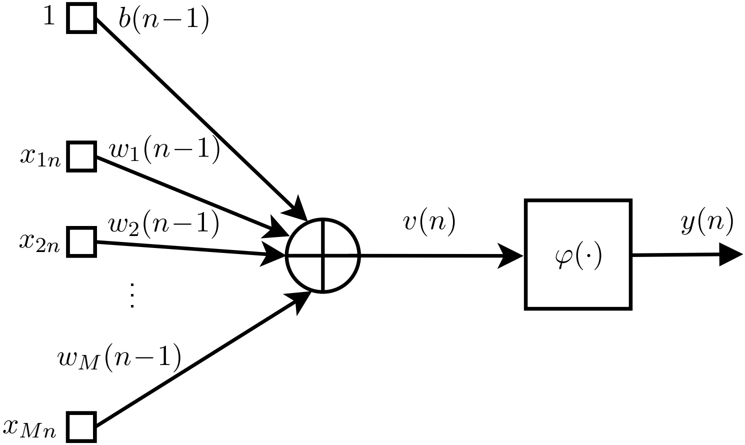
PSI3471 – Fundamentos de Sistemas Eletrônicos Inteligentes

A rede perceptron multicamada

Magno T. M. Silva e Renato Candido

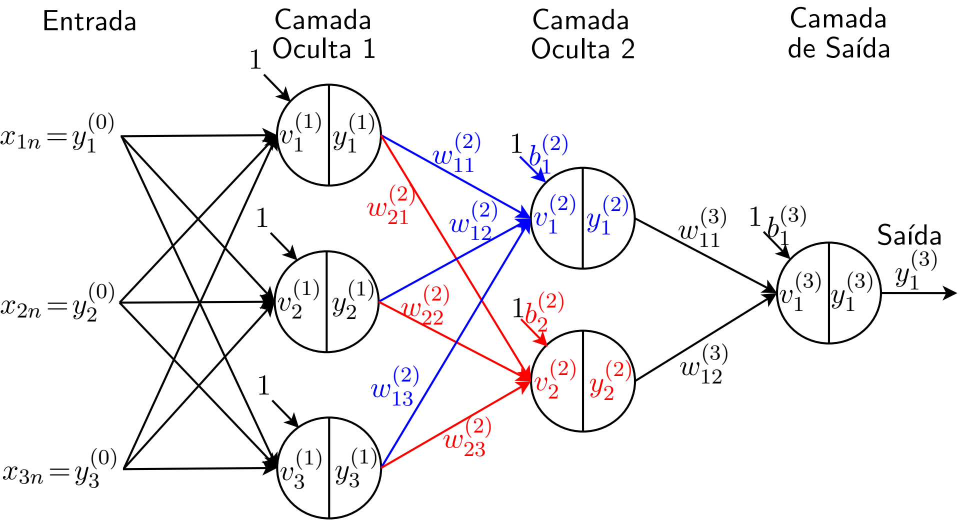
Escola Politécnica da USP

1 Modelo do neurônio: unidade básica da rede MLP



- n-ésimo vetor dos dados: $\mathbf{x}(n) = [\,1\ x_{1n}\ x_{2n}\ \cdots\ x_{Mn}\,]^T$
- vetor de pesos: $\mathbf{w}(n) = [\,b(n)\ w_1(n)\ \cdots\ w_M(n)\,]^T$
- saída do combinador linear: $v(n) = \mathbf{x}^T(n)\mathbf{w}(n-1)$
- saída do neurônio: $y(n) = \varphi(v(n))$

2 Rede MLP com $L = 3$ camadas e $N_L = 1$ saída



MLP com configuração N_1–N_2–N_3 =3–2–1

3 Notação e cálculo da saída da Camada 2

- $\omega_{k\ell}^{(j)}$ representa o peso que liga o Neurônio ℓ da camada $j-1$ ao Neurônio k da camada j
- Se o Neurônio k pertencer à primeira camada oculta, devemos trocar "Neurônio ℓ" na frase anterior por "Entrada ℓ"
- Entrada da rede: $[x_{1n} \;\; x_{2n} \;\; \cdots \;\; x_{Mn}]^T$
- Vamos usar a notação: $y_k^{(0)} \triangleq x_{kn}$, $k = 1{,}2, \ldots, M$ e $N_0 \triangleq M$

$$\mathbf{y}^{(0)} \triangleq \begin{bmatrix} y_1^{(0)} \\ y_2^{(0)} \\ \vdots \\ y_{N_0}^{(0)} \end{bmatrix} = \begin{bmatrix} x_{1n} \\ x_{2n} \\ \vdots \\ x_{Mn} \end{bmatrix}$$

