Perceptron Multicamadas e o Algoritmo de Retropropagação do Erro

Prof. Dr. Guilherme de Alencar Barreto

Agosto /2020

Departmento de Engenharia de Teleinformática
Programa de Pós-Graduação em Engenharia de Teleinformática (PPGETI)
Universidade Federal do Ceará (UFC), Fortaleza-CE

gbarreto@ufc.br

1 Definições Preliminares

De início, vamos assumir que existe uma lei matemática $\mathbf{F}(\cdot)$, também chamada aqui de função ou mapeamento, que relaciona um vetor de entrada qualquer, $\mathbf{x} \in \mathbb{R}^{p+1}$, com um vetor de saída, $\mathbf{d} \in \mathbb{R}^q$. Esta relação, representada genericamente na Figura 1, pode ser descrita matematicamente da seguinte forma:

$$\mathbf{d} = \mathbf{F}[\mathbf{x}] \tag{1}$$

em que se assume que $\mathbf{F}(\cdot)$ é totalmente desconhecida, ou seja, não sabemos de antemão quais são as *fórmulas* usadas para associar um vetor de entrada $\mathbf{x}$ com seu vetor de saída $\mathbf{d}$ correspondente.

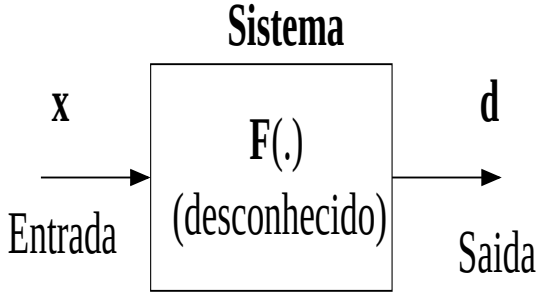

Figura 1: Representação simplificada de um mapeamento entrada-saída genérico.

O mapeamento $\mathbf{F}(\cdot)$ pode ser tão simples quanto um mapeamento linear, tal como

$$\mathbf{d} = \mathbf{M}\mathbf{x} \tag{2}$$

em que $\mathbf{M}$ é uma matriz de dimensão $(p+1) \times q$. Contudo, $\mathbf{F}(\cdot)$ pode ser bastante complexo, envolvendo relações não-lineares entre as variáveis de entrada e saída. É justamente o funcionamento da relação matemática $\mathbf{F}(\cdot)$ que se deseja *imitar* através do uso de algoritmos adaptativos, tais como as redes neurais.

Supondo que a única fonte de informação que nós temos a respeito de $\mathbf{F}(\cdot)$ é conjunto finito de N pares entrada-saída observados (ou medidos), ou seja:

$$\begin{array}{cc} \mathbf{x}_1, & \mathbf{d}_1 \\ \mathbf{x}_2, & \mathbf{d}_2 \\ \vdots & \vdots \\ \mathbf{x}_N, & \mathbf{d}_N \end{array} \tag{3}$$

Os pares entrada-saída mostrados acima podem ser representados de maneira simplificada como $\{\mathbf{x}_\mu, \mathbf{d}_\mu\}$, em que μ é um apenas indíce simbolizando o μ-ésimo par do conjunto de dados. Uma maneira de se adquirir conhecimento sobre $\mathbf{F}(\cdot)$ se dá exatamente através dos uso destes pares.

Para isto pode-se utilizar uma rede neural qualquer para implementar um mapeamento entrada-saída aproximado, representado como $\hat{\mathbf{F}}(\cdot)$, tal que

$$\mathbf{y}_\mu = \hat{\mathbf{F}}[\mathbf{x}_\mu] \tag{4}$$

em que $\mathbf{y}_\mu$ é a saída gerada pela rede neural em resposta ao vetor de entrada $\mathbf{x}_\mu$. Esta saída, espera-se, seja muito próxima da saída real $\mathbf{d}_\mu$. Dá-se o nome de *Aprendizado Indutivo* ao processo de obtenção da relação matemática geral $\hat{\mathbf{F}}(\cdot)$ a partir de apenas alguns pares $\{\mathbf{x}_\mu, \mathbf{d}_\mu\}$ disponíveis.

A seguir será mostrado uma das principais arquiteturas de redes neurais adaptativas usadas com o propósito de obter uma representação aproximada de um mapeamento entrada-saída generíco.

2 Perceptron Multicamadas

Estamos considerando nas definições e cálculos a seguir uma arquitetura de rede neural do tipo **perceptron multicamadas** (MLP - *multilayer perceptron*) com apenas uma camada oculta de neurônios treinados com o algoritmo de **retropropagação do erro** (*Error Backpropagation*).

Os neurônios da camada oculta (primeira camada de processamento) são representados conforme mostrado na Figura 2a, enquanto os neurônios da camada de saída segunda camada de processamento) são representados conforme mostrado na Figura 2b.

O vetor de pesos associado a cada neurônio i da camada oculta, também chamada de *camada escondida* ou *camada intermediária*, é representado como

$$\mathbf{w}_i = \begin{pmatrix} w_{i0} \\ \vdots \\ w_{ip} \end{pmatrix} = \begin{pmatrix} \theta_i \\ \vdots \\ w_{ip} \end{pmatrix} \tag{5}$$

em que θ_i é o limiar (*bias* ou *threshold*) associado ao neurônio i. Os neurônios desta camada são chamados de neurônios ocultos por não terem acesso direto à sáida da rede MLP, onde são calculados os erros.

De modo semelhante, o vetor de pesos associado a cada neurônio k da camada de saída é representado como

$$\mathbf{m}_k = \begin{pmatrix} m_{k0} \\ \vdots \\ m_{kq} \end{pmatrix} = \begin{pmatrix} \theta_k \\ \vdots \\ m_{kq} \end{pmatrix} \tag{6}$$

em que θ_k é o limiar associado ao neurônio de saída k.

O treinamento da rede MLP se dá em duas etapas, que são descritas a seguir.

