

Otimização via Gradiente Descendente

Prof. Dr. Guilherme de Alencar Barreto

Outubro/2015

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

gbarreto@ufc.br

1 Definições Preliminares

Considere uma função escalar (contínua) de p variáveis, $f : \mathbb{R}^p \rightarrow \mathbb{R}$; ou seja, $f(\cdot)$ é uma função que produz uma saída escalar $y \in \mathbb{R}$ e que possui p ($p \geq 1$) variáveis de entrada. Formalmente, podemos escrever

$$y = f(\mathbf{x}) = f(x_1, x_2, \dots, x_p), \quad (1)$$

em que \mathbf{x} é um vetor cujas componentes são as variáveis x_i , $i = 1, \dots, p$. Como exemplos, para $p = 1$, temos que a parábola é uma função dada por

$$y = f(x) = (x - a)^2 + b, \quad (2)$$

cujos gráficos para $a = b = 5$ no plano cartesiano está mostrada na Figura 1a. A função equivalente em três dimensões é chamada de parabolóide, sendo escrita matematicamente como

$$z = f(x, y) = (x - a)^2 + (y - b)^2 + c, \quad (3)$$

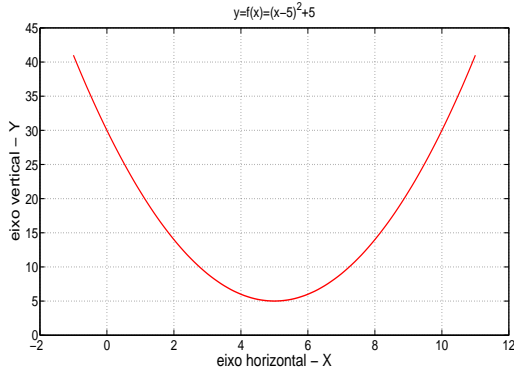
em que a , b e c são constantes reais. O gráfico desta função é uma superfície em 3 dimensões, mostrada na Figura 1b.

Note que as funções acima são contínuas e de variação suaves (i.e. não possuem interrupções ao longo do seu domínio, nem mudanças bruscas em seus gráficos). Além disso, são também funções convexas, ou seja, possuem apenas um ponto extremo, chamado de *mínimo global* neste caso.

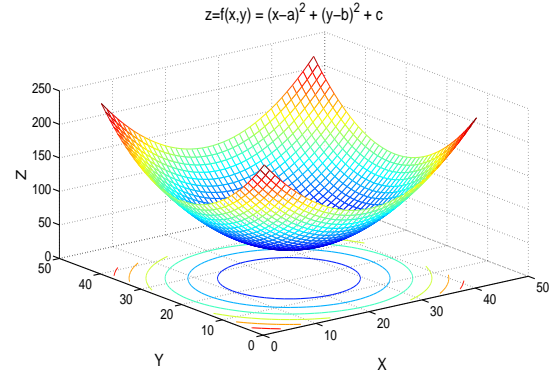
Se a função mostrada na Eq. (2) for usada como função-custo ou função-objetivo em algum problema de minimização, o valor da variável x para o qual $y = f(x)$ produz seu maior valor é chamado de valor ótimo de x , simbolizado como x_{opt} . Se o problema envolver duas variáveis, como a função mostrada na Eq. (3), busca-se um vetor ótimo $\mathbf{x}_{opt} = [x_{opt} \ y_{opt}]^T$, em que T simboliza o vetor-transposto.

De um modo geral, o problema de minimização de funções (sem restrições) pode ser formalizado matematicamente da seguinte maneira. O vetor $\mathbf{x}_{opt} \in \mathbb{R}^p$ é o vetor-ótimo se

$$f(\mathbf{x}_{opt}) < f(\mathbf{x}), \quad \forall \mathbf{x} \neq \mathbf{x}_{opt}. \quad (4)$$



(a)



(b)

Figura 1: Representação gráfica de funções no plano cartesiano e em 3 dimensões. (a) $y = f(x) = (x - 5)^2 + 5$, (b) $z = (x - 20)^2 + (y - 20)^2 + 50$.

ou, de maneira alternativa, como

$$\mathbf{x}_{opt} = \arg \min_{\forall \mathbf{x}} f(\mathbf{x}), \quad (5)$$

Nas próximas seções são apresentadas duas técnicas para obter o vetor-ótimo para uma função convexa, ambas utilizando o vetor-gradiente da função de interesse.

1.1 Método Não-Iterativo

Este é o método mais simples e direto quando a função é convexa, contínua e diferenciável de ordem 1 (i.e. a primeira derivada existe). Para este fim, basta determinar o vetor-gradiente da função de interesse e igualá-lo ao vetor-nulo de dimensão p . Formalmente, esta técnica pode ser escrita como

$$\frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} = \mathbf{0}, \quad (6)$$

que, em termos das componentes do vetor-gradiente, pode também ser escrita como

$$\begin{bmatrix} \frac{\partial f(x_1, \dots, x_p)}{\partial x_1} \\ \frac{\partial f(x_1, \dots, x_p)}{\partial x_2} \\ \vdots \\ \frac{\partial f(x_1, \dots, x_p)}{\partial x_p} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad (7)$$

1.1.1 Exemplo 1

Tomando como exemplo a função mostrada na Eq. (3), temos que o vetor-gradiente para esta função é dado por

$$\begin{bmatrix} \frac{\partial f(x,y)}{\partial x} \\ \frac{\partial f(x,y)}{\partial y} \end{bmatrix} = \begin{bmatrix} 2(x - a) \\ 2(y - b) \end{bmatrix}, \quad (8)$$

de modo, que ao igualarmos cada componente a zero, teremos

$$\begin{bmatrix} 2(x - a) \\ 2(y - b) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad (9)$$

de onde obtemos o vetor-ótimo como $\mathbf{x}_{opt} = [a \ b]^T$.

1.2 Método Iterativo

Em muitas ocasiões não é possível obter uma solução-ótima imediata e fechada para o vetor-ótimo da função de interesse através da técnica mostrada na Eq. (6). Nestes casos, uma solução iterativa, normalmente utilizando uma equação recursiva, surge como a opção mais viável.

Um procedimento muito comum de se chegar iterativamente ao vetor-ótimo \mathbf{x}_{opt} ou a uma vizinhança deste, é conhecido como método do gradiente descendente (ou gradiente estocástico). Para este fim, utiliza-se a seguinte equação recursiva:

$$\mathbf{x}(n+1) = \mathbf{x}(n) - \alpha \frac{\partial f(\mathbf{x}(n))}{\partial \mathbf{x}(n)}, \quad (10)$$

tal que a constante $0 < \alpha \ll 1$ é chamada de *passo de adaptação*. Note que $\mathbf{x}(n)$ denota o valor do vetor-solução \mathbf{x} na iteração n , logo o termo $\frac{\partial f(\mathbf{x}(n))}{\partial \mathbf{x}(n)}$ denota o valor instantâneo do gradiente de $f(\mathbf{x})$ para $\mathbf{x} = \mathbf{x}(n)$. A equação (10) é interpretada da seguinte forma:

O vetor-solução atual, $\mathbf{x}(n)$, é modificado na direção do vetor gradiente de $f(\cdot)$, em um sentido que percorre uma trajetória de descida - daí, a razão por trás do uso do sinal (-) na Eq. (10) e do termo *descendente* no nome do algoritmo.

Em outras palavras, o vetor-solução, $\mathbf{x}(n)$, vai sendo modificado incrementalmente, na direção de máxima variação de $f(\cdot)$, com sinal negativo porque estamos tratando de um problema de minimização; logo, buscamos por um ponto de mínimo (local ou global). Se o problema for o de encontrar o ponto em que ocorre o máximo valor (pico ou moda) de uma função ou de maximização de uma função-custo, o sinal de menos (-) deve ser trocado por um de mais (+).

À medida que a equação recursiva do método do gradiente é executada, espera-se que ao longo de vários ciclos, a diferença entre $\mathbf{x}(n+1)$ e $\mathbf{x}(n)$ vai diminuindo. Em termos mais formais, à medida que $n \rightarrow \infty$, tem-se que $\|\mathbf{x}(n+1) - \mathbf{x}(n)\| \rightarrow 0$, em que $\|\mathbf{v}\|$ denota a norma euclidiana do vetor \mathbf{v} .

Se um ponto de mínimo (local ou global) for alcançado, teremos $\mathbf{x}(n+1) \approx \mathbf{x}(n)$, de onde resulta que

$$\frac{\partial f(\mathbf{x}(n))}{\partial \mathbf{x}(n)} = \mathbf{0}, \quad (11)$$

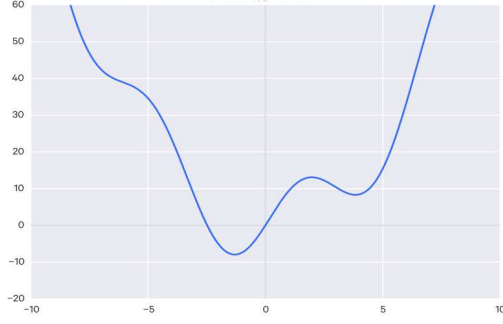
que é a condição a ser satisfeita por um extremo da função de interesse, conforme explicitado na Eq. (6).

Observação: Como a Eq. (10) é recursiva, faz-se necessário definir um valor inicial $\mathbf{x}(0)$ para o vetor-solução em $n = 0$. Tem-se então duas situações que podem acontecer quando o método do gradiente é utilizado em otimização. São elas:

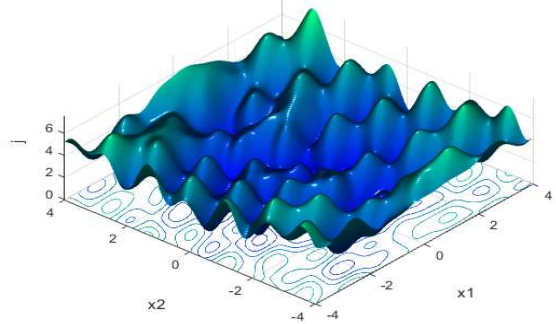
Função Convexa - Neste caso, a função tem apenas um ponto mínimo, chamado de global.

Assim, qualquer que seja o valor atribuído a $\mathbf{x}(0)$, o algoritmo sempre convergirá para a solução ótima após algumas iterações. As funções mostradas na Fig. (1) são exemplos de funções convexas.

Função Não-Convexa - Neste caso, a função tem vários pontos mínimos, chamados de mínimos locais. Logo, o algoritmo convergirá para a solução (ponto de mínimo) mais próximo do vetor inicial $\mathbf{x}(0)$ e não sairá mais deste, uma vez que para qualquer ponto ótimo (seja local ou global) teremos $\frac{\partial f(\mathbf{x})}{\partial \mathbf{x}_{opt}} = \mathbf{0}$.



(a)



(b)

Figura 2: Exemplos de funções não-convexas no plano cartesiano e em 3 dimensões..

1.2.1 Exemplo 2

Tomando como exemplo a função mostrada na Eq. (3), temos que o vetor-gradiente na iteração n para esta função é dado por

$$\begin{bmatrix} \frac{\partial f(x(n), y(n))}{\partial x(n)} \\ \frac{\partial f(x(n), y(n))}{\partial y} \end{bmatrix} = \begin{bmatrix} 2(x(n) - a) \\ 2(y(n) - b) \end{bmatrix}, \quad (12)$$

de modo, que a Eq. (10) para este exemplo passa a ser escrita como

$$\begin{bmatrix} x(n+1) \\ y(n+1) \end{bmatrix} = \begin{bmatrix} x(n) \\ y(n) \end{bmatrix} - 2\alpha \begin{bmatrix} x(n) - a \\ y(n) - b \end{bmatrix}, \quad (13)$$

em que α é o passo de adaptação. Na Fig. 3 estão mostradas as trajetórias $\{x(n)\}_{n=0}^{50}$ e $\{y(n)\}_{n=0}^{50}$ para $a = b = 20$, $c = 50$, $\alpha=0,1$, $x(0) = 2$ e $y(0) = 25$. Note que as trajetórias convergem para os valores ótimos $x_{opt} = y_{opt} = 20$ após algumas iterações. A convergência pode ser acelerada se escolhermos um passo de adaptação maior (e.g. $\alpha=0,25$). Código Matlab/Octave para esta simulação está disponível na Fig. (4).

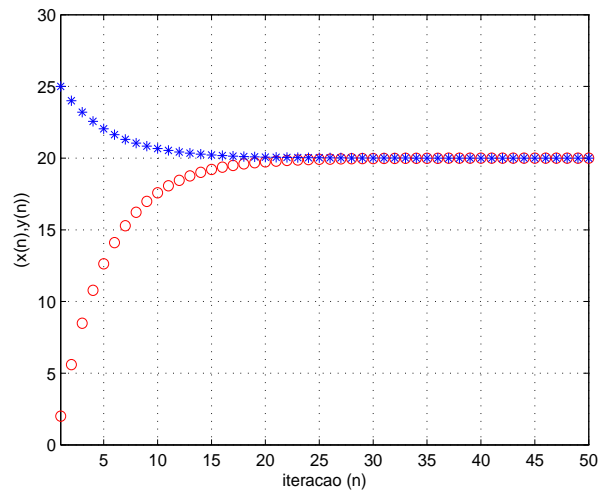


Figura 3: Trajetórias das coordenadas $x(n)$ (círculos vermelhos) e $y(n)$ (asteriscos azuis) para $a = b = 20$, $c = 50$, $\alpha=0,1$, $x(0) = 2$ e $y(0) = 25$.

```
% Metodo do gradiente descendente (funcao de duas variaveis)
% Funcao: z=f(x,y)=(x-a)^2+(y-b)^2+c
% Vetor-gradiente: dz(v)/dv = [dz(x,y)/dx dz(x,y)/dy], onde v=[x y]^T
% Regra recursiva: v(n+1)=v(n)-eta*dz/dv
% Autor: Guilherme Barreto
% Data: 23/11/2015

clc; clear;

v(:,1)=[2;25]; % vetor de condicoes iniciais

eta=0.1; % Passo de aprendizagem

a=20; b=a; c=50; % constantes da funcao
for n=1:50,
    gradvec(1,n)=2*(v(1,n)-a);
    gradvec(2,n)=2*(v(2,n)-b);
    v(:,n+1)=v(:,n)-eta*gradvec(:,n);
end

figure;
plot(1:51,v(1,:), 'ro'); hold on;
plot(1:51,v(2,:), 'b*'); grid;
axis([1 50 0 1.5*a]);
xlabel('iteracao (n)');
ylabel('(x(n),y(n))'); hold off
```

Figura 4: Código Matlab/Octave para Exemplo 2.

Perceptron Logístico e Regra Delta Generalizada

Prof. Dr. Guilherme de Alencar Barreto

Junho/2025

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

gbarreto@ufc.br

1 Introdução

Esta seção discute o efeito de se utilizar uma não-linearidade suave na saída do modelo de neurônio da rede Perceptron simples, em vez de uma não-linearidade abrupta, tal como a função sinal. Esta mudança implicará em alterações na regra de aprendizagem que passará a ser chamada de regra delta generalizada.

Esta pequena mudança promove um grande impacto na aplicabilidade da rede Perceptron simples, uma vez que permite sua aplicação também em problemas de regressão. Recorde que antes a rede Perceptron simples era usada apenas em problemas de classificação de padrões. Isto é possível porque a regra de aprendizado será baseado no gradiente do erro quadrático instantâneo, como no modelo Adaline. Assim, o modelo de neurônio logístico a ser discutido nesta nota de aula, unifica os modelos Adaline e Perceptron simples, que possuem origens distintas [1, 2].

O Perceptron simples com função de ativação sigmoideal (i.e. em forma de S) passa a ser chamado doravante de *Perceptron simples Logístico* ou simplesmente de Perceptron logístico (PL). Aqui iremos discutir duas funções sigmoideais muito utilizadas nos modelos atuais de redes neurais artificiais, a saber, a função sigmoide logística e a função tangente hiperbólica. Porém, pode-se usar qualquer função não-linear suave, que seja diferenciável até segunda ordem, pelo menos.

Manteremos a mesma notação para o vetor de entrada ($\mathbf{x} \in \mathbb{R}^{(p+1) \times 1}$), ativação do i -ésimo neurônio ($u_k \in \mathbb{R}$), saída do k -ésimo neurônio ($y_k \in (0, +1)$ ou $y_k \in (-1, +1)$), peso sináptico conectando a j -ésima entrada ao i -ésimo neurônio ($w_{kj} \in \mathbb{R}$), vetor de pesos do k -ésimo neurônio ($\mathbf{w}_k \in \mathbb{R}^{(p+1) \times 1}$) e número de neurônios (q), melhor definidos a seguir.

O vetor de entrada da rede PL é então definido como

$$\mathbf{x}(t) = \begin{bmatrix} x_0(t) \\ x_1(t) \\ \vdots \\ x_j(t) \\ \vdots \\ x_p(t) \end{bmatrix}_{(p+1) \times 1} = \begin{bmatrix} -1 \\ x_1(t) \\ \vdots \\ x_j(t) \\ \vdots \\ x_p(t) \end{bmatrix}_{(p+1) \times 1} \quad (1)$$

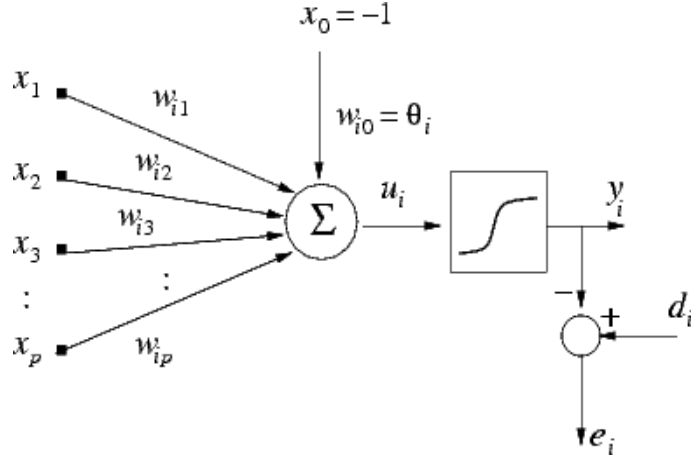


Figura 1: Arquitetura do i -ésimo neurônio da rede Perceptron logístico.

em que $x_j(t)$ denota uma componente qualquer do vetor de entrada $\mathbf{x}(t)$ e t indica o instante de apresentação deste vetor à rede.

O vetor de pesos associado ao i -ésimo neurônio é representado da seguinte forma:

$$\mathbf{w}_k(t) = \begin{bmatrix} w_{k0}(t) \\ w_{k1}(t) \\ \vdots \\ w_{kj}(t) \\ \vdots \\ w_{kc}(t) \end{bmatrix}_{(p+1) \times 1} = \begin{bmatrix} \theta_k(t) \\ w_{k1}(t) \\ \vdots \\ w_{kj}(t) \\ \vdots \\ w_{kc}(t) \end{bmatrix}_{(p+1) \times 1}, \quad (2)$$

em que w_{kj} é o peso sináptico que conecta a entrada j ao k -ésimo neurônio de saída, θ_k define um limiar (*threshold* ou *bias*) associado ao k -ésimo neurônio de saída.

Por fim, o vetor de saídas desejadas (ou saídas-alvo) é representado por um vetor de q componentes, ou seja

$$\mathbf{d}(t) = \begin{bmatrix} d_1(t) \\ \vdots \\ d_k(t) \\ \vdots \\ d_c(t) \end{bmatrix}_{c \times 1}, \quad (3)$$

em que $d_k(t)$ a saída desejada para o k -ésimo neurônio, $k = 1, \dots, c$.

Importante: Em uma rede PL, cada neurônio possui o seu próprio vetor de pesos \mathbf{w}_k . Assim, uma rede com q neurônios terá $p \times q$ pesos sinápticos w_{ij} e q limiares θ_k , resultando em um total de $(p+1) \times q$ parâmetros ajustáveis. Estes parâmetros deverão ser ajustados por meio de uma regra de atualização recursiva denominada **Regra Delta Generalizada**.

2 Regra Delta Generalizada

A ativação do k -ésimo neurônio da rede PL é calculada como segue

$$\begin{aligned}
 u_k(t) &= \sum_{j=1}^p w_{kj}(t)x_j(t) - \theta_k(t) \\
 &= \sum_{j=1}^p w_{kj}(t)x_j(t) + w_{k0}(t)x_0(t) \\
 &= \sum_{j=0}^p w_{kj}(t)x_j(t) \\
 &= \mathbf{w}_k^T(t)\mathbf{x}(t) = \mathbf{x}^T(t)\mathbf{w}_k(t)
 \end{aligned} \tag{4}$$

em que foi feito $x_0 = -1$ e $w_{k0} = \theta_k$. O sobrescrito T indica a operação de transposição dos vetores.

Como agora existe uma não-linearidade suave na saída do neurônio, conforme ilustrado na Figura 1, então a sua saída passa a ser representada como

$$y_k(t) = \phi(u_k(t)) = \phi(\mathbf{w}_k^T(t)\mathbf{x}(t)) = \phi_k(t), \tag{5}$$

em que $\phi(\cdot)$ é uma não-linearidade do tipo sigmoideal, possuindo a forma da letra S esticada. Dois tipos são comumente usados, a sigmoide logística

$$y_k(t) = \phi_k(t) = \frac{1}{1 + \exp\{-u_k(t)\}}, \tag{6}$$

e a tangente hiperbólica

$$y_k(t) = \phi_k(t) = \frac{1 - \exp\{-u_k(t)\}}{1 + \exp\{-u_k(t)\}}. \tag{7}$$

O domínio tanto da função sigmoide logística, quanto da tangente hiperbólica é a reta dos números reais. Contudo, a imagem da função sigmoide logística está restrita ao intervalo $(0, 1)$, enquanto a da tangente hiperbólica está restrita ao intervalo $(-1, +1)$. De um extremo a outro a função é sempre crescente. O gráfico dessas duas funções estão mostrados na Figura 2.

As versões vetoriais das Eqs. (4) e (5) são dadas por

$$\mathbf{u}(t) = \begin{bmatrix} u_1(t) \\ u_2(t) \\ \vdots \\ u_c(t) \end{bmatrix} = \mathbf{W}\mathbf{x}(t) \tag{8}$$

em que $\mathbf{W} \in \mathbb{R}^{c \times (p+1)}$ é a matriz de pesos da rede PL. Nesta matriz, os vetores de pesos \mathbf{w}_k , $k = 1, \dots, q$, estão organizados como linhas de uma matriz \mathbf{W} , ou seja

$$\mathbf{W} = \begin{bmatrix} - & \mathbf{w}_1^T & - \\ - & \mathbf{w}_2^T & - \\ \vdots & \vdots & \vdots \\ - & \mathbf{w}_c^T & - \end{bmatrix}, \tag{9}$$

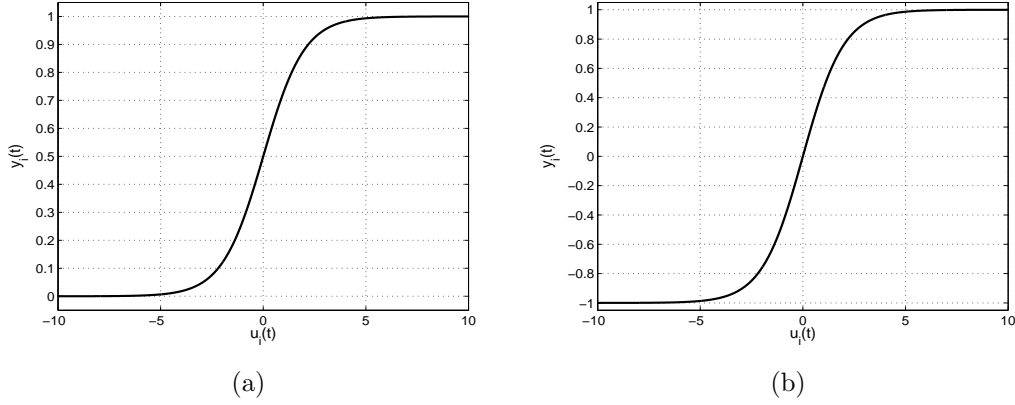


Figura 2: (a) Função sigmoide logística e (b) tangente hiperbólica.

em que $\dim(\mathbf{W}) = c \times (p + 1)$. O vetor com as saídas dos q neurônios da rede é dada por

$$\mathbf{y}(t) = \begin{bmatrix} y_1(t) \\ y_2(t) \\ \vdots \\ y_c(t) \end{bmatrix} = \begin{bmatrix} \phi(u_1(t)) \\ \phi(u_2(t)) \\ \vdots \\ \phi(u_c(t)) \end{bmatrix} = \begin{bmatrix} \phi(\mathbf{w}_1^T \mathbf{x}(t)) \\ \phi(\mathbf{w}_2^T \mathbf{x}(t)) \\ \vdots \\ \phi(\mathbf{w}_c^T \mathbf{x}(t)) \end{bmatrix} \quad (10)$$

O desempenho do k -ésimo neurônio da rede PL no instante t , como aproximador do mapeamento $\mathbf{F}(\cdot)$, é avaliado a partir do erro quadrático instantâneo definido como

$$J_k(t) = \frac{1}{2} e_k^2(t), \quad (11)$$

$$= \frac{1}{2} (d_k(t) - y_k(t))^2, \quad (12)$$

$$= \frac{1}{2} (d_k(t) - \phi(u_k(t)))^2, \quad (13)$$

$$= \frac{1}{2} (d_k(t) - \phi(\mathbf{w}_k^T(t) \mathbf{x}(t)))^2, \quad (14)$$

em que $e_k(t) = d_k(t) - y_k(t)$ é o erro do i -ésimo neurônio em resposta à apresentação do par entrada-saída atual $(\mathbf{x}(t), \mathbf{d}(t))$.

Assim, define-se a seguir uma medida global do desempenho da rede PL, chamada de **erro quadrático médio** (EQM), com base nos erros produzidos para todos os pares entrada-saída $\{\mathbf{x}(t), \mathbf{d}(t)\}$:

$$J[\mathbf{W}] = \frac{1}{2N} \sum_{t=1}^N \sum_{k=1}^c J_k(t) = \frac{1}{2N} \sum_{t=1}^N \sum_{k=1}^c e_k^2(t) = \frac{1}{2N} \sum_{k=1}^c \sum_{t=1}^N [d_k(t) - y_k(t)]^2 \quad (15)$$

em que \mathbf{W} denota o conjunto de todos os parâmetros ajustáveis do modelo (pesos e limiares).

