Redes Neurais Artificiais

Multilayer Perceptron

Prof. João Marcos Meirelles da Silva

www.professores.uff.br/jmarcos

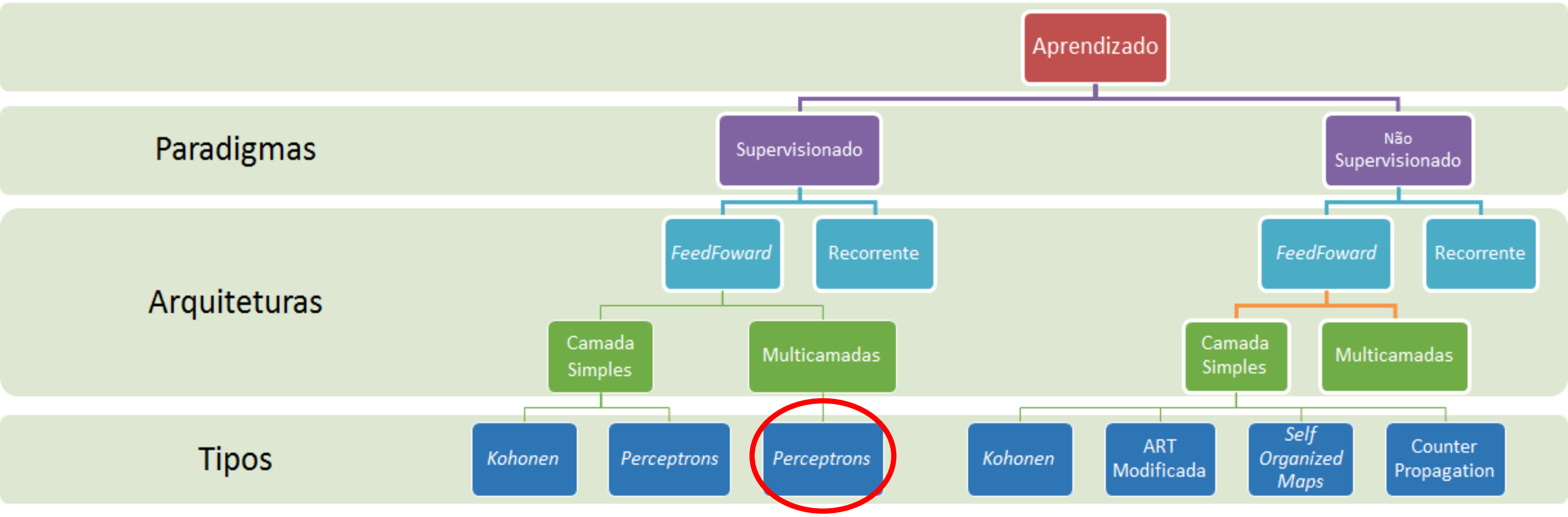
Departamento de Eng. de Telecomunicações
Escola de Engenharia
Universidade Federal Fluminense

Ementa

Redução de Dimensionalidade

Tipos de Dados

Detecção de Anomalias



Objetivo da Aula



Conhecer a rede *multilayer perceptron*, suas funcionalidades e modos de treinamento.

Multilayer Perceptron

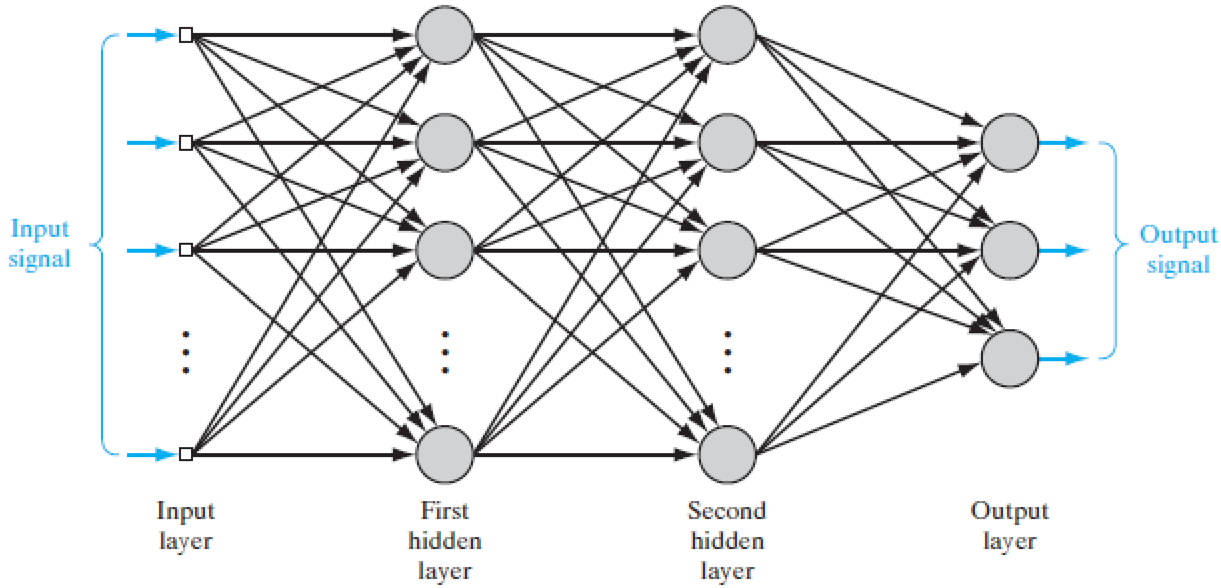


Figura 1: Rede *multilayer perceptron* com duas camadas escondidas.