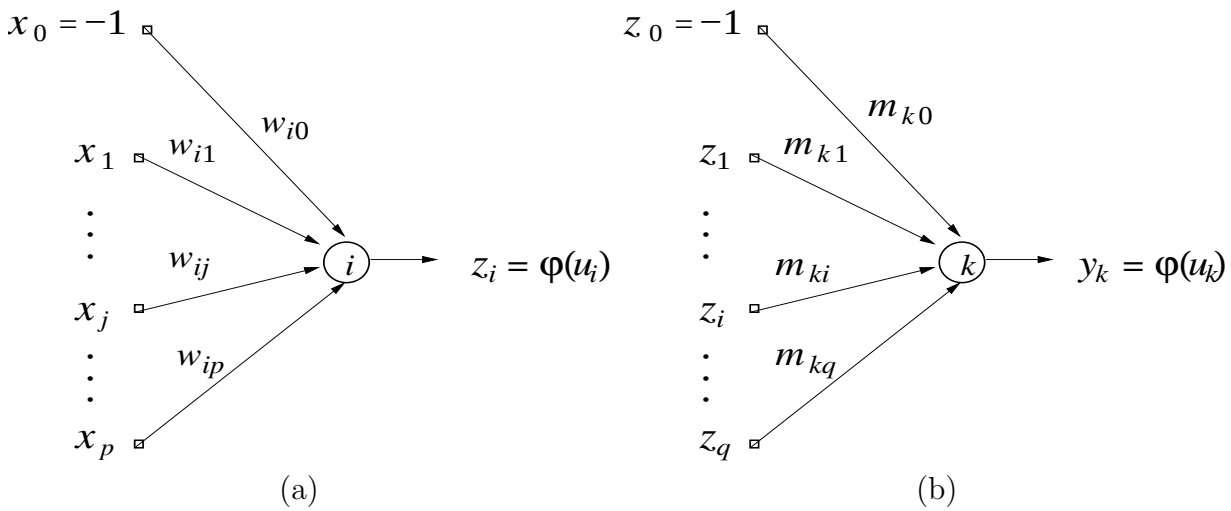

Figura 2: (a) Neurônio da camada oculta. (b) Neurônio da camada de saída.

3 Fase 1: Sentido Direto

Esta etapa de funcionamento da rede MLP envolve o cálculo das ativações e saídas de todos os neurônios da camada oculta e de todos os neurônios da camada de saída. Assim, o fluxo da informação se dá das unidades de entrada para os neurônios de saída, passando pelos neurônios da camada oculta. Por isso, diz-se que o informação está fluindo no sentido **direto** (*forward*), ou seja

Entrada → Camada Oculta → Camada de Saída

Assim, após a apresentação de um vetor de entrada $\mathbf{x}$, na iteração t, o primeiro passo é calcular as ativações dos neurônios da camada oculta:

$$u_i(t) = \sum_{j=0}^{p} w_{ij}(t)x_j(t) = \mathbf{w}_i^T(t)\mathbf{x}(t), \quad i = 1, \ldots, q \tag{7}$$

em que T indica o vetor (ou matriz) transposto e q indica o número de neurônios da camada oculta. Em seguida, as saídas correspondentes são calculadas como

$$z_i(t) = \phi(u_i(t)) = \phi\left(\mathbf{w}_i^T(t)\mathbf{x}(t)\right) = \phi\left(\sum_{j=0}^{p} w_{ij}(t)x_j(t)\right) \tag{8}$$

tal que a função de ativação ϕ assume geralmente uma das seguintes formas:

$$z_i(t) = \frac{1}{1+\exp[-u_i(t)]}, \qquad \text{(sigmóide logística)} \tag{9}$$

$$z_i(t) = \frac{1-\exp[-u_i(t)]}{1+\exp[-u_i(t)]}, \qquad \text{(tangente hiperbólica)} \tag{10}$$

As ativações e saídas dos q neurônios ocultos podem ser escritas em forma matricial, permitindo uma implementação mais compacta em ambientes de programação Octave/Matlab/Scilab. Assim, o vetor de ativações e o vetor de saídas dos neurônios ocultos no instante t são dados por

$$\mathbf{u}(t) = \mathbf{W}(t)\mathbf{x}(t) \qquad \text{e} \qquad \mathbf{z}(t) = \phi(\mathbf{u}(t)), \tag{11}$$

em que a matriz de parâmetros ajustáveis (i.e. pesos e limiares) $\mathbf{W}(t)$ tem dimensões $q \times (p+1)$, sendo definida como

$$\mathbf{W}(t) = \begin{bmatrix} \mathbf{w}_1^T(t) \\ \mathbf{w}_2^T(t) \\ \vdots \\ \mathbf{w}_i^T(t) \\ \vdots \\ \mathbf{w}_q^T(t) \end{bmatrix}_{q\times(p+1)} = \begin{bmatrix} w_{10} & w_{11} & w_{12} & \cdots & w_{1p} \\ w_{20} & w_{21} & w_{22} & \cdots & w_{2p} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ w_{i0} & w_{i1} & w_{i2} & \cdots & w_{ip} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ w_{q0} & w_{q1} & w_{q2} & \cdots & w_{qp} \end{bmatrix}_{q\times(p+1)}, \tag{12}$$

de tal forma que a j-ésima linha contém o vetor de pesos do j-ésimo neurônio oculto, $j = 1, \ldots, q$.

O segundo passo consiste em repetir as operações das Eqs. (7) e (8) para os neurônios da camada de saída:

$$a_k(t) = \sum_{i=0}^{q} m_{ki}(t) z_i(t) = \mathbf{m}_k^T(t)\bar{\mathbf{z}}(t), \qquad k = 1, \ldots, c \tag{13}$$

em que c é o número de neurônios de saída. O vetor $\bar{\mathbf{z}}(t) = [-1 \;\; \mathbf{z}(t)]^T$ inclui a entrada $z_0(t) = -1$ correspondente aos limiares dos neurônios de saída. Note que as saídas dos neurônios da camada oculta, $z_i(t)$, fazem o papel de entrada para os neurônios da camada de saída.

Em seguida, as saídas dos neurônios da camada de saída são calculadas como

$$y_k(t) = \phi(a_k(t)) = \phi\left(\mathbf{m}_k^T(t)\bar{\mathbf{z}}(t)\right) = \phi\left(\sum_{i=0}^{q} m_{ki}(t) z_i(t)\right) \tag{14}$$

tal que a função de ativação ϕ assume geralmente uma das formas definidas nas Eqs. (9) e (10).

De modo similar ao que foi feito para as ativações e saídas dos neurônios ocultos, é possível escrevê-las na forma vetorial da seguinte forma:

$$\mathbf{a}(t) = \mathbf{M}(t)\bar{\mathbf{z}}(t) \qquad \text{e} \qquad \mathbf{y}(t) = \phi(\mathbf{a}(t)), \tag{15}$$

tal que a matriz de parâmetros ajustáveis (i.e. pesos e limiares) $\mathbf{M}(t)$ tem dimensões $c \times (q+1)$, sendo definida como

$$\mathbf{M}(t) = \begin{bmatrix} \mathbf{m}_1^T(t) \\ \mathbf{m}_2^T(t) \\ \vdots \\ \mathbf{m}_k^T(t) \\ \vdots \\ \mathbf{m}_c^T(t) \end{bmatrix}_{c\times(q+1)} = \begin{bmatrix} m_{10} & m_{11} & m_{12} & \cdots & m_{1q} \\ m_{20} & m_{21} & m_{22} & \cdots & m_{2q} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ m_{k0} & m_{k1} & m_{k2} & \cdots & m_{kq} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ m_{c0} & m_{c1} & m_{c2} & \cdots & m_{cq} \end{bmatrix}_{c\times(q+1)}, \tag{16}$$

de tal forma que a k-ésima linha contém o vetor de pesos do k-ésimo neurônio de saída, $k = 1, \ldots, c$.