Os parâmetros da rede PL devem ser especificados a fim de que os neurônios produzam sempre uma saída $y_k(t)$ bem próxima da saída esperada $d_k(t)$ para um vetor de entrada $\mathbf{x}(t)$ qualquer. Em outras palavras, o funcional $J_k(t)$ deve ter o menor valor possível para aquele conjunto de dados específico. Dá-se o nome de parâmetros ótimos aos valores de \mathbf{w}_k que alcançam este objetivo.

Um procedimento iterativo para se chegar aos parâmetros ótimos envolve o uso do método de otimização baseada no gradiente descendente:

$$\mathbf{w}_k(t+1) = \mathbf{w}_k(t) - \alpha \frac{\partial J_k(t)}{\partial \mathbf{w}_k(t)} \quad (16)$$

tal que a variável $0 < \alpha < 1$ é a taxa de aprendizagem. Utilizando a regra da cadeia, a derivada presente na Eq. (16) pode ser escrita da seguinte forma:

$$\frac{\partial J_k(t)}{\partial \mathbf{w}_k(t)} = \frac{dJ_k(t)}{de_k(t)} \frac{de_k(t)}{dy_k(t)} \frac{dy_k(t)}{du_k(t)} \frac{\partial u_k(t)}{\partial \mathbf{w}_k(t)} \quad (17)$$

em que os resultados das quatro derivadas para a rede PL são mostrados a seguir:

$$\frac{dJ(t)}{de_k(t)} = e_k(t) \quad (18)$$

$$\frac{de_k(t)}{dy_k(t)} = -1 \quad (19)$$

$$\frac{dy_k(t)}{du_k(t)} = \frac{d\phi(u_k(t))}{du_k(t)} = \phi'_k(t) \quad (20)$$

$$\frac{\partial u_k(t)}{\partial \mathbf{w}_k(t)} = \mathbf{x}(t) \quad (21)$$

Assim, temos que a regra recursiva de ajuste dos pesos, $w_j(t)$, é dada por

$$\begin{aligned} \mathbf{w}_k(t+1) &= \mathbf{w}_k(t) - \alpha \frac{\partial J_k(t)}{\partial \mathbf{w}_k(t)}, \\ &= \mathbf{w}_k(t) + \alpha e_k(t) \phi'_k(t) \mathbf{x}(t), \\ &= \mathbf{w}_k(t) + \alpha \delta_k(t) \mathbf{x}(t), \end{aligned} \quad (22)$$

em que $\delta_k(t)$ é chamado de gradiente local do k -ésimo neurônio. A regra de aprendizagem mostrada na Eq. (22) é conhecida como *regra delta generalizada* ou *regra LMS generalizada*, para diferenciar da regra LMS original que é também conhecida como regra delta. Em todo caso, se a função de ativação $\phi(u_k(t))$ for a função identidade, ou seja, $\phi(u_k(t)) = u_k(t)$, teremos a regra LMS usual.

A derivada instantânea¹ da função de ativação $\phi_k(t)$, denotada por $\phi'_k(t)$, para a função logística é dada por

$$\phi'_k(t) = \frac{d\phi_k(t)}{du_k(t)} = \frac{d(1 + \exp\{-u_k(t)\})^{-1}}{du_k(t)}, \quad (23)$$

$$= (-1) \cdot (1 + \exp\{-u_k(t)\})^{-2} \cdot \frac{d\exp\{-u_k(t)\}}{du_k(t)}, \quad (24)$$

$$= (-1) \cdot (-1) \cdot (1 + \exp\{-u_k(t)\})^{-2} \cdot \exp\{-u_k(t)\}, \quad (25)$$

$$= \frac{\exp\{-u_k(t)\}}{(1 + \exp\{-u_k(t)\})^2} = \frac{1}{1 + \exp\{-u_k(t)\}} \cdot \frac{\exp\{-u_k(t)\}}{1 + \exp\{-u_k(t)\}}, \quad (26)$$

$$= \phi_k(t)[1 - \phi_k(t)] = y_k(t)[1 - y_k(t)], \quad (27)$$

e para a função tangente hiperbólica (exercício!) temos que

$$\phi'_k(t) = \frac{1}{2}(1 - \phi_k^2(t)) = \frac{1}{2}(1 - y_k^2(t)). \quad (28)$$

¹Derivada no instante atual t .

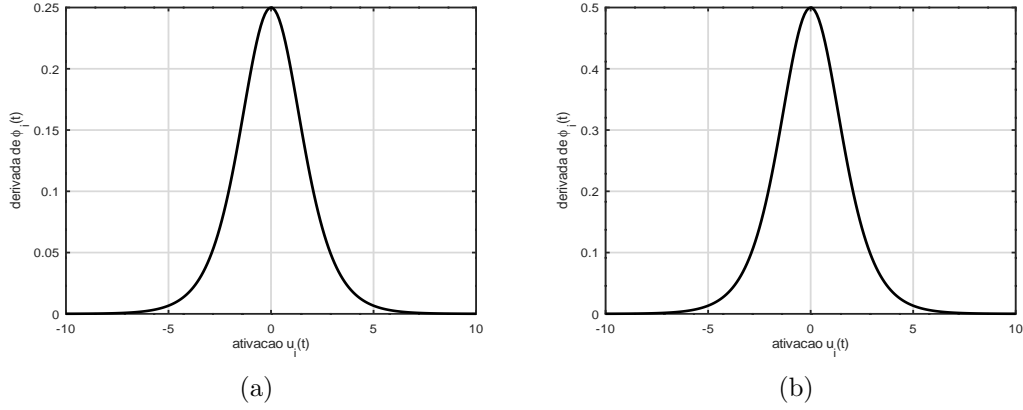


Figura 3: (a) Função sigmoide logística e (b) tangente hiperbólica.

Uma das vantagens em se utilizar as funções sigmoide logística e tangente hiperbólica reside no fato de suas derivadas instantâneas não precisam ser computadas via métodos numéricos, já que podem ser escritas em função do valor da saída $y_k(t)$ no instante t . Os gráficos das derivadas das funções de ativação sigmoide logística e tangente hiperbólica estão mostrados na Figura 3.

Além disso, a derivada da função de ativação é usada no cálculo do gradiente local do erro do k -ésimo neurônio, $\delta_k(t)$, que pode ser entendido como elemento modulador do erro $e_k(t)$. Assim, valores altos do erro (positivos ou negativos) gerarão valores pequenos da derivada $\phi'_k(t)$, o que confere certa estabilidade ao aprendizado da rede PL ao longo do tempo. Tal estabilidade é verificada por uma curva de aprendizagem (convergência do EQM) mais suave que a da regra LMS convencional. Porém, essa modulação do erro pode também causar o problema conhecido como *paralisia da rede*, em que os pesos não são modificados mesmo para erros grandes, principalmente no começo do processo de aprendizado. Para mitigar a ocorrência de paralisia, recomenda-se iniciar os pesos com valores bem pequenos.

A rede PL pode ser utilizada tanto para aproximação de funções, quanto para classificação de padrões.

3 Versão Matricial da Regra LMS Generalizada

Organizando os gradientes locais dos c neurônios de saída no instante t em um vetor de gradientes locais da seguinte forma:

$$\boldsymbol{\delta}(t) = \begin{bmatrix} \delta_1(t) \\ \delta_2(t) \\ \vdots \\ \delta_c(t) \end{bmatrix}_{c \times 1} \quad (29)$$

podemos escrever a regra de aprendizagem da rede PL da seguinte maneira:

$$\mathbf{W}(t+1) = \mathbf{W}(t) + \alpha \boldsymbol{\delta}(t) \mathbf{x}^T(t), \quad (30)$$

em que esta expressão atualiza a matriz de pesos \mathbf{W} , de dimensões $c \times (p+1)$, de uma só vez a cada instante t . Cada termo da Eq. (30) deve ter dimensões compatíveis, de modo que para isso ser possível o termo de correção envolve o produto externo do vetor de erros $\mathbf{e}(t)$ com o vetor de entrada $\mathbf{x}(t)$. É por isso que o vetor de entrada aparece transposto nessa equação.

A versão matricial da regra de aprendizado da rede PL não é muito comum de se ver em livros, porém é extremamente vantajosa para implementação em Octave/Matlab, R e Python, devido à maior velocidade que confere à execução do algoritmo. Usando a Eq. (30) a matriz de pesos \mathbf{W} é atualizada de uma vez a cada apresentação de um vetor de entrada. Note que isto só é possível porque o funcionamento de cada neurônio é independente do funcionamento dos demais, dada a natureza local da regra desta regra de aprendizado.

4 Outras Funções de Ativação

As funções sigmoidais descritas anteriormente ainda são muito usadas em modelos clássicos de redes neurais. Porém, em modelos de redes neurais convolucionais, que com frequência assumem uma escala muito grande, tem se dado preferência há outras funções de menor custo computacional. Descreveremos algumas delas a seguir.

4.1 Rectified Linear Unit (ReLU)

A função de ativação ReLU é uma opção muito popular, que produz a entrada diretamente se ela for positiva e zero caso contrário. Matematicamente, tem-se

$$y_k(t) = \phi(u_k(t)) = \max(0, u_k(t)) = \frac{u_k(t) + |u_k(t)|}{2}. \quad (31)$$

O gráfico da função ReLU no intervalo $(-10, 10)$ para uma grade de valores de u com incremento 0,1 está mostrado na Figura 4a. Algumas propriedades da função de ativação ReLU são listadas abaixo.

- Domínio: $u_k \in (-\infty, +\infty)$. Imagem: $y_k(t) \in [0, \infty)$.
- Usada como padrão na maioria das camadas ocultas de redes convolucionais profundas por ser computacionalmente leve e por ajudar a mitigar o problema do desvanecimento (ou enfraquecimento) do gradiente (*vanishing gradient problem*) em redes multicamadas.
- A derivada desta função não satura. Matematicamente, tem-se

$$\phi'_k(t) = \frac{d\phi(u_k(t))}{du_k(t)} = \begin{cases} 1, & u_k(t) > 0 \\ 0, & u_k(t) \leq 0 \end{cases} \quad (32)$$

- Produz ativação esparsa; ou seja, muitos neurônios são desativados; ou seja, produzem saída nula, o que melhora a eficiência computacional da rede.

Apesar de possuir muitas vantagens, ela também possui algumas desvantagens, sendo a principal o problema da **ReLU moribunda** (*dying ReLU*). Se muitos neurônios produzirem saídas nulas (especialmente para entradas negativas), eles podem “morrer” durante o treinamento e não serem mais ativados.

4.2 Leaky ReLU

Trata-se de uma variação da função ReLU que permite um gradiente pequeno e diferente de zero para entradas negativas, mitigando potencialmente o problema da ReLU moribunda. Matematicamente, tem-se a seguinte expressão para esta função de ativação:

$$y_k(t) = \begin{cases} u_k(t), & u_k(t) \geq 0 \\ \alpha u_k(t), & u_k(t) < 0 \end{cases} \quad (33)$$

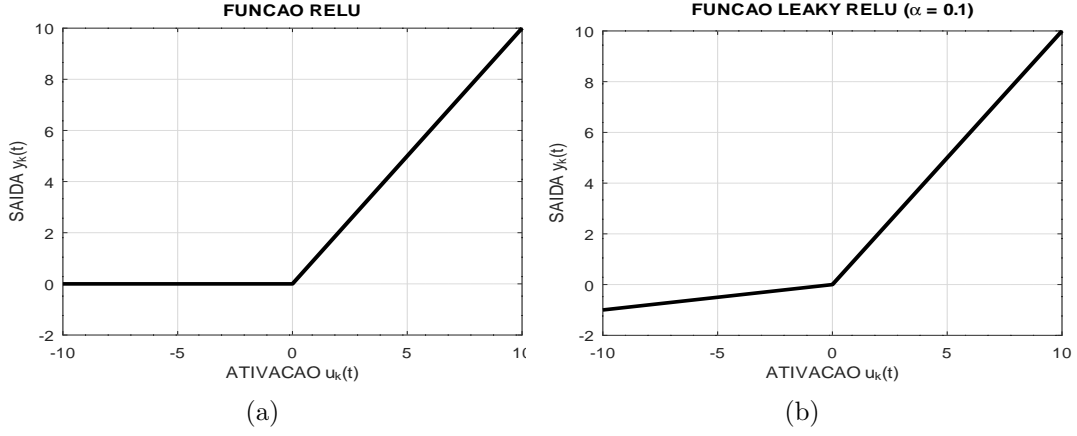


Figura 4: Funções de ativação: (a) ReLU e (b) Leaky ReLU.

em que $0 < \alpha \ll 1$ é uma constante muito pequena, assumindo geralmente o valor $\alpha = 0,01$. O gráfico da função Leaky ReLU no intervalo $(-10, 10)$ para uma grade de valores de u com incremento 0,1 está mostrado na Figura 4b.

Algumas propriedades da função de ativação *leaky* ReLU são listadas abaixo:

- Domínio: $u_k \in (-\infty, +\infty)$. Imagem: $y_k(t) \in (-\infty, +\infty)$.
- É computacionalmente leve e ajuda a mitigar o problema da ReLU moribunda.
- Também ajuda a mitigar o problema de neurônios desativados (neurônios mortos).
- A pequena inclinação para valores negativos de ativação pode tornar o treinamento e, conseqüentemente, a convergência da rede mais lenta.
- A derivada desta função é dada por

$$\phi'_k(t) = \frac{d\phi(u_k(t))}{du_k(t)} = \begin{cases} 1, & u_k(t) \geq 0 \\ \alpha, & u_k(t) < 0 \end{cases} \quad (34)$$

4.3 Exponential Linear Unit (ELU)

Esta função de ativação se assemelha à *Leaky ReLU*, mas com uma curva suave para ativações negativas, o que pode levar a um aprendizado mais rápido. Matematicamente, tem-se a seguinte expressão para esta função de ativação:

$$y_k(t) = \begin{cases} u_k(t), & u_k(t) \geq 0 \\ \alpha (e^{u_k(t)} - 1), & u_k(t) < 0 \end{cases} \quad (35)$$

A saída da ELU para valores positivos da ativação $u_k(t)$ é a própria ativação; ou seja, é uma função identidade para $u_k(t) \geq 0$. Se a entrada for negativa, a curva de saída será ligeiramente suavizada em direção à constante α . Quanto maior for esta constante, mais negativa será a saída para entradas negativas. O gráfico da função ELU no intervalo $(-10, 10)$ para uma grade de valores de u com incremento 0,1 está mostrado na Figura 5a.

Algumas propriedades da função de ativação ELU são listadas abaixo:

- Domínio: $u_k \in (-\infty, +\infty)$. Imagem: $y_k(t) \in (-\alpha, +\infty)$.

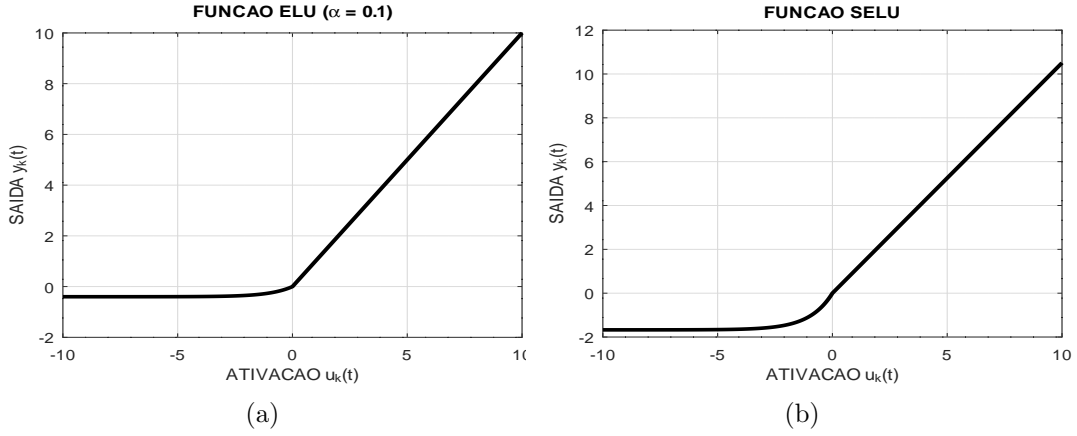


Figura 5: Funções de ativação: (a) ELU e (b) SeLU.

- Tende a induzir uma convergência mais rápida da rede que a função ReLU.
- Frequentemente conduz a um melhor desempenho de generalização que a função ReLU.
- É contínua e diferenciável para todo o domínio da função.
- Mitiga o problema do desvanecimento do gradiente.
- Mitiga o problema da explosão do gradiente.
- Mitiga o problema da ReLU moribunda.
- A derivada desta função é dada por

$$\phi'_k(t) = \frac{d\phi(u_k(t))}{du_k(t)} = \begin{cases} 1, & u_k(t) \geq 0 \\ \alpha e^{u_k(t)}, & u_k(t) < 0 \end{cases} \quad (36)$$

Como aspecto negativo, possui um custo computacional maior que a ReLU, devido à não linearidade da função exponencial aplicada aos valores de entrada negativos.

4.4 Scaled Exponential Linear Unit (SELU)

Esta função de ativação foi projetada para lidar com gradientes que desaparecem ou explodem. Matematicamente, ela é muito similar à função ELU:

$$y_k(t) = \begin{cases} \lambda u_k(t), & u_k(t) \geq 0 \\ \alpha (e^{u_k(t)} - 1), & u_k(t) < 0 \end{cases} \quad (37)$$

em que $\lambda \approx 1,0507$ a fim de promover a autonormalização da saída (média = 0, variância = 1) e $\alpha \approx 1,67326$. O gráfico da função SELU no intervalo $(-10, 10)$ para uma grade de valores de u com incremento 0,1 está mostrado na Figura 5b.

Basicamente, esta função de ativação mantém as mesmas propriedades da função ELU com o benefício adicional da autonormalização.

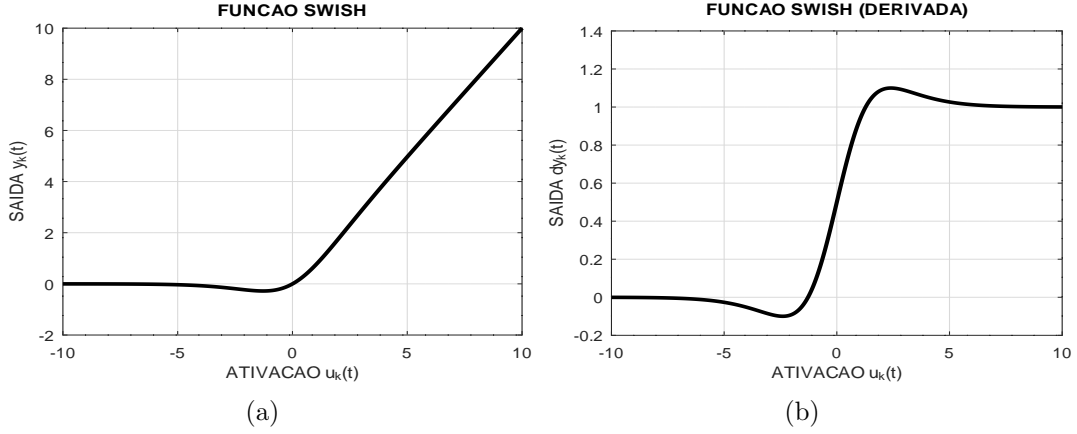


Figura 6: Função de ativação Swish e sua derivada.

4.5 Swish

Trata-se de uma função de ativação que combina a simplicidade da ReLU com algumas vantagens das funções sigmoidais. Matematicamente, tem-se

$$y_k(t) = \phi(u_k(t)) = u_k(t) \cdot \text{logsig}(u_k(t)), \quad (38)$$

em que $\text{logsig}(u)$ denota a função de ativação sigmoide logística. O gráfico da função Swish e de sua derivada no intervalo $(-10, 10)$ para uma grade de valores de u com incremento 0,1 está mostrado na Figura 6. Fica como exercício o cálculo matemático da derivada da função Swish.

A função de ativação Swish tem basicamente as mesmas vantagens que as funções ELU e SELU, sendo utilizadas principalmente em redes neurais muito profundas. Esta função foi proposta por pesquisadores da Google como uma alternativa à popular função de ativação ReLU².

5 Variações da Regra do Gradiente Descendente

Do mesmo modo pelo qual existem diversas funções de ativação, também existem variantes do método clássico de gradiente descendente estocástico (*stochastic gradient descent*, SGD) aqui instanciado pela Eq. (16). A partir desta equação, chega-se à regra recursiva de ajuste dos parâmetros (pesos e limiares) da rede PL mostrada na Eq. (23).

Nas subseções a seguir apresentaremos 7 variantes da regra LMS generalizada. Em geral, todas estas variantes são extensões bem simples da regra LMS generalizada, não oferecendo muito custo computacional ao processo de atualização dos pesos da rede PL.

5.1 LMS Normalizada

Tomando com o referência a Eq. (16), a regra LMS normalizada é escrita como

$$\mathbf{w}_k(t+1) = \mathbf{w}_k(t) + \frac{\alpha}{\gamma + \|\mathbf{x}(t)\|^2} \delta_k(t) \mathbf{x}(t), \quad (39)$$

$$= \mathbf{w}_k(t) + \frac{\alpha}{\gamma + \mathbf{x}(t)^T \mathbf{x}(t)} \delta_k(t) \mathbf{x}(t), \quad (40)$$

²<https://arxiv.org/pdf/1710.05941v1>

em que $\|\mathbf{x}\|^2 = \mathbf{x}^T \mathbf{x}$ é a norma euclidiana quadrática do vetor \mathbf{x} e $0 < \gamma \leq 1$ é uma constante usada para evitar divisão por zero. É comum usar $\gamma = 1$.

A regra LMS normalizada promove uma estabilização do aprendizado em função da norma do vetor de entrada $\mathbf{x}(t)$. Se a norma do vetor $\mathbf{x}(t)$ for alta (i.e., $\|\mathbf{x}\|^2 \gg \gamma$), situação comum quando há outliers, a taxa de aprendizado é dividida por $\gamma + \|\mathbf{x}\|^2$, desacelerando o aprendizado, evitando assim que o vetor de pesos seja atualizado de maneira brusca. Se a norma do vetor $\mathbf{x}(t)$ for pequena (i.e., $\|\mathbf{x}\|^2 \ll \gamma$), a taxa de aprendizado é aumentada para α/γ , acelerando o aprendizado sem desestabilizá-lo.

5.2 Leaky LMS

A regra leaky LMS é escrita como

$$\mathbf{w}_k(t+1) = (1 - \lambda)\mathbf{w}_k(t) + \alpha\delta_k(t)\mathbf{x}(t), \quad (41)$$

em que a constante $0 < \lambda < 1$ é chamada de parâmetro de fuga (*leaky*) ou de decaimento (*decay*).

A regra leaky LMS é popularmente conhecido na área de redes neurais artificiais como regularização por decaimento dos pesos³ (*weight decay regularization*). No caso em que o vetor de pesos \mathbf{w}_k não é atualizado (i.e., $\delta_k(t) = 0$), seu valor será diminuído de uma fração $-\lambda\mathbf{w}_k(t)$. Isso faz com que o vetor não assuma valores elevados, mantendo a sua norma pequena de modo equivalente à regularização L2, também conhecida como regularização de Tikhonov (ver material sobre regressão via método dos mínimos quadrados ordinários).

Curiosidade: As regras LMS normalizada e leaky podem ser combinadas para gerar uma variante: a regra LMS leaky normalizada.

5.3 LMS com Termo de Momento

A regra LMS com termo de momento é escrita como

$$\mathbf{w}_k(t+1) = \mathbf{w}_k(t) + \alpha\delta_k(t)\mathbf{x}(t) + \lambda\Delta\mathbf{w}_k(t-1), \quad (42)$$

em que $0 < \lambda < 1$ é o fator de momento e o termo $\Delta\mathbf{w}_k(t-1)$ é a correção aplicada ao vetor de pesos no passo anterior:

$$\Delta\mathbf{w}_k(t-1) = \alpha\delta_k(t-1)\mathbf{x}(t-1), \quad (43)$$

$$= \mathbf{w}_k(t) - \mathbf{w}_k(t-1). \quad (44)$$

A regra LMS com termo de momento confere estabilidade ao processo de atualização do vetor de pesos $\mathbf{w}_k(t)$ a cada instante t . O termo de momento $\lambda\Delta\mathbf{w}_k(t-1)$ evita que mudanças bruscas ocorram em $\mathbf{w}_k(t)$. Daí a origem do termo *momento*, referindo-se ao conceito físico de *momento de inércia*. Em outras palavras, a introdução do termo $\lambda\Delta\mathbf{w}_k(t-1)$ confere maior *inércia* (ou seja, maior dificuldade de variação) nas correções em $\mathbf{w}_k(t)$.

Um resultado muito interessante sobre essa variante da regra LMS é que para funções quadráticas (e.g., erro médio quadrático), ela é uma versão do método do gradiente conjugado⁴.

³Anders Krogh & John A Hertz (1992). “A simple weight decay can improve generalization”. In: Advances in Neural Information Processing Systems, pp. 950-957.

⁴<http://www.nacad.ufrj.br/~amit/bk03nn.pdf>

5.4 Variantes Adagrad/RMSProp/ADAM

Nesta seção, descreveremos a regra LMS/Adagrad e duas variantes, a saber, a regra RMSProp (*Root Mean Square Propagation*) e a regra ADAM (*Adaptive Moment Estimation*).

O principal conceito por trás da regra LMS/Adagrad é a ideia de ter uma taxa de aprendizado adaptativa, que dependa da soma histórica dos gradientes quadráticos usados na atualização do vetor de pesos⁵. Assim, para cada instante t , o gradiente da função custo (e.g., MSE) com relação ao k -ésimo vetor de pesos é calculado, assim como na regra LMS padrão:

$$\mathbf{g}_k(t) = \frac{\partial J_k(t)}{\partial \mathbf{w}_k(t)}, \quad (45)$$

de modo que o a soma acumulada G_t do quadrado dos gradientes é dada por

$$G_k(t) = G_k(t-1) + \|\mathbf{g}_k(t)\|^2, \quad (46)$$

$$= G_k(t-1) + \left\| \frac{\partial J_k(t)}{\partial \mathbf{w}_k(t)} \right\|^2, \quad (47)$$

$$= G_k(t-1) + \delta_k^2(t) \mathbf{x}^T(t) \mathbf{x}(t). \quad (48)$$

Assim, a regra LMS/Adagrad é escrita como

$$\mathbf{w}_k(t+1) = \mathbf{w}_k(t) + \frac{\alpha}{\sqrt{\gamma + G_k(t)}} \delta_k(t) \mathbf{x}(t), \quad (49)$$

em que $0 < \gamma < 1$ é uma constante pequena usada para evitar a divisão por zero. O termo $\frac{1}{\sqrt{\gamma + G_k(t)}}$ reduz efetivamente a taxa de aprendizado para os vetores de pesos que têm um gradiente acumulado grande. Em contrapartida, os vetores de pesos com gradientes acumulados menores terão uma taxa de aprendizado maior.