- Em cada neurônio da primeira camada oculta, esse vetor é ponderado e somado ao *bias*. A saída de cada neurônio é calculada, aplicando-se a função de ativação $\varphi(\cdot)$ ao resultado dessa combinação linear

4 Notação e cálculo da saída da Camada 2

- Vetor de saídas dos neurônios da primeira camada oculta:

$$\mathbf{y}^{(1)} = \begin{bmatrix} y_1^{(1)} \\ y_2^{(1)} \\ \vdots \\ y_{N_1}^{(1)} \end{bmatrix}$$

- Vetor de entradas da Camada 2:

$$\mathbf{x}^{(2)} = \begin{bmatrix} 1 \\ \mathbf{y}^{(1)} \end{bmatrix}$$

- Matriz de *biases* e os pesos da Camada 2:

$$\mathbf{W}^{(2)} = \begin{bmatrix} b_1^{(2)} & w_{11}^{(2)} & w_{12}^{(2)} & w_{13}^{(2)} \\ b_2^{(2)} & w_{21}^{(2)} & w_{22}^{(2)} & w_{23}^{(2)} \end{bmatrix}_{2\times 4} \triangleq \begin{bmatrix} \mathbf{w}_1^{(2)} \\ \mathbf{w}_2^{(2)} \end{bmatrix}.$$

- Observação: $\mathbf{w}_k^{(j)}$, $k = 1, 2, \ldots, N_j$, são vetores linha

5 Notação e cálculo da saída da Camada 2

- Saídas dos combinadores lineares da Camada 2:

$$\mathbf{v}^{(2)} = \begin{bmatrix} v_1^{(2)} \\ v_2^{(2)} \end{bmatrix} = \begin{bmatrix} \mathbf{w}_1^{(2)}\mathbf{x}^{(2)} \\ \mathbf{w}_2^{(2)}\mathbf{x}^{(2)} \end{bmatrix} = \mathbf{W}^{(2)}\mathbf{x}^{(2)}$$

- Saídas da Camada 2:

$$\mathbf{y}^{(2)} = \varphi(\mathbf{v}^{(2)}) = \begin{bmatrix} \varphi(v_1^{(2)}) \\ \varphi(v_2^{(2)}) \end{bmatrix} = \begin{bmatrix} y_1^{(2)} \\ y_2^{(2)} \end{bmatrix}$$

6 Cálculo progressivo para uma rede MLP genérica

- Entrada, saídas dos CLs e saídas da camada j:

$$\mathbf{x}^{(j)} \triangleq \begin{bmatrix} 1 \\ \mathbf{y}^{(j-1)} \end{bmatrix}, \quad \mathbf{v}^{(j)} \triangleq \begin{bmatrix} v_1^{(j)} \\ v_2^{(j)} \\ \vdots \\ v_{N_j}^{(j)} \end{bmatrix}, \quad \mathbf{y}^{(j)} \triangleq \begin{bmatrix} y_1^{(j)} \\ y_2^{(j)} \\ \vdots \\ y_{N_j}^{(j)} \end{bmatrix},$$

  $j = 1, 2, \ldots, L$ e $\mathbf{y}^{(0)}$ o vetor de entradas da rede

- Matriz de pesos da camada j:

$$\mathbf{W}^{(j)} \triangleq \begin{bmatrix} b_1^{(j)} & w_{11}^{(j)} & w_{12}^{(j)} & \ldots & w_{1N_{j-1}}^{(j)} \\ b_2^{(j)} & w_{21}^{(j)} & w_{22}^{(j)} & \ldots & w_{2N_{j-1}}^{(j)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ b_{N_j}^{(j)} & w_{N_j1}^{(j)} & w_{N_j2}^{(j)} & \ldots & w_{N_jN_{j-1}}^{(j)} \end{bmatrix}_{(N_j)\times(N_{j-1}+1)} = \begin{bmatrix} \mathbf{w}_1^{(j)} \\ \mathbf{w}_2^{(j)} \\ \vdots \\ \mathbf{w}_{N_j}^{(j)} \end{bmatrix}$$

7 Cálculo progressivo para uma rede MLP genérica

- Vetor de saídas dos CLs da camada j:

$$\mathbf{v}^{(j)} = \begin{bmatrix} \mathbf{w}_1^{(j)}\mathbf{x}^{(j)} \\ \mathbf{w}_2^{(j)}\mathbf{x}^{(j)} \\ \vdots \\ \mathbf{w}_{N_j}^{(j)}\mathbf{x}^{(j)} \end{bmatrix} = \mathbf{W}^{(j)}\mathbf{x}^{(j)}$$

- Vetor de saídas da camada j:

$$\mathbf{y}^{(j)} = \varphi\left(\mathbf{v}^{(j)}\right)$$

em que a função de ativação $\varphi(\cdot)$ é aplicada a cada elemento do vetor $\mathbf{v}^{(j)}$