O diagrama de fluxo de sinais mostrado na Figura 3 ilustra as etapas sucessivas que compõem o processamento da informação ao longo do sentido direto da rede MLP. A representação vetorial/matricial facilita o entendimento dos vários estágios desse processamento, além de fornecer um modo de implementação mais eficiente em ambientes de programação a la Octave/Matlab/Scilab. Neste, sentido o vetor de saídas no instante t é rapidamente calculado como

$$\mathbf{y}(t) = \phi\left(\mathbf{M}\phi\left(\mathbf{W}\mathbf{x}(t)\right)\right). \tag{17}$$

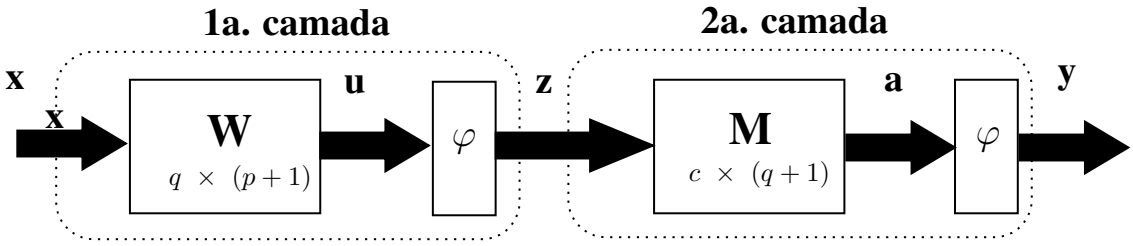

Figura 3: Diagrama de fluxo de sinais da fase direta da rede MLP(p,q,c).

Comentário 1 - A partir da Eq. (17) pode-se perceber facilmente a importância da função de ativação linear ϕ na execução de uma transformação não-linear entre as camadas successivas de processamento. Caso a função ϕ entre a camada oculta e a de saída fosse linear (e.g. $\phi(u_i) = u_i$), a saída da rede seria dada por

$$\mathbf{y}(t) = \phi\left(\mathbf{MWx}(t)\right) = \phi\left(\mathbf{Kx}\right), \tag{18}$$

em que $\mathbf{K}$ é a matriz resultante do produto das matrizes $\mathbf{M}$ e $\mathbf{W}$. Em termos computacionais, isto equivale a dizer que a saída da rede MLP seria equivalente à saída de uma rede perceptron simples. Em outras palavras, se não existisse a transformação não-linear entre camadas, não haveria ganhos de desempenho ao se ter uma rede com arquitetura multicamadas.

4 Fase 2: Sentido Inverso

Esta etapa de funcionamento da rede MLP envolve o cálculo dos gradientes locais e o ajuste dos pesos de todos os neurônios da camada oculta e da camada de saída. Assim, o fluxo de sinais (informação) se dá dos neurônios de saída para os neurônios da camada oculta. Por isso, diz-se que o informação está fluindo no sentido **inverso** (*backward*), ou seja:

Camada de Saída $\rightarrow$ Camada Oculta

Assim, após os cálculos das ativações e saídas levados a cabo na Fase 1, o primeiro passo da Fase 2 consiste em calcular os gradientes locais dos neurônios da camada de saída:

$$\delta_k(t) = e_k(t)y'_k(t), \quad k = 1, \ldots, c \tag{19}$$

em que $y'_k(t) = \frac{dy_k(t)}{du_k(t)} = \frac{d\phi(u_k(t))}{du_k(t)}$ é a derivada instantânea[1] do k-ésimo neurônio de saída no instante t, e $e_k(t)$ é o erro entre a saída desejada $d_k(t)$ do k-ésimo neurônio e a sua saída gerada, $y_k(t)$:

$$e_k(t) = d_k(t) - y_k(t), \qquad k = 1, \ldots, c \tag{20}$$

A derivada $\phi'(u_k(t))$ assume diferentes formas, dependendo da escolha da função de ativação. Assim, temos as seguintes possibilidades:

$$y'_k(t) = y_k(t)[1 - y_k(t)], \quad \text{Se } \phi(u_k(t)) \text{ é a função logística} \tag{21}$$

$$y'_k(t) = \frac{1}{2}(1 - y_k^2(t)), \quad \text{Se } \phi(u_k(t)) \text{ é a tangente hiperbólica} \tag{22}$$

[1]Derivada no instante atual t.

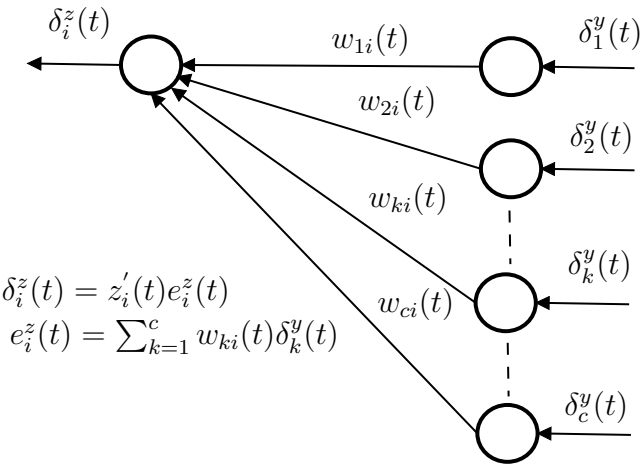

Figura 4: Cálculo do gradiente local do i-ésimo neurônio oculto ($\delta_i^z(t)$) na iteração t.

É útil organizar os erros, as derivadas e os gradientes locais dos neurônios de saída na forma vetorial. Deste modo, o vetor de gradientes locais ($\boldsymbol{\delta}^y(t)$), o vetor de erros ($\mathbf{e}^y(t)$), e o vetor de derivadas da saída ($\mathbf{y}'(t)$) são definidos como

$$\boldsymbol{\delta}^y(t) = \begin{bmatrix} \delta_1^y(t) \\ \vdots \\ \delta_k^y(t) \\ \vdots \\ \delta_c^y(t) \end{bmatrix}_{c\times 1} = \begin{bmatrix} e_1(t)y_1'(t) \\ \vdots \\ e_k(t)y_k'(t) \\ \vdots \\ e_c(t)y_c'(t) \end{bmatrix}_{c\times 1} = \mathbf{e}^y(t) \circ \mathbf{y}'(t), \tag{23}$$

em que o símbolo ($\circ$) denota o produto de Hadamard, ou seja, o produto componente-a-componente entre dois vetores.

