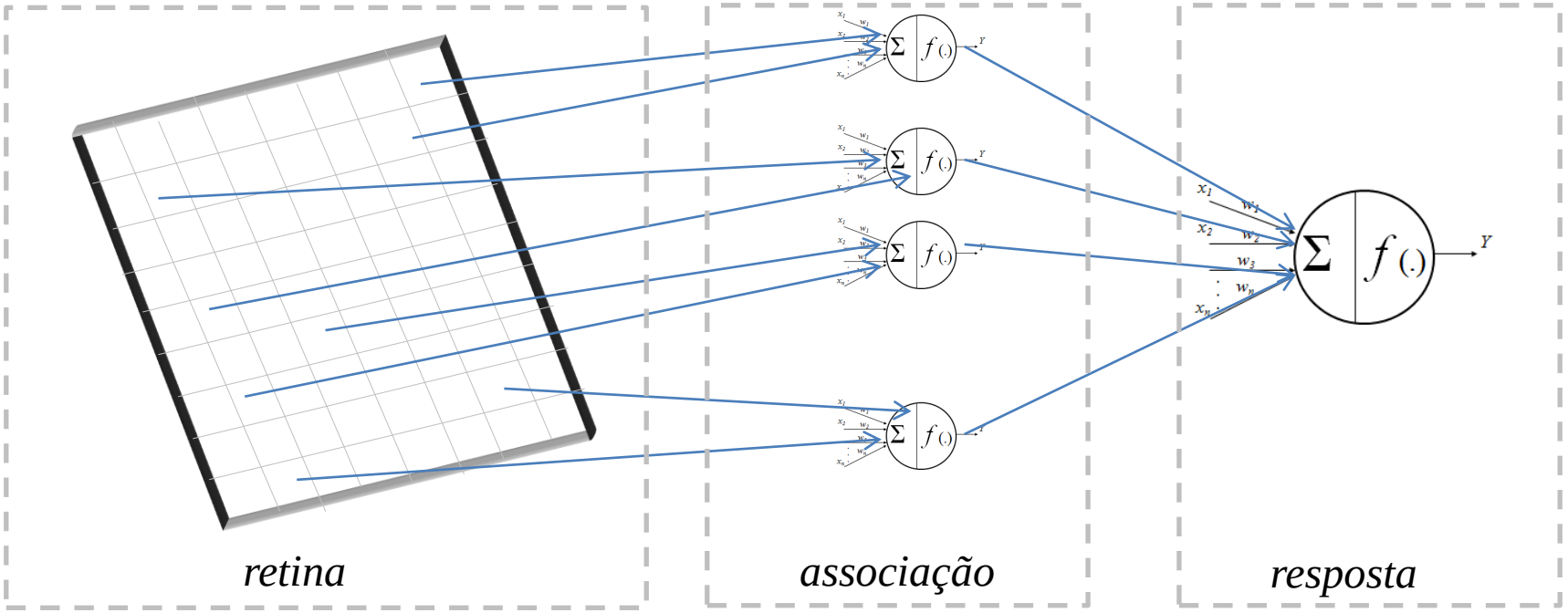
Tarefas que as RNAs podem desempenhar

tarefas	exemplos de aplicações
Classificação	- Reconhecimento de caracteres - Reconhecimento de imagens - Diagnóstico (médico, equipamentos, etc.) - Análise de risco de crédito - Detecção de fraudes - Detecção de falhas em sistemas industriais
Categorização	- Agrupamento de seqüências de DNA - Mineração de dados - Análise de expressão gênica - Agrupamentos de clientes
Previsão	- Previsão do tempo - Previsão financeira (câmbio, bolsa, etc.) - Modelagem de sistemas dinâmicos - Previsão de seqüências de DNA

(Braga, 2007)

O *Perceptron*

- Modelo proposto por Frank Rosenblatt composto por neurônios MCP e regra de aprendizado (Rosenblatt, 1962). A topologia original proposta tem unidades de entrada que formam a *retina*, um nível intermediário formado pelas *unidades de associação* e por um nível de saída formado pelas *unidades de resposta*.