8 Cálculo regressivo: o algoritmo *backpropagation*

- Erros dos neurônios da camada de saída da rede:

$$e_\ell(n) = d_\ell(n) - y_\ell^{(L)}(n)$$

$\ell = 1, 2, \cdots, N_L$

- Função custo a ser minimizada:

$$J_{\mathrm{MSE}} = \frac{1}{N_L} \sum_{\ell=1}^{N_L} e_\ell^2(n)$$

- Há outras opções de função custo, mas vamos considerar o MSE nesta dedução
- Vamos considerar o modo de treinamento estocástico em que os pesos e *biases* são atualizados a cada dado de treinamento $n = 1, 2, \ldots, N_t$

9 Cálculo regressivo: o algoritmo *backpropagation*

- Método do gradiente estocástico para atualizar os pesos *biases* da camada j:

$$\mathbf{W}^{(j)}(n) = \mathbf{W}^{(j)}(n-1) - \eta \frac{\partial J_{MSE}}{\partial \mathbf{W}^{(j)}(n-1)},$$

em que η é um passo de adaptação

- Matriz de gradientes:

$$\frac{\partial J_{MSE}}{\partial \mathbf{W}^{(j)}(n-1)} = \begin{bmatrix} \frac{\partial J_{MSE}}{\partial b_1^{(j)}(n-1)} & \frac{\partial J_{MSE}}{\partial w_{11}^{(j)}(n-1)} & \frac{\partial J_{MSE}}{\partial w_{12}^{(j)}(n-1)} & \cdots & \frac{\partial J_{MSE}}{\partial w_{1N_{j-1}}^{(j)}(n-1)} \\ \frac{\partial J_{MSE}}{\partial b_2^{(j)}(n-1)} & \frac{\partial J_{MSE}}{\partial w_{21}^{(j)}(n-1)} & \frac{\partial J_{MSE}}{\partial w_{22}^{(j)}(n-1)} & \cdots & \frac{\partial J_{MSE}}{\partial w_{2N_{j-1}}^{(j)}(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial J_{MSE}}{\partial b_{N_j}^{(j)}(n-1)} & \frac{\partial J_{MSE}}{\partial w_{N_j 1}^{(j)}(n-1)} & \frac{\partial J_{MSE}}{\partial w_{N_j 2}^{(j)}(n-1)} & \cdots & \frac{\partial J_{MSE}}{\partial w_{N_j N_{j-1}}^{(j)}(n-1)} \end{bmatrix}$$

10 Cálculo regressivo: o algoritmo *backpropagation*

- Na k-ésima linha da matriz $\frac{\partial J_{MSE}}{\partial \mathbf{W}^{(j)}(n–1)}$ temos o vetor gradiente

$$\boldsymbol{\nabla}_{\mathbf{w}_k^{(j)}} J_{\text{MSE}} = \frac{\partial J_{\text{MSE}}}{\partial \mathbf{w}_k^{(j)}(n-1)} =$$

$$\left[\frac{\partial J_{MSE}}{\partial b_k^{(j)}(n–1)} \quad \frac{\partial J_{MSE}}{\partial w_{k1}^{(j)}(n–1)} \quad \frac{\partial J_{MSE}}{\partial w_{k2}^{(j)}(n–1)} \quad \cdots \quad \frac{\partial J_{MSE}}{\partial w_{kN_{j-1}}^{(j)}(n–1)} \right]$$

- Assim, podemos escrever

$$\frac{\partial J_{MSE}}{\partial \mathbf{W}^{(j)}(n-1)} = \begin{bmatrix} \boldsymbol{\nabla}_{\mathbf{w}_1^{(j)}} J_{MSE} \\ \boldsymbol{\nabla}_{\mathbf{w}_2^{(j)}} J_{MSE} \\ \vdots \\ \boldsymbol{\nabla}_{\mathbf{w}_{N_j}^{(j)}} J_{MSE} \end{bmatrix}$$

11 Cálculo regressivo: o algoritmo *backpropagation*

- Vetor gradiente do Neurônio k da camada de saída ($j = L$):