O segundo passo da Fase 2 consiste em calcular o gradiente local do i-ésimo neurônio da camada oculta, $i = 1, \ldots, q$:

$$\delta_i^z(t) = z_i'(t) \sum_{k=1}^{c} m_{ki}(t)\delta_k^y(t), \tag{24}$$

$$= z_i'(t)e_i^z(t), \tag{25}$$

em que $e_i^z(t) = \sum_{k=1}^c m_{ki}(t)\delta_k^y(t)$ pode ser entendido como o erro do i-ésimo neurônio oculto, computado a partir da combinação linear dos gradientes locais dos neurônios da camada oculta. Esta operação está ilustrada na Fig. 4.

A derivada $z_i'(t) = \phi'(u_i(t))$ pode ser calculada por uma das seguintes formas:

$$z_i'(t) = \frac{dz_i(t)}{du_i(t)} = z_i(t)[1 - z_i(t)], \quad \text{Se } \phi(\cdot) \text{ é a função logística} \tag{26}$$

$$z_i'(t) = \frac{dz_i(t)}{du_i(t)} = \frac{1}{2}[1 - z_i^2(t)], \quad \text{Se } \phi(\cdot) \text{ é a tangente hiperbólica} \tag{27}$$

O vetor de gradientes locais dos neurônios ocultos ($\boldsymbol{\delta}^z(t)$) pode ser computado por meio da

seguinte operação:

$$\boldsymbol{\delta}^z(t) = \begin{bmatrix} \delta_1^z(t) \\ \vdots \\ \delta_i^z(t) \\ \vdots \\ \delta_q^z(t) \end{bmatrix}_{q\times 1} = \mathbf{z}'(t) \circ \left(\mathbf{M}^T_{[:,2:q+1]}\boldsymbol{\delta}^y(t)\right) \tag{28}$$

$$= \begin{bmatrix} z_1'(t) \\ z_2'(t) \\ \vdots \\ z_i'(t) \\ \vdots \\ z_q'(t) \end{bmatrix}_{q\times 1} \circ \left(\begin{bmatrix} m_{11}(t) & m_{21}(t) & \cdots & m_{k1}(t) & \cdots & m_{c1}(t) \\ m_{12}(t) & m_{22}(t) & \cdots & m_{k2}(t) & \cdots & m_{c2}(t) \\ \vdots & \vdots & \cdots & \vdots & \cdots & \vdots \\ m_{1k}(t) & m_{2k}(t) & \cdots & m_{ki}(t) & \cdots & m_{ck}(t) \\ \vdots & \vdots & \cdots & \vdots & \cdots & \vdots \\ m_{1q}(t) & m_{2q}(t) & \cdots & m_{kq}(t) & \cdots & m_{cq}(t) \end{bmatrix}_{q\times c} \cdot \begin{bmatrix} \delta_1^y(t) \\ \delta_2^y(t) \\ \vdots \\ \delta_k^y(t) \\ \vdots \\ \delta_c^y(t) \end{bmatrix}_{c\times 1} \right) \tag{29}$$

$$= \begin{bmatrix} z_1'(t) \\ z_2'(t) \\ \vdots \\ z_i'(t) \\ \vdots \\ z_q'(t) \end{bmatrix}_{q\times 1} \circ \begin{bmatrix} m_{11}(t)\delta_1^y(t) + m_{21}(t)\delta_2^y(t) + \cdots + m_{k1}(t)\delta_k^y(t) + \cdots + m_{c1}(t)\delta_c^y(t) \\ m_{12}(t)\delta_1^y(t) + m_{22}(t)\delta_2^y(t) + \cdots + m_{k2}(t)\delta_k^y(t) + \cdots + m_{c2}(t)\delta_c^y(t) \\ \vdots \\ m_{1k}(t)\delta_1^y(t) + m_{2k}(t)\delta_2^y(t) + \cdots + m_{ki}(t)\delta_k^y(t) + \cdots + m_{ck}(t)\delta_c^y(t) \\ \vdots \\ m_{1q}(t)\delta_1^y(t) + m_{2q}(t)\delta_2^y(t) + \cdots + m_{kq}(t)\delta_k^y(t) + \cdots + m_{cq}(t)\delta_c^y(t) \end{bmatrix}_{q\times 1} \tag{30}$$

$$= \begin{bmatrix} z_1'(t) \\ z_2'(t) \\ \vdots \\ z_i'(t) \\ \vdots \\ z_q'(t) \end{bmatrix}_{q\times 1} \circ \begin{bmatrix} \sum_{k=1}^{c} m_{k1}(t)\delta_k^y(t) \\ \sum_{k=1}^{c} m_{k2}(t)\delta_k^y(t) \\ \vdots \\ \sum_{k=1}^{c} m_{ki}(t)\delta_k^y(t) \\ \vdots \\ \sum_{k=1}^{c} m_{kq}(t)\delta_k^y(t) \end{bmatrix}_{q\times 1} \tag{31}$$

$$= \begin{bmatrix} z_1'(t) \cdot \sum_{k=1}^{c} m_{k1}(t)\delta_k^y(t) \\ z_2'(t) \cdot \sum_{k=1}^{c} m_{k2}(t)\delta_k^y(t) \\ \vdots \\ z_i'(t) \cdot \sum_{k=1}^{c} m_{ki}(t)\delta_k^y(t) \\ \vdots \\ z_q'(t) \cdot \sum_{k=1}^{c} m_{kq}(t)\delta_k^y(t) \end{bmatrix}_{q\times 1} \tag{32}$$

em que o símbolo $\circ$ denota o produto de Hadamard e $\mathbf{z}'(t)$ é o vetor de derivadas instantâneas das saídas dos neurônios ocultos. A matriz $\mathbf{M}_{[:,2:q+1]}$ é obtida a partir da matriz $\mathbf{M}$ definida em (16) ao eliminar a primeira coluna (i.e. a coluna de limiares). A representação ora adotada remete à forma com que softwares como o Octave/Matlab/Scilab indexam as linhas e colunas de uma matriz. Portanto, a matriz $\mathbf{M}_{[:,2:q+1]}$ é a submatriz extraída da matriz $\mathbf{M}$ original tomando-se todas as linhas (:) e as colunas 2 a $(q+1)$.

