Inatel
Instituto Nacional de Telecomunicações

www.inatel.br

Redes Perceptron

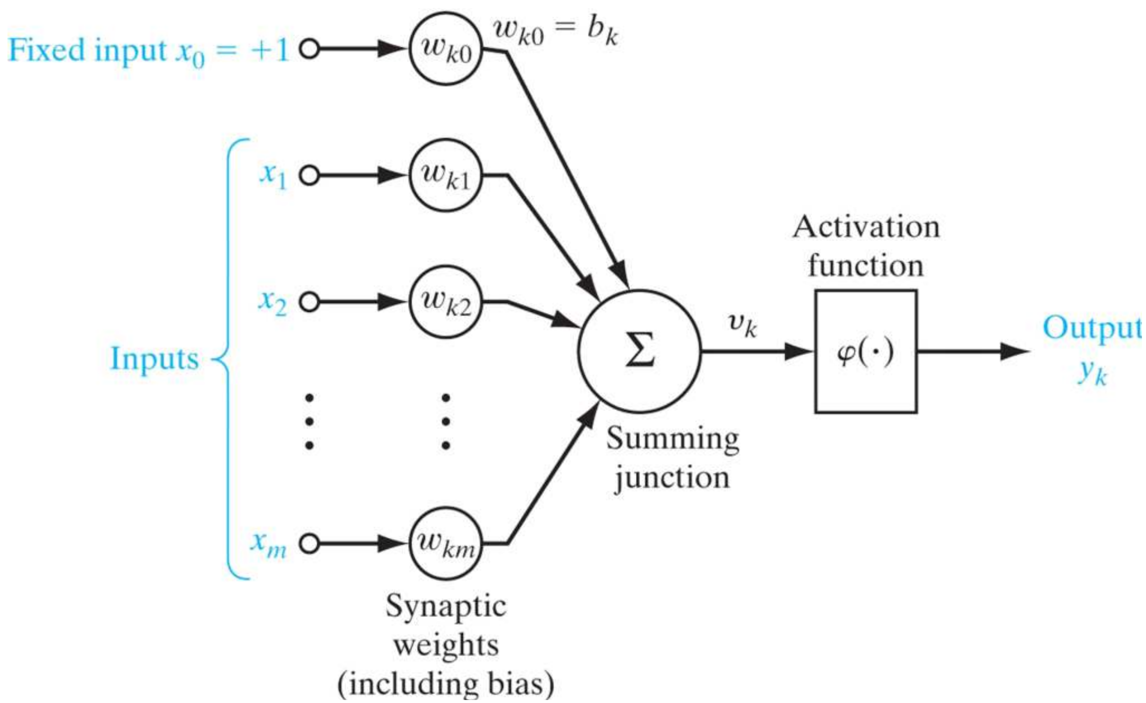
Marcelo Vinícius Cysneiros Aragão
marcelovca90@inatel.br
Av. João de Camargo, 510
Santa Rita do Sapucai - MG
Tel: (35) 3471-9279

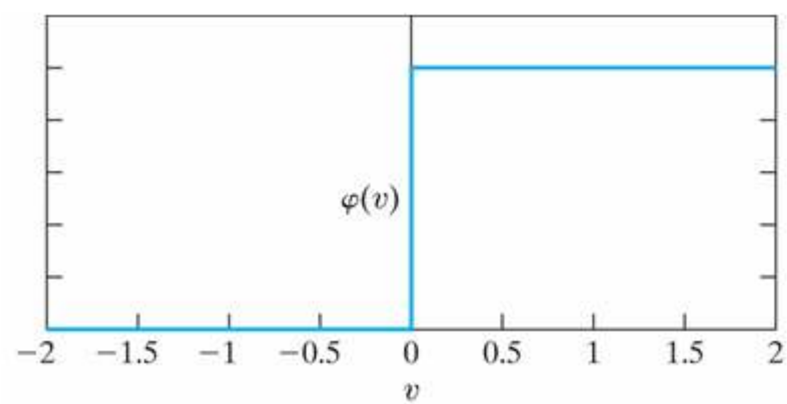
Fev 2018

Perceptrons de camada única

- Idealizado por Rosenblatt (1958), é a forma mais simples de uma rede neural artificial, pois o mesmo é constituído de um único neurônio;
- É considerado uma rede “feed-forward” (alimentação sempre adiante, sem nenhuma realimentação de saída);
- Sua construção é baseada no modelo de neurônio artificial de McCulloch, sendo que sua principal aplicação está na resolução de problemas envolvidos com a classificação de padrões.