$$
\begin{aligned}
\boldsymbol{\nabla}_{\mathbf{w}_k^{(L)}} J_{\mathrm{MSE}} &= \frac{\partial J_{\mathrm{MSE}}}{\partial \mathbf{w}_k^{(L)}(n-1)} = \frac{1}{N_L} \sum_{\ell=1}^{N_L} \frac{\partial e_\ell^2(n)}{\partial \mathbf{w}_k^{(L)}(n-1)} \\
&= \frac{1}{N_L} \frac{\partial e_k^2(n)}{\partial \mathbf{w}_k^{(L)}(n-1)} \\
&= \frac{1}{N_L} \; \frac{\partial e_k^2(n)}{\partial y_k^{(L)}(n)} \; \frac{\partial y_k^{(L)}(n)}{\partial v_k^{(L)}(n)} \; \frac{\partial v_k^{(L)}(n)}{\partial \mathbf{w}_k^{(L)}(n-1)} \\
&= \frac{1}{N_L} 2 e_k(n) \frac{\partial [d_k(n) - y_k^{(L)}(n)]}{\partial y_k^{(L)}(n)} \frac{\partial \varphi\left(v_k^{(L)}(n)\right)}{\partial v_k^{(L)}(n)} \frac{\partial \mathbf{w}_k^{(L)}(n-1) \mathbf{x}^{(L)}(n)}{\partial \mathbf{w}_k^{(L)}(n-1)} \\
&= -\frac{2}{N_L} \; e_k(n) \; \varphi'\left(v_k^{(L)}(n)\right) \; [\mathbf{x}^{(L)}(n)]^T,
\end{aligned}
$$

em que $\varphi'(\cdot)$ representa a derivada de $\varphi(\cdot)$, o que justifica a importância dessa função ser derivável em todos os pontos

12 Cálculo regressivo: o algoritmo *backpropagation*

- Gradiente local da Camada L:

$$\delta_k^{(L)}(n) \triangleq \varphi'(v_k^{(L)}(n))e_k(n)$$

- Vetor gradiente do Neurônio k da camada de saída ($j = L$):

$$\boldsymbol{\nabla}_{\mathbf{w}_k^{(L)}} J_{\mathrm{MSE}} = -\frac{2}{N_L}\, \delta_k^{(L)}(n)\, [\mathbf{x}^{(L)}(n)]^T$$

13 Cálculo regressivo: o algoritmo *backpropagation*

- Gradientes da última camada oculta ($j = L - 1$):

$$\nabla_{\mathbf{w}_k^{(L-1)}} J_{\mathrm{MSE}} = \frac{\partial J_{\mathrm{MSE}}}{\partial \mathbf{w}_k^{(L-1)}(n-1)} = \frac{1}{N_L} \sum_{\ell=1}^{N_L} \frac{\partial e_\ell^2(n)}{\partial \mathbf{w}_k^{(L-1)}(n-1)}$$

- Usando a regra da cadeia sucessivas vezes:

$$\begin{aligned}
\frac{\partial e_\ell^2(n)}{\partial \mathbf{w}_k^{(L-1)}(n-1)} &= 2e_\ell(n) \frac{\partial [d_\ell(n) - y_\ell^{(L)}(n)]}{\partial y_\ell^{(L)}(n)} \frac{\partial y_\ell^{(L)}(n)}{\partial \mathbf{w}_k^{(L-1)}(n-1)} \\
&= -2e_\ell(n) \frac{\partial \varphi(v_\ell^{(L)}(n))}{\partial v_\ell^{(L)}(n)} \frac{\partial v_\ell^{(L)}(n)}{\partial \mathbf{w}_k^{(L-1)}(n-1)}
\end{aligned}$$

14 Cálculo regressivo: o algoritmo *backpropagation*

- Observe que
$$v_\ell^{(L)}(n) = \mathbf{w}_\ell^{(L)}(n-1)\mathbf{x}^{(L)}(n)$$
$$= b_\ell^{(L)}(n-1) + \sum_{m=1}^{N_{L-1}} w_{\ell m}^{(L)}(n-1) y_m^{(L-1)}(n)$$
- No cálculo de $v_\ell^{(L)}(n)$, o único termo que depende de $\mathbf{w}_k^{(L-1)}(n-1)$ é $w_{\ell k}^{(L)}(n-1)y_k^{(L-1)}(n)$, já que a saída $y_k^{(L-1)}(n)$ é calculada utilizando os pesos $\mathbf{w}_k^{(L-1)}(n-1)$

15 Cálculo regressivo: o algoritmo *backpropagation*

- Assim, obtemos

$$\frac{\partial e_\ell^2(n)}{\partial \mathbf{w}_k^{(L-1)}(n-1)} = -2e_\ell(n)\varphi'(v_\ell^{(L)}(n))w_{\ell k}^{(L)}(n-1)\frac{\partial y_k^{(L-1)}(n)}{\partial \mathbf{w}_k^{(L-1)}(n-1)}$$

$$= -2e_\ell(n)\varphi'(v_\ell^{(L)}(n))w_{\ell k}^{(L)}(n-1)\frac{\partial \varphi(v_k^{(L-1)}(n))}{\partial v_k^{(L-1)}(n)}\frac{\partial v_k^{(L-1)}(n)}{\partial \mathbf{w}_k^{(L-1)}(n-1)}$$

$$= -2e_\ell(n)\varphi'(v_\ell^{(L)}(n))w_{\ell k}^{(L)}(n-1)\varphi'(v_k^{(L-1)}(n))\frac{\partial \mathbf{w}_k^{(L-1)}(n-1)\mathbf{x}^{(L-1)}(n)}{\partial \mathbf{w}_k^{(L-1)}(n-1)}$$

$$= -2e_\ell(n)\varphi'(v_\ell^{(L)}(n))w_{\ell k}^{(L)}(n-1)\varphi'(v_k^{(L-1)}(n))[\mathbf{x}^{(L-1)}(n)]^T$$