Comentário 2 - A operação matricial levada a cabo na Eq. (28) é a responsável pelo nome dado ao algoritmo de aprendizado ora em desenvolvimento, a saber, algoritmo de retropropação dos erros (do inglês *error backpropagation*) [1]. Os gradientes locais dos neurônios de saída são projetados na camada oculta pela matriz $\mathbf{M}_{[:,2:q+1]}$. Os valores projetados de volta (ou retropropagados) são então multiplicados pelas derivadas instantâneas $z_i'(t)$ para que os gradientes locais dos neurônios ocultos, $\delta_i^z(t)$, sejam então computados.

O terceiro passo da Fase 2 corresponde ao processo de atualização ou ajuste dos parâmetros (pesos sinápticos e limiares) da rede MLP com uma camada oculta. Assim, para a camada de saída, tem-se que a versão escalar da regra de ajuste dos pesos m_{ki} é dada por

$$\begin{aligned} m_{ki}(t+1) &= m_{ki}(t) + \Delta m_{ki}(t), \\ &= m_{ki}(t) + \alpha \delta_k^y(t) z_i(t), \quad i = 0, \ldots, q; \;\; k = 1, \ldots, c \end{aligned} \tag{33}$$

enquanto que para os pesos dos neurônios ocultos, w_{ij}, a regra de ajuste é escrita como

$$\begin{aligned} w_{ij}(t+1) &= w_{ij}(t) + \Delta w_{ij}(t), \\ &= w_{ij}(t) + \alpha \delta_i^z(t) x_j(t), \quad j = 0, \ldots, p; \;\; i = 1, \ldots, q \end{aligned} \tag{34}$$

em que $\alpha(t)$ é a taxa de aprendizagem.

Para fins de implementação em ambiente Octave/Matlab/Scilab, as versões vetorial e matricial da regra delta generalizada são úteis. As versões vetoriais das regras de ajuste do vetor de pesos do k-ésimo neurônio de saída $\mathbf{m}_k$ e do i-ésimo neurônio oculto são escritas, respectivamente, como

$$\mathbf{m}_k(t+1) = \mathbf{m}_k(t) + \alpha \delta_k^y(t) \bar{\mathbf{z}}(t), \qquad k = 1, \ldots, c, \tag{35}$$

e

$$\mathbf{w}_i(t+1) = \mathbf{w}_i(t) + \alpha \delta_i^z(t) \mathbf{x}(t), \qquad i = 1, \ldots, q. \tag{36}$$

Por fim, as versões matriciais das regras de ajuste de pesos atualizam as matrizes $\mathbf{W}$ e $\mathbf{M}$ de uma vez por meio das seguintes equações:

$$\mathbf{M}(t+1) = \mathbf{M}(t) + \alpha \boldsymbol{\delta}^y(t) \bar{\mathbf{z}}^T(t), \tag{37}$$

e

$$\mathbf{W}(t+1) = \mathbf{W}(t) + \alpha \boldsymbol{\delta}^z(t) \mathbf{x}^T(t). \tag{38}$$

5 Treinamento, Convergência e Generalização

O projeto de uma rede neural envolve a especificação de diversos itens, cujos valores influenciam consideravelmente funcionamento do algoritmo. A seguir especificaremos a lista destes itens juntamente com as faixas de valores que os mesmos podem assumir:

Dimensão do vetor de Entrada (p): Este item pode assumir em tese valores entre 1 e ∞. Porém, existe um limite superior que depende da aplicação de interesse e do custo de se medir (observar) as variáveis x_j. É importante ter em mente que um valor alto para p não indica necessariamente um melhor desempenho para a rede neural, pois pode haver redundância no processo de medição. Neste caso, uma certa medida é, na verdade, a combinação linear de outras medidas, podendo ser descartada sem prejuízo ao desempenho da rede. Quando é muito caro, ou até impossível, medir um elevado número de variáveis x_j, deve-se escolher aquelas que o especialista da área considera como mais relevante ou representativas para o problema. O ideal seria que cada variável x_j, $j = 1, \ldots, p$, "carregasse" informação que somente ela contivesse. Do ponto de vista estatístico, isto equivale a dizer que as variáveis são *independentes* ou *não-correlacionadas* entre si.

Dimensão do vetor de saída (c): Assim como o primeiro item, este também depende da aplicação. Se o interesse está em problemas de aproximação de funções, $\mathbf{y} = F(\mathbf{x})$, o

número de neurônios deve refletir diretamente a quantidades de funções de saída desejadas (ou seja, a dimensão de $\mathbf{y}$).

Se o interesse está em problemas de classificação de padrões, a coisa muda um pouco de figura. Neste caso, o número de neurônios deve codificar o número de classes desejadas.

É importante perceber que estamos chamando as classes às quais pertencem os vetores de dados de uma forma bastante genérica: classe 1, classe 2, ..., etc. Contudo, à cada classe pode estar associado um rótulo (e.g. classe dos empregados, classe dos desempregados, classe dos trabalhadores informais, etc.), cujo significado depende da interpretação que o especialista na aplicação dá a cada uma delas. Estes rótulos normalmente não estão na forma numérica, de modo que para serem utilizados para treinar a rede MLP eles devem ser convertidos apra a forma numérica. A este procedimento dá-se o nome de codificação da saída da rede.

A codificação mais comum define como vetor de saídas desejadas um vetor binário de comprimento unitário; ou seja, apenas uma componente deste vetor terá o valor "1", enquanto as outras terão o valor "0" (ou -1). A dimensão do vetor de saídas desejadas corresponde ao número de classes do problema em questão. Usando esta codificação define-se automaticamente um neurônio de saída para cada classe. Por exemplo, se existem três classes possíveis, existirão três neurônios de saída, cada um representando uma classe. Como um vetor de entrada não pode pertencer a mais de uma classe ao mesmo tempo, o vetor de saídas desejadas terá valor 1 (um) na componente correspondente à classe deste vetor, e 0 (ou -1) para as outras componentes. Por exemplo, se o vetor de entrada $\mathbf{x}(t)$ pertence à classe 1, então seu vetor de saídas desejadas é $\mathbf{d}(t) = [1 \;\; 0 \;\; 0]^T$. Se o vetor $\mathbf{x}(t)$ pertence à classe 2, então seu vetor de saídas desejadas é $\mathbf{d}(t) = [0 \;\; 1 \;\; 0]^T$ e assim por diante para cada exemplo de treinamento.

Número de neurônios na camada oculta (q): Encontrar o número ideal de neurônios da camada oculta não é uma tarefa fácil porque depende de uma série de fatores, muito dos quais não temos controle total. Entre os fatores mais importantes podemos destacar os seguintes:

1. Quantidade de dados disponíveis para treinar e testar a rede.
2. Qualidade dos dados disponíveis (ruidosos, com elementos faltantes, etc.)
3. Número de parâmetros ajustáveis (pesos e limiares) da rede.
4. Nível de complexidade do problema (não-linear, discontínuo, etc.).