Multilayer Perceptron

- A rede apresenta, tipicamente, três camadas: Entrada, escondida e saída
- Capacidade de resolver problemas mais complexos que a rede *perceptron* de camada simples
- Algoritmo de treinamento: *Error back-propagation*
- Capacidade de generalização → respostas razoáveis para padrões nunca vistos pela rede durante o treinamento.

Aprendizado *back-propagation* → redes *back-propagation*

Multilayer Perceptron

A Figura 2 abaixo mostra o esquema de aprendizado supervisionado para a rede *multilayer perceptron*. Os blocos *Environment + Teacher* podem ser substituídos por uma base de dados rotulada.

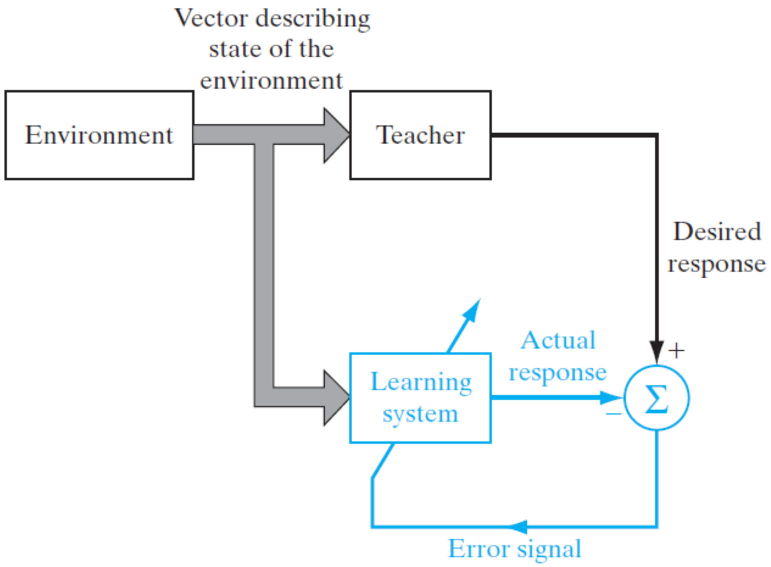


Figura 2: Exemplo de um possível esquema para aprendizado supervisionado.

Multilayer Perceptron

Como exemplo, veja a figura a seguir:

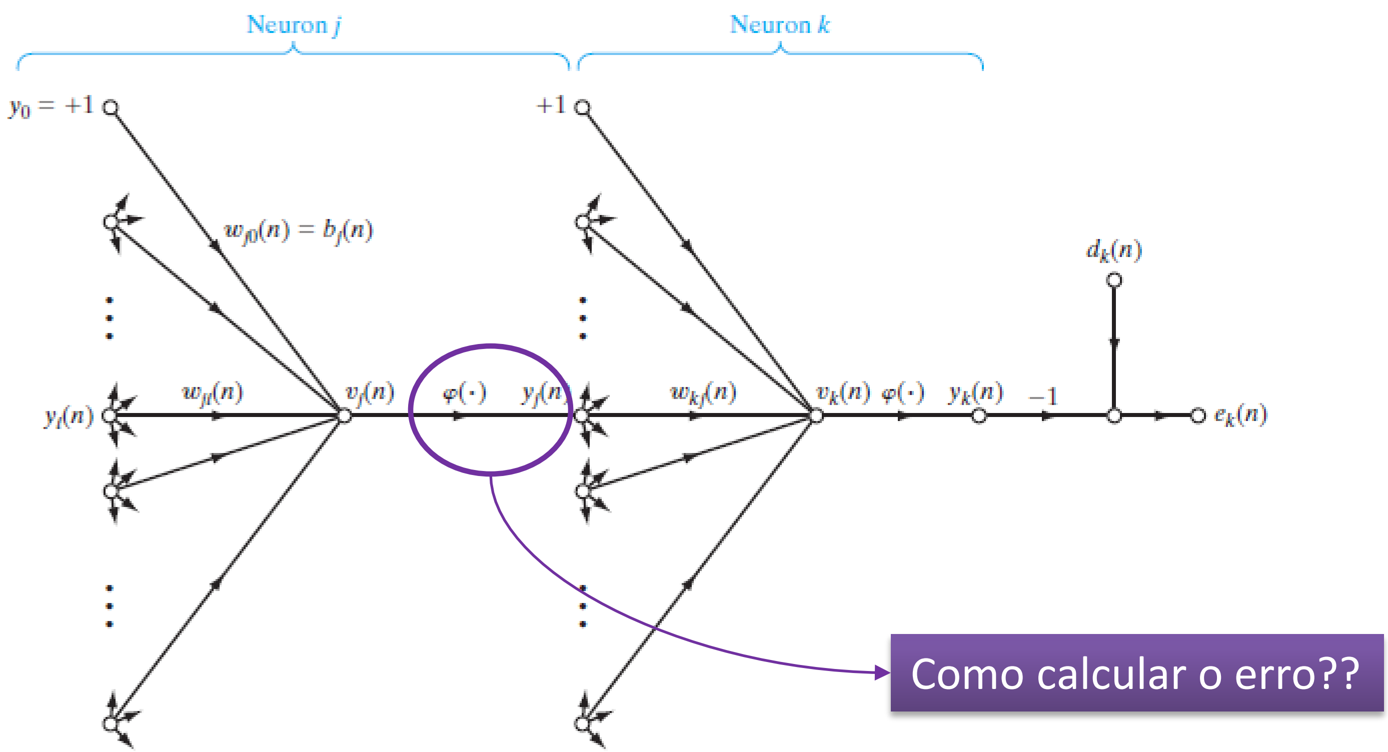


Figura 3: Neurônio *j* como neurônio da camada escondida.