Princípio de funcionamento

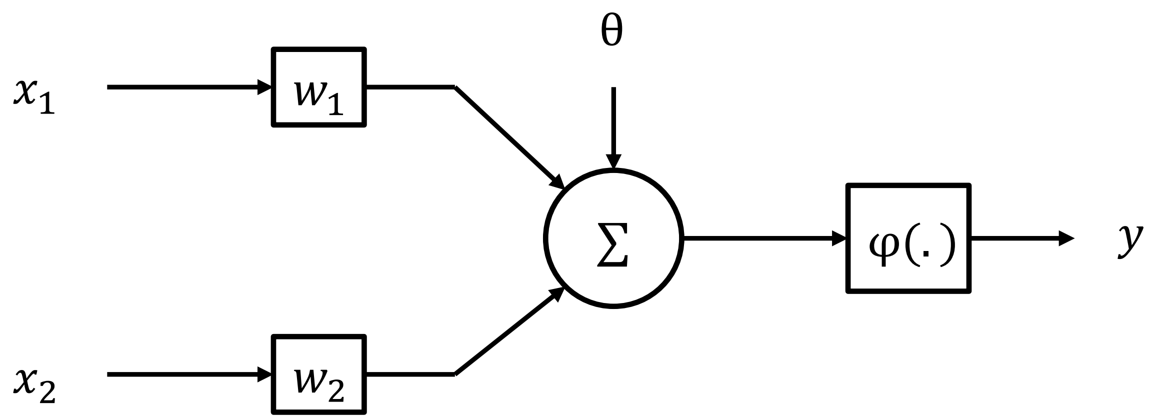




$$y = \begin{cases} +1 & se \quad \sum (w_i * x_i) + b_k \geq 0 \\ -1 & se \quad \sum (w_i * x_i) + b_k < 0 \end{cases}$$

Análise matemática do *Perceptron*

- Para analisar matematicamente o *Perceptron* será considerado um arquitetura com duas entradas;



- A saída do *Perceptron* pode ser escrita em termos matemáticos da seguinte forma:

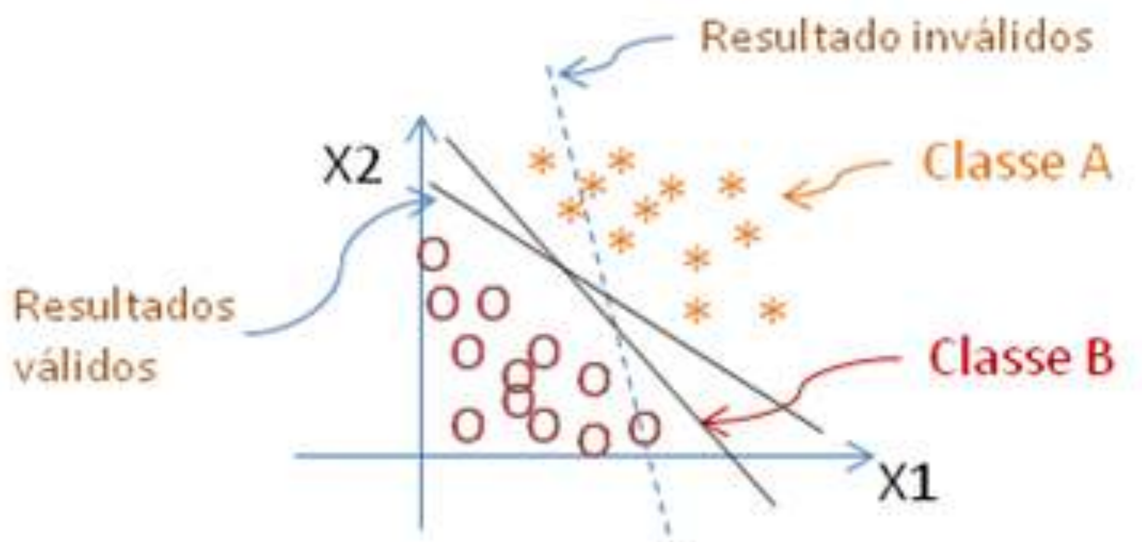
$$y = \begin{cases} +1 \; se \; w_1.x_1 + w_2.x_2 + \theta \geq 0 \\ -1 \; se \; w_1.x_1 + w_2.x_2 + \theta < 0 \end{cases}$$

Análise matemática do Perceptron, cont.