O valor de q é geralmente encontrado por tentativa-e-erro, em função da capacidade de *generalização* da rede (ver definição logo abaixo). Grosso modo, esta propriedade avalia o desempenho da rede neural ante situações não-previstas, ou seja, que resposta ela dá quando novos dados de entrada forem apresentados. Se muitos neurônios existirem na camada oculta, o desempenho será muito bom para os dados de treinamento, mas tende a ser ruim para os novos dados. Se existirem poucos neurônios, o desempenho será ruim também para os dados de treinamento. O valor ideal é aquele que permite atingir as especificações de desempenho adequadas tanto para os dados de treinamento, quanto para os novos dados.

Existem algumas fórmulas heurísticas (*ad hoc*) que sugerem valores para o número de neurônios na camada oculta da rede MLP, porém estas regras devem ser usadas apenas para dar um valor inicial para q. O projetista deve sempre treinar e testar várias vezes

uma dada rede MLP para diferentes valores de q, a fim de se certificar que a rede neural generaliza bem para dados novos, ou seja, não usados durante a fase de treinamento.

Dentre a regras heurísticas citamos a seguir três, que são comumente encontradas na literatura especializada:

Regra do valor médio - De acordo com esta fórmula o número de neurônios da camada oculta é igual ao valor médio do número de entradas e o número de saídas da rede, ou seja:

$$q = \frac{p + c}{2} \tag{39}$$

Regra da raiz quadrada - De acordo com esta fórmula o número de neurônios da camada oculta é igual a raiz quadrada do produto do número de entradas pelo número de saídas da rede, ou seja:

$$q = \sqrt{p \cdot c} \tag{40}$$

Regra de Kolmogorov De acordo com esta fórmula o número de neurônios da camada oculta é igual a duas vezes o número de entradas da rede adicionado de 1, ou seja:

$$q = 2p + 1 \tag{41}$$

Perceba que as regras só levam em consideração características da rede em si, como número de entradas e número de saídas, desprezando informações úteis, tais como número de dados disponíveis para treinar/testar a rede e o erro de generalização máximo aceitável.

Uma regra que define um valor inferior para q levando em consideração o número de dados de treinamento/teste é dada por:

$$q \geq \frac{N - 1}{p + 2} \tag{42}$$

A regra geral que se deve sempre ter em mente é a seguinte: *devemos sempre ter muito mais dados que parâmetros ajustáveis*. Assim, se o número total de parâmetros (pesos + limiares) da rede é dado por $Z = (p + 1) \cdot q + (q + 1) \cdot c$, então devemos sempre tentar obedecer à seguinte relação:

$$N \gg Z \tag{43}$$

Um refinamento da Eq. (43), proposto por Baum & Haussler (1991), sugere que a relação entre o número total de parâmetros da rede (Z) e a quantidade de dados disponíveis (N) deve obedecer à seguinte relação:

$$N > \frac{Z}{\varepsilon} \tag{44}$$

em que $\varepsilon > 0$ é o erro percentual máximo aceitável durante o teste da rede; ou seja, se o erro aceitável é 10%, então $\varepsilon = 0{,}1$. Para o desenvolvimento desta equação, os autores assumem que o erro percentual durante o treinamento não deverá ser maior que $\varepsilon/2$.

Para exemplificar, assumindo que $\varepsilon = 0{,}1$, então temos que $N > 10Z$. Isto significa que para uma rede de Z parâmetros ajustáveis, devemos ter uma quantidade dez vezes maior de padrões de treinamento.

Note que se substituirmos Z na Eq. (44) e isolarmos para q, chegaremos à seguinte expressão que fornece o valor aproximado do número de neurônios na camada oculta:

$$q \approx \left\lceil \frac{\varepsilon N - c}{p + M + 1} \right\rceil \tag{45}$$

em que $\lceil u \rceil$ denota o menor inteiro maior que u.

A Eq. (45) é bastante completa, visto que leva em consideração não só aspectos estruturais da rede MLP (número de entradas e de saídas), mas também o erro máximo tolerado para teste e o número de dados disponíveis. Portanto, seu uso é bastante recomendado.

Funções de ativação (ϕ) e (ϕ): Em tese, cada neurônio pode ter a sua própria função de ativação, diferente de todos os outros neurônios. Contudo, para simplificar o projeto da rede é comum adotar a mesma para todos os neurônios. Em geral, escolhe-se a função logística ou a tangente hiperbólica para os neurônios da camada oculta. Aquela que for escolhida para estes neurônios será adotada também para os neurônios da camada de saída. Em algumas aplicações é comum adotar uma função de ativação linear para os neurônios da camada de saída, ou seja, $\phi(u_k(t)) = C_k \cdot u_k(t)$, onde C_k é uma constante (ganho) positiva. Neste caso, tem-se que $\phi'(u_k(t)) = C_k$. O fato de $\phi(u_k(t))$ ser linear não altera o poder computacional da rede, o que devemos lembrar sempre é que os neurônios da camada oculta devem ter uma função de ativação não-linear, obrigatoriamente.

Critério de Parada e Convergência: A convergência da rede MLP é, em geral, avaliada com base nos valores do erro médio quadrático (ε_{epoca}) por época de treinamento:

$$\varepsilon_{epoca} = \frac{1}{N}\sum_{t=1}^{N}\varepsilon(t) = \frac{1}{2N}\sum_{t=1}^{N}\sum_{k=1}^{c}e_k^2(t) \tag{46}$$

Por outro lado, quando se utiliza a rede para classificar padrões, o desempenho da mesma é avaliado pela *taxa de acerto na classificação*, definida como:

$$P_{epoca} = \frac{\text{Número de vetores classificados corretamente}}{\text{Número de total de vetores}} \tag{47}$$

O gráfico $\varepsilon_{epoca} \times$ número de épocas ou o $P_{epoca} \times$ número de epócas é chamado de *Curva de Aprendizagem* da rede neural.

Em geral, o treinamento da rede neural é interrompido quando ε_{epoca} (ou P_{epoca}) atinge um limite inferior considerado adequado para o problema em questão (por exemplo, $\varepsilon_{epoca} \leq 0,001$ ou $P_{epoca} \approx 0,95$), ou quando o número máximo de épocas permitido é alcançado.