Já a regra RMSProp é uma variante da regra LMS/Adagrad em que a atualização do termo $G_k(t)$ envolve uma combinação convexa dos termos $G_k(t-1)$ e $\|\mathbf{g}_k(t)\|^2$. Matematicamente, tem-se

$$G_k(t) = \lambda G_k(t-1) + (1-\lambda) \|\mathbf{g}_k(t)\|^2, \quad (50)$$

$$= \lambda G_k(t-1) + (1-\lambda) \left\| \frac{\partial J_k(t)}{\partial \mathbf{w}_k(t)} \right\|^2, \quad (51)$$

$$= \lambda G_k(t-1) + (1-\lambda) \delta_k^2(t) \mathbf{x}^T(t) \mathbf{x}(t). \quad (52)$$

em que $0 < \lambda < 1$ é o fator de decaimento (tipicamente assume o valor $\lambda = 0,9$). Em outras palavras, a regra LMS/RMSProp é como uma regra *leaky* LMS/Adagrad.

Por fim, a regra LMS/ADAM combina os benefícios das regras LMS/Adagrad e LMS com termo de momento ao usar tanto a média móvel dos gradientes, usada para suavizar (ou filtrar) variações bruscas nas componentes do vetor gradiente causadas por ruído estocástico, quanto os gradientes quadráticos para adaptar a taxa de aprendizado⁶. A regra LMS/ADAM necessita das seguintes etapas:

1. Estimção do 1o. Termo (vetor gradiente filtrado):

$$\begin{aligned} \mathbf{m}_k(t+1) &= \beta_1 \mathbf{m}_k(t) + (1-\beta_1) \frac{\partial J_k(t)}{\partial \mathbf{w}_k(t)}, \\ &= \beta_1 \mathbf{m}_k(t) + (1-\beta_1) \delta_k(t) \mathbf{x}(t), \end{aligned} \quad (53)$$

em que $0 < \beta_1 < 1$ é uma constante (parâmetro de suavização). Valor típico: $\beta_1 = 0,9$.

⁵<https://jmlr.org/papers/volume12/duchi11a/duchi11a.pdf>

⁶<https://arxiv.org/abs/1412.6980>

2. Estimação do 2o. Termo (variância):

$$\begin{aligned} v_k(t+1) &= \beta_2 v_k(t) + (1 - \beta_2) \left\| \frac{\partial J_k(t)}{\partial \mathbf{w}_k(t)} \right\|^2, \\ &= \beta_2 v_k(t) + (1 - \beta_2) \delta_k^2(t) \mathbf{x}^T(t) \mathbf{x}(t), \end{aligned} \quad (54)$$

em que $0 < \beta_2 < 1$ é uma constante. Valor típico: $\beta_2 = 0,999$.

3. Correção de viés para 1o. e 2o. Termos:

$$\hat{\mathbf{m}}_k(t) = \frac{\mathbf{m}_k(t+1)}{1 - \beta_1} \quad \text{e} \quad \hat{v}_k(t) = \frac{v_k(t)}{1 - \beta_2} \quad (55)$$

4. A Regra LMS/ADAM em si:

$$\mathbf{w}_k(t+1) = \mathbf{w}_k(t) - \frac{\alpha}{\sqrt{\hat{v}_k(t)} + \gamma} \hat{\mathbf{m}}_k(t), \quad (56)$$

em que $0 < \gamma \leq 1$ é uma constante usada para evitar divisão por zero.

Comentário Final: Não há uma função de ativação ou uma variante da regra LMS que funcione melhor que todas as outras para todas as aplicações. Portanto, é necessário experimentar duas ou mais combinações a fim de chegar na melhor solução para o problema em mãos. Comece sempre pela configuração mais básica, tipo logsig+LMS padrão, em seguida vá testando combinações até encontrar uma solução que satisfaça o problema com o menor custo computacional possível.

6 Exercícios Computacionais

Exercício 1 - Qual é a implicação do aparecimento do fator $\phi'_k(t)$ na regra delta generalizada? Dica: Pesquise por *paralisia da rede* em sítios da internet.

Exercício 2 - Escolher um conjunto de dados no Repositório da Universidade da Califórnia em Irvine (<http://www.ics.uci.edu/~mllearn/MLSummary.html>) e utilizar os pares de vetores entrada-saída correspondentes para avaliar o Perceptron simples Logístico em um problema de classificação de padrões e em um problema de regressão. Determinar a curva de aprendizagem (EQM em função da época de treinamento) e os índices de desempenhos correspondentes a cada tarefa. Classificação: taxas de acerto médio, desvio padrão, mediana, mínimo e máximo. Regressão: R2 médio, desvio padrão, mediana, mínimo e máximo.

Referências

- [1] B. Widrow and M. A. Lehr. Perceptrons, adalines, and backpropagation. In M. A. Arbib, editor, *The Handbook of Brain Theory and Neural Networks*, pages 871–877. MIT Press, 2nd edition, 2002.
- [2] B. Widrow and M. A. Lehr. 30 years of adaptive neural networks: Perceptron, madaline and backpropagation. *Proceedings of the IEEE*, 78(9):1415–1442, 1990.

2 REDE MLP PARA CLASSIFICAÇÃO DE PADRÕES

Neste capítulo descreve-se a rede MLP, suas regras de ativação e aprendizagem, apresentadas em uma notação matricial não-usual e que servirá de base para a metodologia proposta.

2.1 Introdução

As redes neurais artificiais (RNAs) são máquinas de aprendizagem não-linear formadas por vários elementos processadores simples, chamados usualmente de neurônios artificiais. O modelo desses neurônios foi proposto inicialmente por McCulloch & Pitts (1943), seguindo a ideia que os neurônios tem capacidade de adaptação em função de estímulos (informação) oriundos do meio em que estão inseridos, lançando mão de processamento paralelo e distribuído para processar e codificar a informação nas conexões com outros neurônios.

As RNAs apresentam-se como uma ferramenta computacional eficiente no tratamento de problemas não-lineares em processamento de sinais (HWANG et al., 1997; ZAKNICH, 2003). Estes problemas requerem mapeamentos entrada-saída não-lineares, e são comumente encontrados em aproximação de funções e classificação de padrões. Este último é tratado no decorrer desta dissertação, enquanto maiores esclarecimentos sobre a aplicação de redes neurais para aproximação de funções podem ser encontrados em Haykin (2001), Príncipe et al. (2000).

Na utilização de RNAs em tarefas de classificação de padrões deve-se associar a cada padrão de entrada (representado por um vetor-coluna de características) uma das classes predefinidas. As chamadas funções discriminantes ou superfícies de decisão serão construídas a partir de um conjunto de padrões de treinamento, com rótulos de classes conhecidas, para separar os padrões de classes diferentes. As regiões de decisão podem ser lineares, li-

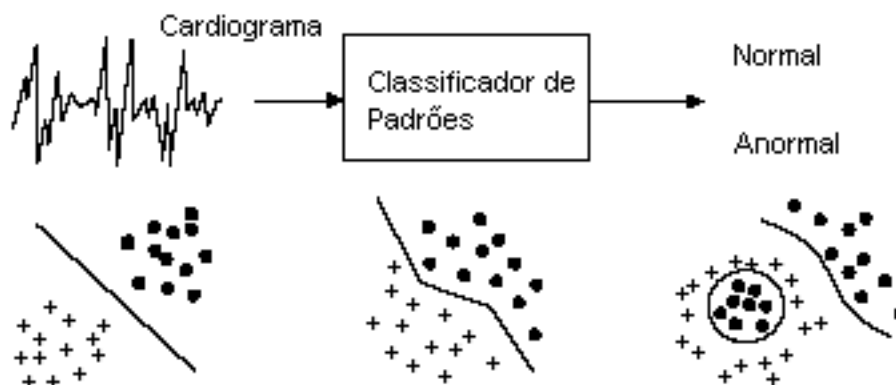


Figura 2.1 Exemplo de uma tarefa de classificação de padrões, adaptada da referência Jain et al. (1996).

neares por partes ou de alguma forma arbitrária, conforme podem ser vistas na Figura 2.1.

Algumas das principais aplicações de RNAs em classificação de padrões são em reconhecimento de caracteres, reconhecimento de imagens, reconhecimento de voz, diagnóstico de patologias, dentre outras. As RNAs são consideradas classificadores semi-paramétricos, uma vez que as funções discriminantes mudam de acordo com a topologia da rede (PRINCIPE et al., 2000).

Das diversas topologias existentes para RNAs uma bastante popular é a rede perceptron, inventada por Rosenblatt (1958) na década de 50. A rede perceptron tem múltiplas entradas totalmente conectadas a uma única camada, cujos neurônios determinam a saída da rede. Esta configuração recebe o nome de perceptron simples (PS) e é aplicável somente a problemas de classificação considerados linearmente separáveis.

Uma outra configuração da rede perceptron é a rede perceptron multicamadas (MLP), obtida a partir da rede PS com a inserção de camadas ocultas de neurônios, ou seja, camadas com elementos de processamento que não estão conectados diretamente com a saída da rede.

De particular interesse para este trabalho, a rede MLP com uma camada oculta, além de poder ser aplicada em problemas de classificação não-lineares, pode ser considerada como um aproximador universal de função (CYBENKO, 1989; FUNAHASHI, 1989; HORNIK et al., 1989). Assim, desde que a rede MLP possua uma quantidade suficiente de neurônios ocultos, ela pode aproximar qualquer função contínua com uma certa precisão.

Portanto, este capítulo tem por objetivo mostrar sucintamente a arquitetura da rede

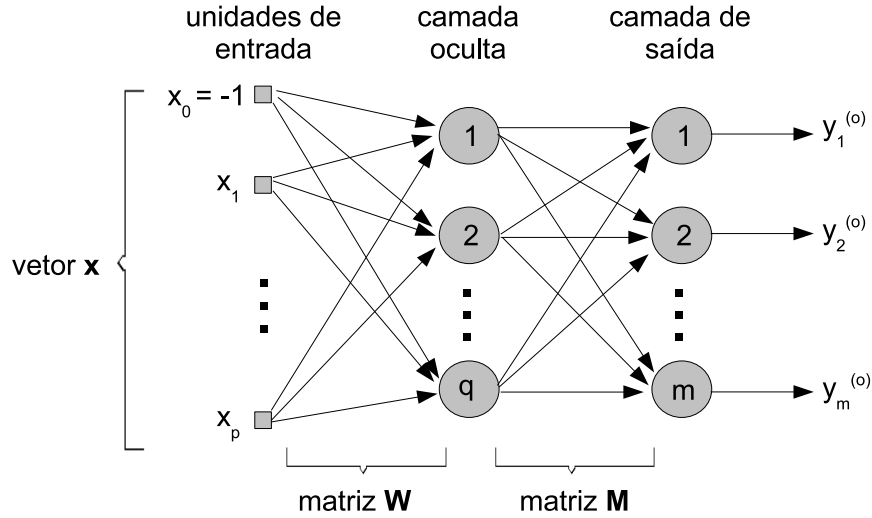


Figura 2.2 Rede MLP com uma camada oculta.

MLP utilizada no decorrer da dissertação. Será feita uma breve discussão sobre os algoritmos envolvidos, fundamentos, características, propriedades e as notações vetoriais e matriciais utilizadas para a rede MLP nos próximos capítulos.

2.2 Arquitetura da Rede MLP

Basicamente uma rede MLP é constituída de um conjunto de unidades de entrada (que recebem os sinais), uma ou mais camadas ocultas (ou escondidas) compostas por neurônios não-lineares, e uma camada de saída composta por um ou mais neurônios que podem ser lineares ou não-lineares. Conforme mencionado anteriormente, os neurônios das camadas intermediárias são chamados de neurônios ocultos ou escondidos pelo fato de não terem acesso direto à saída da rede. Embora possa ter qualquer número de camadas ocultas, o enfoque nesta dissertação é a rede MLP com apenas uma camada de neurônios ocultos.

A Figura 2.2 mostra a arquitetura de uma rede MLP *feedforward* (alimentada adiante) com uma camada oculta e totalmente conectada. Isto significa que um neurônio em qualquer camada da rede é conectado à todas as unidades/neurônios da camada anterior, e que um sinal de entrada ou funcional progride na rede para frente, da esquerda para direita e de camada em camada (HAYKIN, 2001). Nesta arquitetura x_1, x_2, \dots, x_p são as componentes do vetor de entrada \mathbf{x} o qual deve ser acrescido da componente fixa $x_0 = -1$, correspondente ao limiar ou bias. Portanto o vetor \mathbf{x} , de dimensionalidade $p+1$ (incluindo

o limiar), pode então ser representado na iteração t por

$$\mathbf{x}(t) = \begin{bmatrix} -1 \\ x_1(t) \\ \vdots \\ x_p(t) \end{bmatrix}. \quad (2.1)$$

É possível definir a matriz $\mathbf{X} \in \mathbb{R}^{(p+1) \times N}$ reunindo todos os vetores-coluna acima, para as N amostras do conjunto de dados utilizado na etapa de treinamento da rede. A matriz \mathbf{X} pode então ser escrita como

$$\mathbf{X} = [\mathbf{x}(1) \mid \mathbf{x}(2) \mid \dots \mid \mathbf{x}(N)]. \quad (2.2)$$

Na Figura 2.2, q ($2 \leq q < \infty$) representa o número de neurônios da camada oculta, os quais desempenham um papel crucial na rede MLP porque agem como detectores de características (HAYKIN, 2001). Conforme o processo de aprendizagem da rede avança, estes neurônios começam gradualmente a "descobrir" as características salientes presentes nos dados de treinamento. Daí então realizam uma transformação não-linear nos dados de entrada para um novo espaço, chamado espaço oculto ou espaço de características. Neste novo espaço, as classes de interesse em uma tarefa de classificação de padrões podem ser mais facilmente separadas entre si comparativamente ao espaço original de entrada (HAYKIN, 2001).

Na camada de saída da rede MLP, m ($m \geq 1$) designa o número de neurônios de saída, os quais detêm as saídas reais obtidas pela rede. Normalmente o número de neurônios de saída é igual ao número C de classes em um problema de classificação de padrões, na chamada codificação *1-out-of- Q* . Uma outra possibilidade é dizer que m é o menor inteiro igual a ou maior que \sqrt{C} , na denominada *codificação binária simples*.

A matriz $\mathbf{W} \in \mathbb{R}^{q \times (p+1)}$ é formada por todas as conexões (ou pesos) sinápticas entre as unidades de entrada e os neurônios da camada oculta. Portanto, cada elemento w_{ij} de \mathbf{W} na Figura 2.2 representa a conexão sináptica entre a j -ésima entrada e o i -ésimo neurônio da camada oculta. A matriz \mathbf{W} pode então ser representada por

$$\mathbf{W} = \begin{bmatrix} \mathbf{w}_1^T \\ \mathbf{w}_2^T \\ \vdots \\ \mathbf{w}_Q^T \end{bmatrix}, \quad (2.3)$$

em que cada linha $\mathbf{w}_i^T = [w_{i0} \ w_{i1} \ \dots \ w_{ip}]$ corresponde aos pesos sinápticos entre o i -ésimo neurônio oculto e as $p + 1$ unidades de entrada, incluindo o limiar.

Seguindo o mesmo princípio, define-se $\mathbf{M} \in \mathbb{R}^{m \times (q+1)}$ como a matriz formada pelas conexões sinápticas entre os neurônios da camada oculta e os neurônios da camada de saída. Cada entrada m_{ki} de \mathbf{M} representa a conexão sináptica entre o i -ésimo neurônio oculto e o k -ésimo neurônio de saída. \mathbf{M} pode então ser escrita como

$$\mathbf{M} = \begin{bmatrix} \mathbf{m}_1^T \\ \mathbf{m}_2^T \\ \vdots \\ \mathbf{m}_m^T \end{bmatrix}, \quad (2.4)$$

em que cada linha $\mathbf{m}_k^T = [m_{k0} \ m_{k1} \ \dots \ m_{kq}]$ corresponde ao vetor de pesos sinápticos do k -ésimo neurônio de saída, incluindo o limiar.

Uma vez mostrada a arquitetura da rede MLP com uma camada oculta e apresentados os principais parâmetros, faz-se necessária uma breve discussão sobre o algoritmo de retropropagação do erro (*error back-propagation*), utilizado para a aprendizagem supervisionada da rede.

2.3 O Algoritmo de Retropropagação do Erro

A técnica utilizada nesta dissertação para o treinamento de uma rede MLP é o algoritmo de retropropagação do erro (*error back-propagation*), que pode ser visto com maiores detalhes em Principe et al. (2000), Haykin (2001) e Bishop (2005). Este nome foi dado em função de o algoritmo propagar o sinal de erro a partir dos neurônios de saída em direção a todos os neurônios da camada oculta, percorrendo assim o caminho inverso do processamento dos dados de entrada. Foi popularizado no trabalho de Rumelhart et al. (1986), embora idéia similar tenha sido desenvolvida mais cedo por outros pesquisadores, tais como Werbos (1974) e Parker (1985).

De uma forma geral, a técnica de aprendizagem por retropropagação do erro envolve

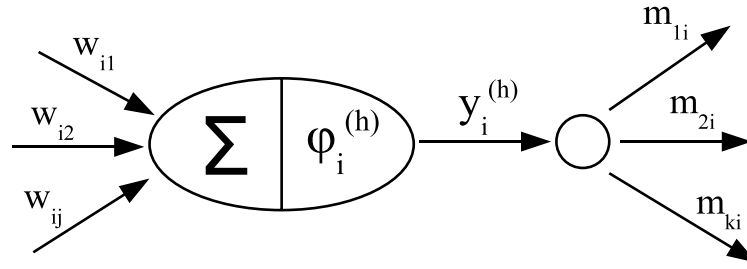


Figura 2.3 Saída de um neurônio oculto.

dois sentidos de propagação de sinais através da rede, o sentido direto e o sentido reverso (ou dual). No primeiro, cada vetor de dados é aplicado a cada uma das unidades de entrada da rede para que seu efeito se propague, camada por camada, até que os neurônios da camada de saída determinem as respostas reais da rede. Nesta fase todos os pesos sinápticos permanecem inalterados. No sentido reverso ocorre a atualização dos pesos sinápticos de acordo com uma regra de correção do erro (função custo). Este sinal de erro é então propagado para trás através da rede, camada por camada. Os pesos sinápticos são ajustados para fazer com que a resposta real da rede se mova aleatoriamente para mais perto da resposta desejada (HAYKIN, 2001). As seções seguintes discutem o treinamento da rede MLP com uma camada oculta nos sentidos direto e reverso.

2.3.1 Sentido Direto da Rede

O sentido direto da rede começa com a apresentação de um vetor de entrada \mathbf{x} na iteração t . A ativação de um neurônio da camada oculta é dada por

$$u_i^{(h)}(t) = \sum_{j=1}^p w_{ij}(t)x_j(t) - \theta_i^{(h)}(t) = \sum_{j=0}^p w_{ij}(t)x_j(t), \quad i = 1, \dots, q \quad (2.5)$$

em que $\theta_i^{(h)}(t)$ é o limiar do i -ésimo neurônio oculto. Define-se $x_0(t) = -1$ e $w_{i0}(t) = \theta_i^{(h)}(t)$ para simplificação da notação. Os demais parâmetros foram definidos na seção anterior.

A saída calculada para o i -ésimo neurônio oculto, conforme ilustrado na Figura 2.3, é então dada por

$$y_i^{(h)}(t) = \varphi_i \left[u_i^{(h)}(t) \right] = \varphi_i \left[\sum_{j=0}^p w_{ij}(t)x_j(t) \right], \quad (2.6)$$

em que $\varphi_i(\cdot)$ é uma não-linearidade do tipo sigmoidal, possuindo a forma da letra S esticada. Normalmente são utilizados dois tipos de função de ativação, a sigmóide logística dada por

$$y(t) = \varphi(t) = \frac{1}{1 + \exp\{-u(t)\}}, \quad (2.7)$$

e a tangente hiperbólica, cuja expressão é dada por

$$y(t) = \varphi(t) = \frac{1 - \exp\{-u(t)\}}{1 + \exp\{-u(t)\}}. \quad (2.8)$$

O domínio destas funções é a reta dos números reais. Contudo, a imagem da função sigmóide logística está restrita ao intervalo $(0, 1)$, enquanto a imagem da tangente hiperbólica está restrita a $(-1, +1)$. De um extremo a outro ambas são sempre crescentes. Os gráficos das funções sigmóide logística e tangente hiperbólica estão mostrados na Figura 2.4.

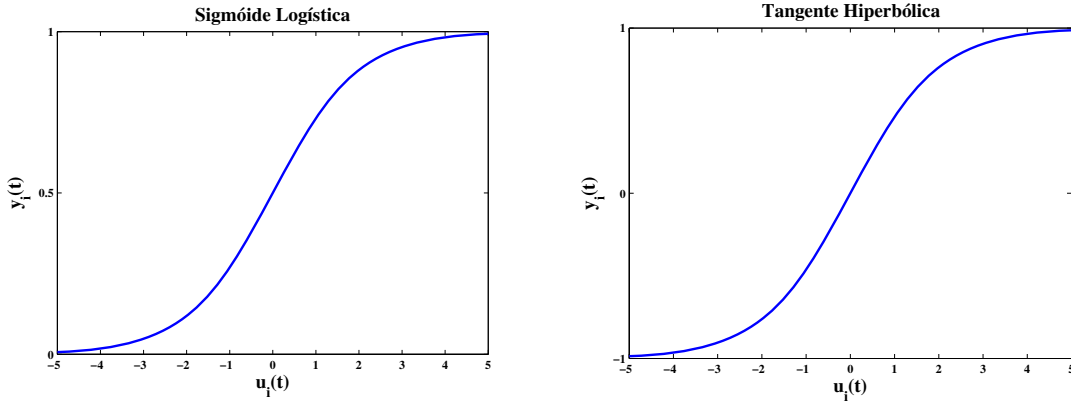
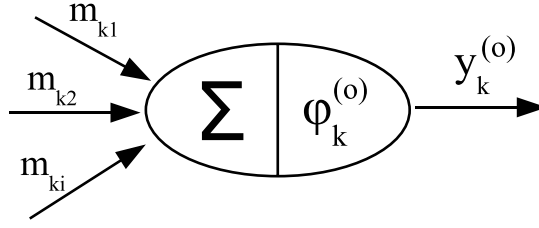


Figura 2.4 Funções de ativação sigmóide logística e tangente hiperbólica.

A Equação (2.6) pode ser representada na forma matricial como

$$\mathbf{Y}^{(h)} = \varphi_i[\mathbf{WX}], \quad (2.9)$$

em que a função de ativação sigmoidal $\varphi_i[\cdot]$ é aplicada a cada componente do produto matricial no cálculo de $\mathbf{Y}^{(h)}$. A matriz $\mathbf{Y}^{(h)} \in \mathbb{R}^{q \times N}$ armazena as saídas dos q neurônios ocultos, calculadas para todos os N exemplos do conjunto de dados utilizados no treinamento da rede. Na forma expandida $\mathbf{Y}^{(h)}$ é representada por

**Figura 2.5** Resposta de um neurônio de saída.

$$\mathbf{Y}^{(h)} = \begin{bmatrix} y_1^{(h)}(1) & y_1^{(h)}(2) & \cdots & y_1^{(h)}(N) \\ y_2^{(h)}(1) & y_2^{(h)}(2) & \cdots & y_2^{(h)}(N) \\ \vdots & \vdots & \vdots & \vdots \\ y_q^{(h)}(1) & y_q^{(h)}(2) & \cdots & y_q^{(h)}(N) \end{bmatrix}. \quad (2.10)$$

De forma similar aos neurônios ocultos, e conforme ilustrado na Figura 2.5, a resposta calculada para o k -ésimo neurônio da camada de saída é dada por

$$y_k^{(o)}(t) = \varphi_k \left[u_k^{(o)}(t) \right] = \varphi_k \left[\sum_{i=0}^q m_{ki}(t) y_i^{(h)}(t) \right], \quad (2.11)$$

em que define-se $y_0^{(h)}(t) = -1$ e $m_{k0}(t) = \theta_k^{(o)}(t)$ é o limiar do k -ésimo neurônio de saída. A Equação (2.11) também pode ser escrita na forma matricial como

$$\mathbf{Y}^{(o)} = \varphi_k \left[\mathbf{M} \mathbf{Y}^{+(h)} \right], \quad (2.12)$$

em que a matriz $\mathbf{Y}^{+(h)}$ é obtida a partir da matriz $\mathbf{Y}^{(h)}$, adicionando-se uma primeira linha com todos os elementos iguais a -1 , equivalente ao limiar da camada oculta conectado a todos os neurônios de saída. $\mathbf{Y}^{+(h)}$ é representada na forma expandida por

$$\mathbf{Y}^{+(h)} = \begin{bmatrix} -1 & -1 & \cdots & -1 \\ y_1^{(h)}(1) & y_1^{(h)}(2) & \cdots & y_1^{(h)}(N) \\ y_2^{(h)}(1) & y_2^{(h)}(2) & \cdots & y_2^{(h)}(N) \\ \vdots & \vdots & \vdots & \vdots \\ y_q^{(h)}(1) & y_q^{(h)}(2) & \cdots & y_q^{(h)}(N) \end{bmatrix}. \quad (2.13)$$

É possível portanto afirmar que a matriz $\mathbf{Y}^{(o)} \in \mathbb{R}^{m \times N}$ armazena as respostas dos m neurônios de saída, calculadas para todos os N vetores do conjunto de treinamento. Na

forma expandida $\mathbf{Y}^{(o)}$ é representada por

$$\mathbf{Y}^{(o)} = \begin{bmatrix} y_1^{(o)}(1) & y_1^{(o)}(2) & \cdots & y_1^{(o)}(N) \\ y_2^{(o)}(1) & y_2^{(o)}(2) & \cdots & y_2^{(o)}(N) \\ \vdots & \vdots & \vdots & \vdots \\ y_m^{(o)}(1) & y_m^{(o)}(2) & \cdots & y_m^{(o)}(N) \end{bmatrix}. \quad (2.14)$$

Uma vez obtidas as respostas para todos os m neurônios de saída, é finalizado o sentido direto do treinamento da rede MLP.