- Expressando a desigualdade através de uma equação do primeiro grau, percebe-se que a fronteira de decisão para este *Perceptron* de duas entradas é representada por uma reta;

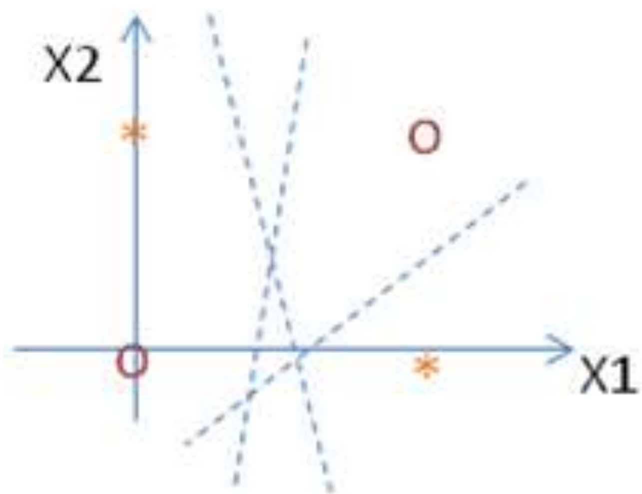
$$w_1.x_1 + w_2.x_2 + b_k = 0$$

- Para a rede *Perceptron*, as classes devem ser "linearmente separáveis"



Problema do XOR

- Para o problema do ou-exclusivo, pode-se utilizar um *Perceptron* de camada única?



X1	X2	y
0	0	0
0	1	1
1	0	1
1	1	0

Treinamento do *Perceptron*

- O processo de treinamento do *Perceptron* está associado ao ajuste dos pesos sinápticos e do limiar da rede com o objetivo de classificar padrões;
- Para o *Perceptron*, a regra de aprendizado utilizada é a regra de Hebb [Hebb, 1949];
- Resumidamente:
  - se a saída reproduzida é coincidente com a saída desejada, os pesos sinápticos e limiar da rede serão então incrementados proporcionalmente aos valores de seus sinais de entrada;
  - caso contrário, os pesos sinápticos e limiar serão decrementados; Este processo é repetido sequencialmente para todas as amostras de treinamento até que a saída do *Perceptron* seja similar a saída desejada para cada amostra;

Treinamento do *Perceptron*, cont.

- Em termos matemáticos, as regras de ajuste dos pesos sinápticos e do limiar do neurônio pode ser expresso da seguinte forma:

$$\begin{cases} w_i^{Atual} = w_i^{Anterior} + \eta.\left(d^{(k)} - y\right).x^{(k)} \\ \theta_i^{Atual} = \theta_i^{Anterior} + \eta.\left(d^{(k)} - y\right).x^{(k)} \end{cases}$$

- Em termos de implementação computacional, fica mais fácil tratar as expressões na forma vetorial:

$$w^{Atual} = w^{Anterior} + \eta.\left(d^{(k)} - y\right).x^{(k)}$$

Treinamento do *Perceptron*, cont.

- Onde:

$w = [\theta \;\; w_1 \; w_2 \ldots w_n]$ é o vetor contendo o limiar e os pesos;

$x^{(k)} = [-1 \quad {x_1}^{(k)} \;\; {x_2}^{(k)} \ldots \;\; {x_n}^{(k)}]$ é a k-ésima amostra de treinamento;

$d^{(k)}$ é o valor desejado para a k-ésima amostra de treinamento;

y é o valor de saída produzido pelo *Perceptron*;

η é uma constante que define a taxa de aprendizagem; Normalmente adota-se $0 < \eta < 1$. Quando muito grande, não converge. Se muito pequena, não chega no resultado.