(Braga, 2007)

Perceptron: algoritmo de aprendizado

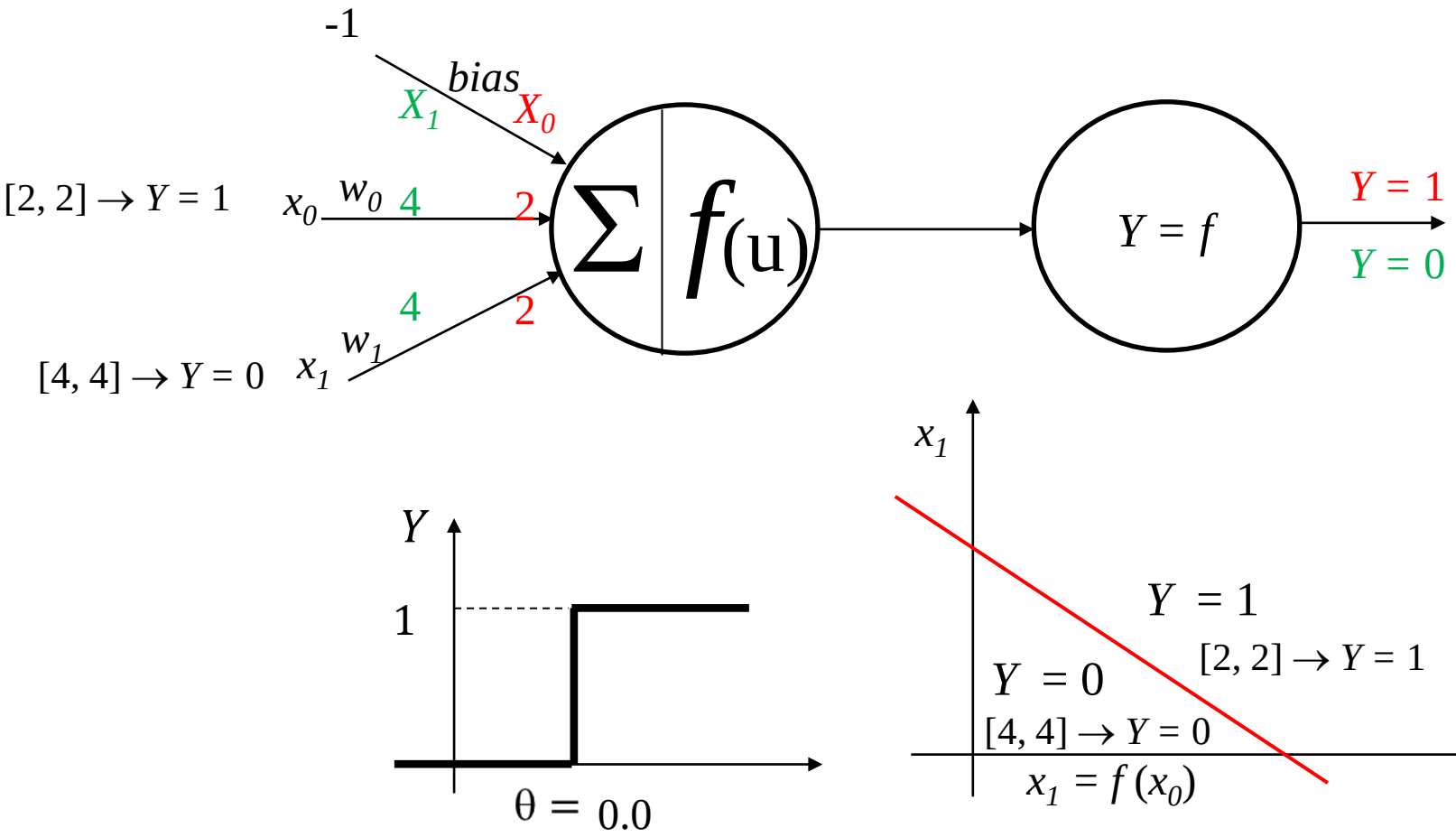
- $w(n)$ = vetor de pesos no instante n;
- $\Delta w(n)$ = valor do incremento a ser aplicado ao vetor de pesos no instante n;
- η = constante que é a medida da rapidez com que o vetor de pesos será atualizado;
- x = vetor de entrada;
- Y = saída atual;
- Yd = saída desejada;
- $e = Yd - Y \rightarrow$ erro.

"O algoritmo de treinamento de um perceptron sempre chega, em um tempo finito, a uma solução para problemas de separação de duas classes linearmente separáveis."

1. Inicializar o valor de η;
2. Inicializar o vetor de pesos w com valores aleatórios;
3. Aplicar a regra de atualização dos pesos $w(n + 1) = w(n) + \eta\, e\, x(n)$ para todos os p pares (x_i, Yd_i) do conjunto de treinamento;
4. Repetir o passo 3 até que $e = 0$ para todos os p elementos do treinamento.