16 Cálculo regressivo: o algoritmo *backpropagation*

- Identificando $\delta_\ell^{(L)}(n)$ na expressão anterior:

$$\boldsymbol{\nabla}_{\mathbf{w}_k^{(L-1)}} J_{\mathrm{MSE}} = -\frac{2}{N_L}\, \varphi'(v_k^{(L-1)}(n))\, \sum_{\ell=1}^{N_L} \delta_\ell^{(L)}(n) w_{\ell k}^{(L)}(n{-}1)[\mathbf{x}^{(L-1)}(n)]^T$$

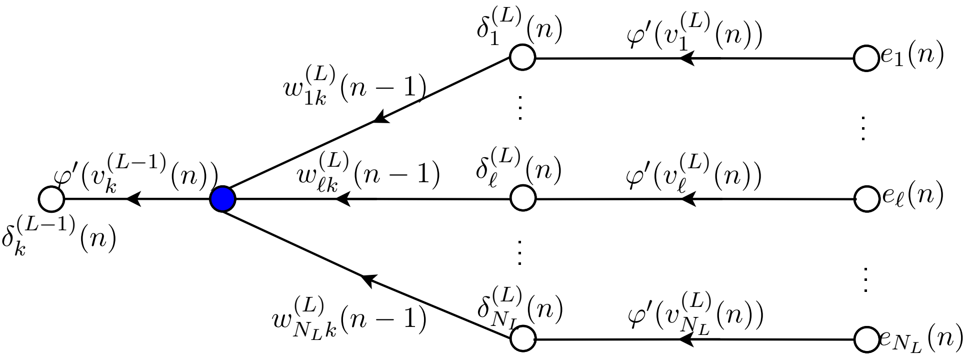
- Gradiente local da Camada $L-1$:

$$\delta_k^{(L-1)}(n) \triangleq \varphi'(v_k^{(L-1)}(n))\, \sum_{\ell=1}^{N_L} \delta_\ell^{(L)}(n) w_{\ell k}^{(L)}(n-1)$$

- Vetor gradiente:

$$\boldsymbol{\nabla}_{\mathbf{w}_k^{(L-1)}} J_{\mathrm{MSE}} = -\frac{2}{N_L} \delta_k^{(L-1)}(n)[\mathbf{x}^{(L-1)}(n)]^T$$

17 Cálculo regressivo: o algoritmo *backpropagation*



Fluxo do sinal na retropropagação considerando as camadas L e $L-1$

18 Cálculo regressivo: o algoritmo *backpropagation*

- Generalizando, define-se o gradiente local para qualquer camada oculta j como

$$\boxed{\delta_k^{(j)}(n) \triangleq \varphi'(v_k^{(j)}(n)) \sum_{\ell=1}^{N_{j+1}} \delta_\ell^{(j+1)}(n) w_{\ell k}^{(j+1)}(n-1).}$$

e para a camada de saída L como

$$\boxed{\delta_k^{(L)}(n) \triangleq \varphi'(v_k^{(L)}(n)) e_k(n).}$$

19 Cálculo regressivo: o algoritmo *backpropagation*

Definem-se os vetores

$$\boldsymbol{\delta}^{(j)}(n) \triangleq \begin{bmatrix} \delta_1^{(j)}(n) \\ \delta_2^{(j)}(n) \\ \vdots \\ \delta_{N_j}^{(j)}(n) \end{bmatrix}, \mathbf{e}(n) \triangleq \begin{bmatrix} e_1(n) \\ e_2(n) \\ \vdots \\ e_{N_L}(n) \end{bmatrix}, \mathbf{d}_{\varphi}^{(j)}(n) \triangleq \begin{bmatrix} \varphi'(v_1^{(j)}(n)) \\ \varphi'(v_2^{(j)}(n)) \\ \vdots \\ \varphi'(v_{N_j}^{(j)}(n)) \end{bmatrix}$$

e a matriz $\mathbf{W}^{(j+1)}(n-1)$ de dimensões $N_{j+1} \times N_j$ (sem a coluna de *biases*):

$$\overline{\mathbf{W}}^{(j+1)}(n-1) \triangleq \begin{bmatrix} w_{11}^{(j+1)}(n-1) & w_{12}^{(j+1)}(n-1) & \ldots & w_{1N_j}^{(j+1)}(n-1) \\ w_{21}^{(j+1)}(n-1) & w_{22}^{(j+1)}(n-1) & \ldots & w_{2N_j}^{(j+1)}(n-1) \\ \vdots & \vdots & \ddots & \vdots \\ w_{N_{j+1}1}^{(j+1)}(n-1) & w_{N_{j+1}2}^{(j+1)}(n-1) & \ldots & w_{N_{j+1}N_j}^{(j+1)}(n-1) \end{bmatrix}.$$