Algoritmo *Perceptron* - treinamento

1) Obter conjunto de amostras de treinamento $\{x^{(k)}\}$;
2) Associar a saída desejada $\{d^{(k)}\}$ para cada amostra obtida;
3) Iniciar o vetor w com valores aleatórios pequenos;
4) Especificar a taxa de aprendizagem $\{\eta\}$;
5) Iniciar o contador de número de épocas $\{época s \leftarrow 0\}$;
6) Repetir as instruções:

    6.1) $\text{erro} \leftarrow "inexiste"$;

    6.2) Para todos pares de treinamento $\{x^{(k)}, d^{(k)}\}$, faça:

        6.2.1) $v \leftarrow w^T * x^k$;

        6.2.2) $y \leftarrow$ degrau(v); (sign no Matlab)

        6.2.3) Se $y \,!= \, d^{(k)}$

            6.2.3.1) então $\begin{cases} w \leftarrow w + \eta * \left(d^{(k)} - y\right) * x^{(k)} \\ erro \leftarrow "existe" \end{cases}$

    6.3) $época \leftarrow época + 1$;

Até que: $\text{erro} == "inexiste"$

Algoritmo *Perceptron* - operação

1) Obter a amostra a ser classificada $\{x\}$;
2) Utilizar o vetor w ajustado durante o treinamento;
3) Executar as seguintes instruções:

    3.1) $v \leftarrow w^T * x$;

    3.2) $y \leftarrow$ degrau(v);   (sign no Matlab)

    3.3) Se $y == -1$

        3.3.1) Então: amostra $\boldsymbol{x} \in \{Classe\ A\}$

    3.4) Se $y == 1$

        3.4.1) Então: amostra $\boldsymbol{x} \in \{Classe\ B\}$

https://code.google.com/p/neural-networks-examples/

Exemplo de treinamento

- Supondo um problema a ser mapeado pelo *Perceptron* com duas entradas $\{x_1, x_2\}$;
- Para um conjunto de quatro amostras de treinamento constituídas dos seguintes valores: $\Omega^{(x)} = \{[2.0\ 3.5]; [6.8\ 5.3]; [2.0\ 2.5]; [8.1\ 4.2]\}$.
- Considerando-se ainda que os respectivos valores de saída para cada uma das amostras seja dado por $\Omega^{(d)} = \{[-1]; [+1]; [-1]; [+1]\}$.
- Escolhendo aleatoriamente os pesos sinápticos iniciais: $w = \{0.84; 0.68; 0.88\}$.

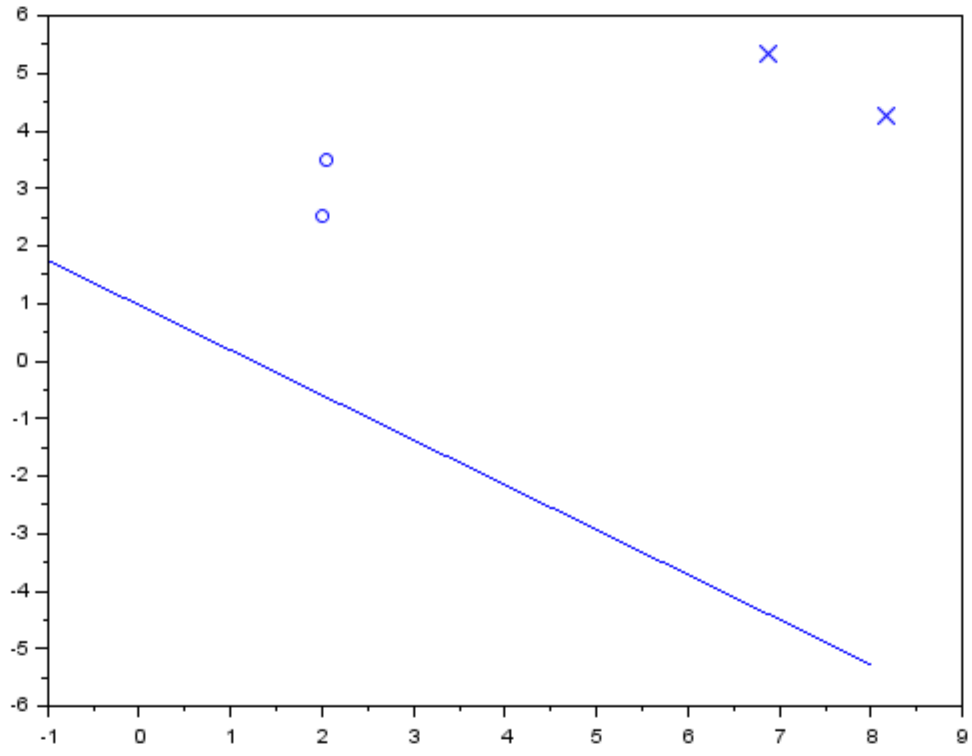
$$\Omega^{(x)} = \begin{array}{c} \\ x^{(1)} \\ x^{(2)} \\ x^{(3)} \\ x^{(4)} \end{array} \begin{array}{c} \begin{array}{ccc} x_0 & x_1 & x_2 \end{array} \\ \begin{bmatrix} -1 & 2.0 & 3.5 \\ -1 & 6.8 & 5.3 \\ -1 & 2.0 & 2.5 \\ -1 & 8.1 & 4.2 \end{bmatrix} \end{array}$$