Uso do Perceptron para classificar padrões ou classes

modelo da rede:



(Braga, 2007)

Algoritmo aplicado do *Perceptron*

apresentar padrão x0 [2, 2], y0 = 1

apresentar bias = -1

apresentar constante de ajuste η = 0.1

apresentar pesos w(t) [-0.5441, 0.5562, -0.4074]

calcular u0 = [bias*w(t)[0] + x0[0]*w(t)[1] + x0[1]*w(t)[2]
limiar = 0; se u0 > 0 então *f*(u0) = 1, senão *f*(u0) = 0

atualizar matriz de pesos:

w[0](t+1) =w[0] (t) + η *(y0- *f*(u0))*bias

w[1](t+1) =w[1] (t) + η *(y0- *f*(u0))*x0[0]

w[2](t+1) =w[2] (t) + η *(y0- *f*(u0))*x0[1]

Algoritmo aplicado do *Perceptron*

apresentar padrão x1 [4, 4], y1 =0

apresentar bias = -1

apresentar constante de ajuste η = 0.1

apresentar pesos w (t+1)[..., ..., ...]

calcular u0 = bias*w(t+1)[0] + x1[0]*w(t+1)[1] + x1[1]*w(t+1)[2]
limiar = 0; se u0 > 0 então *f*(u0) = 1, senão *f*(u0) = 0

atualizar matriz de pesos:

w[0](t+2) =w[0] (t+1) + η *(y1 - *f*(u0))*bias

w[1](t+2) =w[1] (t+1) + η *(y1- *f*(u0))*x1[0]

w[2](t+2) =w[2] (t+1) + η *(y1- *f*(u0))*x1[1]

Algoritmo aplicado do *Perceptron*

apresentar padrão x0 [2, 2], y0 = 1

apresentar bias = -1

apresentar constante de ajuste η = 0.1

apresentar pesos w (t+2)[..., ..., ...]

calcular u0 = bias*w(t+2)[0] + x0[0]*w(t+2)[1] + x0[1]*w(t+2)[2]
limiar = 0; se u0 > 0 então *f*(u0) = 1, senão *f*(u0) = 0

atualizar matriz de pesos:

w[0](t+3) =w[0] (t+2) + η *(y0 - *f*(u0))*bias

w[1](t+3) =w[1] (t+2) + η *(y0 - *f*(u0))*x0[0]

w[2](t+3) =w[2] (t+2) + η *(y0 - *f*(u0))*x0[1]

Algoritmo aplicado do *Perceptron*

apresentar padrão x1 [4, 4], y1 =0

apresentar bias = -1

apresentar constante de ajuste η = 0.1

apresentar pesos w (t+3)[..., ..., ...]

calcular u0 = bias*w(t+3)[0] + x1[0]*w(t+3)[1] + x1[1]*w(t+3)[2]
limiar = 0; se u0 > 0 então *f*(u0) = 1, senão *f*(u0) = 0

atualizar matriz de pesos:

w[0](t+4) =w[0] (t+3) + η *(y1- *f*(u0))*bias

w[1](t+4) =w[1] (t+3) + η *(y1- *f*(u0))*x1[0]

w[2](t+4) =w[2] (t+3) + η *(y1- *f*(u0))*x1[1]

Algoritmo aplicado do *Perceptron*

apresentar padrão x0 [2, 2], y0 = 1

apresentar bias = -1

apresentar constante de ajuste η = 0.1

apresentar pesos w (t+4)[..., ..., ...]

calcular u0 = bias*w(t+4)[0] + x1[0]*w(t+4)[1] + x1[1]*w(t+4)[2]
limiar = 0; se u0 > 0 então *f*(u0) = 1, senão *f*(u0) = 0

atualizar matriz de pesos:

w[0](t+5) =w[0] (t+4) + η *(y0- *f*(u0))*bias

w[1](t+5) =w[1] (t+4) + η *(y0- *f*(u0))*x0[0]

w[2](t+5) =w[2] (t+4) + η *(y0- *f*(u0))*x0[1]

parar iterações quando erro (y – *f*(u0)) = 0
e valores de saída Y = valores esperados