2.3.2 Sentido Reverso da Rede

O sentido reverso ou dual começa na camada de saída, com a determinação do erro $e_k^{(o)}(t)$ produzido por cada um dos neurônios de saída da rede na iteração t , e expresso por

$$e_k^{(o)}(t) = d_k(t) - y_k^{(o)}(t), \quad k = 1, \dots, m \quad (2.15)$$

em que $d_k(t)$ é a saída desejada ou saída-alvo para o k -ésimo neurônio da camada de saída.

O valor instantâneo da energia do erro para o k -ésimo neurônio de saída é dado por $\frac{1}{2}e_k^2(t)$. O valor instantâneo da energia total do erro, também chamado de erro quadrático instantâneo, pode ser obtido somando-se os termos $\frac{1}{2}e_k^2(t)$ para todos os m neurônios de saída, e é escrito como

$$J(t) = \frac{1}{2} \sum_{k=1}^m e_k^2(t) = \frac{1}{2} \sum_{k=1}^m (d_k(t) - y_k^{(o)}(t))^2. \quad (2.16)$$

Para a obtenção da regra de aprendizagem para a rede MLP, a função custo de interesse é o *Erro Quadrático Médio* (EQM) calculado para os N vetores de treinamento, e dado por

$$J(Z) = \frac{1}{N} \sum_{t=1}^N J(t) = \frac{1}{2N} \sum_{t=1}^N \sum_{k=1}^m e_k^2(t) = \frac{1}{2N} \sum_{t=1}^N \sum_{k=1}^m (d_k(t) - y_k^{(o)}(t))^2, \quad (2.17)$$

em que Z é o conjunto de todos os parâmetros (pesos e limiares) da rede, incluindo os elementos das matrizes W e M . Pela Equação (2.17), a função $J(Z)$ pode ser minimizada

ao se minimizar a função $J(t)$. Portanto, o algoritmo de retropropagação do erro aplica uma correção $\Delta w_{ji}(t)$ ao peso sináptico $w_{ji}(t)$, que é proporcional ao gradiente da função $J(t)$, expresso por $\partial J(t)/\partial w_{ji}(t)$.

Com os neurônios de saída, é necessário então determinar o gradiente de $J(t)$ na direção do peso $m_{ki}(t)$ da camada de saída, dado por $\partial J(t)/\partial m_{ki}(t)$. Para se calcular esta derivada faz-se uso da regra da cadeia para fatorá-la em vários termos. Como se tratam de funções contínuas, a fatoração pode ser escrita da seguinte forma

$$\frac{\partial J(t)}{\partial m_{ki}(t)} = \frac{\partial J(t)}{\partial e_k(t)} \frac{\partial e_k(t)}{y_k^{(o)}(t)} \frac{\partial y_k^{(o)}(t)}{\partial u_k^{(o)}(t)} \frac{\partial u_k^{(o)}(t)}{\partial m_{ki}(t)}, \quad (2.18)$$

em que cada uma das derivadas parciais pode ser calculada como

$$\frac{\partial J(t)}{\partial e_k(t)} = e_k(t), \quad (2.19)$$

$$\frac{\partial e_k(t)}{y_k^{(o)}(t)} = -1, \quad (2.20)$$

$$\frac{\partial y_k^{(o)}(t)}{\partial u_k^{(o)}(t)} = \varphi'_k \left[u_k^{(o)}(t) \right] = y'_k(t), \quad (2.21)$$

$$\frac{\partial u_k^{(o)}(t)}{\partial m_{ki}(t)} = y_i^{(h)}(t), \quad (2.22)$$

em que $\varphi'_k[\cdot]$ é a derivada da função de ativação, que normalmente assume uma entre duas formas possíveis, caso a função de ativação seja a sigmóide logística ou a tangente hiperbólica. Para o caso da sigmóide logística sua derivada é dada por

$$y'(t) = \frac{dy(t)}{du_i(t)} = y(t)[1 - y(t)], \quad (2.23)$$

e para a tangente hiperbólica

$$y'(t) = \frac{dy(t)}{du_i(t)} = 0,5[1 - y^2(t)]. \quad (2.24)$$

Os gráficos das derivadas das funções sigmóide logística e tangente hiperbólica estão mostrados na Figura 2.6.

Por fim, o gradiente na Equação (2.18) pode ser simplificado para

$$\frac{\partial J(t)}{\partial m_{ki}(t)} = -e_k(t) \varphi'_k \left[u_k^{(o)}(t) \right] y_i^{(h)}(t). \quad (2.25)$$

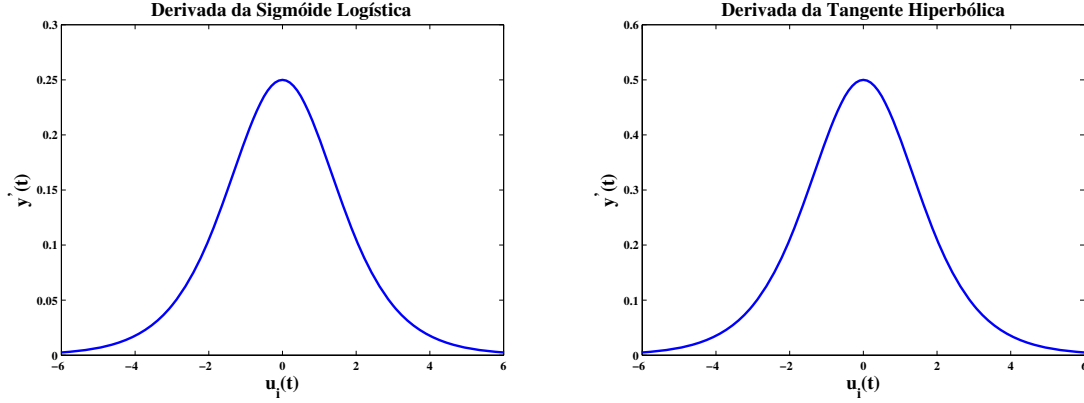


Figura 2.6 Derivadas da sigmóide logística e tangente hiperbólica.

Define-se agora a regra de aprendizagem para a atualização dos pesos dos neurônios de saída como

$$\begin{aligned}
 m_{ki}(t+1) &= m_{ki}(t) + \Delta m_{ki}(t) \\
 &= m_{ki}(t) - \eta \frac{\partial J(t)}{\partial m_{ki}(t)} \\
 &= m_{ki}(t) + \eta e_k(t) \varphi'_k \left[u_k^{(o)}(t) \right] y_i^{(h)}(t),
 \end{aligned} \tag{2.26}$$

em que a constante η é chamada de taxa de aprendizagem ou passo de adaptação, e normalmente assume valores entre 0 e 1. A Equação (2.26) é conhecida como *Regra Delta Generalizada*, a qual é derivada a partir da *Regra Delta*, também chamada de *Regra LMS (Least Mean Square)* ou *Regra de Widrow-Hoff* (WIDROW; HOFF, 1960). A Equação (2.26) pode ser reescrita como

$$m_{ki}(t+1) = m_{ki}(t) + \eta \delta_k^{(o)}(t) y_i^{(h)}(t), \tag{2.27}$$

em que define-se $\delta_k^{(o)}(t)$ como o gradiente local do k -ésimo neurônio de saída (ver Figura 2.7), dado por

$$\delta_k^{(o)}(t) = \varphi'_k(t) e_k^{(o)}(t), \tag{2.28}$$

em que $\varphi'_k(t) = \varphi'_k \left[u_k^{(o)}(t) \right]$ para simplificar a notação.

A atualização dos pesos nos neurônios da camada oculta (matriz \mathbf{W}) segue um proce-

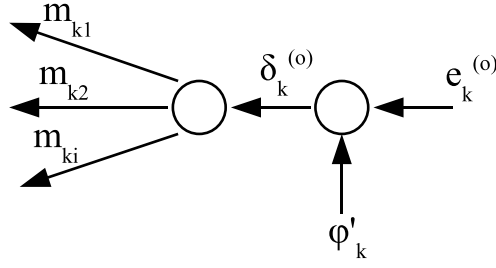


Figura 2.7 Retropropagação do erro em um neurônio de saída.

dimento similar ao que foi feito para os neurônios de saída (matriz \mathbf{M}). Novamente deve ser calculado o gradiente da função custo $J(t)$, agora na direção do peso $w_{ij}(t)$. Com o auxílio da regra da cadeia este gradiente pode ser fatorado como

$$\frac{\partial J(t)}{\partial w_{ij}(t)} = \frac{\partial J(t)}{\partial y_i^{(h)}(t)} \frac{\partial y_i^{(h)}(t)}{\partial u_i^{(h)}(t)} \frac{\partial u_i^{(h)}(t)}{\partial w_{ij}(t)}, \quad (2.29)$$

em que cada derivada parcial é calculada como

$$\frac{\partial J(t)}{\partial y_i^{(h)}(t)} = - \sum_{k=1}^m m_{ki} \varphi'_k \left[u_k^{(o)}(t) \right] e_k(t), \quad (2.30)$$

$$\frac{\partial y_i^{(h)}(t)}{\partial u_i^{(h)}(t)} = \varphi'_i \left[u_i^{(h)}(t) \right], \quad (2.31)$$

$$\frac{\partial u_i^{(h)}(t)}{\partial w_{ij}(t)} = x_j(t). \quad (2.32)$$

Usando novamente a *Regra Delta Generalizada*, a atualização dos pesos dos neurônios ocultos pode ser expressa por

$$\begin{aligned} w_{ij}(t+1) &= w_{ij}(t) + \Delta w_{ij}(t) \\ &= w_{ij}(t) - \eta \frac{\partial J(t)}{\partial w_{ij}(t)} \\ &= w_{ij}(t) + \eta \varphi'_i \left[u_i^{(h)}(t) \right] \sum_{k=1}^m m_{ki} \delta_k^{(o)}(t) x_j(t), \end{aligned} \quad (2.33)$$

em que define-se o gradiente local $\delta_i^{(h)}(t)$ do i -ésimo neurônio da camada oculta (ver Figura 2.8) como

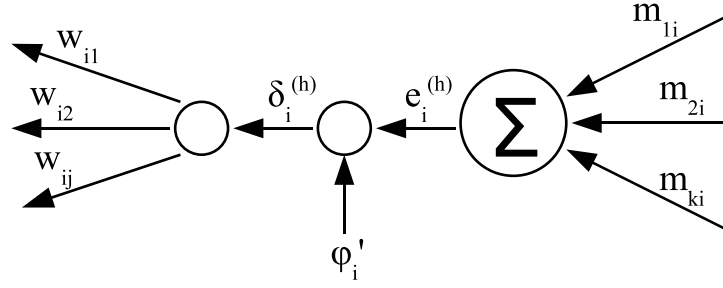


Figura 2.8 Retropropagação do erro em um neurônio oculto.

$$\delta_i^{(h)}(t) = \varphi'_i \left[u_i^{(h)}(t) \right] \sum_{k=1}^m m_{ki} \delta_k^{(o)}(t) = \varphi'_i \left[u_i^{(h)}(t) \right] e_i^{(h)}(t), \quad i = 0, \dots, q, \quad (2.34)$$

em que o termo $e_i^{(h)}(t)$ é considerado o sinal de erro retropropagado no i -ésimo neurônio da camada oculta, uma vez que os sinais de erro na camada oculta são combinações lineares dos verdadeiros erros da rede, os quais são obtidos nos neurônios da camada de saída. Assim, a Equação (2.33) pode finalmente ser reescrita como

$$w_{ij}(t+1) = w_{ij}(t) + \eta \delta_i^{(h)}(t) x_j(t). \quad (2.35)$$

Concluída a fase de treinamento da rede MLP, é então necessário avaliar o seu desempenho para generalização em dados de um conjunto de vetores de teste, até então desconhecidos pela rede.

2.4 Conclusões

Neste capítulo foi apresentada inicialmente a arquitetura da rede MLP *feedforward* com uma camada oculta e totalmente conectada, destacando os principais parâmetros envolvidos. Além disso, foi introduzida uma notação matricial para representar os dados de entrada, as matrizes de pesos e as respostas dos neurônios ocultos e de saída. Esta notação torna mais simples o entendimento da metodologia proposta por esta dissertação.

Na sequência também foi apresentado o algoritmo de retropropagação dos erros (*error backpropagation*) com gradiente descendente, utilizado no treinamento das redes MLP tratadas nesta dissertação. A abordagem para aprendizagem supervisionada da rede foi

dividida nos sentidos direto e reverso, em que neste último utilizou-se a *Regra Delta Generalizada* para atualização de todos os pesos e limiares da rede.

O próximo capítulo discute inicialmente o papel dos neurônios ocultos e das camadas ocultas nas redes MLP e, em seguida, trata de alguns dos métodos comumente utilizados para seleção de modelos neurais, além listar brevemente alguns estudos recentes nesta área.

3 SELEÇÃO DE MODELOS NEURAIS

Este capítulo discute inicialmente a relevância das camadas ocultas e respectivos neurônios nas redes MLP. Na sequência são tratadas heurísticas e técnicas clássicas aplicadas em seleção de modelos neurais, e por fim são citados alguns estudos recentes nesta área.

3.1 Introdução

Embora as RNAs atualmente possuam teoria bem consolidada na comunidade científica, aliada a uma gama de aplicações em inúmeras áreas, ainda hoje escolher a melhor arquitetura neural para resolver um determinado problema, seja de regressão ou de classificação, é uma tarefa difícil que, por muitas vezes, é realizada por tentativa e erro.

De acordo com Alippi (1999) a determinação de um modelo neural, para um certo conjunto de pares entrada-saída, compreende três fases distintas: a seleção do modelo (na qual se escolhe a complexidade da arquitetura neural), a fase de aprendizagem (para determinar os parâmetros do modelo) e a validação (para avaliar a capacidade de generalização do modelo). A arquitetura que associe a menor complexidade com a melhor habilidade na generalização é então escolhida para resolver o problema, seja ele de classificação de padrões ou de aproximação de funções.

A habilidade de uma RNA em generalizar o conhecimento obtido em dados conhecidos, sobre dados desconhecidos, depende da sintonia entre os dados de treinamento e a complexidade da rede. Uma medida simples da complexidade de uma rede MLP com uma camada oculta envolve o número de neurônios ocultos e o correspondente número de parâmetros ajustáveis, o que para uma rede neural é medido pela quantidade de conexões sinápticas. Cada neurônio oculto adicionado aumenta o número de conexões em uma parcela de $(p + 1) + m$, em que p e m designam o número de unidades de entrada e o número de neurônios na camada de saída, respectivamente (a dimensão adicional contabiliza os

limiares para os neurônios ocultos).

Determinar a arquitetura adequada a um dado problema depende não somente da definição do que venha a ser um bom desempenho, mas também da quantidade e qualidade dos dados disponíveis. Entretanto, a partir de um conhecimento razoável sobre as funções da camada oculta, bem como de seus neurônios, é possível escolher uma arquitetura adequada e alcançar soluções eficientes (XIANG et al., 2005).

Este capítulo inicia com uma discussão sobre a relevância dos neurônios ocultos em uma rede neural multicamadas, destacando uma interpretação geométrica desses neurônios e das camadas ocultas. Na sequência são tratadas algumas heurísticas e técnicas tradicionais utilizadas na determinação do número de neurônios ocultos na rede MLP, finalizando com uma breve citação sobre estudos recentes em seleção de modelos.

3.2 O Papel dos Neurônios Ocultos

A aplicação de uma rede perceptron simples em problemas de classificação de padrões é satisfatória, em termos de capacidade de generalização, apenas nos casos em que as classes dos dados são linearmente separáveis (MINSKY; PAPERT, 1988) ou seja, classes que podem ser separadas por um hiperplano.

Os problemas rotulados como não-linearmente separáveis podem ser resolvidos com as redes multicamadas com no mínimo dois neurônios ocultos, todos com funções de ativação não-lineares. A ativação de cada neurônio oculto (ver Equação (2.5)), na iteração t , é dada por

$$u_i^{(h)}(t) = \sum_{j=1}^p w_{ij}(t)x_j(t) - \theta_i^{(h)}(t) = \sum_{j=0}^p w_{ij}(t)x_j(t), \quad i = 1, \dots, q \quad (3.1)$$

em que $\theta_i^{(h)}(t)$ é o limiar do i -ésimo neurônio oculto, $x_0(t) = -1$ e $w_{i0}(t) = \theta_i^{(h)}(t)$. O termo w_{ij} é a conexão sináptica entre a j -ésima entrada ($x_j(t)$), de dimensionalidade p , e o neurônio oculto i . A saída deste neurônio é obtida com a aplicação da função de ativação não-linear sigmoideal $\varphi_i(\cdot)$ (ver Equação (2.6)), e expressa por

$$y_i^{(h)}(t) = \varphi_i \left[u_i^{(h)}(t) \right] = \varphi_i \left[\sum_{j=0}^p w_{ij}(t)x_j(t) \right]. \quad (3.2)$$

Conforme mencionado no capítulo anterior, os neurônios ocultos realizam uma trans-

formação não-linear nos dados de entrada da rede para o espaço oculto, em que a tarefa de classificação de padrões pode ser realizada com maior facilidade. O processo de aprendizagem em redes neurais para classificação de padrões tem por principal objetivo produzir regiões de decisão de formato arbitrário, que sejam capazes de discriminar classes não-linearmente separáveis (FUNAHASHI, 1998; BISHOP, 2005).

O fato de uma rede MLP poder possuir qualquer número de neurônios ocultos pode levar à falsa impressão de que quanto maior o número desses neurônios, melhor será o desempenho da rede. Entretanto, à medida que se aumenta a complexidade do modelo neural, pelo aumento do número de neurônios ocultos, o desempenho da rede melhora até certo limite, depois tende a deteriorar para os dados do conjunto de teste. Esta questão está ligada à ocorrência dos fenômenos de *underfitting* e *overfitting*, que serão tratados nos próximos parágrafos.

3.2.1 *Underfitting* e *Overfitting*

Na aprendizagem supervisionada utilizada em RNAs para tarefas de classificação de padrões, busca-se extrapolar o conhecimento adquirido com os dados de treinamento para novos exemplos de teste, até então desconhecidos pela rede. Entretanto, há um limite na aproximação obtida no conjunto de treinamento e a generalização da rede para o conjunto de teste.

Se a arquitetura neural tem poucos parâmetros é possível que a rede sequer aprenda a informação contida no conjunto de treinamento, resultando em uma pobre capacidade de generalização. Esse fenômeno é conhecido como *underfitting* (subajustamento) do modelo aos dados. Por outro lado, se a rede tem uma quantidade elevada de parâmetros, pode ocorrer uma aprendizagem excessiva da rede aos dados de treinamento. Isto pode incapacitar a rede de reconhecer novos dados diferentes daqueles utilizados na fase de treinamento. Esse fenômeno é conhecido como *overfitting* (sobreajustamento) do modelo aos dados.

Para um melhor entendimento na identificação de ocorrência de *underfitting* e *overfitting*, discute-se agora duas métricas comumente empregadas em Estatística para avaliar modelos preditivos, o viés (erro indutivo) e a variância. O viés corresponde ao erro cometido na aproximação do modelo aos dados utilizados na fase de treinamento. Já a variância representa a sensibilidade do modelo à escolha do conjunto de treinamento.

Assim, os modelos com excesso de parâmetros que caracterizam o *overfitting* aproxi-

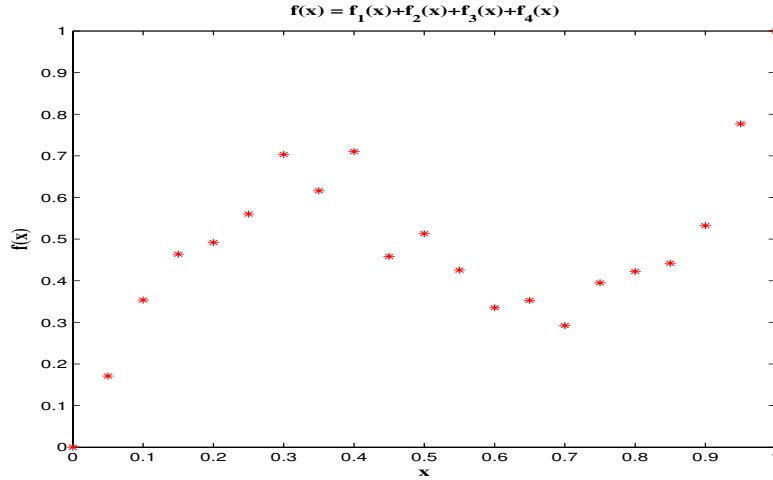


Figura 3.1 Função a ser aproximada no problema de regressão.

mam bem os dados de treinamento, apresentando baixo viés, no entanto são sensíveis à escolha do conjunto de treinamento, apresentando alta variância. Os modelos com poucos parâmetros que caracterizam o *underfitting* não aproximam bem os dados de treinamento, apresentando alto viés, embora sejam consistentes na escolha de diferentes conjuntos de treinamento, apresentando baixa variância.

Para exemplificar a ocorrência de *underfitting* e *overfitting*, será considerada a aplicação de uma rede MLP em um problema de regressão, cuja função a ser aproximada é dada por

$$f(x) = 0,5 + 0,4 \sin(2\pi x) + 0,05x + \varepsilon, \quad (3.3)$$

em que f é uma composição de quatro funções: um nível DC ($f_1(x) = 0,5$), uma senóide ($f_2(x) = 0,4 \sin(2\pi x)$), uma função linear ($f_3(x) = 0,05x$) e $f_4(x) = \varepsilon$ é uma variável (ou número) pseudo-aleatória gaussiana de média zero e desvio-padrão $\sigma_\varepsilon = 0,05$, ou seja

$$\varepsilon \sim N(0, \sigma_\varepsilon^2). \quad (3.4)$$

A Figura 3.1 mostra o gráfico da função f .

Embora o foco desta dissertação esteja voltado para a utilização de redes MLP aplicadas em problemas de classificação de padrões, o problema de regressão tratado agora é um exemplo didático bastante ilustrativo dos fenômenos de *underfitting* e *overfitting*. O pro-

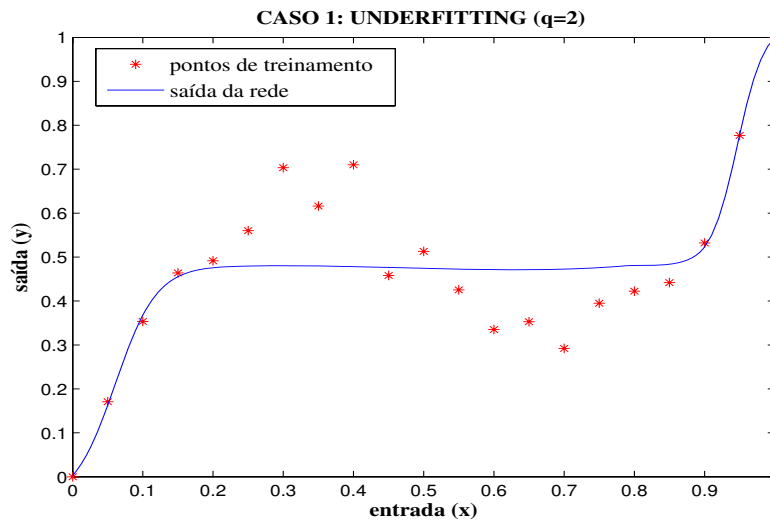


Figura 3.2 Problema de regressão com underfitting.

blema consiste em treinar uma rede MLP com uma entrada e uma saída para aproximar a função f mostrada na Figura 3.1. Para tanto foi treinada uma rede $MLP(1, q, 1)$ com uma entrada e uma saída, e q neurônios na única camada oculta da rede. Foi utilizada a função sigmóide logística para a ativação dos neurônios ocultos, e função linear para os neurônios de saída. Além disso, todos os dados disponíveis foram utilizados no treinamento da rede, e os parâmetros de taxa de aprendizagem, fator de momento e número de épocas adotados foram $\eta = 0,1$, $\alpha = 0,75$ e $N_e = 5.000$, respectivamente.

Caso 1: Considere inicialmente o treinamento de uma rede MLP com $q = 2$ neurônios ocultos, cuja saída é mostrada na Figura 3.2, juntamente com a função f a ser aproximada. Analisando as curvas na Figura 3.2 é possível observar dois aspectos relevantes. Primeiro é que a saída da rede, representada na curva em azul, apresentou alto viés na aproximação da maioria dos pontos da função f , exceção feita a alguns poucos pontos, situados nas extremidades da curva em f . Segundo é que a saída da rede não oscilou em torno dos pontos de aproximação, mantendo uma certa suavidade na curva, o que caracteriza uma baixa variância. Portanto, a saída da rede mostrada na Figura 3.2 é um caso de *underfitting*, apresentando ALTO viés e BAIXA variância.

Caso 2: Considere agora uma rede MLP treinada com $q = 50$ neurônios ocultos, cuja saída é mostrada na Figura 3.3. Percebe-se neste caso que a saída da rede passou por todos os pontos da função f , apresentando portanto viés nulo. Entretanto, a variância da saída da rede em torno da função f é considerável, o que significa dizer que se novos pontos fossem inseridos em f dificilmente se teria uma boa aproximação por parte da

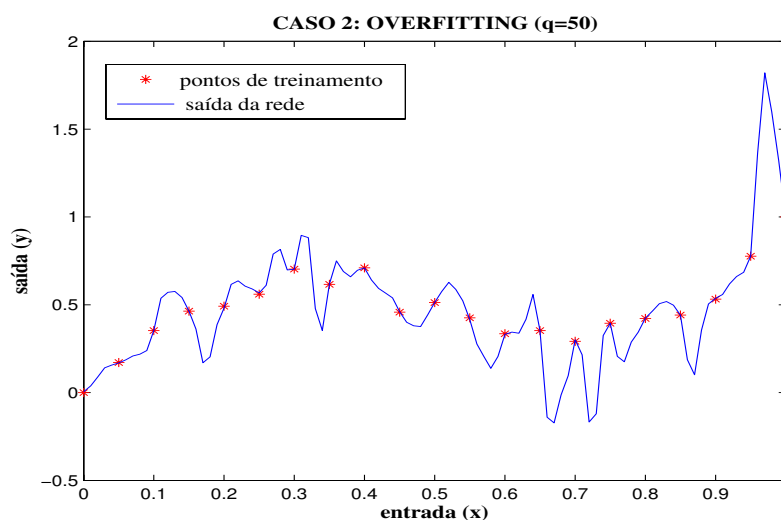


Figura 3.3 Problema de regressão com overfitting.

resposta da rede. Portanto, pode-se dizer que este é um caso de *overfitting*, caracterizado por apresentar BAIXO viés e ALTA variância.