Avaliação da Rede Treinada: Para validar a rede treinada, ou seja, dizer que ela está apta para ser utilizada, é importante testar a sua resposta (saída) para dados de entrada diferentes daqueles vistos durante o treinamento. Estes novos dados podem ser obtidos através de novas medições, o que nem sempre é viável. Durante o teste os pesos da rede não são ajustados.

Para contornar este obstáculo, o procedimento mais comum consiste em treinar a rede apenas com uma parte dos dados selecionados *aleatoriamente*, guardando a parte restante para ser usada para testar o desempenho da rede. Assim, ter-se-á dois conjuntos de dados, um para treinamento, de tamanho $N_1 < N$, e outro de tamanho $N_2 = N - N_1$. Em geral, escolhe-se N_1 tal que a razão N_1/N esteja na faixa de $0,75$ a $0,90$.

Em outras palavras, se $N_1/N \approx 0,75$ tem-se que 75% dos vetores de dados devem ser selecionados aleatoriamente, sem reposição, para serem utilizados durante o treinamento. Os 25% restantes serão usados para testar a rede. O valor de ε_{epoca} calculado com os dados de teste é chamado de *erro de generalização* da rede, pois testa a capacidade da mesma em "extrapolar" o conhecimento aprendido durante o treinamento para novas situações. É importante ressaltar que, geralmente, o erro de generalização é maior do que o erro de treinamento, pois trata-se de um novo conjunto de dados.

6 Dicas para um Bom Projeto da Rede MLP

A seguir são dadas algumas sugestões para aumentar a chance de ser bem-sucedido no projeto de uma rede neural artificial.

Pré-processamento dos pares entrada-saída Antes de apresentar os exemplos de treinamento para a rede MLP é comum mudar a escala original das componentes dos vetores $\mathbf{x}$ e $\mathbf{d}$ para a escala das funções de ativação logística (0 e 1) ou da tangente hiperbólica (-1 e 1). As duas maneiras mais comuns de se fazer esta mudança de escala são apresentadas a seguir:

Procedimento 1: Indicado para quando as componentes x_j do vetor de entrada só assumem valores positivos e a função de ativação, $\phi(u)$, é a função logística. Neste caso, aplicar a seguinte transformação a cada componente de $\mathbf{x}$:

$$x_j^* = \frac{x_j}{x_j^{max}} \tag{48}$$

em que, ao dividir cada x_j pelo seu maior valor $x_j^{max} = \max_{\forall t}\{x_j(t)\}$, tem-se que $x_j^* \in [0, 1]$.

Procedimento 2: Indicado para quando as componentes x_j do vetor de entrada assumem valores positivos e negativos, e a função de ativação, $\phi(u)$, é a função tangente hiperbólica. Neste caso, aplicar a seguinte transformação a cada componente de $\mathbf{x}$:

$$x_j^* = 2\left(\frac{x_j - x_j^{min}}{x_j^{max} - x_j^{min}}\right) - 1 \tag{49}$$

em que $x_j^{min} = \min_{\forall t}\{x_j(t)\}$ é o menor valor de x_j. Neste caso, tem-se que $x_j^* \in [-1, +1]$.

Os dois procedimentos descritos acima também devem ser igualmente aplicados às componentes d_k dos vetores de saída, $\mathbf{d}$, caso estes possuam amplitudes fora da faixa definida pelas funções de ativação.

Taxa de aprendizagem variável: Nas Equações (34) e (33) é interessante que se use uma taxa de aprendizagem variável no tempo, $\alpha(t)$, decaindo até um valor bem baixo com o passar das iterações, em vez de mantê-la fixa por toda a fase de treinamento. Duas opções são dadas a seguir:

$$\alpha(t) = \alpha_0\left(1 - \frac{t}{t_{max}}\right), \quad \text{Decaimento linear} \tag{50}$$

$$\alpha(t) = \frac{\alpha_0}{1+t}, \quad \text{Decaimento exponencial} \tag{51}$$

em que α_0 é o valor inicial da taxa de aprendizagem e t_{max} é o número máximo de iterações:

$$t_{max} = \text{Tamanho do conjunto de treinamento} \times \text{Número máximo de épocas} \tag{52}$$

A idéia por trás das duas equações anteriores é começar com um valor alto para α, dado por $\alpha_0 < 0,5$, e terminar com um valor bem baixo, da ordem de $\alpha \approx 0,01$, a fim de estabilizar o processo de aprendizado.

Termo de momento: Também nas Equações (34) e (33) é interessante que se use um termo adicional, chamado *termo de momento*, cujo objetivo é tornar o processo de modificação dos pesos mais estável. Com este termo, as Equações (34) e (33) passam a ser escritas como:

$$w_{ij}(t+1) = w_{ij}(t) + \alpha\delta_i(t)x_j(t) + \eta\Delta w_{ij}(t-1) \quad (53)$$

$$m_{ki}(t+1) = m_{ki}(t) + \alpha\delta_k(t)z_i(t) + \eta\Delta m_{ki}(t-1) \quad (54)$$

em que $\Delta w_{ij}(t-1) = w_{ij}(t) - w_{ij}(t-1)$ e $\Delta m_{ki}(t-1) = m_{ki}(t) - m_{ki}(t-1)$. A constante η é chamada *fator de momento*. Enquanto α deve ser mantida abaixo de 0,5 por questões de estabilidade do aprendizado, o fator de momento é mantido em geral em valores na faixa [0,5 - 1]. É importante destacar que resultados teóricos recentes demostram que a introdução do termo de momento equivale, na verdade, a uma versão estacionária do método do gradiente conjugado [2].

Função Tangente Hiperbólica: Tem sido demonstrado empiricamente, ou seja, através de simulação computacional que o processo de treinamento converge mais rápido quando se utiliza a função de ativação tangente hiperbólica do que quando se usa a função logística. A justificativa para isto está no fato da tangente hiperbólica ser uma função ímpar, ou seja, $\phi(-u_i) = -\phi(u_i)$. Daí sugere-se utilizar a função tangente hiperbólica sempre que o problema permitir.

Limites menores que os assintóticos: É interessante notar que os valores limites 0 e 1 para a função logística, ou (-1 e $+1$) para a função tangente hiperbólica são valores assintóticos, ou seja, nunca são alcançados na prática. Assim, ao tentarmos forçar a saída rede neural para estes valores assintóticos, os pesos sinápticos, w_{ij} e m_{ki} tendem a assumir valores absolutos muito altos, ou seja, $w_{ij} \to \infty$ e $m_{ki} \to \infty$.

Para evitar este problema, sugere-se elevar de um valor bem pequeno $0 < \epsilon \ll 1$ o limite inferior de $\phi(\cdot)$ e diminuir deste mesmo valor o limite superior de $\phi(\cdot)$. Assim, teríamos a seguinte alteração:

$$-1 \to \epsilon - 1 \quad (55)$$

$$0 \to \epsilon \quad (56)$$

$$1 \to 1 - \epsilon \quad (57)$$

É comum escolher valores dentro da faixa $\epsilon \in [0,01 - 0,05]$.