$$\Omega^{(d)} = \begin{array}{c} d^{(1)} \\ d^{(2)} \\ d^{(3)} \\ d^{(4)} \end{array} \begin{bmatrix} -1 \\ +1 \\ -1 \\ +1 \end{bmatrix}$$

Exemplo de treinamento, cont.

- Após uma época de treinamento:

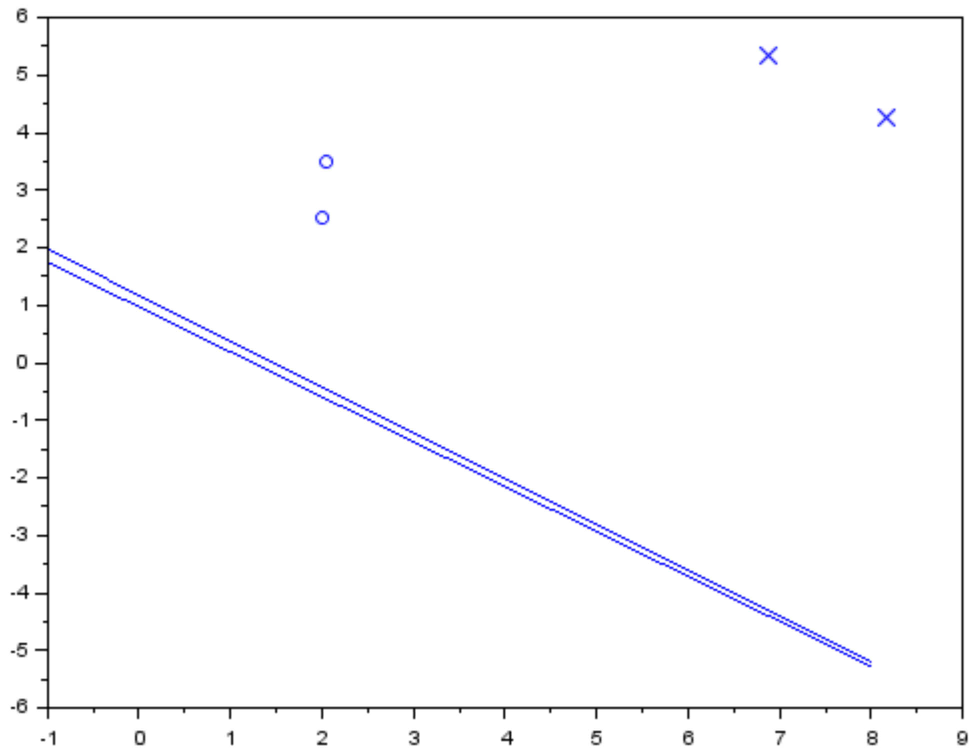
$$w = \{0.88; 0.60; 0.75\}$$



Exemplo de treinamento, cont.