Multilayer Perceptron

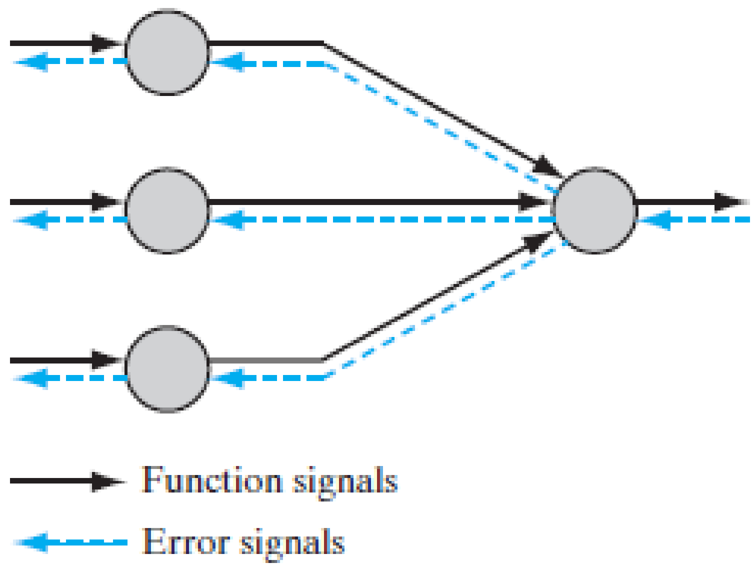


Figura 4: Sinais *feedfoward* (preto) e *back* (azul) *propagations*.

Dois tipos de sinais podem ser identificados na Figura 3 acima:

1. Sinal de entrada (*feedfoward propagation*)
2. Sinal de erro (*error back-propagation*)

Multilayer Perceptron



O treinamento de uma rede via algoritmo *Error Back-propagation* envolve 3 estágios:

1. A propagação do sinal de entrada, camada a camada, a partir do padrão apresentado até a camada de saída;
2. O cálculo e a *retro-propagação* do erro associado;
3. O ajuste dos pesos das sinapses.

Multilayer Perceptron



Observações:

- Após o treinamento, a operação da rede envolve somente a primeira fase;
- Mesmo que a fase de treinamento seja muito lenta, a operação da rede, depois de treinada, é muito rápida;
- Em geral, o número de camadas escondidas depende da complexidade do problema e do número de padrões que rede necessita aprender.

Back-propagation Algorithm

Notação:

- Os índices *i*, *j* e *k* referem-se a neurônio de camadas diferentes na rede:

  1. *i* → neurônio da camada anterior à camada *j*;
  2. *j* → neurônio da camada *j*;
  3. *k*→ neurônio da camada posterior à camada *j*

- Na *n-ésima* iteração, o *n-ésimo* padrão é apresentado à rede;

Back-propagation Algorithm

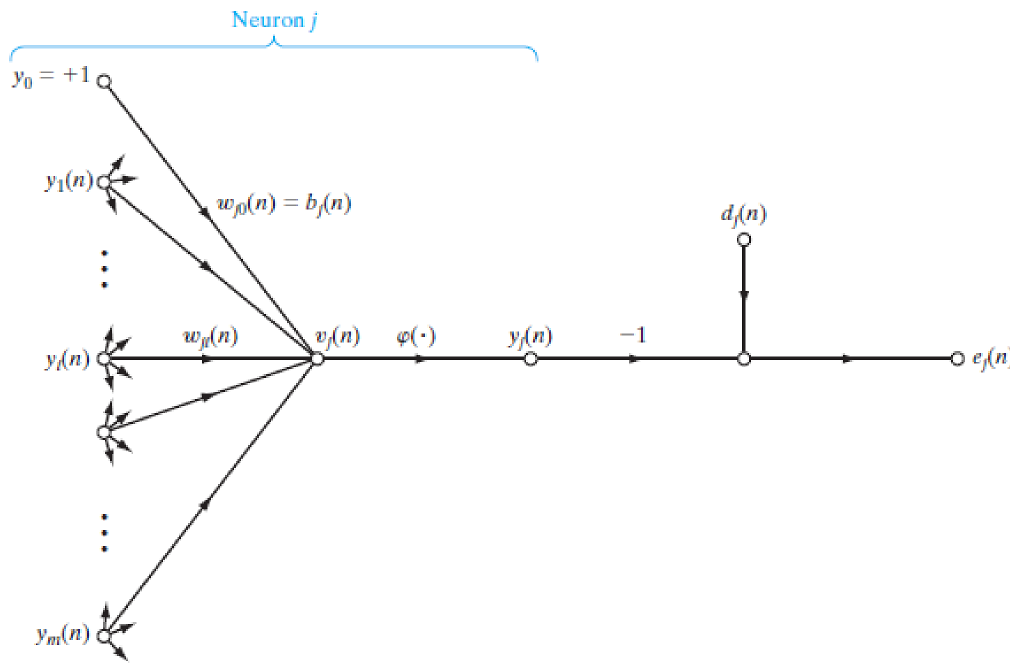


Figura 5: Diagrama simplificado de um neurônio ***j*** na camada de saída da rede neural.