Classificação de padrões: Quando se treina a rede MLP para classificar padrões é comum usar a codificação de saída descrita na Seção 5, em que na especificação do vetor de saídas desejadas assume-se o valor de saída unitário (1) para o neurônio que representa a classe e nulo (0) para os outros neurônios. Conforme dito no item anterior este valores são assintóticos e portanto, dificilmente serão observados durante a fase de teste.

Assim para evitar ambigüidades durante o cálculo da taxa de acerto P_{epoca} durante as fases de treinamento e teste define-se como a classe do vetor de entrada atual, $\mathbf{x}(t)$, como sendo a classe representada pelo neurônio que tiver maior valor de saída. Em palavras, podemos afirmar que se o índice do neurônio de maior saída é k^*, ou seja

$$k^*(t) = \arg\max_{\forall k}\{y_k(t)\} \quad (58)$$

então a classe de $\mathbf{x}(t)$ é a classe k^*.

Generalização: A rede MLP é um dos algoritmos de aproximação mais poderosos que existem, conforme atestado por uma gama de teoremas matemáticos. Contudo, todo este poder computacional, se não for utilizado adequadamente, não necessariamente implica em uma rede que seja capaz de generalizar adequadamente.

Por generalização adequada entende-se a habilidade da rede em utilizar o conhecimento armazenado nos seus pesos e limiares para gerar saídas coerentes para novos vetores de entrada, ou seja, vetores que não foram utilizados durante o treinamento. A generalização é considerada boa quando a rede, durante o treinamento, foi capaz de capturar (aprender) adequadamente a relação entrada-saída do mapeamento de interesse.

O bom treinamento de uma rede MLP, de modo que a mesma seja capaz de lidar com novos vetores de entrada, depende de uma série de fatores, dentre os quais podemos listar os seguintes

1. Excesso de graus de liberdade de uma rede MLP, na forma de elevado número de parâmetros ajustáveis (pesos e limiares).
2. Excesso de parâmetros de treinamento, tais como taxa de aprendizagem, fator de momento, número de camadas ocultas, critério de parada, dimensão da entrada, dimensão da saída, método de treinamento, separação dos conjuntos de treinamento e teste na proporção adequada, critério de validação, dentre outros.

Em particular, no que tange ao número de parâmetros ajustáveis, uma das principais conseqüências de um treinamento inadequado é a ocorrência de um subdimensionamento ou sobredimensionamento da rede MLP, o que pode levar, respectivamente, à ocorrência de *underfitting* (subajustamento) ou *overfitting* (sobreajustamento) da rede aos dados de treinamento. Em ambos os casos, a capacidade de generalização é ruim.

Dito de maneira simples, o subajuste da rede aos dados ocorre quando a rede não tem poder computacional (i.e. neurônios na camada oculta) suficiente para aprender o mapeamento de interesse. No outro extremo está o sobreajuste, que ocorre quando a rede tem neurônios ocultos demais (dispostos em uma ou duas camadas ocultas) e passar a memorizar os dados de treinamento. O ajuste ideal é obtido para um número de camadas ocultas e neurônios nestas camadas que confere à rede um bom desempenho durante a fase de teste, quando sua generalização é avaliada.

Uma das técnicas mais utilizadas para treinar a rede MLP, de modo a garantir uma boa generalização é conhecido como parada prematura (*early stopping*). Para este método funcionar, primeiramente devemos separar os dados em dois conjuntos, o de treinamento e o de teste, conforme mencionado na Seção 5. Em seguida, o conjunto de treinamento é ainda dividido em duas partes, uma para estimação dos parâmetros da rede propriamente dito e outra para validação durante o treinamento. O conjunto de validação deve ser usado de tempos em tempos (por exemplo, a cada 5 épocas de treinamento) para cálculo do erro quadrático médio de generalização. Durante a validação, os pesos e limiares da rede não são ajustados.

A idéia do método da parada prematura é interromper o treinamento a partir do momento em que o erro quadrático médio calculado para o conjunto de validação assumir uma tendência de crescimento. Argumenta-se que esta tendência de crescimento do erro é um indicativo de que a rede está começando a se especializar demais nos dados usados para estimação dos parâmetros. A avaliação final da generalização é feita usando-se o conjunto de teste. O método de parada prematura pode ser melhor visualizado através das curvas

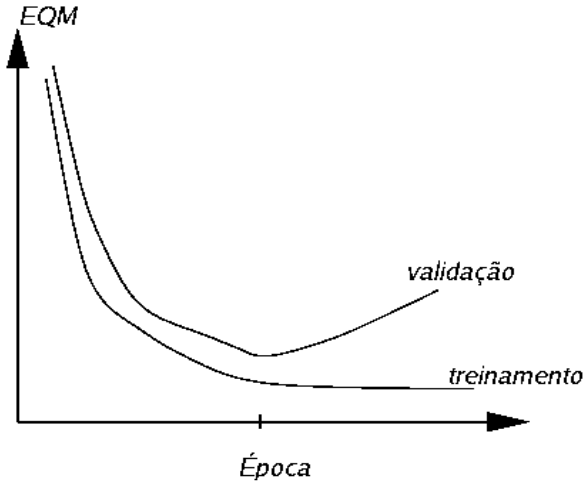

Figura 5: Curvas de aprendizagem para conjuntos de estimação e validação.

de aprendizagem do erro quadrático médio para o conjunto de estimação e para o conjunto de validação, conforme mostradas na Figura 5.

7 Exercícios Computacionais

Exercício 1 - Usar os pares de vetores entrada-saída disponíveis no arquivo `mlp_exe1.zip` para avaliar a rede MLP em um problema de classificação de padrões. Pede-se determinar a curva de aprendizagem da rede MLP e a taxa de acerto média na classificação.

Exercício 2 - Usar a rede MLP para aproximar a função

$$y(x_1, x_2) = \sin^2(x_1) \cdot \cos^2(x_2) + x_1 x_2^3 \tag{59}$$

através da geração de 500 pares entrada-saída de treinamento. Pede-se determinar o gráfico da função $z(x, y)$ usando a fórmula mostrada na Eq. (59) e através da rede MLP. Determinar também a curva de aprendizagem do modelo e o erro médio de generalização.

Referências

[1] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[2] A. Bhaya and E. Kaszkurewicz. Steepest descent with momentum for quadratic functions is a version of the conjugate gradient method. *Neural Networks*, 17(1):65–71, 2004.