Caso 3: Por fim, considere o treinamento de uma rede MLP com $q = 4$ neurônios ocultos, cuja saída juntamente com a função f são mostradas na Figura 3.4. Este caso apresentou viés menor que o obtido no caso 1 com *underfitting*, ao mesmo tempo em que apresentou uma menor variância que a obtida no caso 2 com *overfitting*. Percebe-se então que a saída da rede na Figura 3.4 é uma boa aproximação da função f , uma vez que manteve o equilíbrio entre viés e variância, indicativo de um ajuste de curva mais adequado (*goodfitting*). Portanto, deve-se buscar situações de *goodfitting* (bom ajuste) em vez dos casos extremos de *underfitting* ou *overfitting*.

3.2.2 Interpretação Geométrica

O entendimento do papel dos neurônios ocultos da rede MLP pode ser melhor aprendido se a estes neurônios for dada uma interpretação geométrica. Dessa forma, Park et al. (1991) afirmam que cada neurônio comporta-se como um separador linear, e os coeficientes do hiperplano que define esse separador são determinados pelos pesos sinápticos e limiares daquele neurônio.

Segundo Lippmann (1989), classificadores hiperplanares geram regiões de decisão complexas, ou seja, de formas arbitrárias, usando neurônios ocultos que formam limites de decisão hiperplanares no espaço varrido pelos dados de entrada. Nessa análise, os pesos adaptativos entre as unidades de entrada e os neurônios ocultos formam hiperplanos de

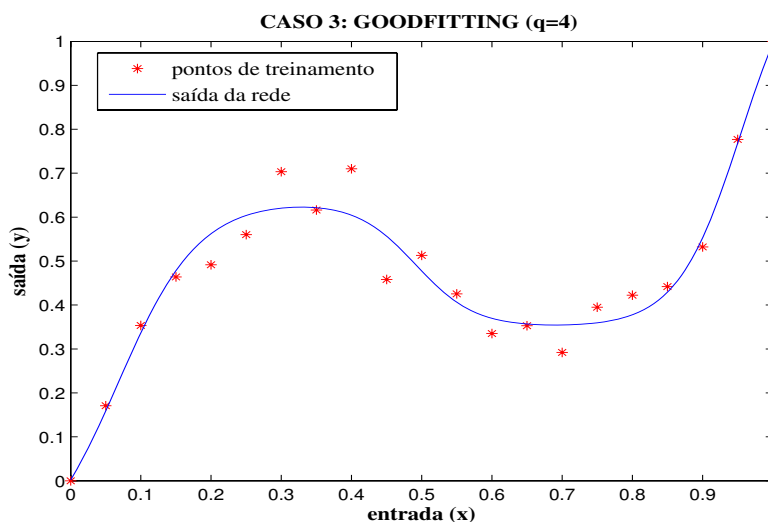


Figura 3.4 Problema de regressão com goodfitting.

decisão no espaço de entrada, e cada região de decisão é coberta pelos segmentos desses diversos hiperplanos (VAUGHN, 1999; LIPPMANN, 1989).

De acordo com Xiang et al. (2005) a dificuldade em estimar o número de hiperplanos necessários para construir a forma geométrica básica de uma função-alvo aumenta com a complexidade dos dados de entrada. Em particular, quando a dimensionalidade dos dados é maior que dois, o que acontece na grande maioria dos problemas de classificação de padrões, torna-se praticamente impossível determinar a forma geométrica básica da função-alvo.

Gori & Scarselli (1998) provaram que, independente da função de ativação dos neurônios ocultos, as arquiteturas com menos neurônios na primeira camada oculta em relação ao número de entradas da rede, não podem produzir superfícies de separação fechadas. Se o número de neurônios ocultos for maior que o número de entradas, a MLP poderá criar superfícies de separação abertas ou fechadas. Além disso, nenhuma escolha da função sigmoideal nos neurônios transforma superfícies de separação abertas em superfícies fechadas.

Draghici (2002) argumenta que se as conexões sinápticas de uma rede neural são números reais, os hiperplanos implementados com esses pesos podem ser alocados em quaisquer posições pelo algoritmo de treinamento. Se esse algoritmo for apropriado, a solução dos pesos da rede será encontrada para que todos os padrões de classes opostas sejam separados. Por outro lado, se os pesos forem restritos a poucos valores inteiros, as posições disponíveis para os hiperplanos durante o treinamento são muito escassas.

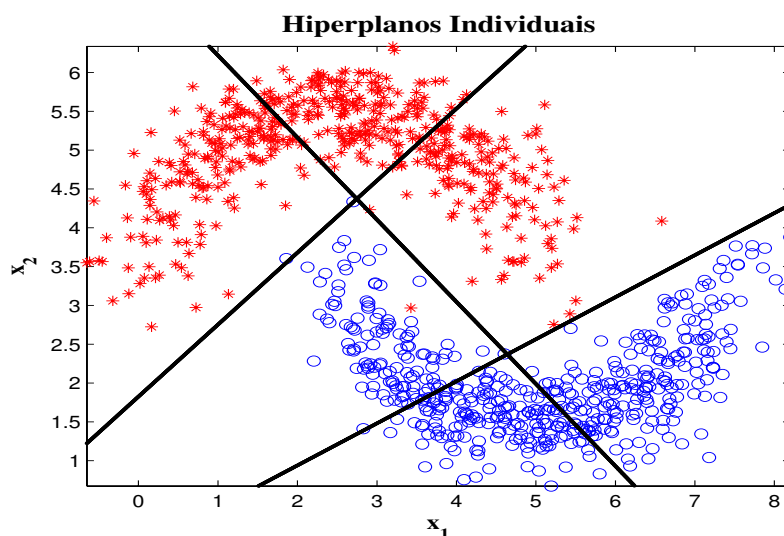


Figura 3.5 Hiperplanos individuais da rede MLP com 3 neurônios ocultos, para o conjunto banana.

Para um melhor entendimento e visualização do papel dos neurônios ocultos, considere por exemplo a Figura 3.5, em que é mostrada uma distribuição bidimensional de dados, com 1001 amostras divididas em duas classes representadas em azul e vermelho. Este conjunto de dados é sintético e possui distribuição entre as classes com aparência de duas bananas. As retas mostradas na Figura 3.5 representam os hiperplanos, obtidos após a fase de treinamento de uma rede MLP com uma camada oculta, formada por três neurônios. A região de decisão na saída da rede, conforme ilustrada na Figura 3.6, é obtida a partir da resposta dos neurônios de saída, como uma combinação dos hiperplanos dos neurônios da camada oculta.

3.2.3 Número de Camadas Ocultas

A princípio não existem restrições acerca do número de camadas ocultas em uma rede MLP, embora geralmente sejam utilizadas redes com uma ou duas camadas de neurônios ocultos. A seguir são apresentados alguns estudos sobre essa questão, embora o foco desta dissertação esteja voltado para a rede MLP com uma única camada oculta.

De acordo com Jain et al. (1996) as redes MLP com uma ou duas camadas ocultas podem apresentar superfícies de decisão arbitrárias, cujas complexidades são limitadas pelo número de neurônios ocultos. A Figura 3.7 resume a interpretação geométrica feita por Jain et al. (1996) para a rede MLP com uma e duas camadas ocultas, considerando o espaço de entrada bidimensional.

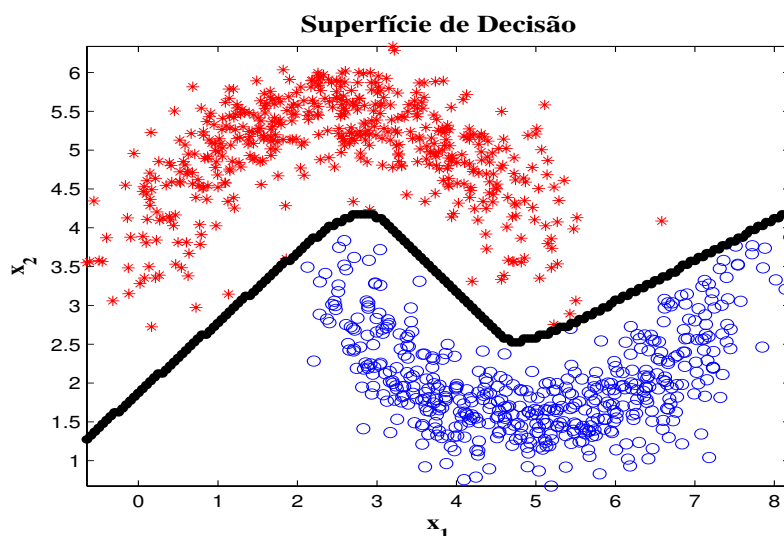


Figura 3.6 Superfície de decisão da rede MLP com 3 neurônios ocultos, para o conjunto banana.

A Figura 3.7 mostra o problema de classificação da porta lógica ou-exclusivo de duas entradas, o qual é classicamente caracterizado como não-linearmente separável (HAYKIN, 2001). Neste caso, as redes MLP com uma ou duas camadas ocultas são igualmente eficazes na resolução do problema, apresentando no entanto diferentes regiões de decisão. De uma forma geral, Jain et al. (1996) argumentam que as redes MLP com uma camada oculta são capazes de produzir regiões de decisão convexas¹, e as redes com duas camadas de neurônios ocultos são capazes de produzir regiões de decisão não-convexas (ver Figura 3.7).

Seguindo esta linha de pensamento Stinchcombe & White (1989) afirmam ainda que uma rede neural com uma ou mais camadas ocultas pode formar superfícies de decisão arbitrárias se um número suficientemente alto de neurônios são usados nas camadas ocultas. Afirmam também que não haveria diferença nas capacidades de separação de ambas as arquiteturas, muito embora elas venham a diferir em outros aspectos, tais como a performance de separação para um número finito de neurônios ocultos ou a generalização realizada.

Sartori & Antsaklis (1991) e Huang & Huang (1991) provaram que uma rede neural *feedforward* com uma camada oculta possuindo $N - 1$ neurônios pode estabelecer quaisquer relações entrada-alvo, entre as N amostras do conjunto de treinamento e suas respectivas saídas-alvos, de forma exata. Na mesma linha de raciocínio Tamura & Tateishi (1997)

¹Uma região poligonal é dita convexa quando não apresenta reentrâncias no corpo da mesma. Isto significa que todo segmento de reta cujas extremidades estão nesta região estará totalmente contido na região poligonal. A negação desta afirmação é aplicada para o caso de uma região poligonal não-convexa.

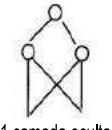
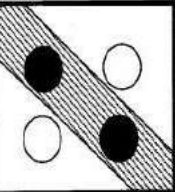
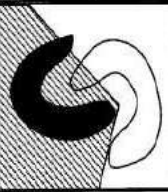
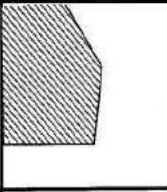
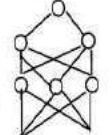
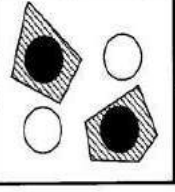
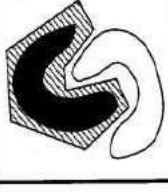
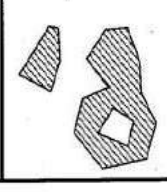
Rede MLP	Descrição das regiões de decisão	Problema do OU-exclusivo	Classes com regiões não-convexas	Formas das regiões de decisão
 1 camada oculta	Arbitrária (complexidade limitada pelo número de neurônios ocultos)			
 2 camadas ocultas	Arbitrária (complexidade limitada pelo número de neurônios ocultos)			

Figura 3.7 Interpretação geométrica do papel dos neurônios ocultos em redes MLP, adaptada da referência Jain et al. (1996).

construíram uma rede com duas camadas ocultas com $(N/2) + 3$ neurônios ocultos e mostraram que tal rede pode chegar às mesmas N relações entrada-alvo com um erro arbitrariamente menor que o encontrado por Sartori & Antsaklis (1991) e Huang & Huang (1991).

Villiers & Barnard (1992) não encontraram diferenças significativas na performance de redes neurais com uma ou duas camadas ocultas. No entanto, chegaram a conclusão de que as redes com duas camadas ocultas são mais tendenciosas a chegar em pontos de mínimos locais durante a etapa de treinamento, utilizando os algoritmos *backpropagation* e gradiente conjugado.

Segundo Chester (1990), o problema de uma rede neural *feedforward* com uma camada oculta é que os neurônios interagem uns com os outros de forma global, o que torna difícil melhorar uma aproximação em um ponto sem piorar em um outro. Em redes com duas camadas ocultas, os efeitos dos neurônios podem ser isolados e as aproximações em diferentes regiões podem ser ajustadas independentemente umas das outras.

Xiang et al. (2005) interpretaram a rede MLP de duas camadas ocultas como uma combinação linear de múltiplas redes MLP de uma camada oculta, as quais compartilham os mesmos neurônios ocultos mas com diferentes neurônios de saída. O número de neurônios na segunda camada oculta é então o número destas redes de única camada oculta que constroem subfunções que serão combinadas na aproximação da função desejada.

Nesta mesma linha de pensamento, Principe et al. (2000) associam o número de neu-

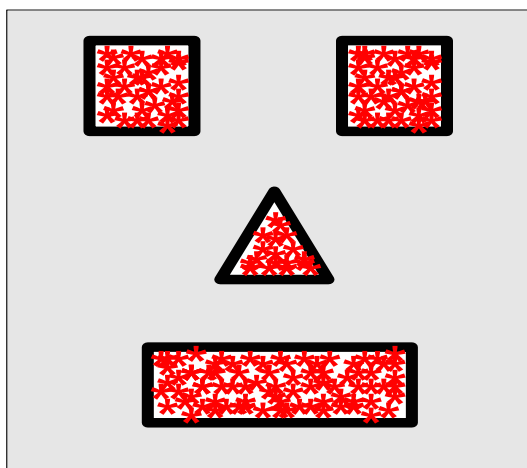


Figura 3.8 Conjunto de dados máscara, adaptado da referência Principe et al. (2000).

rônios na primeira camada oculta da rede MLP com o número de hiperplanos necessários no espaço dos dados de entrada. Da mesma forma, o número de neurônios na segunda camada oculta é associado ao número de regiões no espaço de entrada necessárias para a aproximação em questão. O neurônio de saída simplesmente combina estas regiões para produzir o mapeamento entrada-saída desejado. Para ilustrar esta ideia considere a distribuição bidimensional na Figura 3.8, que pode ser pensada como um problema de classificação em que uma das classes é espalhada no espaço de entrada e representada pelos pontos em vermelho, e a outra classe é a região representada pela cor cinza.

O objetivo é treinar uma rede MLP com duas camadas ocultas para estabelecer a classificação correta dos dados na Figura 3.8, em que se percebe claramente a existência de quatro regiões distintas no espaço de entrada. Estas regiões tem aparência que assemelha-se a uma máscara, em que os olhos podem ser aproximados por dois quadrados, o nariz por um triângulo e a boca por um retângulo.

A figura 3.9 ilustra a topologia da rede MLP utilizada neste problema. São mostrados treze neurônios na primeira camada oculta, em virtude dos treze hiperplanos necessários, sendo três para o nariz, quatro para um dos olhos, dois para o outro (aproveita dois do primeiro olho) e quatro para a boca. De acordo com Principe et al. (2000), são necessários ainda quatro neurônios na segunda camada oculta da rede, um para cada região mostrada (olhos, nariz e boca). O neurônio de saída simplesmente realiza uma combinação das respostas da camada anterior.

Este exemplo é bastante ilustrativo para entender o papel dos neurônios em cada camada oculta, no entanto é meramente didático. Primeiro porque normalmente a estrutura

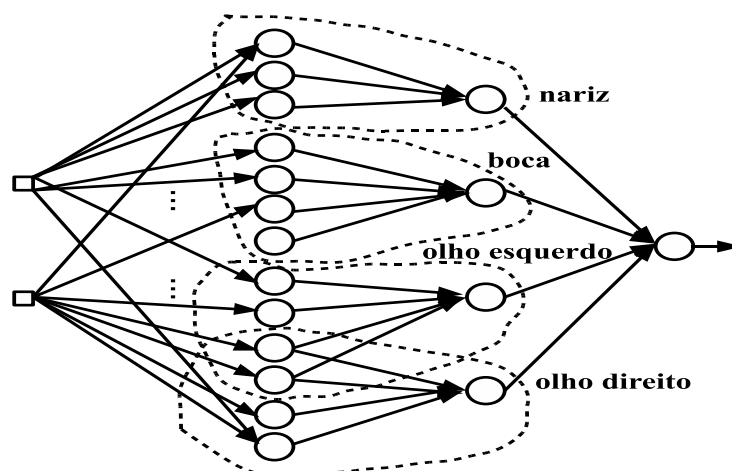


Figura 3.9 Rede MLP com duas camadas ocultas, adaptada da referência Principe et al. (2000).

dos dados é desconhecida, segundo porque raramente se tem dados de dimensionalidade dois. Estes fatores contribuem para inviabilizar o conhecimento prévio do quantitativo e da forma dos hiperplanos e regiões de decisão.

Diante do que foi exposto nos parágrafos anteriores, percebe-se opiniões por vezes diferentes sobre a melhor arquitetura a ser escolhida no que diz respeito ao número de camadas ocultas e ao número de neurônios por camada. Entretanto a opção por utilizar a rede MLP com uma única camada oculta é atrativa, uma vez que alia estrutura simplificada e desempenho satisfatório para aplicações em classificação de padrões. Além disso, a metodologia proposta nesta dissertação pode, a princípio, ser aplicada em redes MLP com duas ou mais camadas ocultas.

As próximas seções apresentam algumas técnicas comumente empregadas na determinação do número de neurônios ocultos em RNAs.

3.3 Regras Heurísticas

Para o usuário que utiliza uma rede MLP como uma caixa preta, a tarefa de encontrar o número satisfatório de neurônios na camada oculta normalmente é feita através da tentativa e erro, o que normalmente demanda muito tempo e esforço computacional. Nos parágrafos seguintes serão apresentadas algumas regras simples que servem como ponto de partida para esta estimativa.

Alguns softwares comerciais são bastante empregados na implementação de redes neu-

rais artificiais. Um deles é o *Neuralware*², que sugere uma arquitetura de maneira tal que existam cinco amostras do conjunto de treinamento para cada conexão sináptica da rede. Desta forma, o número de neurônios ocultos é dado por

$$q = \frac{N}{5(m+p)}, \quad (3.5)$$

em que N é o número de vetores no conjunto de treinamento, m é a dimensão da saída da rede e p é a dimensão dos dados de entrada.

Outro software bastante utilizado na comunidade científica é o *Weka* (WITTEN; FRANK, 2005), que trata-se de uma coleção de algoritmos de aprendizagem de máquinas para resolver problemas do mundo real em mineração de dados. O *Weka* estima como padrão

$$q = \frac{p+m}{2}, \quad (3.6)$$

para o número de neurônios ocultos na rede MLP com uma única camada oculta, baseada na regra do valor médio sugerida por Blum (1992) e Masters (1993). Além desta regra, Blum (1992) e Masters (1993) ainda estabelecem algumas outros critérios para estimar o número de neurônios ocultos nas redes com uma camada oculta. São elas

$$q = 2p + 1 \quad (\text{regra de Kolmogorov}) \quad (3.7)$$

$$2\sqrt{p} + m \leq q \leq 2p + 1 \quad (\text{regra de Fletcher-Gloss}) \quad (3.8)$$

$$q = \sqrt{mp} \quad (\text{regra da raiz quadrada}) \quad (3.9)$$

As equações acima baseiam-se na busca de uma arquitetura piramidal para a rede MLP, sem levar em conta portanto aspectos como a quantidade de amostras disponíveis para o treinamento, a presença de ruído nos dados e a complexidade do mapeamento entrada-saída.

Os parâmetros funcionais da rede MLP tais como a dimensão e o número de amostras dos dados de entrada, a escala dos pesos e a maior e menor distância entre classes adjacentes podem influenciar no número de neurônios ocultos (DRAGHICI, 2002). Por outro lado, Daqi & Yan (2005) consideram que o número de neurônios ocultos está apenas rela-

²<http://www.neuralware.com>

cionado ao número de classes, assim como as regiões e formas de distribuição das amostras no espaço de entrada, sem relação com o número de amostras ou com a dimensionalidade do espaço de entrada. Daqi & Yan (2005) estabelecem a desigualdade

$$2^q + 1 \geq C \quad \text{ou} \quad q \geq \log_2(C - 1), \quad (3.10)$$

em que C representa o número de classes em problemas de classificação. No entanto, a determinação da distribuição das amostras em um espaço de alta dimensão torna-se bastante difícil. Desta forma os autores apresentam ainda a seguinte expressão empírica para selecionar o número inicial de neurônios ocultos

$$q_0 = \lfloor 2 \log_2(p + C - 1) \rfloor \geq 2, \quad (3.11)$$

em que $\lfloor \cdot \rfloor$ é a função que retorna o maior inteiro menor ou igual ao seu argumento.

Em problemas de seleção de modelos neurais, a quantidade e qualidade das amostras de treinamento também é um fator importante. Baum & Haussler (1989) provaram que, em uma rede neural com uma camada oculta e usada para classificação binária, são necessários N exemplos para o conjunto de treinamento, expresso pela desigualdade

$$N \geq \frac{32Z}{\epsilon} \ln \left(\frac{32Z}{\epsilon} \right), \quad (3.12)$$

em que \ln é o logaritmo natural, ϵ é um parâmetro de exatidão e Z é o número total de pesos da rede, dado por

$$Z = (p + 1) \times q + (q + 1) \times m. \quad (3.13)$$

Uma vez satisfeita a Equação (3.12) e a condição de que o erro relativo cometido para os exemplos do conjunto de treinamento seja menor que $\epsilon/2$, a rede apresentará via de regra um desempenho satisfatório na generalização (BAUM; HAUSSLER, 1989).

Haykin (2001) simplifica a Equação (3.12) levando em conta considerações práticas e assume que para uma boa generalização da rede, o tamanho do conjunto de treinamento deve satisfazer a seguinte condição

$$N = O \left(\frac{Z}{\epsilon} \right), \quad (3.14)$$

em que ϵ é a fração dos erros cometidos com os dados do conjunto de teste e $O(\cdot)$ denota

a ordem da quantidade entre parênteses. Para um erro de 10%, por exemplo, o número necessário de exemplos de treinamento deve ser da ordem de 10 vezes maior que o número de parâmetros livres da rede.

Uma contribuição adicional sobre a capacidade de aproximação das redes neurais *feedforward* é atribuída a Barron (1994), salientando que para algumas classes de funções suaves, o parâmetro definido como o primeiro momento absoluto da distribuição das magnitudes de Fourier (C_f) é determinante na estimação dos neurônios ocultos,

$$q \sim C_f \left(\frac{N}{p \log N} \right)^{\frac{1}{2}}, \quad (3.15)$$

na busca pela otimização do erro quadrático médio. Neste trabalho o valor de C_f foi simplificado e adotado como unitário, uma vez que sua determinação não é um processo simples (BARRON, 1993, 1994).

Looney (1996) considerou, para a arquitetura de uma rede neural *feedforward*, a utilização da codificação *1-out-of-Q* para os neurônios de saída ($m = C$) ou, para um valor grande de C , o número de neurônios de saída pode ser encontrado como $m = \log_2 C$. No caso em que $m < \log_2 C$ o processo de aprendizagem torna-se mais difícil, mas ainda possível. Uma vez que o número N de amostras de treinamento é um valor pré-estabelecido, o número q de neurônios ocultos torna-se o principal parâmetro na busca por uma arquitetura satisfatória. Assim, Looney (1996) estabeleceu três formas para estimar o número de neurônios ocultos:

- usar o valor limite inferior de neurônios dado por $q = \log_2 C$ na camada oculta, e adicionar mais neurônios até que a rede aprenda bem;
- começar com um número relativamente grande de neurônios na única camada oculta da rede, maior do que teoricamente seria necessário ($q = [(c - 1)/2 + (\log_2 c)/2]$), para que o número de subclasses³ c atenda à condição $c \geq 2C$. Em seguida, deve-se treinar a rede e então podar os neurônios ou pesos que não contribuam significativamente para o aprendizado (e retrainar após a poda);
- iniciar a rede com duas camadas ocultas de tamanhos q_1 e q_2 , em que $q_1 > q_2$ ($q_1 = 2\sqrt{C}$ e $q_2 = \sqrt{C}$), adicionar neurônios à primeira camada escondida conforme necessário, e finalmente podar a rede e retrainá-la.

³As classes de uma população finita de vetores de características podem ser decompostas em subclasses convexas e linearmente separáveis. Estas subclasses podem ser separadas por um número apropriado de hiperplanos q , que pode variar de um limite inferior $q = \log_2 c$ ($c = 2^q$) a um limite superior $q = c - 1$.

De uma forma geral, algumas das regras e técnicas apresentadas nos parágrafos anteriores possuem formulação teórica em suas fundamentações, outras são estabelecidas com base em experimentações exaustivas, e ainda há aquelas que apenas sugerem uma idéia para a tentativa inicial, ou seja, uma espécie de "chute".

3.4 Técnicas de Seleção de Modelos

Existem vários métodos disponíveis na literatura que tratam de seleção de modelos neurais. Há técnicas baseadas em inferências estatísticas (MURATA et al., 1994; VAPNIK, 1998), em que se busca a partir de uma família de redes neurais o conjunto de parâmetros ótimo para aproximar a distribuição condicional do sistema (MURATA et al., 1994). Outras baseiam-se em algoritmos construtivos (FAHLMAN; LEBIERE, 1990; MOODY, 1994), em que o treinamento da rede neural inicia com apenas um neurônio oculto e são adicionados outros neurônios à medida em que a rede torna-se incapaz de aprender novos padrões.