Back-propagation Algorithm



Back-propagation Algorithm ⇒ Gradiente de Erro

O sinal de erro na saída do neurônio *j* na iteração *n* é dado por:

$$e_j(n) = d_j(n) - y_j(n) \tag{1}$$

Definimos o valor instantâneo do erro de energia para o neurônio j como sendo $0.5e_j^2(n)$.

Back-propagation Algorithm

Assim, o valor instantâneo do erro de energia total E(n) é obtido somando-se o valor instantâneo sobre todos os neurônios da camada de saída.

$$E(n) = \frac{1}{2}\sum_{j=1}^{m_L} e_j^2(n) \qquad (2)$$

Back-propagation Algorithm



Seja N o número total de padrões (exemplos) contidos no conjunto de treinamento. O valor médio do erro quadrático de energia é obtido somando-se E(n), ∀n e normalizando-o em relação ao tamanho N do conjunto:

$$E_{avg} = \frac{1}{N} \sum_{n=1}^{N} E(n) \qquad (3)$$

⇒ E(n) e E_{avg} é função dos pesos e do bias da rede!

Back-propagation Algorithm

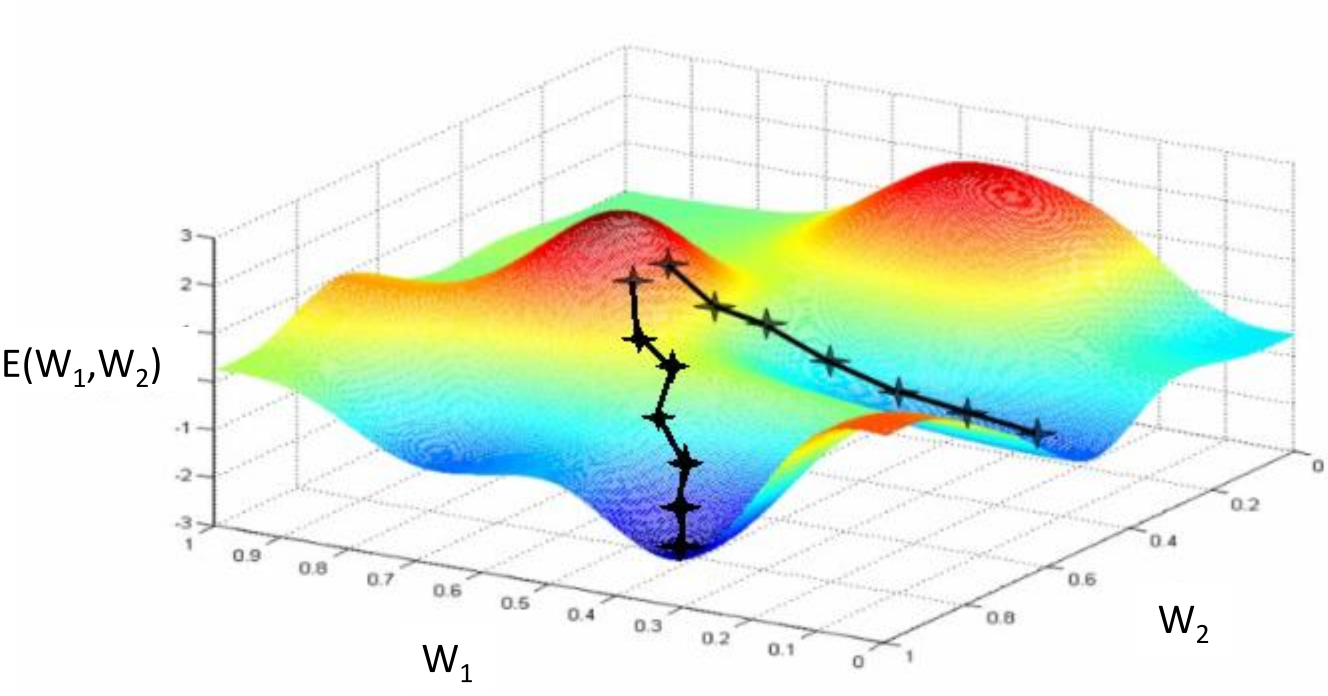


Figura 6: Gradiente aplicado na superfície de erro.

Back-propagation Algorithm

O chamado campo local induzido (*induced local field*), $v_j(n)$, associado ao neurônio *j*, é dado por:

$$v_j(n) = \sum_{i=0}^{m} w_{ji}(n) y_i(n) \qquad (4)$$

onde *m* é o número total de entradas (excluindo o *bias*) aplicadas ao neurônio *j*.

Back-propagation Algorithm

O peso w_{j0} (Correspondente à entrada fixa y_0 = +1) possui valor igual a b_j. Logo, a saída do neurônio *j* é dada por:

$$y_j(n) = \sum_{i=0}^{m} \varphi_j(v_j(n)) \qquad (5)$$

Back-propagation Algorithm



O algoritmo *back-propagation* aplica uma correção $\Delta w_{ji}(n)$ na sinapse w_{ji}, proporcional à derivada parcial $\partial E(n)/\partial w_{ji}(n)$.

$$\frac{\partial E(n)}{\partial w_{ji}} = \frac{\partial E(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial w_{ji}(n)} \qquad (6)$$

$\partial E(n)/\partial w_{ij}(n) \Rightarrow$ Determina a direção de busca no espaço de busca da sinapse w_{ij}.