20 Cálculo regressivo: o algoritmo *backpropagation*

Assim, para a camada de saída temos

$$\boxed{\boldsymbol{\delta}^{(L)}(n) = \mathbf{d}_{\varphi}^{(L)}(n) \odot \mathbf{e}(n)}$$

e para as camadas ocultas

$$\boxed{\boldsymbol{\delta}^{(j)}(n) = \mathbf{d}_{\varphi}^{(j)}(n) \odot \left\{ \left[\overline{\mathbf{W}}^{(j+1)}(n-1) \right]^T \; \boldsymbol{\delta}^{(j+1)}(n) \right\}}$$

em que $\odot$ representa a multiplicação elemento por elemento entre dois vetores.

21 Cálculo regressivo: o algoritmo *backpropagation*

- É comum incorporar $2/N_L$ ao passo de adaptação η
- Atualização dos pesos do Neurônio k da Camada j:

$$\boxed{\mathbf{w}_k^{(j)}(n) = \mathbf{w}_k^{(j)}(n-1) + \eta\delta_k^{(j)}(n)[\mathbf{x}^{(j)}(n)]^T}$$

$k = 1, 2, \ldots, N_j, \;\; j = 1, 2 \ldots, L.$

- Definindo a matriz

$$\mathbf{\Delta}_\delta^{(j)}(n) = \boldsymbol{\delta}^{(j)}(n)[\mathbf{x}^{(j)}(n)]^T$$

a atualização da matriz de pesos da camada j fica

$$\boxed{\mathbf{W}^{(j)}(n) = \mathbf{W}^{(j)}(n-1) + \eta\mathbf{\Delta}_\delta^{(j)}(n).}$$

- Como a implementação matricial é mais eficiente, essa forma de atualização é preferida

22 Inicialização

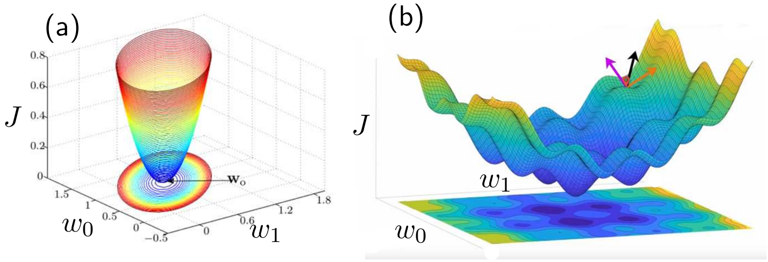
- Se inicializarmos os pesos e *biases* com zero (como no LMS), dependendo da função de ativação, esses parâmetros não serão atualizados
- Para exemplificar, vamos considerar a tangente hiperbólica

$$\varphi(v) = \tanh(v) = \frac{e^v - e^{-v}}{e^v + e^{-v}}$$

  Se inicializarmos os pesos e *biases* com zero como $\tanh(0) = 0$, os gradientes serão nulos e a atualização dos pesos e *biases* não ocorre
- Costuma-se inicializar esses parâmetros a partir de uma distribuição uniforme, por exemplo, valores uniformemente distribuídos no intervalo $[-10^{-2}, 10^{-2}]$

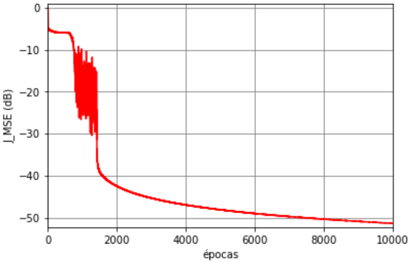
23 Mínimos locais

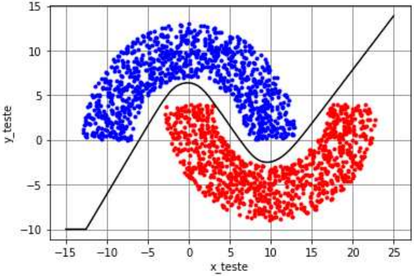
- Diferente do LMS, a função custo minimizada pelo *backpropagation* tem inúmeros mínimos locais
- Não temos mais uma função convexa e o gradiente é nulo nesses pontos de mínimo
- O *backpropagation* para de atualizar nesses pontos e atinje uma solução subótima



a) Função custo do MSE a ser minimizada pelo LMS; b) Função custo do MSE, a ser minimizada pelo *backpropagation*. Fonte: [https://www.youtube.com/watch?v=Suevq-kZdIw]