Algumas metodologias fundamentam-se no treinamento de redes com um número relativamente grande de neurônios ocultos com o intuito de, em uma fase posterior, aplicar técnicas de poda para a eliminação de conexões sinápticas e/ou neurônios (HASSIBI et al., 1992; LECUN et al., 1990; LEVIN et al., 1994). Existem ainda as metodologias de poda baseadas em parâmetros de regularização de complexidade, tais como decaimento de pesos (RUMELHART et al., 1989; KROGH; HERTZ, 1995), eliminação de pesos (WEIGEND; RUMELHART, 1991) e suavização aproximativa (MOODY; ROGNVALDSSON, 1997).

Outras técnicas como a validação cruzada (*hold-out*) (STONE, 1974) e a validação cruzada múltipla são bastante utilizadas também para a seleção de modelos neurais. Nestas, o objetivo é escolher dentre um conjunto de topologias de redes candidatas, a que apresente melhores resultados de acordo com um critério estabelecido.

Esta dissertação restringe-se a discutir algumas das abordagens mais comuns em seleção de modelos neurais, descritas em livros como Haykin (2001), Principe et al. (2000) e Bishop (2005). Portanto, as seções seguintes descrevem algumas dessas técnicas usuais, as quais inclusive serão utilizadas para comparativo com a metodologia proposta neste trabalho.

3.4.1 Validação Cruzada

O processo de aprendizagem por retropropagação na rede MLP estabelece um mapeamento de entrada-saída de acordo com as amostras do conjunto de treinamento e seus respectivos rótulos. Esse mapeamento é alcançado às custas da parametrização dos pesos sinápticos e limiares da rede, na busca pela minimização da função erro quadrático no conjunto de treinamento.

De acordo com Kearns (1996) a seleção de um modelo neural envolve dois procedimentos. O primeiro consiste em escolher o número apropriado de parâmetros da rede, e o segundo em ajustar esses parâmetros. Os dados do conjunto de treinamento são utilizados nos dois passos deste processo. De uma forma geral, o ajuste dos parâmetros é determinado pelo algoritmo de aprendizagem e o problema da seleção resume-se à escolha da arquitetura (número de pesos sinápticos).

Neste contexto, o critério de validação cruzada (*cross-validation*) destaca-se como uma ferramenta bastante utilizada na escolha e avaliação em aplicações de predição estatística (STONE, 1974). A técnica consiste na divisão controlada ou não-controlada dos dados amostrais em dois subconjuntos disjuntos, sendo um para a predição do modelo e o outro para avaliar o seu desempenho.

No caso da rede MLP, inicialmente os dados disponíveis são divididos aleatoriamente em um conjunto de treinamento e um conjunto de teste. O conjunto de treinamento é então dividido em dois outros conjuntos disjuntos. Um é utilizado para selecionar o modelo e outro para testar ou validar o modelo. Tratam-se respectivamente do conjunto de estimação e do conjunto de validação. O procedimento para estimação/validação prossegue da seguinte forma (HAYKIN, 2001):

- Após um determinado número de épocas de treinamento, os pesos e limiares permanecem fixos e a rede opera no sentido direto. O erro quadrático médio é então calculado em todo o conjunto de validação e portanto é chamado de erro de validação;
- Após a conclusão da fase da validação, o treinamento é reiniciado para uma nova quantidade de épocas (normalmente fixa), e o processo volta a se repetir.

Entretanto, este procedimento pode ocasionar em *overfitting* para as amostras do conjunto de validação. Assim, o desempenho da rede selecionada deve ser confirmado nos dados do conjunto de teste, que é diferente dos conjuntos de treinamento e validação.

Surge agora uma questão que deve ser respondida em uma etapa de pré-aplicação da validação cruzada. Trata-se da proporção dos dados de treinamento para estimação e validação da rede. Considera-se que das N amostras do conjunto de dados de treinamento, $(1 - b)N$ são designadas para o treinamento e as restantes bN para a validação. Em seu trabalho, Kearns (1996) desenvolveu um tratamento analítico aliado a diversas simulações computacionais, observando que

- Quando a complexidade da função alvo desejada é pequena em relação ao tamanho N do conjunto de treinamento, o erro de predição é insensível ao valor de b . Em outras palavras, o desempenho da validação cruzada é relativamente independente da escolha de b ;
- Quando a complexidade da função alvo torna-se de maior relevância em relação a N , a escolha de b tem um maior efeito no desempenho da validação cruzada;
- Existe um valor fixo de b que fornece resultados dito satisfatórios para um grande número de complexidades de funções.

Diante disso, Kearns (1996) sugeriu a escolha de $b = 0,2$. Significa dizer que 80% dos dados disponíveis para o treinamento são utilizados para a fase de estimação dos parâmetros, enquanto os 20% restantes servem para a validação.

Por fim, de acordo com o critério de validação cruzada, selecionar a melhor arquitetura neural implica agora em escolher aquela que forneça o menor erro de validação. A técnica de validação cruzada descrita até aqui é também conhecida como método *hold-out* (HAYKIN, 2001). As duas próximas seções tratam de duas variações da validação cruzada que também podem ser empregadas em seleção de modelos neurais, a validação cruzada com parada prematura e a validação cruzada múltipla.

3.4.1.1 Validação Cruzada com Parada Prematura

Durante a fase de treinamento da rede MLP a função custo (erro quadrático médio) decresce com o aumento do número de épocas, sendo minimizada em direção a um mínimo local ou a um mínimo global. Inicialmente o decréscimo é de forma rápida e depois segue lentamente até um ponto de mínimo na superfície de erro.

O fato de o erro quadrático diminuir com o número de épocas ao longo do treinamento pode conduzir a uma falsa ideia de que quanto maior o número de épocas, melhor será

a capacidade de generalização da rede. No entanto, isso pode ocasionar na ocorrência de *overfitting*. Diante disso, a validação cruzada pode ser usada para detectar o início do excesso de treinamento e então interrompê-lo, evitando o *overfitting*.

Neste caso, o conjunto de validação é utilizado para dar uma boa indicação do melhor momento para finalizar o treinamento. Normalmente o aprendizado alcançado pela rede não funciona tão bem para as amostras de validação. Enquanto o erro no treinamento decresce monotonamente para um número crescente de épocas, o erro de validação decresce monotonamente para um mínimo e então começa a crescer durante a evolução do treinamento (HAYKIN, 2001). A Figura 3.10 mostra o comportamento dos erros quadráticos médios de treinamento e validação em função do número de épocas de treinamento, no gráfico chamado de curva de aprendizagem, para um conjunto de dados do mundo real denominado *ionosphere*⁴.

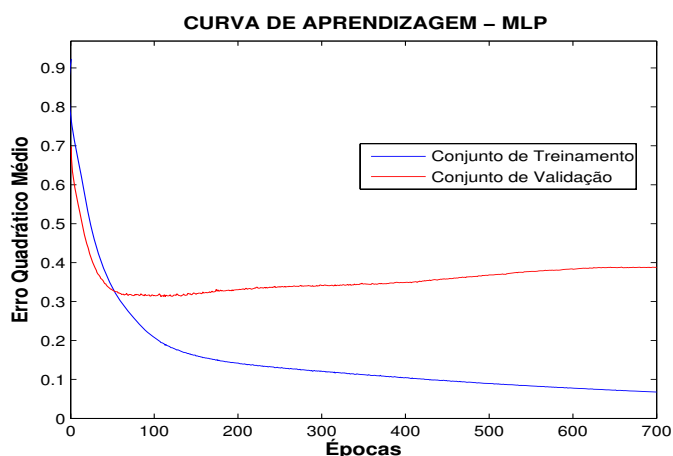


Figura 3.10 Curva de aprendizagem para o conjunto ionosphere.

A técnica de validação cruzada com parada prematura sugere então a interrupção do treinamento tão logo que o erro determinado para as amostras de validação seja maior que da última vez em que ele foi verificado. Esta ideia sugere que o aprendizado da rede após esse ponto trata-se apenas do ruído presente nas amostras de treinamento (HAYKIN, 2001).

Embora existam referências mais antigas sobre o método, merece destaque a análise estatística da parada prematura feita por Amari et al. (1996) para aplicações utilizando a rede MLP com uma camada oculta. Após numerosas simulações computacionais ele identificou duas tendências no comportamento da rede, de acordo com o número N de amostras de treinamento e o número total Z de parâmetros da rede:

⁴<http://www.ics.uci.edu/~mlearn/MLRepository.html>

- Quando Z é grande, a melhor estratégia é usar apenas $1/\sqrt{2Z}$ amostras para o conjunto de validação, enquanto a maioria restante vai compor o conjunto de treinamento;
- Quando $N > 30Z$, teoria e simulações mostraram que a validação cruzada não é necessária, devido ao erro de generalização tornar-se pior com a utilização das amostras de validação para interromper o treinamento;
- Para faixas intermediárias, em que $N < 30Z$, certamente ocorrerá excesso de ajuste e, portanto, a validação cruzada com parada prematura melhora significativamente a capacidade de generalização da rede.

Amari et al. (1996) determinaram o valor do parâmetro ótimo (b_{otimo}) para a partição dos dados de treinamento entre estimação e validação, o qual é expresso por

$$b_{otimo} = 1 - \frac{\sqrt{2Z - 1} - 1}{2(Z - 1)} \simeq 1 - \frac{1}{\sqrt{2Z}}, \quad (3.16)$$

em que a aproximação acima é válida para grandes valores de Z .

Embora a validação cruzada com parada prematura seja uma heurística que alia simplicidade e relativa eficiência, muitos dados reais possuem curvas de aprendizagem bem diferentes das vistas na Figura 3.10. Baldi & Chauvin (1991) mostraram que redes lineares com n entradas e n saídas podem possuir acima de n mínimos. Para o caso da rede MLP a situação pode ser ainda pior. Diante disso, Prechelt (1998) estabeleceu diferentes critérios de parada com base em simulações exaustivas com dados reais para melhorar a generalização da rede, muito embora as custas de um aumento no tempo de treinamento.

3.4.1.2 Validação Cruzada Múltipla

Uma variação da validação cruzada bastante utilizada é a validação cruzada múltipla (*k-fold cross-validation*). Consiste em dividir os N exemplos disponíveis do conjunto de treinamento em k ($k > 1$) subconjuntos (HAYKIN, 2001). Admite-se normalmente que N é divisível por k , e o treinamento é então realizado com $k - 1$ subconjuntos e a validação com aquele que restou. Este procedimento é repetido por k vezes, sempre validando o modelo com um subconjunto diferente. Ao final, o desempenho é avaliado pela média dos erros quadráticos de validação, obtidos nas k diferentes rodadas.

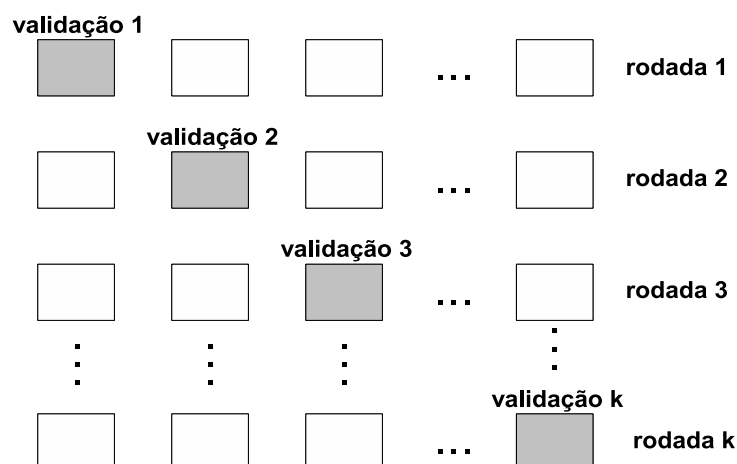


Figura 3.11 Ilustração esquemática da validação cruzada múltipla.

A representação esquemática da validação cruzada pode ser vista na Figura 3.11. A principal vantagem deste método é que todos os dados são utilizados tanto no treinamento quanto na determinação do erro de validação, ao passo em que quando se aumenta o valor de k tem-se uma grande proporção de dados na fase de treinamento.

Alguns trabalhos utilizaram a validação cruzada múltipla para seleção de arquiteturas neurais. Liu (2006), por exemplo, utilizou *10-fold cross-validation* e *12-fold cross-validation* em dados do mundo real (*australian credit card* e *diabetes*, respectivamente) para analisar a estabilidade de um conjunto de redes neurais artificiais. Setiono (2001) utilizou um algoritmo construtivo baseado em *10-fold cross-validation* para seleção de modelos em redes MLP com uma camada oculta, e aplicou seu método em simulações com trinta e dois conjuntos de dados.

Embora a validação cruzada múltipla forneça uma boa estimativa sobre a generalização das redes neurais, há uma nítida desvantagem em sua aplicação. Trata-se do fato de requerer uma quantidade significativa de tempo de processamento, uma vez que cada rodada para generalização da rede envolve k rodadas de treinamento.

As técnicas descritas nos parágrafos seguintes dedicam-se a seleção de modelos com base na poda de conexões sinápticas consideradas de menor relevância, de acordo com critérios de sensibilidade.

3.4.2 Optimal Brain Surgeon (OBS)

OBS é uma técnica desenvolvida por Hassibi et al. (1992) com o objetivo de, a partir do treinamento de uma rede neural razoavelmente grande, eliminar os pesos sinápticos

considerados irrelevantes. Por consequência é possível estabelecer uma arquitetura de menor complexidade, melhorando a capacidade de generalização.

Inicialmente o treinamento da rede é realizado para que se atinja ao menos um mínimo local na função de custo, considerada neste caso como o erro quadrático médio (EQM). Le-Cun et al. (1990) usou expansão em série de Taylor para aproximar o EQM, como pode ser visto na expressão a seguir

$$J(\bar{\theta} + \Delta\theta) = J(\bar{\theta}) + g^T(\bar{\theta})\Delta\theta + \frac{1}{2}\Delta\theta^T \mathbf{H}(\bar{\theta})\Delta\theta + O(\|\Delta\theta\|^3), \quad (3.17)$$

em que $(\cdot)^T$ representa a operação de transposição e $\Delta\theta$ é a perturbação no vetor de pesos, em torno do ponto de operação $(\bar{\theta})$, devido a retirada de um peso específico. $g^T(\bar{\theta})$ e $\mathbf{H}(\bar{\theta})$ são respectivamente o vetor gradiente e a matriz Hessiana ($\mathbf{H}(\bar{\theta}) = d^2 J(\bar{\theta})/d\theta^2$) calculados no ponto de operação $(\bar{\theta})$. Esta matriz armazena todas as derivadas de segunda ordem da função custo. Maiores informações sobre o cálculo recursivo da matriz Hessiana podem ser encontradas em Hassibi et al. (1992), Medeiros (2008) e Haykin (2001).

Considerando que o treinamento convergiu para um mínimo local ou no melhor caso, para o mínimo global, o termo do gradiente na Equação (3.17) pode ser ignorado. Da mesma forma o último termo da equação também pode ser desconsiderado, se for admitido que a superfície de erro em torno deste mínimo é aproximadamente quadrada. Assim, a Equação (3.17) pode ser simplificada para

$$J(\bar{\theta} + \Delta\theta) = J(\bar{\theta}) + \frac{1}{2}\Delta\theta^T \mathbf{H}(\bar{\theta})\Delta\theta, \quad (3.18)$$

e a variação no erro de saída em função da remoção de um peso pode ser dada por

$$\Delta J = J(\bar{\theta} + \Delta\theta) - J(\bar{\theta}) = \Delta\theta^T \frac{\mathbf{H}(\bar{\theta})}{2} \Delta\theta. \quad (3.19)$$

O objetivo agora é zerar um dos pesos da rede para minimizar o aumento no erro, dado pela Equação (3.19). Esta minimização é sujeita à condição $\Delta\theta_i + \theta_i = 0$, ou seja, sujeito a remoção do peso θ_i , o que recai num problema de otimização com restrições, cuja solução é obtida a partir do *lagrangiano*

$$L = \Delta\theta^T \frac{\mathbf{H}(\bar{\theta})}{2} \Delta\theta - \zeta(\mathbf{1}_i^T \Delta\theta + \theta_i), \quad (3.20)$$

em que ζ é o *multiplicador de Lagrange* e $\mathbf{1}_i$ é um vetor unitário cujos elementos são todos zeros, exceto o i -ésimo elemento. Usando as derivadas funcionais, empregando a restrição

Tabela 3.1 Algoritmo de implementação do OBS

-
-
1. Treinar uma arquitetura neural $MLP(p, q, m)$ com número elevado de pesos sinápticos, para minimizar o erro quadrático médio.
 2. Determinar de forma recursiva a inversa da matriz Hessiana (HASSIBI et al., 1992), (MEDEIROS, 2008), (HAYKIN, 2001).
 3. Utilizar a Equação (3.21) para calcular as sensibilidades de todos os parâmetros da rede, ordenando-as com base em suas magnitudes.
 4. Eliminar o peso com a menor sensibilidade.
 5. Utilizar a Equação (3.22) para atualização dos vetores de pesos da rede obtida. Recomenda-se retreinar a rede sempre que um peso ou uma pequena parte deles (3 – 5%) seja eliminada (HAYKIN, 2001; HASSIBI et al., 1992; CORANI; GUARISO, 2005).
 6. Avaliar o desempenho da rede podada nos conjuntos de treinamento e teste.
 7. Voltar ao passo 2 enquanto o procedimento de poda produzir resultados satisfatórios, de acordo com o critério de poda pré-estabelecido.
-

imposta e usando inversão de matrizes é possível fornecer uma expressão que reflita a variação do erro devido a remoção do peso θ_i , e a modificação ótima do vetor de peso $\Delta\theta$, dados respectivamente por

$$S_i = \Delta J(i) = \frac{1}{2} \frac{\theta_i^2}{[\mathbf{H}(\bar{\theta})^{-1}]_{i,i}}, \quad (3.21)$$

$$\Delta\theta = \frac{\theta_i^2}{[\mathbf{H}(\bar{\theta})^{-1}]_{i,i}} \mathbf{H}(\bar{\theta})^{-1} \mathbf{1}_i, \quad (3.22)$$

em que $[\mathbf{H}(\bar{\theta})^{-1}]_{i,i}$ é o elemento (i, i) da matriz inversa de $\mathbf{H}(\bar{\theta})$ e S_i é conhecida como a saliência ou sensibilidade do peso θ_i , que pode ser interpretada como a mudança na função custo causada pela anulação do peso em questão (LECUN et al., 1990). O algoritmo de implementação do OBS está descrito na Tabela 3.1.

3.4.3 Optimal Brain Damage (OBD)

Embora criada antes mesmo da formulação do OBS, a técnica OBD proposta por LeCun et al. (1990) pode ser considerada como um caso particular do OBS. A Equação (3.17) pode ser escrita na forma escalar, considerando uma dada perturbação $\Delta\theta$ do vetor de

parâmetros, para a função custo (erro) como

$$\Delta J = \sum_i g_i \Delta \theta_i + \frac{1}{2} \sum_i h_{ii} \Delta \theta_i^2 + \frac{1}{2} \sum_{i \neq j} h_{ij} \Delta \theta_i \Delta \theta_j, \quad (3.23)$$

em que o termo de ordem superior foi desconsiderado pela mesma justificativa dada no OBS. Os g_i 's são as componentes do gradiente, os $\Delta \theta_i$'s são as componentes de $\Delta \theta$, h_{ii} e h_{ij} são os termos diagonais e não-diagonais da matriz Hessiana, respectivamente. As componentes do gradiente e os termos não-diagonais da matriz Hessiana são escritas como

$$g_i = \frac{\partial J}{\partial \theta_i}, \quad h_{ij} = \frac{\partial^2 J}{\partial \theta_i \partial \theta_j}. \quad (3.24)$$

A simplificação referente ao mínimo local ou global da função erro considerada para OBS vale igualmente para o caso de OBD. Além disto, LeCun et al. (1990) aproxima a matriz Hessiana desprezando os termos fora da diagonal principal. Desta forma, os termos g_i e h_{ij} na Equação (3.24) são desprezados, e a variação do erro na saída devido a remoção de um peso torna-se

$$\Delta J = \frac{1}{2} \sum_i h_{ii} \Delta \theta_i^2, \quad (3.25)$$

e a sensibilidade de cada peso é dada por

$$S_i = \frac{h_{ii} \theta_i^2}{2}, \quad (3.26)$$

uma vez que não é necessário calcular inversão de matrizes, pelo fato da matriz Hessiana ser aproximada como diagonal. Assim, seus termos diagonais são dados pela expressão analítica

$$h_{kk} = \sum_{(i,j) \in V_k} \frac{\partial^2 J}{\partial w_{ij}^2}, \quad (3.27)$$

em que um simples parâmetro θ_k pode controlar uma ou mais conexões sinápticas, ou seja, $w_{ij} = \theta_k$ para todo $(i, j) \in V_k$, em que V_k é o conjunto de pares de índices. A Tabela (3.2) mostra o algoritmo de implementação de OBD.

Tabela 3.2 Algoritmo de implementação do OBD

-
-
1. Escolher uma arquitetura neural $\text{MLP}(p, q, m)$ com número elevado de pesos sinápticos.
 2. Treinar a rede até a obtenção de uma solução razoável, de acordo com critério pré-estabelecido.
 3. Utilizar a Equação (3.27) para calcular os termos diagonais h_{kk} para cada um dos parâmetros da rede.
 4. Utilizar a Equação (3.26) para calcular as sensibilidades de todos os parâmetros da rede, ordenando-as com base em suas magnitudes.
 5. Eliminar um número pré-fixado de pesos com as menores sensibilidades.
 6. Voltar ao passo 2.
-

3.4.4 Outras Técnicas

Os estudos relacionados com a seleção de modelos neurais, na busca pela arquitetura ótima para resolver um determinado problema, ainda desperta o interesse por parte de muitos pesquisadores. Nos últimos anos, diversas técnicas e metodologias foram propostas a esse respeito.

Xiang et al. (2005) estabeleceram diretrizes, com base na interpretação geométrica da rede MLP em aproximação de funções, para selecionar o número de camadas ocultas e o número de neurônios ocultos em cada camada. Foram realizadas exaustivas simulações em redes com no máximo dois nós de entrada e apenas um neurônio de saída. Eles concluíram que sua interpretação geométrica é ainda válida à medida que se aumenta a dimensão do espaço de entrada, embora as formas geométricas das funções alvos sejam difíceis de se determinar.

Delogu et al. (2008) propuseram um algoritmo construtivo baseado em programação linear para projetar redes com uma camada oculta, usadas como classificadores. Segundo os autores a aplicação do algoritmo permite determinar automaticamente o número de neurônios ocultos e os pesos sinápticos da rede.

Gómez et al. (2009) selecionaram o número de neurônios ocultos em arquiteturas neurais, com base no uso de uma medida de complexidade para estruturas com funções booleanas, nomeada complexidade de generalização. No entanto, esse método pode ser

aplicado somente a dados binários, uma vez que a definição de complexidade de generalização é válida apenas para funções booleanas.

Trenn (2008) usou a ordem de aproximação de funções através dos polinômios de *Taylor* para estimar o número de neurônios ocultos nas redes MLP. Essa análise foi no entanto puramente teórica, sem levar em conta por exemplo questões práticas como o número de dados disponível para treinamento e o ajuste dos parâmetros da rede.

A técnica de poda de conexões sinápticas realizada em Medeiros & Barreto (2007) levaram à eliminação completa de neurônios ocultos em uma rede MLP com uma camada oculta, totalmente conectada. A análise foi fundamentada na correlação entre os erros produzidos pelos neurônios de saída e os erros retropropagados associados aos neurônios da camada oculta.

De particular interesse para esta dissertação, a técnica da decomposição em valores singulares (SVD) foi utilizada por Teoh et al. (2006) para estimar o número de neurônios ocultos nas redes MLP. Nesse trabalho as saídas das funções de ativação na camada oculta serviram de base para a aplicação da técnica.

Ludermir et al. (2006) combinaram as técnicas de *simulated annealing* (arrefecimento simulado) e *tabu search* (busca tabu) para implementar um processo automático na busca de redes MLP com pequenas topologias e altas capacidades de generalização. Essa técnica foi testada em problemas de classificação e predição.

Lauret et al. (2006) propôs um algoritmo de poda de neurônios ocultos em uma rede neural totalmente conectada com uma camada oculta, baseado na análise de sensibilidade global da saída do modelo. Nesse caso, a relevância de cada neurônio oculto é determinada pela análise da decomposição de Fourier da variância da saída do modelo. Cada neurônio oculto representa uma fração dessa variância, o que pode sugerir uma indicação daqueles neurônios que podem ser eliminados.

Alguns trabalhos usaram algoritmos genéticos (SAXENA; SAAD, 2007; BENARDOS; VOSNIAKOS, 2007) e outros algoritmos revolucionários, tais como *Particle Swarm Optimization* (YU et al., 2008), para determinar a melhor topologia de rede para um dado problema, incluindo o número de camadas ocultas, o número de neurônios por camada, o número atributos de entrada relevantes, pesos e limiares. No entanto, essas metodologias consomem muito tempo de processamento e normalmente não podem ser usadas em aplicações em tempo-real.

3.5 Conclusões

Este capítulo tratou da seleção de modelos neurais na rede MLP, em aplicações de classificação de padrões ou predição. Todos os critérios e metodologias citados fundamentam-se na busca pela menor arquitetura neural capaz de extrapolar o conhecimento adquirido na fase de treinamento para amostras até então desconhecidas pela rede.

Inicialmente foram discutidos aspectos relevantes dos neurônios das camadas ocultas na rede MLP, inclusive com uma interpretação geométrica dada a rede. Embora o foco deste trabalho esteja voltado para as redes com uma única camada oculta, foi discutido sobre a escolha do número dessas camadas. Para os fins propostos nesta dissertação é possível estender sua aplicação as redes com mais de uma camada oculta.

Na sequência foram tratadas algumas regras heurísticas de bastante utilização em seleção de modelos, as quais, em sua maioria, são decorrências de extensivas experimentações com diversos problemas e, via de regra, se apresentam como uma idéia inicial na escolha pela melhor arquitetura.