- Após duas épocas de treinamento:

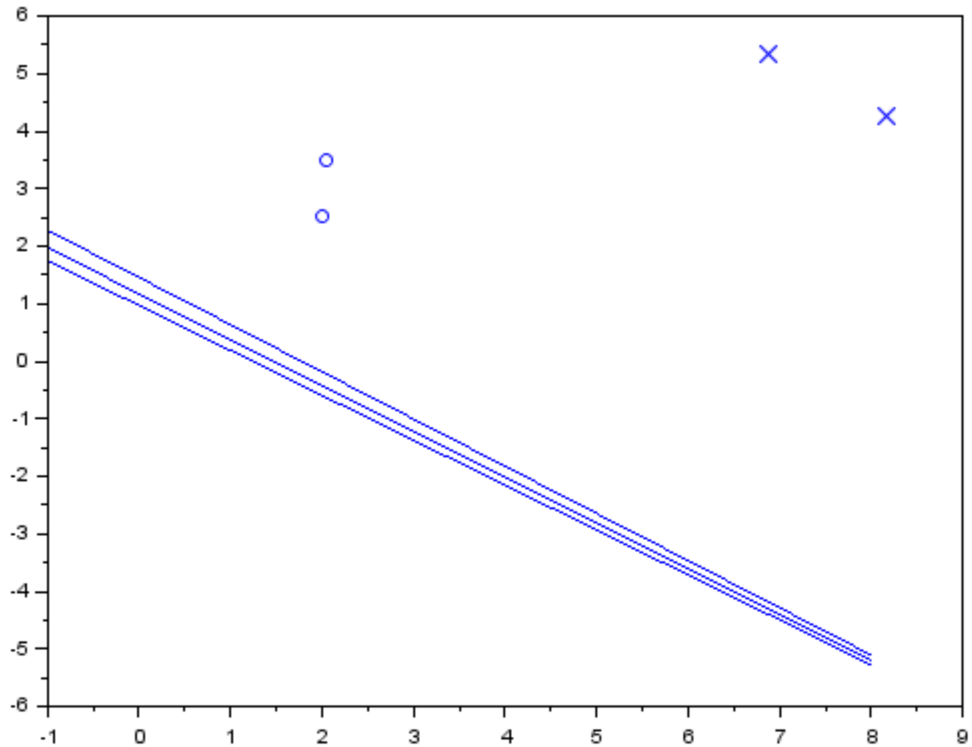
$$w = \{0.93; 0.52; 0.64\}$$



Exemplo de treinamento, cont.

- Após três épocas de treinamento:

$$w = \{0.97; 0.44; 0.51\}$$

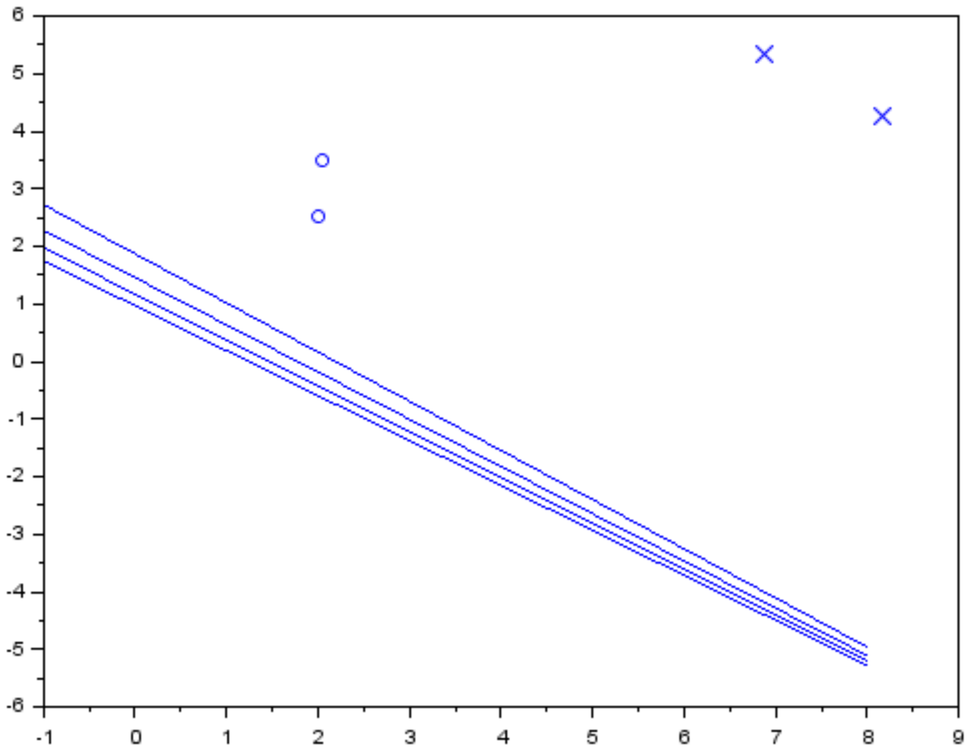


Inatel
Instituto Nacional de Telecomunicações

Exemplo de treinamento, cont.