Back-propagation Algorithm

Diferenciando ambos os lados da equação (2) em relação a $e_j(n)$, temos:

$$\frac{\partial E(n)}{\partial e_j(n)} = e_j(n) \tag{7}$$

Diferenciando ambos os lados da equação (1) em relação a $y_j(n)$, temos:

$$\frac{\partial e_j(n)}{\partial y_j(n)} = -1 \tag{8}$$

Back-propagation Algorithm

Diferenciando ambos os lados da equação (5) em relação a $v_j(n)$, temos:

$$\frac{\partial y_j(n)}{\partial v_j(n)} = \varphi_j'(v_j(n)) \tag{9}$$

Diferenciando ambos os lados da equação (4) em relação a $w_{ij}(n)$, temos:

$$\frac{\partial v_j(n)}{\partial w_{ji}(n)} = y_i(n) \tag{10}$$

Back-propagation Algorithm

Através da substituição das equações (7) a (10) em (6):

$$\frac{\partial E(n)}{\partial w_{ji}(n)} = -e_j(n)\varphi_j'(v_j(n))y_i(n) \qquad (11)$$

A correção $\Delta w_{ij}(n)$ aplicada à sinapse w_{ij} é definida pela Regra Delta:

$$\Delta w_{ji}(n) = -n\frac{\partial E(n)}{\partial w_{ji}(n)} \qquad (12)$$

onde *n* é a taxa de aprendizado do algoritmo *back-propagation*.

Back-propagation Algorithm



Substituindo a equação (11) em (12), temos:

$$\Delta w_{ji}(n) = n\delta_j(n) y_i(n) \qquad (13)$$

onde o gradiente local $\delta_j(n)$ é definido por:

$$\delta_j(n) = -\frac{\partial E(n)}{\partial v_j(n)}$$

$$\delta_j(n) = -\frac{\partial E(n)}{\partial e_j(n)}\frac{\partial e_j(n)}{\partial y_j(n)}\frac{\partial y_j(n)}{\partial v_j(n)}$$

$$\delta_j(n) = e_j(n)\varphi_j'(v_j(n)) \qquad (14)$$

Back-propagation Algorithm

O cálculo do ajuste das sinapses $\Delta w_{ij}(n)$ depende do erro de saída $e_j(n)$ do neurônio *j*, mas:

Se o neurônio *j* for da camada escondida, como fazer para aplicar a equação (13), uma vez que não há como determinar um sinal de erro $e_j(n)$?

Back-propagation Algorithm

Neste caso, o cálculo do gradiente para este neurônio deve ser redefinido. Veja a figura a seguir:

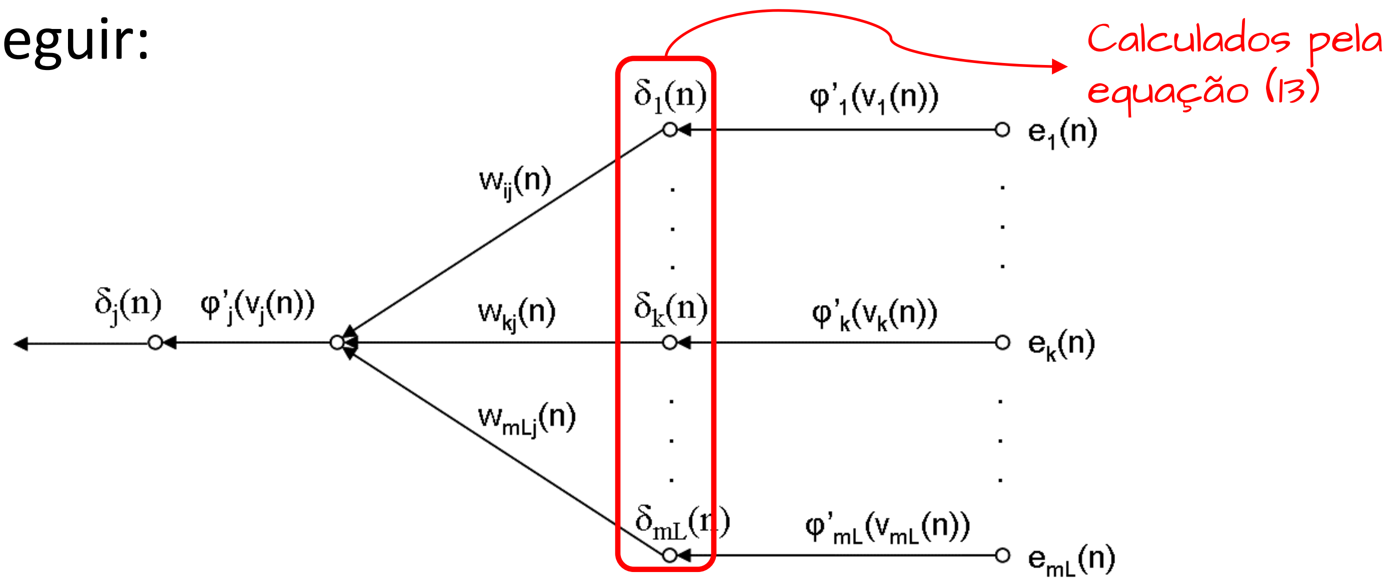
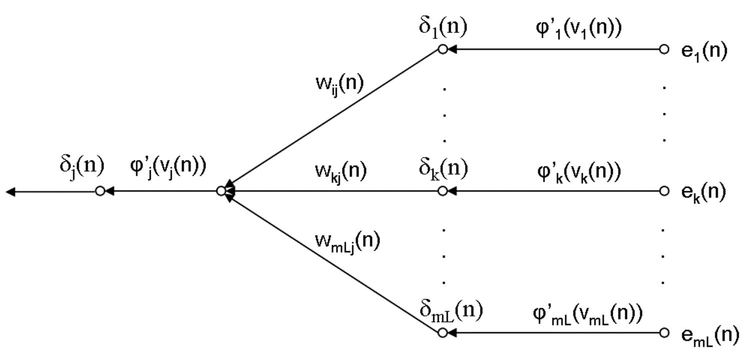