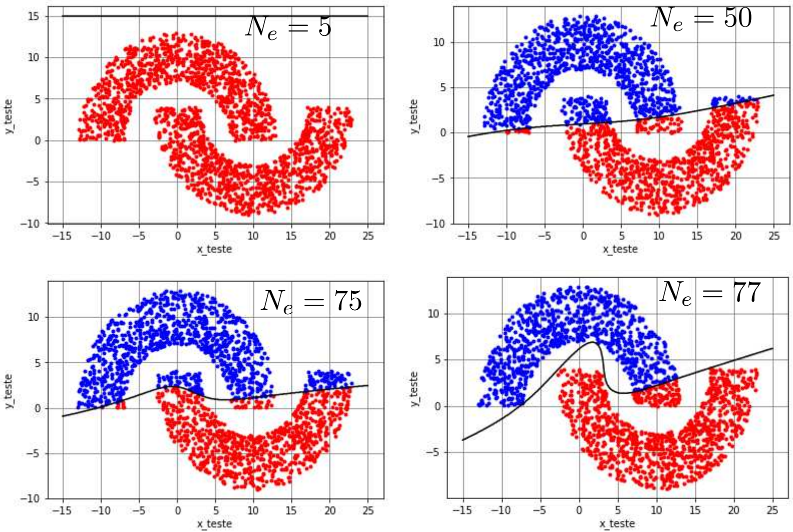
24 MLP nas meias-luas



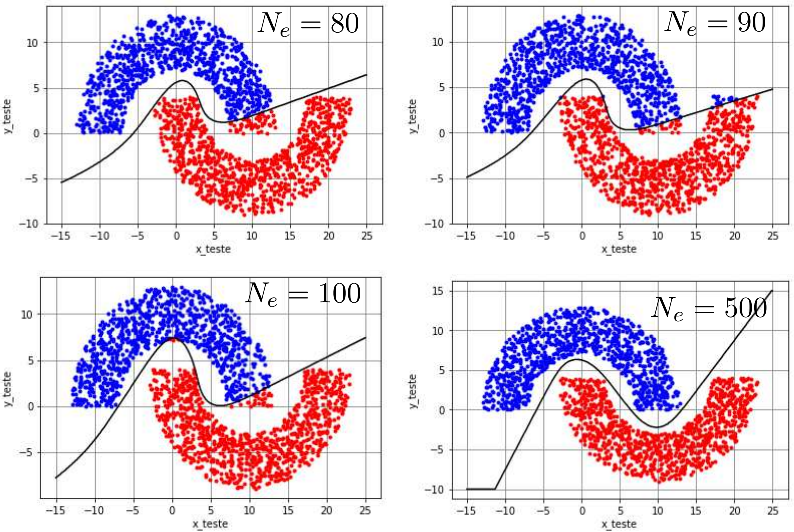


O problema de classificação das meias-luas ($r_1 = 10$, $r_2 = -4$ e $r_3 = 6$). Função custo ao longo das épocas de treinamento (figura à esquerda);

25 Evolução da fronteira de separação



26 Evolução da fronteira de separação

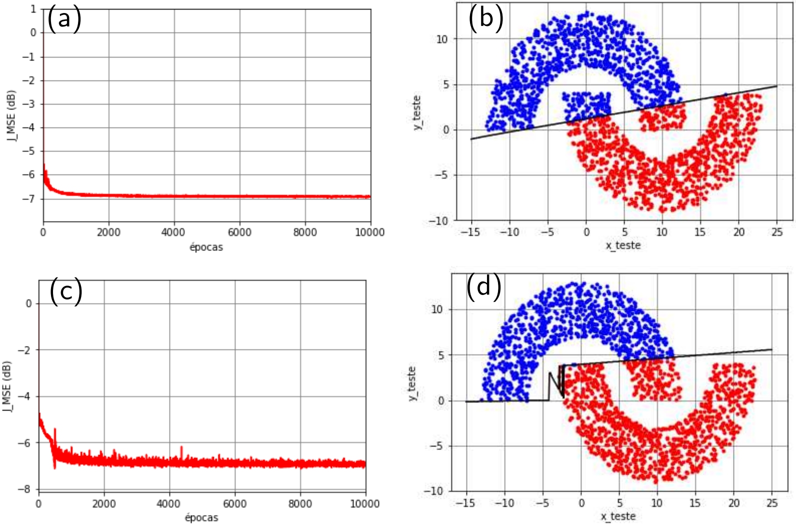


27 Evolução da fronteira de separação

Taxas de erro de classificação

N_e	5	50	75	77	80	90	100	500
Taxa de erro (%)	50,00	10,20	9,35	4,85	2,85	2,20	1,55	0,00