- Após quatro épocas de treinamento:

$$w = \{1.0; 0.36; 0.39\}$$

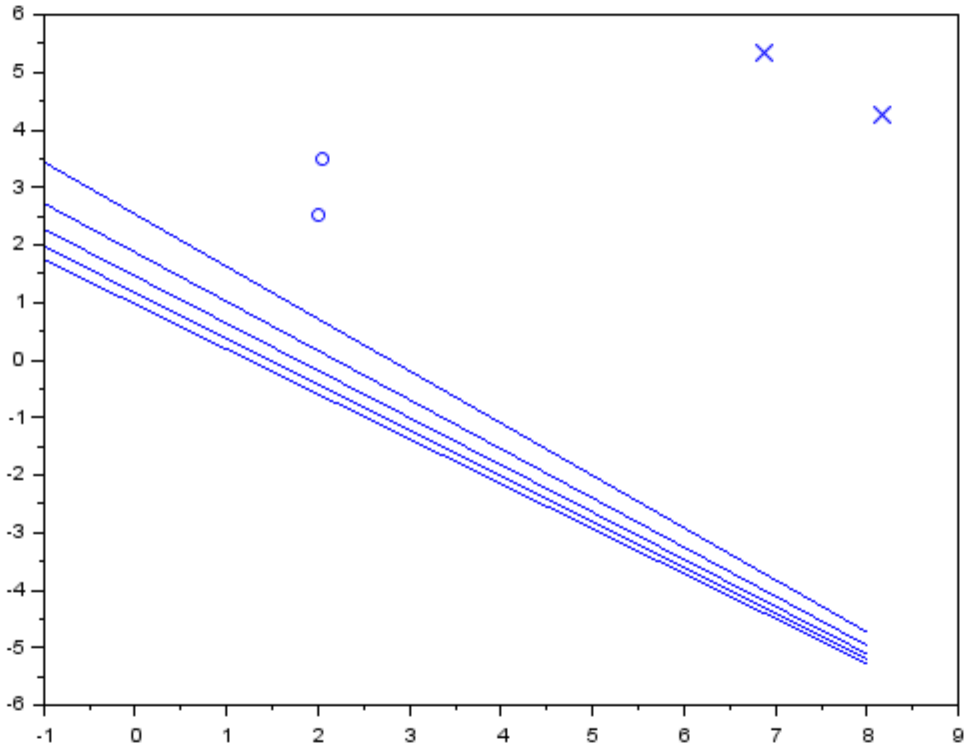


www.inatel.br

Exemplo de treinamento, cont.

- Após cinco épocas de treinamento:

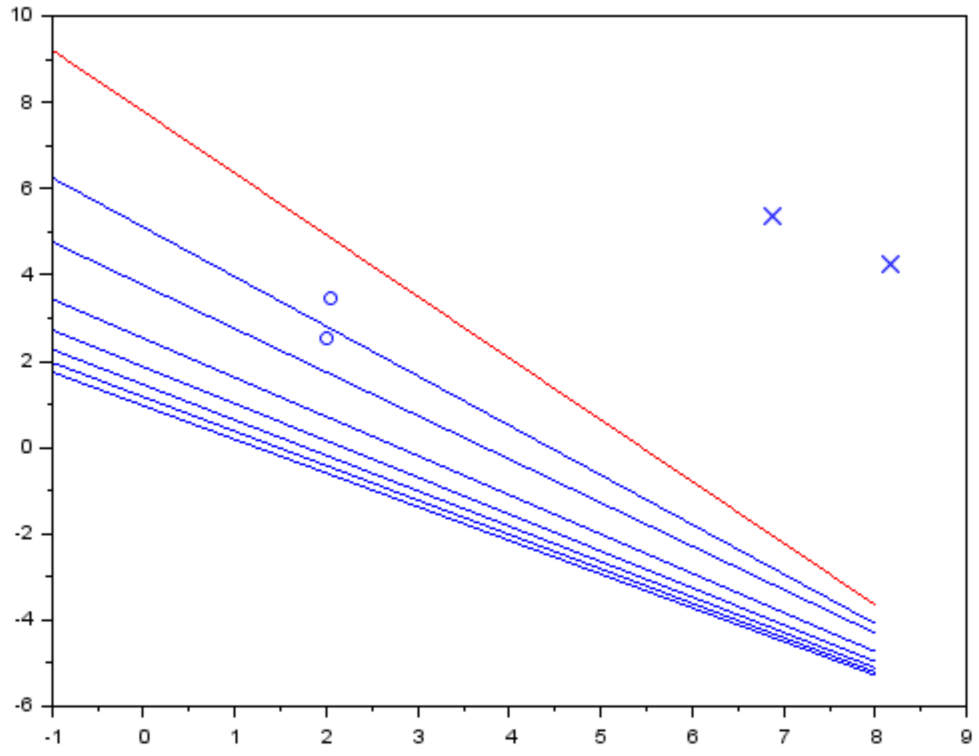
$$w = \{1.04; 0.28; 0.27\}$$



Exemplo de treinamento, cont.

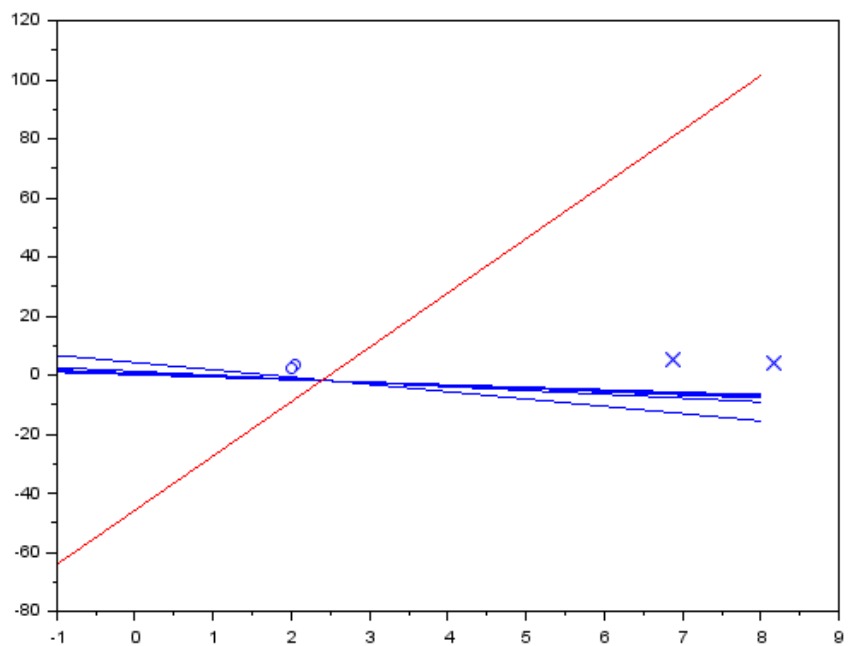
- Após oito épocas de treinamento:

$$w = \{1.09; 0.2; 0.14\}$$



Exemplo de treinamento, cont.

- Usando o mesmo problema anterior porém escolhendo outros valores, aleatoriamente, para os pesos sinápticos iniciais, encontra-se uma solução após 7 épocas de treinamento resultando nos pesos sinápticos finais $w = \{0.28; 0.11; -0.01\}$.



Aspectos práticos sobre o *Perceptron*

- Podem existir infinitas soluções para a rede dependendo dos pesos sinápticos iniciais escolhidos. Logo a solução não é ótima e o número de épocas varia;
- A rede divergirá se o problema não for linearmente separável;
- Usando a faixa de separabilidade entre as classes forem muito estreitas, o processo de treinamento pode implicar em instabilidade. Neste caso utiliza-se uma taxa de aprendizagem $\{\eta\}$ bem pequena.
- Quanto mais próxima a superfície de decisão estiver da fronteira de separabilidade, menos épocas serão necessárias para a rede convergir.
- A normalização das entradas para domínios apropriados contribui para o incremento do desempenho da rede.

Código no GitHub

https://github.com/marcelovca90/nn-python

SingleLayerPerceptronHebbian.py

Fim

www.inatel.br

Inatel
Instituto Nacional de Telecomunicações

Marcelo Vinícius C. Aragão
marcelovca90@inatel.br