Uso de *Perceptron* para classificar padrões ou classes

```
public class perceptron {
    double w [][] = {{- 0.5441},{0.5562},{-0.4074}};  //pesos de entrada
    double x0[] = {-1, 2, 2}; //entradas
    double x1[] = {-1, 4, 4};
    //x[0] = x1[0] = -1 = entrada do bias
    //w[0][0]= - 0.5441 = peso do bias
    double func_ativacao;
    double limiar = 0.0;
    double u0;
    int f;  int y0 = 1; int y1 = 0;
    double taxa_aprendizado = 0.1;

    public void iteração() {
        int n = 0;

        while (n < 5){
w = entrada0(w);

  for (int i = 0; i < w.length; i++)
     System.out.println("pesos w = \t" + w[i][0]);
     System.out.println("valor de f = \t" + f);

 w = entrada1(w);

   for (int i = 0; i < w.length; i++)
     System.out.println("pesos w = \t" + w[i][0]);
     System.out.println("valor de f = \t" + f);

    n = n + 1;
     System.out.println("número de treinamentos " + n + "\n");

    }
}
```

```
public double[][] entrada0(double [][]w){
    System.out.println("entrada 0 ");
    this.w = w;
    u0 = 0;
    for (int i = 0; i < x0.length; i++)
          u0 += x0[i]*w[i][0];
        // System.out.println("x0[" + i + "] = " + x0[i] );
        // System.out.println("w[" + i + "][0] = " + w[i][0] );
         System.out.println("u0 = " + u0 );

 if (u0 > limiar)
    f = 1;
 else f = 0;

 for (int i = 0; i < x0.length; i++){   //3 linhas
     w[i][0]= w[i][0]+ taxa_aprendizado*(y0-f)*x0[i];
    // System.out.println("pesos w = \t" + w[i][0]);
 }

 return w;}

  public double[][] entrada1(double [][] w){
      System.out.println("entrada 1 ");
      this.w = w;
      u0 = 0;
    for (int i = 0; i < x1.length; i++)
          u0 += x1[i]*w[i][0];
         System.out.println("u0 = " + u0 );

 if (u0 > limiar)
    f = 1;
 else f = 0;

 for (int i = 0; i < w.length; i++){
      w[i][0]= w[i][0]+ taxa_aprendizado*(y1-f)*x1[i];
//      System.out.println("pesos w = \t" + w[i][0]);
 }

 return w;
  }
```

Uso de *Perceptron* para classificar padrões ou classes

Uso de *Perceptron* para classificar padrões ou classes

```
    public void testa_rede(double[][] w, double []x0, double []x1 ){
         System.out.println("TESTE DA REDE NEURAL \n "  );

         this.w = w;
         x0 = this.x0;
         x1 = this.x1;

         for (int i = 0; i < w.length; i++){
           System.out.println("pesos resultante do treinamento  "  );
           System.out.println("w[" + i + "][0] = " + w[i][0] );
         }
         u0 = 0;
       for (int i = 0; i < x0.length; i++)
          u0 += x0[i]*w[i][0];
 if (u0 >  limiar)
    f = 1;
 else f = 0;
System.out.println("\nu0 = " + u0 );
System.out.println("teste da entrada x0 saída y = " + f );

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

Uso de *Perceptron* para classificar padrões ou classes

testes

```
public class teste_perceptron {

public static void main(String args[]){

perceptron p = new perceptron();
p.iteração();
p.testa_rede( p.w, p.x0, p.x1);