28 Mínimos locais



Meias-luas ($r_1 = 10$, $r_2 = -4$ e $r_3 = 6$). (a) e (c) Função custo ao longo das épocas de treinamento; (b) e (d) Dados de teste ($N_{\text{teste}} = 2 \times 10^3$) e curva de separação das regiões obtida com uma rede MLP (3–2–1) treinada em *mini-batch* com o algoritmo *backpropagation* ($N_0 = 2$, $\eta = 0{,}1$, $N_t = 10^3$, $N_b = 50$ e $N_e = 10^4$); função de ativação tangente hiperbólica e pesos e *biases* inicializados com distribuição uniforme no intervalo $[-10^{-2}, 10^{-2}]$.

29 Mínimos locais

- A rede MLP é capaz de proporcionar soluções não lineares que podem ser adequadas a vários problemas de classificação e regressão
- É importe utilizar técnicas que façam com que o algoritmo *backpropagation* não fique parado em mínimos locais

30 MLP como aproximador universal de funções

- Uma rede MLP treinada com o algoritmo *backpropagation* realiza um mapeamento entrada-saída de forma não linear
- Considere uma rede MLP com N_0 entradas e N_L saídas:
  - A relação entrada-saída da rede define um mapeamento de um espaço Euclidiano de entrada de dimensão N_0 a um espaço Euclidiano de saída de dimensão N_L
  - O espaço de dimensão N_L é infinitamente e continuamente diferenciável desde que a função de ativação também seja
- Qual o número mínimo de camadas ocultas que a rede MLP precisa ter para fornecer uma aproximação de qualquer mapeamento contínuo?
- A resposta para essa pergunta envolve o Teorema da Aproximação Universal.

31 Teorema da Aproximação Universal

Seja $\varphi(\cdot)$ uma função contínua, não constante, limitada e monotônica crescente. Vamos utilizar I_{N_0} para denotar o hipercubo unitário $[0, 1]^{N_0}$ de dimensão N_0. O espaço de funções contínuas em I_{N_0} é denotado por $C(I_{N_0})$. Então, dada qualquer função $f \in C(I_{N_0})$ e $\varepsilon > 0$, existe um inteiro N_1 e conjuntos de constantes reais α_i, b_i e w_{ij}, $i = 1, 2, \ldots, N_1$ e $j = 1, 2, \ldots, N_0$ tal que se pode definir

$$F(x_1, x_2, \ldots, x_{N_0}) = \sum_{i=1}^{N_1} \alpha_i \varphi \left(\sum_{j=1}^{N_0} w_{ij} x_j + b_i \right)$$

como uma aproximação da função $f(\cdot)$, ou seja,

$$|F(x_1, x_2, \ldots, x_{N_0}) - f(x_1, x_2, \ldots, x_{N_0})| < \varepsilon$$

para todos $x_1, x_2, \ldots, x_{N_0}$ do espaço de entrada.

32 MLP e o Teorema da Aproximação Universal

O Teorema da Aproximação Universal é diretamente aplicável à rede MLP:

- A tangente hiperbólica satisfaz as condições impostas para a função $\varphi(\cdot)$: é não constante, limitada e monotonicamente crescente
- $$\sum_{i=1}^{N_1} \alpha_i \varphi \left(\sum_{j=1}^{N_0} w_{ij} x_j + b_i \right)$$

  representa a saída de uma MLP descrita a seguir:

  1. a rede possui N_0 entradas, indicadas por $x_1, x_2, \ldots, x_{N_0}$, e uma única camada oculta composta por N_1 neurônios
  2. o neurônio oculto i tem pesos $w_{i1}, w_{i2}, \ldots, w_{iN_0}$ e *bias* b_i;
  3. a saída da rede é uma combinação linear das saídas dos neurônios ocultos, com α_1, $\alpha_2, \ldots,$ α_{N_1} sendo os pesos da saída

33 MLP como aproximador universal

- Uma única camada oculta é suficiente para que uma rede MLP obtenha uma aproximação uniforme para um determinado conjunto de treinamento, representado pelas entradas $x_1, x_2, \ldots, x_{N_0}$, e uma saída desejada $f(x_1, x_2, \ldots, x_{N_0})$
- As redes MLP são conhecidas como aproximadores universais de funções
- Esse resultado é conhecido como Teorema de Cybenko (1988)
- O teorema não nos diz que uma única camada oculta é ótima no sentido de tempo de aprendizado, simplicidade de implementação ou capacidade de generalização.
- Mais detalhes podem ser encontrados, por exemplo, em [Haykin, 2009].