Perceptron Multicamadas (MLP)

Prof. Dr. Ajalmar Rocha

Disciplina: Inteligência Computacional Aplicada (ICA)
Programa de Pós-Graduação em Eng. de Telecomunicações (PPGET)
Instituto Federal do Ceará (IFCE)

Agosto/2013

Problemas Linearmente Separáveis

- Problemas **linearmente separáveis** podem ser resolvidos por uma rede neural treinada com uma única camada.
- Classificar dados para as portas OR e AND são exemplos deste tipo de problema.

$$\textbf{Porta OR} = \left[\begin{array}{cc|c} \mathbf{p} & \mathbf{q} & \mathbf{p} \vee \mathbf{q} \\ \hline 0 & 0 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{array}\right] \quad \textbf{Porta AND} = \left[\begin{array}{cc|c} \mathbf{p} & \mathbf{q} & \mathbf{p} \wedge \mathbf{q} \\ \hline 0 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{array}\right]$$

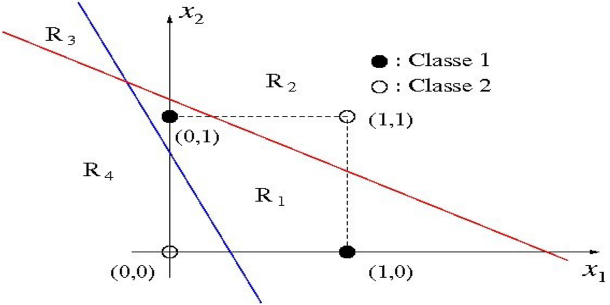
Problemas Não Linearmente Separáveis

- Problemas não-linearmente separáveis não podem ser resolvidos por uma rede neural treinada com uma única camada.
- Este tipo de problema exige uma rede neural do tipo multicamadas com saídas não lineares.
- Classificar dados para a porta XOR é um exemplo deste tipo de problema.

$$\textbf{Porta XOR} = \left[\begin{array}{cc|c} \textbf{p} & \textbf{q} & \textbf{p xor q} \\ \hline 0 & 0 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{array}\right]$$

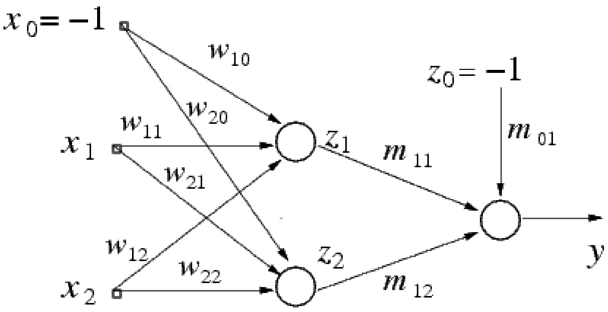
Rede para Classificar dados para o XOR

- Com dois neurônios conseguimos dividir o espaço de entrada em quatro regiões, $\{R_i\}_{i=1}^4$.
- Considere uma reta resolvendo o problema para a **porta AND** e a outra para a **porta OR**.



Rede para Classificar dados para o XOR

- Considere ainda uma rede descrita segundo a arquitetura abaixo.
- A saída para cada neurônio é calculada com base na função degrau.

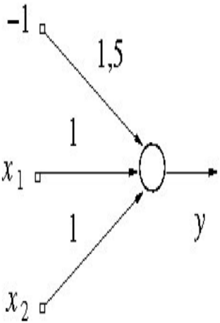


Rede para Classificar dados para o XOR

- A saída para os neurônios da camada oculta são calculadas com base na função degrau.

$$\textbf{Porta AND} = \left[\begin{array}{cc|c} \mathbf{x_1} & \mathbf{x_2} & \mathbf{z_1} \\ \hline 0 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{array}\right] \quad \textbf{Porta OR} = \left[\begin{array}{cc|c} \mathbf{x_1} & \mathbf{x_2} & \mathbf{z_2} \\ \hline 0 & 0 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{array}\right]$$

Pesos do Neurônio de M-P para porta AND (Lembram???)