Por fim foram apresentadas algumas das metodologias mais usuais na busca pelas menores e melhores topologias. Foi dado um maior destaque nas técnicas de validação cruzada, validação cruzada múltipla, OBS e OBD, as quais serão utilizadas para comparação e validação dos resultados obtidos com a metodologia proposta.

O capítulo a seguir é dedicado as teorias da Decomposição em Valores Singulares (SVD) e da Análise de Componentes Principais (PCA), que tratam-se das ferramentas que constituem a base da proposta deste trabalho.

Referências

- ALIPPI, C. FPE-based criteria to dimension feedforward neural topologies. *IEEE Transactions on Circuits and Systems: Fundamental Theory and Applications*, v. 46, n. 8, p. 962–973, 1999.
- AMARI, S.; MURATA, N.; MULLER, K. R. Statistical theory of overtraining - is cross-validation asymptotically effective? *Advances in Neural Information Processing Systems*, v. 8, p. 176–182, 1996.
- BALDI, P.; CHAUVIN, Y. Temporal evolution of generalization during learning in linear networks. *Neural Computation*, v. 3, p. 589–603, 1991.
- BALDI, P.; HORNIK, K. Neural networks and principal component analysis: learning from examples without local minima. *Neural Networks*, v. 2, n. 1, p. 53–58, 1991.
- BARRON, A. R. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information Theory*, v. 39, n. 3, p. 930–945, 1993.
- BARRON, A. R. Approximation and estimation bounds for artificial neural networks. *Machine Learning*, v. 14, n. 1, p. 115–133, 1994.
- BAUM, E. B.; HAUSSLER, D. What size net gives valid generalization? *Neural Computation*, v. 1, p. 81–90, 1989.
- BENARDOS, P. G.; VOSNIAKOS, G. C. Optimizing feedforward artificial neural network architecture. *Engineering Applications of Artificial Intelligence*, v. 20, n. 3, p. 365–382, 2007.
- BISHOP, C. M. *Neural Networks for Pattern Recognition*. 1st. ed. [S.l.]: Oxford University Press, 2005.
- BLUM, A. *Neural Networks in C++: An Object-Oriented Framework for Building Connectionist Systems*. [S.l.]: Wiley, 1992.
- BORRA, S.; CIACCIO, A. D. Measuring the prediction error. a comparison of cross-validation, bootstrap and covariance penalty methods. *Computational Statistics and Data Analysis*, v. 54, p. 2976–2989, 2010.
- BOURLARD, H.; KAMP, Y. Auto-association by multilayer perceptrons and singular value decomposition. *Biological Cybernetics*, v. 59, n. 4-5, p. 291–294, 1988.
- CAO, D.; YANG, B. An improved face recognition algorithm based on svd. In: *International Conference on Computer and Automation Engineering (ICCAE'2010)*. [S.l.: s.n.], 2010. v. 3, p. 109–112.

- CHESTER, D. Why two hidden layers are better than one. In: *IEEE Proc. Int. Conf. Neural Networks*,. Washington: [s.n.], 1990. v. 1, p. 265–268.
- CLARK, D.; SCHRETER, Z.; ADAMS, A. A quantitative comparison of dystal and backpropagation. In: *Seventh Australian Conference on Neural Networks*. [S.l.: s.n.], 1996. p. 132–137.
- CORANI, G.; GUARISO, G. An application of pruning in the design of neural networks for real time flood forecasting. *Neural Computing and Applications*, v. 14, n. 1, p. 66–77, 2005.
- COTTRELL, G. W.; MUNRO, P.; ZIPSER, D. Learning internal representations from gray-scale images: An example of extensional programming. In: *Ninth Annual Conference of the Cognitive Science Society*. [S.l.: s.n.], 1987. p. 462–473.
- CYBENKO, G. Approximation by superposition of sigmoidal functions. *Mathematics of Control, Signal and Systems*, v. 2, p. 303–314, 1989.
- DAQI, G.; YAN, J. Classification methodologies of multilayer perceptrons with sigmoid activation functions. *Pattern Recognition*, v. 38, p. 1469–1482, 2005.
- DELOGU, R.; FANNI, A.; MONTISCI, A. Geometrical synthesis of MLP neural networks. *Neurocomputing*, v. 71, p. 919–930, 2008.
- DRAGHICI, S. On the capabilities of neural networks using limited precision weights. *Neural Networks*, v. 15, p. 395–414, 2002.
- EMBRECHTS, M. J.; HARGIS, B. J.; LINTON, J. D. An augmented efficient backpropagation training strategy for deep autoassociative neural networks. In: *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*. [S.l.: s.n.], 2010. p. 141–146.
- FAHLMAN, S. E.; LEBIERE, C. The cascade-correlation learning architecture. *Touretzky, 1990*, p. 524–532, 1990.
- FUNAHASHI, K. On the approximate realization of continuous mappings by neural networks. *Neural Networks*, v. 2, n. 3, 1989.
- FUNAHASHI, K. Multilayer neural networks and bayes decision theory. *Neural Networks*, v. 11, p. 209–213, 1998.
- GILL, P. E.; MURRAY, W.; WRIGHT, M. H. *Numerical Linear Algebra and Optimization*. [S.l.]: Addison-Wesley, 1991.
- GOLOVKO, V. A. et al. Dimensionality reduction and attack recognition using neural networks approaches. In: *International Joint Conference on Neural Networks (IJCNN'2007)*. [S.l.: s.n.], 2007. p. 2734–2739.
- GOLUB, G. H.; Van Loan, C. F. *Matrix Computations*. 3rd. ed. [S.l.]: The John-Hopkins University Press, 1996.
- GÓMEZ, I.; FRANCO, L.; JEREZ, J. M. Neural network architecture selection: Can function complexity help? *Neural Processing Letters*, v. 30, p. 71–87, 2009.

- GONZALEZ, R. C.; WOODS, R. E. *Processamento de Imagens Digitais*. 1st. ed. [S.l.]: Editora Edgard Blücher Ltda., 2000.
- GOOD, R. P.; KOST, D.; CHERRY, G. A. Introducing a unified pca algorithm for model size reduction. *IEEE Transactions on Semiconductor Manufacturing*, v. 23, n. 2, p. 201–209, 2010.
- GORI, M.; SCARSELLI, F. Are multilayer perceptrons adequate for pattern recognition and verification? *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 20, n. 11, p. 1121–1132, 1998.
- HASSIBI, B.; STORK, D. G.; WOLFF, G. J. Optimal brain surgeon and general network pruning. In: *IEEE International Conference on Neural Networks*. San Francisco: [s.n.], 1992. v. 1, p. 293–299.
- HAYKIN, S. *Neural networks: a comprehensive foundation*. 2nd. ed. [S.l.]: Pearson Prentice Hall, 2001.
- HINTON, G. E.; SALAKHUTDINOV, R. R. Reducing the dimensionality of data with neural networks. *Science*, v. 313, n. 5786, p. 504–507, 2006.
- HORNIK, K.; STINCHCOMBE, M.; WHITE, H. Multilayer feedforward networks are universal approximators. *Neural Networks*, v. 2, p. 359–366, 1989.
- HUANG, S. C.; HUANG, Y. F. Bounds on number of hidden neurons in multilayer perceptrons. *IEEE Transactions on Neural Networks*, v. 2, p. 47–55, 1991.
- HWANG, J. N. et al. The past, present and future of neural networks for signal processing. *IEEE Signal Processing Magazine*, v. 14, n. 6, p. 28–48, 1997.
- JAIN, A. K.; MAO, J.; MOHIUDDIN, K. M. Artificial neural networks: A tutorial. *Computer*, v. 29, n. 3, p. 31–44, March 1996.
- KEARNS, M. A bound on the error of cross validation using the approximation and estimation rates, with consequences for the training-test split. *Advances in Neural Information Processing Systems*, v. 8, p. 183–189, 1996.
- KONSTANTINIDES, K.; YAO, K. Statistical analysis of effective singular values in matrix rank determination. *IEEE Transactions on Acoustics, Speech and Signal Processing*, v. 36, n. 5, p. 757–763, 1988.
- KROGH, A.; HERTZ, J. A. A simple weight decay can improve generalization. In: MOODY, J. E.; HANSON, S. J.; LIPPMANN, R. P. (Ed.). *Advances in Neural Information Processing Systems 4*. [S.l.]: Morgan Kaufmann, 1995.
- LAURET, P.; FOCK, E.; MARA, T. A. A node pruning algorithm based on a fourier amplitude sensitivity test method. *IEEE Transactions on Neural Networks*, v. 17, n. 2, p. 273–293, 2006.
- LAY, D. C. *Linear algebra and its applications*. 3rd. ed. [S.l.]: Addison-Wesley, 2002.
- LECUN, Y.; DENKER, J. S.; SOLLA, S. A. Optimal brain damage. In: *Advances in Neural Information Processing Systems*. San Mateo: [s.n.], 1990. v. 2, p. 598–605. CA: Morgan Kaufmann.

- LEMME, A.; REINHART, R. F.; STEIL, J. J. Efficient online learning of a non-negative sparse autoencoder. In: *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*. [S.l.: s.n.], 2010. p. 1–5.
- LEVIN, A. U.; LEEN, T. K.; MOODY, J. E. Fast pruning using principal components. In: COWAN, J.; TESAUTO, G.; ALSPECTOR, J. (Ed.). *Advances in Neural Information Processing 6*. [S.l.]: Morgan Kaufmann, 1994.
- LI, M.; XING, Y.; LUO, R. A novel feature extraction approach based on pca and improved fisher score applied in speaker verification. In: *International Conference on Audio, Language and Image Processing (ICALIP'2008)*. [S.l.: s.n.], 2008. p. 56–60.
- LIPPMANN, R. P. Pattern classification using neural networks. *Communications Magazine*, v. 11, p. 47–64, 1989.
- LIU, Y. Create stable neural networks by cross-validation. In: *International Joint Conference on Neural Networks*. [S.l.: s.n.], 2006. p. 3925–3928.
- LOONEY, C. G. Advances in feedforward neural networks: Demystifying knowledge acquiring black boxes. *IEEE Transactions on Knowledge and Data Engineering*, v. 8, n. 2, p. 211–226, April 1996.
- LUDERMIR, T. B.; YAMAZAKI, A.; ZANCHETTIN, C. An optimization methodology for neural network weights and architectures. *IEEE Transactions on Neural Networks*, v. 17, n. 6, p. 1452–1459, 2006.
- LUO, Y.; XIONG, S.; WANG, S. A pca based unsupervised feature selection algorithm. In: *Second International Conference on Genetic and Evolutionary Computing*. [S.l.: s.n.], 2008. p. 299–302.
- MASTERS, T. *Practical Neural Network Recipes in C++*. 3rd. ed. [S.l.]: Academic Press, 1993.
- MCCULLOCH, W. S.; PITTS, W. H. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, v. 5, n. 1, 1943.
- MEDEIROS, C. M. de S. *Uma contribuicao ao problema de selecao de modelos neurais usando o principio de maxima correlacao dos erros*. Tese (Doutorado) — Departamento de Engenharia de Teleinformatica, Universidade Federal do Ceara, 2008.
- MEDEIROS, C. M. S.; BARRETO, G. A. Pruning the multilayer perceptron through the correlation of backpropagated errors. In: *7th International Conference on Intelligent Systems Design and Applications (ISDA'07)*. [S.l.: s.n.], 2007. p. 64–69.
- MINSKY, M. L.; PAPERT, S. A. *Perceptrons*. [S.l.]: Cambridge, MA, 1988.
- MOODY, J. E. Prediction risk and architecture selection for neural networks. In: CHERKASSKY, V.; FRIEDMAN, J. H.; WECHSLER, H. (Ed.). *From Statistics to Neural Networks: Theory and Pattern Recognition Applications*. [S.l.]: Springer, NATO ASI Series F, 1994.

- MOODY, J. E.; ROGNVALDSSON, T. Smoothing regularizers for projective basis function networks. *Advances in Neural Information Processing Systems*, v. 9, p. 585–591, 1997.
- MURATA, N.; YOSHIZAWA, S.; AMARI, S. I. Network information criterion - determining the number of hidden units for an artificial neural network model. *IEEE Transactions on Neural Networks*, v. 5, n. 6, p. 865–872, 1994.
- OSOWSKI, S.; MAJKOWSKI, A.; CICHOCKI, A. Robust pca neural networks for random noise reduction of the data. In: *International Conference on Acoustics, Speech and Signal Processing (ICASSP'1997)*. [S.l.: s.n.], 1997. v. 4, p. 3397–3400.
- PARK, S.; KIM, J. H.; CHUNG, H.-S. A training algorithm for discrete multilayer perceptrons. In: *International Symposium on Circuit and Systems*. [S.l.: s.n.], 1991. p. 1493–1496.
- PARK, Y. R.; MURRAY, T. J.; CHEN, C. Predicting sun spots using a layered perceptron neural network. *IEEE Transactions on Neural Networks*, v. 7, n. 2, p. 501–505, 1996.
- PARKER, D. B. *Learning Logic*. [S.l.], 1985.
- PRASANTHA, H. S.; SHASHIDHARA, H. L.; MURTHY, K. N. B. Image compression using svd. In: *International Conference on Computational Intelligence and Multimedia Applications (ICCIMA'2007)*. [S.l.: s.n.], 2007. p. 143–145.
- PRECHELT, L. Automatic early stopping using cross validation: Quantifying the criteria. *Neural Networks*, v. 11, n. 4, p. 761–767, 1998.
- PRINCIPE, J. C.; EULIANO, N. R.; LEFEBVRE, W. C. *Neural and Adaptive Systems: Fundamentals Through Simulations*. 1st. ed. [S.l.]: John Wiley & Sons, Inc., 2000.
- PSICHOGIOS, D.; UNGAR, L. SVD-NET: An algorithm that automatically selects network structure. *IEEE Transactions on Neural Networks*, v. 5, n. 3, p. 513–515, 1994.
- Rocha Neto, A. R.; BARRETO, G. A. On the application of ensembles of classifiers to the diagnosis of pathologies of the vertebral column: A comparative analysis. *IEEE Latina America Transactions*, v. 7, n. 4, p. 487–496, 2009.
- ROSENBLATT, F. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, v. 65, p. 386–408, 1958.
- RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Learning internal representations by error propagation. *Nature*, v. 323, p. 533–566, 1986.
- RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Connectionist learning procedures. *Artificial Intelligence*, v. 40, p. 185–234, 1989.
- SARTORI, M. A.; ANTSAKLIS, P. J. A simple method to derive bounds on the size and to train multilayer neural networks. *IEEE Transactions on Neural Networks*, v. 2, n. 4, p. 467–471, 1991.

- SAXENA, A.; SAAD, A. Evolving an artificial neural network classifier for condition monitoring of rotating mechanical systems. *Applied Soft Computing*, v. 7, n. 1, p. 441–454, 2007.
- SETIONO, R. Feedforward neural network construction using cross-validation. *Neural Computation*, v. 13, n. 12, p. 2865–2877, 2001.
- STAHLBERGER, A.; RIEDMILLER, M. Fast network pruning and feature extraction using the unit-obs algorithm. In: MOZER, M.; JORDAN, M.; PETSCHKE, T. (Ed.). *Advances in Neural Information Processing*. [S.l.: MIT Press, 1996. v. 9, p. 655–661.
- STINCHCOMBE, M.; WHITE, H. Universal approximation using feedforward networks with non-sigmoids hidden layer activation functions. In: *Proc. Conf. Neural Networks*, [S.l.: s.n.], 1989. p. 613–618.
- STONE, M. Cross-validators choice and assessment of statistical predictions. *Journal of the Royal Statistical Society*, v. 36, n. 2, p. 111–147, 1974.
- STRANG, G. *Linear algebra and its applications*. 4th. ed. [S.l.: Brooks Coley, 2005.
- TAMURA, S.; TATEISHI, M. Capabilities of a four-layered feedforward neural network: Four layers versus three. *IEEE Transactions on Neural Networks*, v. 8, n. 2, p. 251–255, 1997.
- TAMURA, S. et al. Determination of the number of redundant hidden units in a three-layered feedforward neural network. In: *International Joint Conference on Neural Networks (IJCNN'1993)*. [S.l.: s.n.], 1993. v. 1, p. 335–338.
- TEOH, E. J.; C.TAN, K.; XIANG, C. Estimating the number of hidden neurons in a feedforward network using the singular value decomposition. *IEEE Transactions on Neural Networks*, v. 17, n. 6, p. 1623–1629, 2006.
- TRENN, S. Multilayer perceptrons: Approximation order and necessary number of hidden units. *IEEE Transactions on Neural Networks*, v. 19, n. 5, p. 836–844, 2008.
- VAPNIK, V. N. *Statistical Learning Theory*. [S.l.: John Wiley & Sons, Inc., 1998.
- VAUGHN, M. L. Derivation of the multilayer perceptron weight constraints for direct network interpretation and knowledge discovery. *Neural Networks*, v. 12, p. 1259–1271, 1999.
- VILLIERS, J. de; BARNARD, E. Backpropagation neural nets with one and two hidden layers. *IEEE Transactions on Neural Networks*, v. 4, n. 1, p. 136–141, 1992.
- WEIGEND, A.; RUMELHART, D. The effective dimension of the space of hidden units. In: *International Joint Conference on Neural Networks (IJCNN'91)*. [S.l.: s.n.], 1991. v. 3, p. 2069–2074.
- WERBOS, P. J. *Beyond regression: new tools for prediction and analysis in the behavioural sciences*. Tese (Doutorado) — Harvard University, 1974.
- WIDROW, B.; HOFF, M. E. Adaptive switching circuits. In: *IRE WESCON Convention Record-Part 4*. [S.l.: s.n.], 1960. p. 96–104.

- WITTEN, I. H.; FRANK, E. *Data Mining: Practical Machine Learning Tools and Techniques*. 2nd. ed. [S.l.]: Morgan Kaufmann, 2005.
- XIANG, C.; DING, S. Q.; LEE, T. H. Geometrical interpretation and architecture selection of MLP. *IEEE Transactions on Neural Networks*, v. 16, n. 1, p. 84–96, 2005.
- YU, J.; WANG, S.; XI, L. Evolving artificial neural networks using an improved PSO and DPSO. *Neurocomputing*, v. 71, n. 4–6, p. 1054–1060, 2008.
- YU, Y. D.; KANG, D. S.; KIM, D. Color image compression based on vector quantization using pca and lebld. In: *IEEE Region 10 Conference (TENCON'1999)*. [S.l.: s.n.], 1999. v. 2, p. 1259–1262.
- ZAKNICH, A. Neural networks for intelligent signal processing. *World Scientific*, 2003.

Perceptron Multicamadas e o Algoritmo de Retropropagação do Erro

Prof. Dr. Guilherme de Alencar Barreto

Agosto /2020

Departamento 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:

$$\mathbf{d} = \mathbf{F}[\mathbf{x}] \quad (1)$$

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.

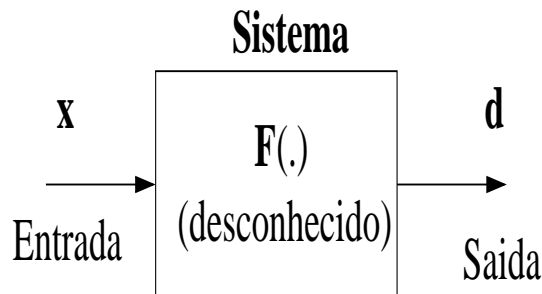


Figura 1: Representação simplificada de um mapeamento entrada-saída genérico.

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

$$\mathbf{d} = \mathbf{M}\mathbf{x} \quad (2)$$

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:

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

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 índice 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

$$\mathbf{y}_\mu = \hat{\mathbf{F}}[\mathbf{x}_\mu] \quad (4)$$

em que \mathbf{y}_μ é a saída gerada pela rede neural em resposta ao vetor de entrada \mathbf{x}_μ . Esta saída, espera-se, seja muito próxima da saída real \mathbf{d}_μ . 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 genérico.

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

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

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 saída 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

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

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.

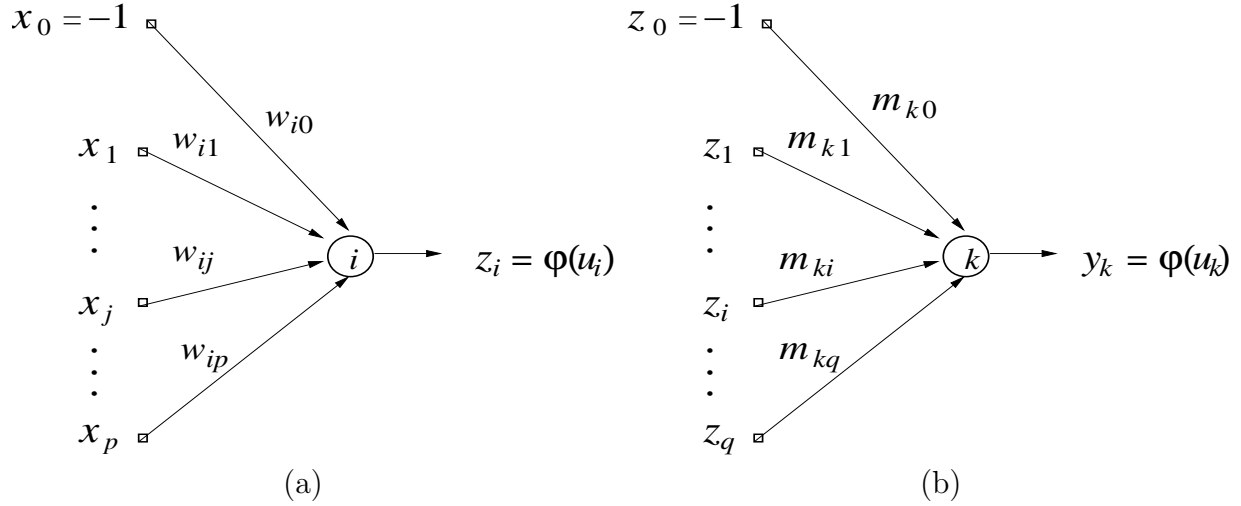


Figura 2: (a) Neurônio da camada oculta. (b) Neurônio da camada de saída.

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 a informação está fluindo no sentido **direto** (*forward*), ou seja

Entrada \rightarrow Camada Oculta \rightarrow 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:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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:

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

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

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

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

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

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

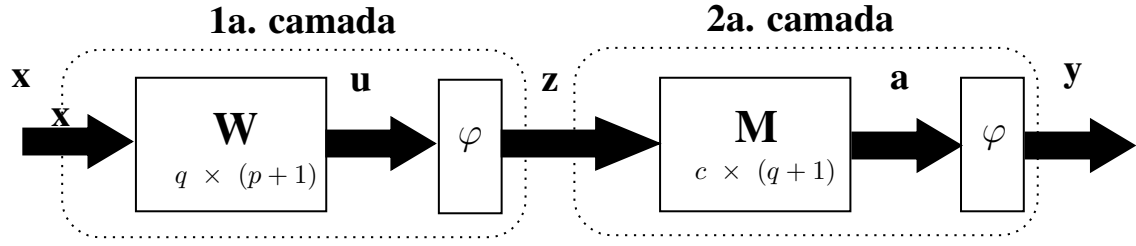


Figura 3: Diagrama de fluxo de sinais da fase direta da rede $MLP(p,q,c)$.

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 sucessivas 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

$$\mathbf{y}(t) = \phi(\mathbf{M}\mathbf{W}\mathbf{x}(t)) = \phi(\mathbf{K}\mathbf{x}), \quad (18)$$

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:

$$\delta_k(t) = e_k(t)y'_k(t), \quad k = 1, \dots, c \quad (19)$$

em que $y'_k(t) = \frac{dy_k(t)}{du_k(t)} = \frac{d\phi(u_k(t))}{du_k(t)}$ é a derivada instantânea¹ 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, $y_k(t)$:

$$e_k(t) = d_k(t) - y_k(t), \quad k = 1, \dots, c \quad (20)$$

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

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

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

¹Derivada no instante atual t .

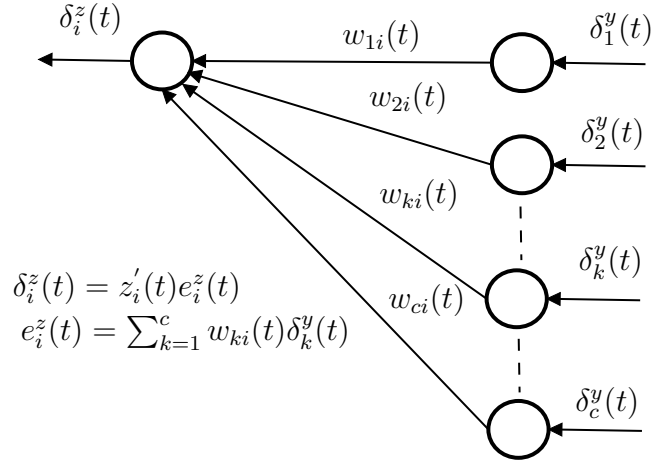


Figura 4: Cálculo do gradiente local do i -ésimo neurônio oculto ($\delta_i^z(t)$) na iteração t .