Figura 7: Esquema de retro-propagação do erro para cálculo do gradiente $\delta_j(n)$, a partir dos gradientes da camada de saída.

Back-propagation Algorithm

Por analogia com o raciocínio anterior para o cálculo do gradiente da camada de saída, temos:

$$\delta_j(n) = \varphi_j'(v_j(n)) \sum_{k=1}^{m_L} \delta_k(n) w_{kj}(n) \qquad (15)$$



Back-propagation Algorithm

Correção dos pesos das sinapses:

$$\Delta w_{ji} = n\delta_j(n)y_i(n) \qquad (16)$$

- Se o neurônio *j* pertencer a camada de saída:

$$\delta_j(n) = e_j(n)\varphi_j'(v_j(n)) \qquad (17)$$

- Se o neurônio *j* pertencer a uma camada escondida:

$$\delta_j(n) = \varphi_j'(v_j(n))\sum_{k=1}^{m_l}\delta_k(n)w_{kj}(n) \qquad (18)$$

Back-propagation Algorithm

Resta agora atentarmos para a função de ativação. Se utilizarmos a tangente hiperbólica:

$$\varphi_j\left(v_j(n)\right) = a\tanh\left(bv_j(n)\right), \qquad (a,b) > 0 \qquad (19)$$

e diferenciando (18) em relação a $v_j(n)$:

$$\begin{aligned} \varphi_j' &= ab\,\mathrm{sech}^2 bv_n(n) \\ &= ab(1 - \tanh^2 bv_j(n)) \\ &= \frac{b}{a}\left[a - y_j(n)\right]\left[a + y_j(n)\right] \end{aligned} \qquad (20)$$

Back-propagation Algorithm

Desta forma, o gradiente local pode ser expresso das seguintes formas:

- Para um neurônio *j* da camada de saída:

$$\delta_j(n) = e_j(n)\varphi_j'(v_j(n))$$
$$= \frac{b}{a}[d_j(n) - y_j(n)][a - y_j(n)][a + y_j(n)] \qquad (21)$$

- Para um neurônio *j* de uma camada escondida:

$$\delta_j(n) = \varphi_j'(v_j(n)) \sum_{k=1}^{m_L} \delta_k(n) w_{kj}(n)$$
$$= \frac{b}{a}[a - y_j(n)][a + y_j(n)] \sum_{k=1}^{m_l} \delta_k(n) w_{kj}(n) \qquad (22)$$