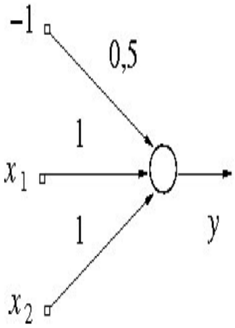


$w_1 = w_2 = 1$ e $\theta = 1{,}5$

$y = 1$, se $u \geqslant 0$.

$y = 0$, se $u < 0$.

Pesos do Neurônio de M-P para porta OR (Lembram???)



$w_1 = w_2 = 1$ e $\theta = 0{,}5$

$y = 1$, se $u \geqslant 0$.

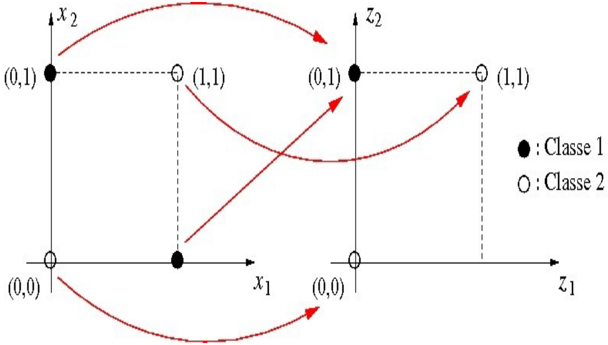
$y = 0$, se $u < 0$.

- Perceba que para o neurônio de saída as entradas são dadas pelos valores de $[z_1 \;\; z_2]^T$, obtidos a partir de $[x_1 \;\; x_2]^T$.

$$\textbf{Porta XOR} = \left[\begin{array}{cc|cc|c} \mathbf{x_1} & \mathbf{x_2} & \mathbf{z_1} & \mathbf{z_2} & \mathbf{y} \\ \hline 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{array}\right]$$

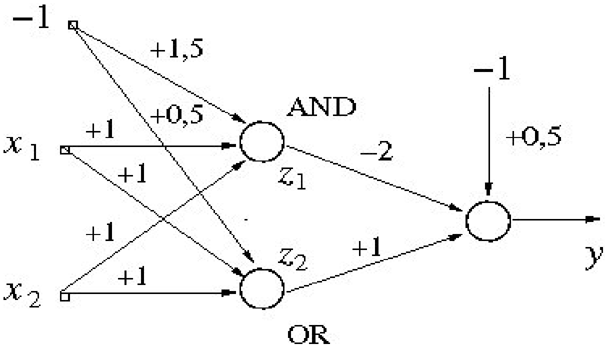
Rede para Classificar dados para o XOR

- A representação dos dados em relação a z_1 e z_2 é mostrado a seguir.
- Como pode se perceber uma reta agora é capaz de classificar o problema.



Rede para Classificar dados para o XOR

- Uma rede com duas camadas capaz de resolver o problema da **porta XOR** é apresentada a seguir.



Perceptron Multicamadas

- Uma rede Perceptron Multicamadas (*Multilayer Perceptron - MLP*) típica possui:
  1. Camada de entrada contendo as componentes do vetor de entrada $\mathbf{x}(t) = [x_0 \;\; x_1 \ldots x_k \ldots x_p]^T$ no tempo t;
  2. Camada oculta de neurônios, tal que $\{\mathbf{w}_i(t)\}_{i=1}^{q}$; e
  3. Camada de saída de neurônios, tal que $\{\mathbf{m}_j(t)\}_{j=1}^{m}$.

Perceptron Multicamadas

- Podemos - em termos de representação - usar a seguinte expressão para descrever uma rede MLP contendo apenas uma camada oculta: $\mathbf{MLP(p, q, m)}$ em que p é o tamanho do vetor de entrada; q é a quantidade de neurônios na camada oculta; e m é o número de neurônios na camada de saída.
- Similarmente para descrever uma rede MLP contendo duas camadas ocultas, usa-se: $\mathbf{MLP(p, q_2, q_1, m)}$ em que q_1 e q_2 são respectivamente as quantidades de neurônios nas primeira e segunda camadas ocultas.