    }

}
```

entrada 0
u0 = 0.8417000000000001
pesos w = -0.5441
pesos w = 0.5562
pesos w = -0.4074
valor de f = 1
entrada 1
u0 = 1.1393000000000004
pesos w = -0.44410000000000005
pesos w = 0.1562
pesos w = -0.8074
valor de f = 1
número de treinamentos 1

entrada 0
u0 = -0.8583
pesos w = -0.5441
pesos w = 0.3562
pesos w = -0.6073999999999999
valor de f = 0
entrada 1
u0 = -0.46069999999999967
pesos w = -0.5441
pesos w = 0.3562
pesos w = -0.6073999999999999
valor de f = 0
número de treinamentos 2

entrada 0
u0 = 0.04170000000000007
pesos w = -0.5441
pesos w = 0.3562
pesos w = -0.6073999999999999
valor de f = 1
entrada 1
u0 = -0.46069999999999967
pesos w = -0.5441
pesos w = 0.3562
pesos w = -0.6073999999999999
valor de f = 1
número de treinamentos 3

entrada 0
u0 = 0.04170000000000007
pesos w = -0.5441
pesos w = 0.3562
pesos w = -0.6073999999999999
valor de f = 1
entrada 1
u0 = -0.46069999999999967
pesos w = -0.5441
pesos w = 0.3562
pesos w = -0.6073999999999999
valor de f = 0
número de treinamentos 4

entrada 0
u0 = 0.04170000000000007
pesos w = -0.5441
pesos w = 0.3562
pesos w = -0.6073999999999999
valor de f = 1
entrada 1
u0 = -0.46069999999999967
pesos w = -0.5441
pesos w = 0.3562
pesos w = -0.6073999999999999
valor de f = 0
número de treinamentos 5

TESTE DA REDE NEURAL

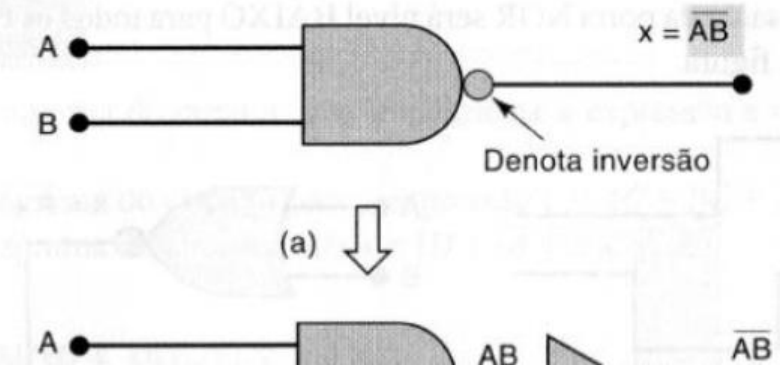
pesos resultante do treinamento
w[0][0] = -0.5441
pesos resultante do treinamento
w[1][0] = 0.3562
pesos resultante do treinamento
w[2][0] = -0.6073999999999999

u0 = 0.04170000000000007
teste da entrada x0 saída y = 1
u0 = -0.46069999999999967
teste da entrada x1 saída y = 0

Prováveis ajustes nos Neurônios MCP para solução das funções booleanas “NAND”, “OU” e “NOU”

profº Mauricio Conceição Mario

Porta e função booleana "NE"



		AND	NAND
A	B	AB	$\overline{AB}$
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

Nand_booleana

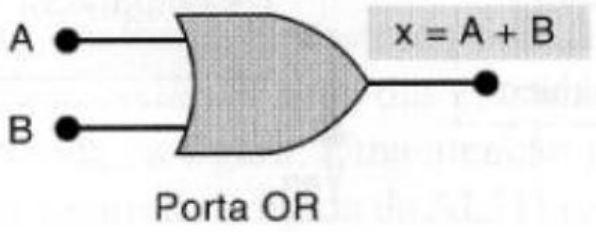
```
double w1 = -1.0; double w2 = -1.0;
double x1[] = {0.0, 0.0, 1.0, 1.0};
double x2[] = {0.0, 1.0, 0.0, 1.0};
double func_ativacao;
double limiar = -1.1;
```

Porta e função booleana "OU"

OR

A	B	x = A + B
0	0	0
0	1	1
1	0	1
1	1	1

(a)



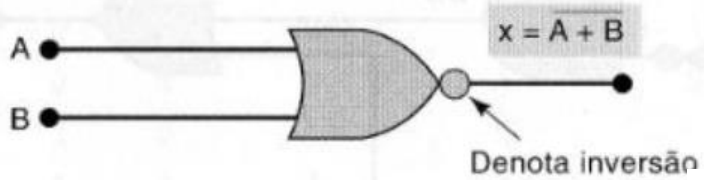
Porta OR

(b)

Or_booleana

```
double w1 = 1.0; double w2 = 1.0;
double x1[] = {0.0, 0.0, 1.0, 1.0};
double x2[] = {0.0, 1.0, 0.0, 1.0};
double func_ativacao;
double limiar = 0.9;
```

Porta e funcão booleana “NOU”



		OR	NOR
A	B	A + B	$\overline{A + B}$
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

Nor_booleana

```
double w1 = -1.0; double w2 = -1.0;
double x1[] = {0.0, 0.0, 1.0, 1.0};
double x2[] = {0.0, 1.0, 0.0, 1.0};
double func_ativacao;
double limiar = -0.1;
```

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.