Exemplo

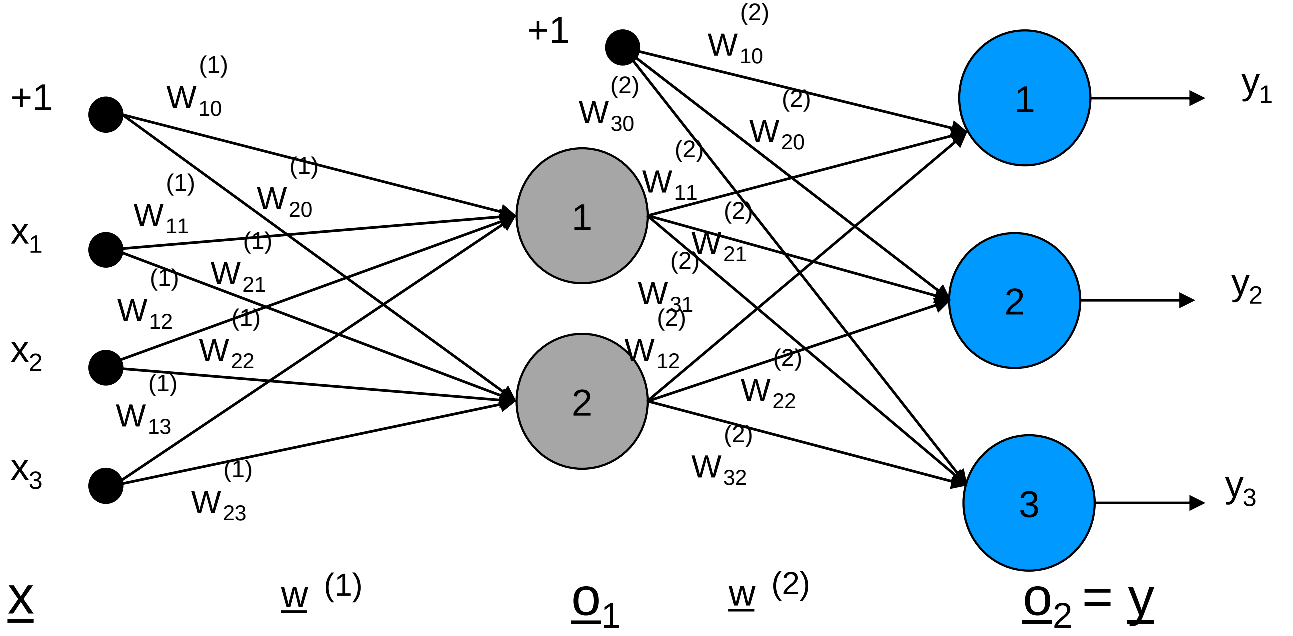


Figura 8: Vamos tomar como exemplo uma rede 3 x 2 x 3.

Exemplo

1) Inicialização:

Pesos das sinapses devem ser iniciados com valores em torno de zero.

2) Padrões de treinamento:

Apresentar um padrão $\underline{x}$ qualquer na camada de entrada da rede

Exemplo



3) Propagação da Entrada (cálculo do campo induzido – camada escondida)

$$v_1^{(1)} = w_{10}^{(1)} + w_{11}^{(1)}x_1 + w_{12}^{(1)}x_2 + w_{13}^{(1)}x_3$$

$$v_2^{(1)} = w_{20}^{(1)} + w_{21}^{(1)}x_1 + w_{22}^{(1)}x_2 + w_{23}^{(1)}x_3$$

Em notação matricial:

$$\begin{bmatrix} v_1^{(1)} \\ v_2^{(1)} \end{bmatrix} = \begin{bmatrix} w_{10}^{(1)} & w_{11}^{(1)} & w_{12}^{(1)} & w_{13}^{(1)} \\ w_{20}^{(1)} & w_{21}^{(1)} & w_{22}^{(1)} & w_{23}^{(1)} \end{bmatrix} \begin{bmatrix} +1 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad (1)$$

Exemplo



Logo:

$$\underline{v}^{(1)} = W^{(1)}\underline{x}_b \qquad (2)$$

onde $\underline{\boldsymbol{v}}^{(1)} \in \mathbb{R}^{\mathbf{2x1}}, \boldsymbol{W}^{(1)} \in \mathbb{R}^{\mathbf{2x4}}, \underline{\boldsymbol{x}}_{\boldsymbol{b}} = [+\mathbf{1} \quad \underline{\boldsymbol{x}}]^{\boldsymbol{T}} \in \mathbb{R}^{\mathbf{4x1}}$

A saída da camada escondida é dada por:

$$\underline{o}^{(1)} = \tanh(\underline{v}^{(1)}) \qquad (3)$$

onde $\underline{\boldsymbol{o}}^{(1)} \in \mathbb{R}^{\mathbf{2x1}}$.

Exemplo

Para a camada de saída, temos:

$$v_1^{(2)} = w_{10}^{(2)} + w_{11}^{(2)} o_1 + w_{12}^{(2)} o_2$$
$$v_2^{(2)} = w_{20}^{(2)} + w_{21}^{(2)} o_1 + w_{22}^{(2)} o_2$$
$$v_3^{(2)} = w_{30}^{(2)} + w_{31}^{(2)} o_1 + w_{32}^{(2)} o_2$$

Em notação matricial:

$$\begin{bmatrix} v_1^{(2)} \\ v_2^{(2)} \\ v_3^{(2)} \end{bmatrix} = \begin{bmatrix} w_{10}^{(2)} & w_{11}^{(2)} & w_{12}^{(2)} \\ w_{20}^{(2)} & w_{21}^{(2)} & w_{22}^{(2)} \\ w_{30}^{(2)} & w_{31}^{(2)} & w_{32}^{(2)} \end{bmatrix} \begin{bmatrix} +1 \\ o_1 \\ o_2 \end{bmatrix} \qquad (4)$$

Exemplo



Logo:

$$\underline{v}^{(2)} = W^{(2)}\underline{o}_b \tag{5}$$

onde $\underline{\boldsymbol{v}}^{(2)} \in \mathbb{R}^{\boldsymbol{3x1}}, \boldsymbol{W}^{(2)} \in \mathbb{R}^{\boldsymbol{3x3}}, \underline{\boldsymbol{o}}_{\boldsymbol{b}} = [+\boldsymbol{1} \quad \underline{\boldsymbol{o}}]^{\boldsymbol{T}} \in \mathbb{R}^{\boldsymbol{3x1}}$

A saída da rede é dada por:

$$\underline{o}^{(2)} = y = \tanh(\underline{v}^{(2)}) \tag{6}$$

onde $\underline{\boldsymbol{o}}^{(2)} \in \mathbb{R}^{\boldsymbol{3x1}}$.