Treinamento de uma rede MLP

- O treinamento de uma rede MLP se da em dois sentidos:
  1. **direto** (ou *forward*), em que o fluxo de sinais se dá na ordem: (**Entrada** $\rightarrow$ **Camada de Oculta** $\rightarrow$ **Camada de Saída**); e
  2. **inverso** (ou *backward*), em que o fluxo de sinais se dá na seguinte ordem: (**Camada de Saída** $\rightarrow$ **Camada de Oculta**)
- O processo realizado no **sentido direto** visa obter a saída da rede considerando o conjunto atual de parâmetros.
- O processo realizado no **sentido inverso** visa calcular o erro e realizar a atualização do vetor de pesos para cada um dos neurônios da rede.

Perceptron Multicamadas: Sentido Direto

- A i-ésima variável de ativação do **neurônio da camada oculta** pode, portanto, ser calculada por

$$u_i = x_0 w_{i0} + x_1 w_{i1} + \cdots + x_k w_{ik} + \cdots + x_p w_p. \qquad (1)$$

- Enquanto a i-ésima variável de saída do neurônio oculto pode ser calculada por

$$h_i = sigmoidal(u_i) \qquad (2)$$

em que $sigmoidal(u_i)$ é uma função sigmóide (logística ou tangente hiperbólica).

- Assim, a saída para a camada é

$$\mathbf{h}^T = [h_1 \dots h_i \dots h_q]^T \qquad (3)$$

Perceptron Multicamadas: Sentido Direto

- Similarmente, a j-ésima variável de ativação do **neurônio da camada saída** pode, portanto, ser calculada por

$$u_j = h_0 m_{j0} + h_1 m_{j1} + \cdots + h_i m_{ji} + \cdots + h_q m_{jq}. \qquad (4)$$

- Enquanto a j-ésima variável de saída do neurônio oculto pode ser calculada por

$$y_j = sigmoidal(u_j) \qquad (5)$$

em que $sigmoidal(u_j)$ é uma função sigmóide (logística ou tangente hiperbólica).

- Assim, a saída para a camada é

$$\mathbf{y}^T = [y_1 \ldots y_j \ldots y_m]^T \qquad (6)$$

Perceptron Multicamadas: Sentido Inverso

- O erro para o j-ésimo **neurônio da camada de saída** pode ser obtido diretamente por

$$e_j(t) = d_j - y_j, \tag{7}$$

uma vez que se sabe a saída desejada.

- Assim, a regra de aprendizagem pode ser utilizada para atualizar os pesos dos **neurônios de saída**

$$m_{ji}(t+1) = m_{ji}(t) + \eta {y_j}'(t) e_j(t) h_i(t) \tag{8}$$

em que ${y_j}'(t)$, como dito anteriormente, é a derivada de uma função sigmóide. Ou ainda, em notação vetorial

$$\mathbf{m}_j(t+1) = \mathbf{m}_j(t) + \eta {y_j}'(t) e_j(t) \mathbf{h}(t) \tag{9}$$

Perceptron Multicamadas: Sentido Inverso

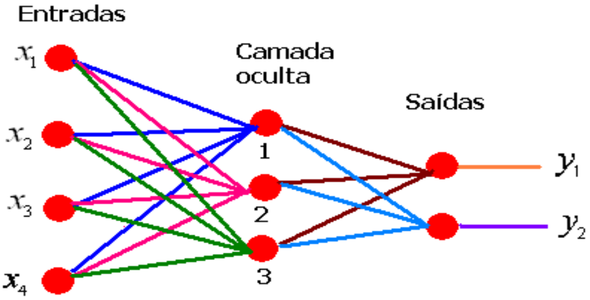
- Até aqui tudo bem, pois temos o erro na camada de saída.
- No entanto, como calcular o erro para os neurônios da camada oculta? Uma vez que não temos nela a saída desejada.
- Para resolver tal problema foi proposto o algoritmo **backpropagation**.

Um dos trabalhos de proposição do backpropagation

D. E. Rumelhart, G. E. Hinton, & R. J. Williams (1986). "Learning representations by backpropagating errors". Nature, 323:533536, 1986.

Perceptron Multicamadas: Sentido Inverso

- Vamos observar mais uma vez uma rede neural MLP típica.



Perceptron Multicamadas: Sentido Inverso

- A regra de aprendizagem para atualização dos pesos dos **neurônios oculta** é dada por

$$w_{ik}(t+1) = w_{ik}(t) + \eta h_i{}'(t) e_i(t) x_k(t). \tag{10}$$

Na notação vetorial, pode ser descrito ainda por

$$\mathbf{w}_i(t+1) = \mathbf{w}_i(t) + \eta h_i{}'(t) e_i(t) \mathbf{x}(t) \tag{11}$$

em que $h_i{}'(t)$, como dito anteriormente, é a derivada de uma função sigmóide.

Perceptron Multicamadas: Sentido Inverso

- Entretanto, o erro retropropagado para i-ésimo **neurônio da camada de oculta** deve ser obtido com base na contribuição relativa dos erros $e_j(t)$ obtidos para **os neurônios da camada de saída**, a saber:

$$e_i(t) = \sum_{j=1}^{m} m_{ji} y_j{}'(t) e_j(t). \tag{12}$$

obtido a partir da minimização de uma função custo

$$J = \frac{1}{2} \sum_{j=1}^{m} (d_j - sigmoidal(\sum_{i=1}^{q} h_i(t) m_{ji}(t)))^2$$

que considera a influência do neurônio oculto nos erros obtidos pelos neurônios de saída.

Perceptron Multicamadas: Sentido Inverso

- Em resumo, a regra de aprendizagem para os neurônios da camada oculta é dada por

$$\mathbf{w}_i(t+1) = \mathbf{w}_i(t) + \eta h_i'(t) \left(\sum_{j=1}^{m} m_{ji}(t) y_j'(t) e_j(t) \right) \mathbf{x}(t) \quad (13)$$

Minimização da Função Custo para Neurônios Ocultos

- A função custo

$$J = \frac{1}{2}\sum_{j=1}^{m}(e_j(t))^2$$

$$J = \frac{1}{2}\sum_{j=1}^{m}(d_j(t) - y_j(t))^2$$

$$J = \frac{1}{2}\sum_{j=1}^{m}(d_j(t) - sigmoidal(\sum_{i=1}^{q} h_i(t)m_{ji}(t)))^2$$

  considera a influência que o neurônios ocultos tem no erro obtido pelos neurônios de saída.

- Como se sabe para obter um regra de aprendizagem basta que se minimize essa função em relação ao vetor de pesos da camada escondida $\mathbf{w}_i$

Minimização da Função Custo para Neurônios Ocultos

- Assim, considere que a derivada da função custo

$$\frac{\partial J}{\partial w_{ik}} = \frac{\partial}{\partial w_{ik}} \frac{1}{2} \sum_{j=1}^{m} (d_j - sigmoidal(\sum_{i=1}^{q} h_i(t) m_{ji}(t)))^2 \quad (14)$$

pode ser apresentada como a soma dos erros de cada um dos neurônios da camada de saída

$$\frac{\partial J}{\partial w_{ik}} = \frac{\partial}{\partial w_{ik}} \frac{1}{2} (e_1^2(t) + e_2^2(t) + \cdots + e_j^2(t) + \cdots + e_m^2(t)) (15)$$

onde m é o número de neurônios de saída. Portanto, a derivada da Eq. (15) pode ser apresentada da seguinte forma:

$$\frac{\partial J}{\partial w_{ik}} = \frac{1}{2} (\frac{\partial}{\partial w_{ik}} e_1^2(t) + \cdots + \frac{\partial}{\partial w_{ik}} e_j^2(t) + \cdots + \frac{\partial}{\partial w_{ik}} e_m^2(t)) (16)$$

Minimização da Função Custo para Neurônios Ocultos

- Analisando separadamente os termos do somatório apresentado na Eq. (16), a saber:

$$\frac{\partial e_j^2(t)}{\partial w_{ik}} = \frac{\partial}{\partial w_{ik}}(d_j(t) - y_j(u_j(t)))^2 \tag{17}$$