Exemplo

4) Calculando o erro:

$$e_1 = d_1 - y_1$$
$$e_2 = d_2 - y_2$$
$$e_3 = d_3 - y_3$$

No caso desse exemplo (rede autoassociativa), temos que $\underline{d} \equiv \underline{x}$. Na forma vetorial:

$$\underline{e} = \underline{x} - \underline{y}\,, \qquad \underline{e} \in \mathbb{R}^{3x1} \qquad (7)$$

Exemplo



5) Agora vamos retropropagar os erros. Vamos inicialmente calcular os gradientes (camada de saída):

$$\delta_1^{(2)} = e_1(1 - y_1^2)$$
$$\delta_2^{(2)} = e_2(1 - y_2^2)$$
$$\delta_3^{(2)} = e_3(1 - y_3^2)$$

Na forma matricial:

$$\underline{\delta}^{(2)} = diag(\underline{e})[\underline{1} - diag\left(\underline{y}\right)\underline{y}] \qquad (\ 8\ )$$

Em python, a equação (8) fica:

```
delta2 = np.dot(np.diagflat(e), (1 - np.dot(np.diagflat(Y), Y)))
```

Exemplo

Para a camada escondida, temos:

$$\underline{v}_\delta = W^{(2)^T}\underline{\delta}^{(2)} \qquad (9)$$

onde $\underline{v}_\delta \in \mathbb{R}^{2x1}, W^{(2)^T} \in \mathbb{R}^{3x3}$ e $\underline{\delta}^{(2)} \in \mathbb{R}^{3x1}$.

Calculando os gradientes da camada escondida (camada 1) de forma escalar:

$$\delta_1^{(1)} = (1 - o_1^2)v_{\delta_1}$$
$$\delta_2^{(1)} = (1 - o_2^2)v_{\delta_2}$$

Exemplo

Em notação matricial:

$$\underline{\delta}^{(1)} = diag\left[\underline{1} - diag\left(\underline{o}\right)\underline{o}\right]\underline{v}_{\delta} \qquad (10)$$

Agora que temos o gradiente das camadas (1) e (2), podemos proceder com a atualização dos pesos das sinapses:

$$W^{(1)} = W^{(1)} + \eta\underline{\delta}^{(1)}\underline{x}_b$$

$$W^{(2)} = W^{(2)} + \eta\underline{\delta}^{(2)}\underline{o}_b$$

Resumo do Algoritmo



De forma genérica, para uma rede MLP de três camadas m-N-L, temos:

m – Número de neurônios da camada de entrada
N – Número de neurônios da camada escondida ou intermediária
L - Número de neurônios da camada de saída

$\underline{X}$ - Vetor de entrada.
$\underline{X}_b$ – Vetor de entrada com bias.
$\underline{o}$ - Vetor de saída da camada escondida.
o_b – Vetor de entrada da camada de saída.

Resumo do Algoritmo

$W^{(1)}$ – Matriz de pesos das camada escondida:

$$W^{(1)} = \begin{bmatrix} w_{10}^{(1)} & w_{11}^{(1)} & \cdots & w_{1m}^{(1)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{N0}^{(1)} & w_{N1}^{(1)} & \cdots & w_{Nm}^{(1)} \end{bmatrix}, \qquad W^{(1)} \in \mathbb{R}^{Nx(m+1)}$$

$W^{(2)}$ – Matriz de pesos das camada de saída:

$$W^{(2)} = \begin{bmatrix} w_{10}^{(2)} & w_{11}^{(2)} & \cdots & w_{1N}^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{L0}^{(2)} & w_{L1}^{(2)} & \cdots & w_{LN}^{(2)} \end{bmatrix}, \qquad W^{(2)} \in \mathbb{R}^{Lx(N+1)}$$

Resumo do Algoritmo



- Propagação:

$$\underline{v}^{(1)} = W^{(1)}\underline{X}_b \qquad (23)$$

$$\underline{o} = tanh\left(\underline{v}^{(1)}\right) \qquad (24)$$

$$\underline{v}^{(2)} = W^{(2)}\underline{o}_b \qquad (25)$$

$$\underline{y} = tanh\left(\underline{v}^{(2)}\right) \qquad (26)$$

- Cálculo do erro:

$$\underline{e} = \underline{d} - \underline{y}, \qquad \underline{e} \in \mathbb{R}^{Lx1} \qquad (27)$$

Resumo do Algoritmo

- Retropagação:

$$\underline{\delta}^{(2)} = diag(\underline{e})\left[\underline{1} - diag\left(\underline{y}\right)\underline{y}\right], \qquad \underline{\delta}^{(2)} \in \mathbb{R}^{Lx1} \qquad (28)$$

$$\underline{v}_{\delta} = W^{(2)^T}\underline{\delta}^{(2)}, \qquad \underline{v}_{\delta} \in \mathbb{R}^{(N+1)x1} \qquad (29)$$

$$\underline{\delta}^{(1)} = diag\left[\underline{1} - diag(\underline{o})\underline{o}\right]\underline{v}_{\delta}, \qquad \underline{\delta}^{(1)} \in \mathbb{R}^{mx1} \qquad (30)$$

- Atualização dos Pesos:

$$W^{(1)} = W^{(1)} + \eta\, diag\left(\underline{\delta}^{(1)}\right)\underline{x}_b \qquad (31)$$

$$W^{(2)} = W^{(2)} + \eta\, diag\left(\underline{\delta}^{(2)}\right)\underline{o}_b \qquad (32)$$

Referências



1. *Neural Networks and Learning Machines*, 3rd. Edition, Simon Haykin
2. *Fundamental of Neural Networks - Architectures, Algorithms and Applications*, Laurene Fausett
3. *Pattern Classification*, Richard O. Duda, Peter E. Hart, David G. Stork