- Essa derivada é bastante conhecida e resulta em

$$\frac{\partial e_j^2(t)}{\partial w_{ik}} = (2)(-1)e_j(t)y_j^{'}\frac{\partial}{\partial w_{ik}}u_j(t) \tag{18}$$

ou seja

$$\frac{\partial e_j^2(t)}{\partial w_{ik}} = -2e_j(t)y_j^{'}\frac{\partial}{\partial w_{ik}}u_j(t) \tag{19}$$

Minimização da Função Custo para Neurônios Ocultos

- Lembrando que $u_j(t)$ representa as contribuições dadas pelos neurônios i conectados a j. Logo, a derivada de

$$\frac{\partial}{\partial w_{ik}} u_j(t) = \frac{\partial}{\partial w_{ik}} (\sum_{i=1}^{q} m_{ji}(t) h_i(t)). \tag{20}$$

Minimização da Função Custo para Neurônios Ocultos

- Além disto, como somente o neurônio i da camada escondida tem o peso w_{ik} como entrada, a derivada da Eq. (20) reduz-se às seguintes equações

$$\frac{\partial}{\partial w_{ik}} u_j(t) = \frac{\partial}{\partial w_{ik}}(m_{ji}(t) h_i(t)), \quad e \tag{21}$$

$$\frac{\partial}{\partial w_{ik}} u_j(t) = m_{ji}(t) \frac{\partial}{\partial w_{ik}}(h_i(t)). \tag{22}$$

- Aplicando-se a regra da cadeia na Eq. (22) é obtido

$$\frac{\partial}{\partial w_{ik}} u_j(t) = m_{ji}(t) h_i^{'}(t) \frac{\partial}{\partial w_{ik}}(u_i(t)). \tag{23}$$

Minimização da Função Custo para Neurônios Ocultos

- Uma vez que $u_i(t)$ corresponde a soma ponderada das entradas conectadas no neurônio, a derivada $\frac{\partial}{\partial w_{ik}}(u_i(t))$ permite que a partir da equação

$$\frac{\partial}{\partial w_{ik}} u_j(t) = m_{ji}(t) h_i^{'}(t) \frac{\partial}{\partial w_{ik}}(u_i(t)). \tag{24}$$

seja obtida a seguinte expressão

$$\frac{\partial}{\partial w_{ik}} u_j(t) = m_{ji}(t) h_i^{'}(t) x_k(t). \tag{25}$$

Minimização da Função Custo para Neurônios Ocultos

- Substituindo a Eq. (25) na Eq. (19) temos

$$\frac{\partial e_j^2(t)}{\partial w_{ik}} = (-2)e_j(t)y_j^{'}(t)m_{ji}(t)h_i^{'}(t)x_k(t) \tag{26}$$

- Além disto, a Eq. (16) também pode ser representada por

$$\frac{\partial J}{\partial w_{ik}} = \frac{1}{2}\left(\sum_{j=1}^{m}\frac{\partial}{\partial w_{ik}}e_j^2(t)\right) \tag{27}$$

o que resulta ao se considerar a Eq. (26) em

$$\frac{\partial J}{\partial w_{ik}} = \frac{1}{2}\left(\sum_{j=1}^{m}(-2)e_j(t)y_j^{'}(t)m_{ji}(t)h_i^{'}(t)x_k(t)\right) \tag{28}$$

Minimização da Função Custo para Neurônios Ocultos

- Como os termos $h_i'(t)$ e $x_k(t)$ na Eq. (28) não dependem de j podemos obter a seguinte expressão

$$\frac{\partial J}{\partial w_{ik}} = -h_i'(t)x_k(t)\left(\sum_{j=1}^{m} e_j(t)y_j'(t)m_{ji}(t)\right) \tag{29}$$

- Como o ajuste dos pesos é realizado no sentido contrário ao gradiente, temos que $\Delta w \propto -\nabla J$. Ou seja,

$$\Delta w_{ij}(t) = \eta h_i'(t)\left(\sum_{j=1}^{m} e_j(t)y_j'(t)m_{ji}(t)\right)x_k(t) \tag{30}$$

OBRIGADO