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

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

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, $i = 1, \dots, q$:

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

$$= z_i'(t) e_i^z(t), \quad (25)$$

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:

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

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

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

seguinte operação:

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

$$= \begin{bmatrix} z'_1(t) \\ z'_2(t) \\ \vdots \\ z'_i(t) \\ \vdots \\ z'_q(t) \end{bmatrix}_{q \times 1} \circ \left(\begin{bmatrix} m_{11}(t) & m_{21}(t) & \cdots & m_{k1}(t) & \cdots & m_{c1}(t) \\ m_{12}(t) & m_{22}(t) & \cdots & m_{k2}(t) & \cdots & m_{c2}(t) \\ \vdots & \vdots & \cdots & \vdots & \cdots & \vdots \\ m_{1k}(t) & m_{2k}(t) & \cdots & m_{kk}(t) & \cdots & m_{ck}(t) \\ \vdots & \vdots & \cdots & \vdots & \cdots & \vdots \\ m_{1q}(t) & m_{2q}(t) & \cdots & m_{kq}(t) & \cdots & m_{cq}(t) \end{bmatrix}_{q \times c} \cdot \begin{bmatrix} \delta_1^y(t) \\ \delta_2^y(t) \\ \vdots \\ \delta_k^y(t) \\ \vdots \\ \delta_c^y(t) \end{bmatrix}_{c \times 1} \right) \quad (29)$$

$$= \begin{bmatrix} z'_1(t) \\ z'_2(t) \\ \vdots \\ z'_i(t) \\ \vdots \\ z'_q(t) \end{bmatrix}_{q \times 1} \circ \begin{bmatrix} m_{11}(t)\delta_1^y(t) + m_{21}(t)\delta_2^y(t) + \cdots + m_{k1}(t)\delta_k^y(t) + \cdots + m_{c1}(t)\delta_c^y(t) \\ m_{12}(t)\delta_1^y(t) + m_{22}(t)\delta_2^y(t) + \cdots + m_{k2}(t)\delta_k^y(t) + \cdots + m_{c2}(t)\delta_c^y(t) \\ \vdots \\ m_{1k}(t)\delta_1^y(t) + m_{2k}(t)\delta_2^y(t) + \cdots + m_{kk}(t)\delta_k^y(t) + \cdots + m_{ck}(t)\delta_c^y(t) \\ \vdots \\ m_{1q}(t)\delta_1^y(t) + m_{2q}(t)\delta_2^y(t) + \cdots + m_{kq}(t)\delta_k^y(t) + \cdots + m_{cq}(t)\delta_c^y(t) \end{bmatrix}_{q \times 1} \quad (30)$$

$$= \begin{bmatrix} z'_1(t) \\ z'_2(t) \\ \vdots \\ z'_i(t) \\ \vdots \\ z'_q(t) \end{bmatrix}_{q \times 1} \circ \begin{bmatrix} \sum_{k=1}^c m_{k1}(t)\delta_k^y(t) \\ \sum_{k=1}^c m_{k2}(t)\delta_k^y(t) \\ \vdots \\ \sum_{k=1}^c m_{ki}(t)\delta_k^y(t) \\ \vdots \\ \sum_{k=1}^c m_{kq}(t)\delta_k^y(t) \end{bmatrix}_{q \times 1} \quad (31)$$

$$= \begin{bmatrix} z'_1(t) \cdot \sum_{k=1}^c m_{k1}(t)\delta_k^y(t) \\ z'_2(t) \cdot \sum_{k=1}^c m_{k2}(t)\delta_k^y(t) \\ \vdots \\ z'_i(t) \cdot \sum_{k=1}^c m_{ki}(t)\delta_k^y(t) \\ \vdots \\ z'_q(t) \cdot \sum_{k=1}^c m_{kq}(t)\delta_k^y(t) \end{bmatrix}_{q \times 1} \quad (32)$$

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 retropropagação 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

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

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

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

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

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

e

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

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:

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

e

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

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 , $j = 1, \dots, p$, “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 quantidade 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 para 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, descontínuo, 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 as 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:

$$q = \frac{p + c}{2} \quad (39)$$

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:

$$q = \sqrt{p \cdot c} \quad (40)$$

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:

$$q = 2p + 1 \quad (41)$$

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:

$$q \geq \frac{N - 1}{p + 2} \quad (42)$$

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:

$$N \gg Z \quad (43)$$

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:

$$N > \frac{Z}{\varepsilon} \quad (44)$$

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:

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

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:

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

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:

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

O gráfico $\varepsilon_{epoca} \times$ número de épocas ou o $P_{epoca} \times$ número de épocas 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} :

$$x_j^* = \frac{x_j}{x_j^{max}} \quad (48)$$

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} :

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

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:

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

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

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

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

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 demonstram 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, $w_{ij} \rightarrow \infty$ e $m_{ki} \rightarrow \infty$.

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:

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

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

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

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 estes 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 consequências 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

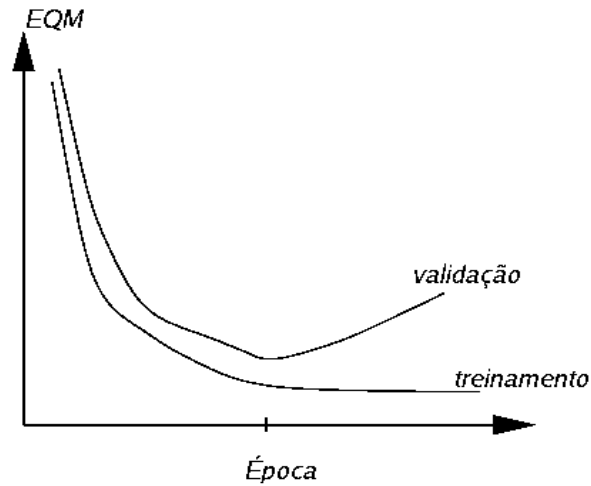


Figura 5: Curvas de aprendizagem para conjuntos de estimacão e validacão.

de aprendizagem do erro quadrático médio para o conjunto de estimacão e para o conjunto de validacão, 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 classificacão de padrões. Pede-se determinar a curva de aprendizagem da rede MLP e a taxa de acerto média na classificacão.

Exercício 2 - Usar a rede MLP para aproximar a funcão

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

através da geracão de 500 pares entrada-saída de treinamento. Pede-se determinar o gráfico da funcão $z(x, y)$ 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 generalizacão.

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.

Perceptron Simples

Prof. Dr. Guilherme de Alencar Barreto

JUL/2020

Departamento de Engenharia de Teleinformática
Programa de Pós-Graduação em Engenharia de Teleinformática (PPGETI)
Curso de Graduação em Engenharia de Teleinformática (CGGETI)
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:

$$\mathbf{d} = \mathbf{F}[\mathbf{x}] \quad (1)$$

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.

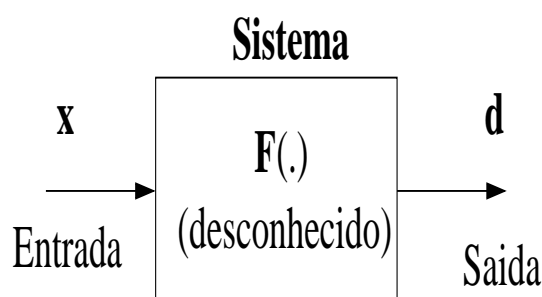


Figura 1: Representação simplificada de um mapeamento entrada-saída genérico.

O mapeamento $\mathbf{F}(\cdot)$ pode representar diversos problemas de interesse prático, tais como problemas de *aproximação de função* ou de *classificação de padrões*. De modo resumido, problemas de classificação de padrões são aqueles em que se deseja construir um programa de computador que categorize o vetor de entrada em uma dentre várias classes disponíveis. Por exemplo, de posse de um vetor contendo medidas que descrevem determinado processo industrial, deseja-se saber se este processo está operando em modo seguro (classe normal) ou inseguro (classe anormal).

Em problemas de aproximação de função, também chamados por Estatísticos de *problemas de regressão*, deseja-se conhecer a dependência numérica entre uma variável de saída e p variáveis de entrada. Por exemplo, qual a relação entre o preço hoje de uma certa ação na bolsa de valores e seus valores ontem e anteontem? Há outras variáveis importantes para descrever esta relação de modo mais preciso? Por exemplo, o preço do dólar ontem e anteontem.

Em aproximação de funções a saída é, em geral, dada por números reais, enquanto em classificação de padrões a saída é normalmente representada por números binários.

Independentemente da aplicação de interesse, o mapeamento $\mathbf{F}(\cdot)$ pode ser tão simples quanto um mapeamento linear, tal como

$$\mathbf{d} = \mathbf{M}\mathbf{x} \quad (2)$$

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*¹ 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

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

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 índice 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

$$\mathbf{y} = \hat{\mathbf{F}}[\mathbf{x}], \quad (4)$$

em que \mathbf{y} é a saída gerada pela rede neural que, espera-se, seja muito próxima da saída real \mathbf{d} . 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 um dos primeiros modelos matemáticos adaptativos usados com o propósito de obter uma representação aproximada de um mapeamento entrada-saída qualquer.

2 Modelo Perceptron Simples

Nesta seção descreveremos um tipo elementar de algoritmo adaptativo, chamado **Perceptron Simples**, composto de apenas um elemento processador (neurônio). Este modelo foi proposto por Rosenblatt no seguinte artigo:

F. Rosenblatt (1958). “The Perceptron: A probabilistic model for information storage and organization in the brain”, *Psychological Review*, vol. 65, p. 386-408.

O modelo Perceptron simples é considerado como a primeira arquitetura neural inventada. Seu bloco construtivo é o neurônio artificial de McCulloch & Pitts (neurônio M-P), mostrado na Figura 2, proposto no seguinte artigo:

¹Termo, digamos assim, mais tecnicamente apropriados do que *imitar* seriam *aproximar* ou *modelar*.

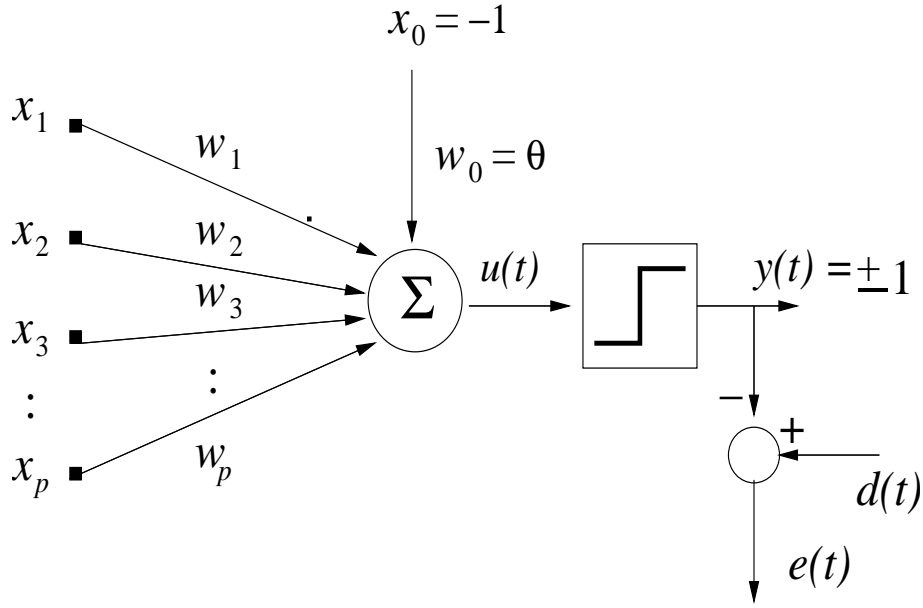


Figura 2: Arquitetura do neurônio da rede neural Perceptron.

W. S. McCulloch and W. Pitts (1943). “A logical calculus of the ideas immanent in nervous activity”, *Bulletin of Mathematical Biophysics*, vol. 5, p. 115-133.

O vetor de entrada do Perceptron é então definido como

$$\mathbf{x}(t) = \begin{pmatrix} x_0(t) \\ x_1(t) \\ \vdots \\ x_j(t) \\ \vdots \\ x_p(t) \end{pmatrix} = \begin{pmatrix} -1 \\ x_1(t) \\ \vdots \\ x_j(t) \\ \vdots \\ x_p(t) \end{pmatrix} \quad (5)$$

em que $x_j(t)$ denota uma componente qualquer do vetor de entrada $\mathbf{x}(t)$ e t indica o instante de apresentação deste vetor à rede.

O vetor de saídas desejadas é representado por um vetor de q componentes, ou seja

$$\mathbf{d}(t) = \begin{pmatrix} d_1(t) \\ \vdots \\ d_i(t) \\ \vdots \\ d_q(t) \end{pmatrix}, \quad (6)$$

em que $d_i(t)$ a saída desejada para o i -ésimo neurônio.

O vetor de pesos associado ao i -ésimo neurônio é representado da seguinte forma:

$$\mathbf{w}_i(t) = \begin{pmatrix} w_{i0}(t) \\ w_{i1}(t) \\ \vdots \\ w_{ij}(t) \\ \vdots \\ w_{ip}(t) \end{pmatrix} = \begin{pmatrix} \theta_i(t) \\ w_{i1}(t) \\ \vdots \\ w_{ij}(t) \\ \vdots \\ w_{ip}(t) \end{pmatrix}, \quad (7)$$

em que w_{ij} é o peso sináptico que conecta a entrada j ao i -ésimo neurônio, θ_i define um limiar (*threshold* ou *bias*) associado ao i -ésimo neurônio.

Importante: Cada neurônio da rede Perceptron possui o seu próprio vetor de pesos \mathbf{w}_i . Assim, uma rede com q neurônios terá $p \times q$ pesos sinápticos w_{ij} e q limiares θ_i , resultando em um total de $(p + 1) \times q$ parâmetros ajustáveis. Estes parâmetros deverão ser ajustados por meio de uma regra de atualização recursiva denominada **Regra de Aprendizagem do Perceptron**.

3 Funcionamento do Perceptron Simples

Após a apresentação de um vetor de entrada \mathbf{x} , na iteração t , a ativação $u_i(t)$ do i -ésimo neurônio de saída é calculada por meio da seguinte expressão:

$$\begin{aligned} u_i(t) &= \sum_{j=1}^p w_{ij}(t)x_j(t) - \theta_i \\ &= \sum_{j=1}^p w_{ij}(t)x_j(t) + w_{i0}(t)x_0(t) \\ &= \sum_{j=0}^p w_{ij}(t)x_j(t) \\ &= \mathbf{w}_i^T(t)\mathbf{x}(t) \end{aligned} \quad (8)$$

em que foi feito $x_0(t) = -1$ e $w_{i0}(t) = \theta_i(t)$. O sobrescrito T indica a operação de transposição dos vetores. Note que a ativação do neurônio no instante t é simplesmente o produto-escalar do vetor de entrada $\mathbf{x}(t)$ com o vetor de pesos $\mathbf{w}_i(t)$ do i -ésimo neurônio. Assim, a rede Perceptron faz uma verificação de quais vetores de pesos são mais semelhantes ao vetor de entrada atual.

Do ponto de vista geométrico, valores positivos de $u_i(t)$ indicam que o ângulo entre os vetores $\mathbf{x}(t)$ e $\mathbf{w}_i(t)$ é menor do que 90 graus² (Figura 3a).

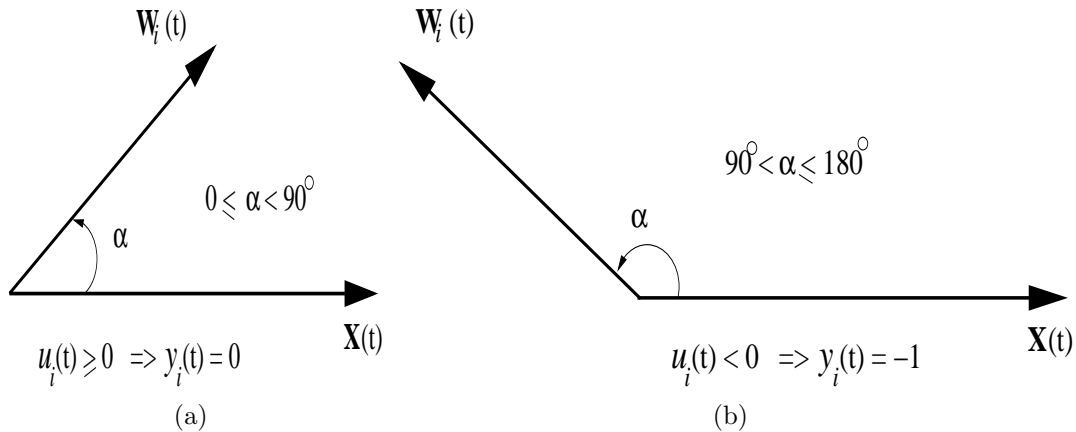


Figura 3: Produto interno entre o vetor de entrada atual $\mathbf{x}(t)$ e o vetor de pesos $\mathbf{w}_i(t)$ do i -ésimo neurônio da rede Perceptron.

²Por definição, o produto-escalar entre dois vetores \mathbf{x} e \mathbf{y} é dado por $\mathbf{x} \cdot \mathbf{y} = \|\mathbf{x}\| \cdot \|\mathbf{y}\| \cdot \cos \beta$, em que β é o ângulo entre os vetores.

A saída atual do Perceptron é dada pela aplicação da função *sinal*, dada por:

$$y_i(t) = \text{sinal}(u_i(t)) = \begin{cases} +1, & u_i(t) \geq 0 \\ -1, & u_i(t) < 0 \end{cases} \quad (9)$$

Note que como a saída $y_i(t)$ pode assumir apenas dois possíveis valores (isto é, a saída é binária!), o Perceptron é comumente aplicado a problemas de classificação de padrões. Conforme mostrado na Eq. (8), a ativação do i -ésimo neurônio é o resultado do produto-escalar do vetor de entrada atual com o seu vetor de pesos. Assim, a saída $y_i(t)$ indica se o vetor de pesos $\mathbf{w}_i(t)$ pode ser considerado similar o suficiente ao vetor $\mathbf{x}(t)$. Caso afirmativo, então $y_i = +1$, indicando o disparo do neurônio. Caso contrário, a saída é $y_i(t) = -1$.

Em particular pode-se fazer a seguinte afirmação:

O algoritmo Perceptron é basicamente utilizado em tarefas de classificação de padrões em que os dados ou as classes são linearmente separáveis.

Exercício 1 - Explicar geometricamente a afirmação de que o Perceptron só é capaz de classificar dados linearmente separáveis. Dica: usar as funções lógicas *AND*, *OR* e *XOR* para desenvolver o raciocínio.

4 Função-Objetivo do Perceptron Simples

Nesta seção demonstraremos que o Perceptron Simples é um classificador linear ótimo, ou seja, sua regra de aprendizagem conduz à minimização de uma função-custo ou função-objetivo específica. Por ser uma máquina (algoritmo) capaz de aprender, os pesos sinápticos do Perceptron Simples devem ser determinados a fim de que a saída $y_i(t)$ gerada esteja bem próxima da saída desejada $d_i(t)$ associada a um certo vetor de entrada $\mathbf{x}(t)$.

Uma escolha razoável para a função-objetivo do Perceptron Simples seria aquela que quantificasse a *probabilidade média de erros de classificação*, ou seja, que fornecesse a probabilidade média de o Perceptron tomar uma decisão em favor de uma classe específica quando o vetor de entrada pertencesse na realidade a outra classe.

A título de derivação teórica, vamos considerar apenas um neurônio de saída, de modo que o problema de classificação envolve apenas duas classes (classe 1 - ω_1 e classe 2 - ω_2). Por existir apenas um neurônio passaremos a representar o vetor de pesos deste simplesmente por \mathbf{w} , eliminando o índice i . Para a análise que se segue adotar a função sinal como função quantizadora da saída.

Vamos também redefinir cada vetor de entrada \mathbf{x} , de modo que este passe a ser representado como

$$\mathbf{z} = \begin{cases} \mathbf{x}, & \text{se } \mathbf{x} \in \omega_1. \\ -\mathbf{x}, & \text{se } \mathbf{x} \in \omega_2. \end{cases} \quad (10)$$

O propósito da Definição (10) é fazer com que o produto $\mathbf{w}^T \mathbf{z}$ seja sempre positivo para todo \mathbf{z} . Por definição, se $\mathbf{w}^T \mathbf{z} > 0$ para todo \mathbf{z} disponível, então o problema de classificação é dito ser linearmente separável. Contudo, na prática, mesmo que não consigamos obter $\mathbf{w}^T \mathbf{z} > 0$ para todos os vetores de entrada, buscamos uma solução para \mathbf{w} que produza $\mathbf{w}^T \mathbf{z} > 0$ para tantos vetores de entrada quanto possível.

Em outras palavras, buscamos minimizar o erro de classificação dos vetores de entrada. Matematicamente, este procedimento pode ser representado pela seguinte função-objetivo[1]:

$$J[\mathbf{w}] = \sum_{\mathbf{z}_k \in \mathcal{Z}} (-\mathbf{w}^T \mathbf{z}_k), \quad (11)$$

em que \mathbf{z}_k denota o k -ésimo vetor de entrada mal-classificado (i.e. classificado erroneamente) e \mathcal{Z} é o conjunto dos vetores mal-classificados.

Visto que a função $J[\mathbf{w}]$ é contínua, podemos lançar mão de um procedimento baseado na derivada primeira de uma função para encontrar sua solução ótima. No presente caso, utilizamos um método iterativo conhecido como método do gradiente descendente³ para derivar uma regra de ajuste recursivo do vetor de pesos \mathbf{w} . Assim, temos que

$$\mathbf{w}^{novo} = \mathbf{w}^{atual} + \Delta \mathbf{w} \quad (12)$$

$$= \mathbf{w}^{atual} - \alpha \frac{\partial J[\mathbf{w}]}{\partial \mathbf{w}} \quad (13)$$

em que \mathbf{w}^{atual} corresponde o valor atual de \mathbf{w} , enquanto \mathbf{w}^{novo} denota o valor depois de ajustado. A constante $0 < \alpha \ll 1$ é chamada de passo ou taxa de aprendizagem.

A derivada $\frac{\partial J[\mathbf{w}]}{\partial \mathbf{w}}$ é dada por

$$\frac{\partial J[\mathbf{w}]}{\partial \mathbf{w}} = \sum_{\mathbf{z}_k \in \mathcal{Z}} (-\mathbf{z}_k), \quad (14)$$

que nada mais é do que a soma dos vetores mal-classificados. Substituindo este resultado na Definição (13) chegamos a

$$\mathbf{w}^{novo} = \mathbf{w}^{atual} + \alpha \sum_{\mathbf{z}_k \in \mathcal{Z}} \mathbf{z}_k. \quad (15)$$

O tipo de treinamento que usa a regra de aprendizagem mostrada na Equação (15) é algumas vezes chamada de treinamento **época-a-época** ou **em lote** (*batch*), uma vez que todos os vetores mal-classificados são usados ao mesmo tempo para atualizar \mathbf{w} . Muitas vezes é preferível atualizar \mathbf{w} logo que um erro de classificação ocorre. Este procedimento é comumente chamado de treinamento **padrão-a-padrão** (*pattern-by-pattern*) ou **seqüencial**. Neste caso, a regra de aprendizagem passa a ser escrita como

$$\mathbf{w}(t+1) = \mathbf{w}(t) + \alpha \mathbf{z}(t), \quad (16)$$

em que t denota o instante de apresentação do vetor de entrada $\mathbf{z}(t)$.

Em termos da notação original, a regra de aprendizagem mostrada na Eq. (16) pode a ser escrita como:

$$\mathbf{w}(t+1) = \mathbf{w}(t) + \alpha e(t) \mathbf{x}(t), \quad (17)$$

em que $e(t) = d(t) - y(t)$ corresponde ao erro de classificação do vetor de entrada $\mathbf{x}(t)$. Note que quando a classificação do vetor de entrada é correta, o erro correspondente é nulo (i.e. $e(t) = 0$), não havendo ajuste do vetor \mathbf{w} . Só há ajuste quando ocorre uma classificação incorreta.

Para o caso em que há q neurônios, a regra de ajuste do vetor de pesos do i -ésimo neurônio é dada por

$$\mathbf{w}_i(t+1) = \mathbf{w}_i(t) + \alpha e_i(t) \mathbf{x}(t), \quad (18)$$

em que $e_i(t) = d_i(t) - y_i(t)$ corresponde ao erro de classificação do i -ésimo neurônio.

Considerando a regra de aprendizagem do perceptron apenas do ponto de vista do peso sináptico w_{ij} podemos escrever

$$w_{ij}(t+1) = w_{ij}(t) + \alpha e_i(t) x_j(t), \quad i = 1, \dots, q \quad j = 0, 1, \dots, p \quad (19)$$

em que x_j corresponde à j -ésima entrada. Analisando a Eq. (19), podemos concluir que atualização do peso w_{ij} depende apenas de informação *local*, ou seja, de informação disponível apenas

³Também chamado de método da descida mais íngreme (*method of steepest descent*).

naquela sinapse do neurônio i , seja através da entrada por meio de $x_j(t)$, seja na saída por meio de $e_i(t)$.

Para finalizar, vamos considerar uma notação matricial para a regra de aprendizagem do Perceptron, já apresentada sob uma notação vetorial na Eq. (18) e sob uma notação escalar na Eq. (19). A notação matricial permite que o Perceptron seja implementado em poucas linhas quando usamos softwares do tipo Matlab/Octave/Scilab para simulação computacional.

Considere que os vetores de pesos \mathbf{w}_i estão organizados como linhas de uma matriz \mathbf{W} , ou seja

$$\mathbf{W} = \begin{pmatrix} \mathbf{w}_1^T \\ \mathbf{w}_2^T \\ \vdots \\ \mathbf{w}_q^T \end{pmatrix}, \quad (20)$$

em que $\dim(\mathbf{W}) = q \times (p + 1)$. Organizando os erros gerados pelos q neurônios de saída no instante t em um vetor de erros $\mathbf{e}(t) = [e_1(t) \ e_2(t) \ \cdots \ e_q(t)]^T$, podemos escrever a regra de aprendizagem do Perceptron Simples da seguinte maneira:

$$\mathbf{W}(t + 1) = \mathbf{W}(t) + \alpha \mathbf{e}(t) \mathbf{x}^T(t), \quad (21)$$

em que esta expressão atualiza a matriz de pesos \mathbf{W} , de dimensões $q \times (p + 1)$, de uma só vez. Cada termo da Eq. (21) deve ter dimensões compatíveis, de modo que para isso ser possível o termo de correção envolve o produto externo do vetor de erros $\mathbf{e}(t)$ com o vetor de entrada $\mathbf{x}(t)$. É por isso que o vetor de entrada aparece transposto nessa equação.

A versão matricial da regra de aprendizado do Perceptron não é muito comum de se ver em livros, porém é extremamante vantajosa para implementação em Octave, Matlab e Scilab, devido à maior velocidade que confere à execução do algoritmo. Usando a Eq. (21) a matriz de pesos \mathbf{W} é atualizada de uma vez a cada apresentação de um vetor de entrada. Note que isto só é possível porque o funcionamento de cada neurônio é independente do funcionamento dos demais, dada a natureza local da regra desta regra de aprendizado.

Exercício 2 - Obter através de argumentos geométricos a regra de aprendizagem do Perceptron mostrada na Eq. (18). Sugestão: avaliar as condições em que o Perceptron erra a classificação de um dado vetor de entrada, e o que o Perceptron deveria fazer para passar a não errar àquela classificação.

4.1 Metodologias de Treinamento e Avaliação

O processo de ajuste dos pesos, também chamado de *fase de aprendizado* ou *fase de treinamento* do Perceptron Simples equivale a um aprendizado que depende de um sinal de saída conhecido previamente, ou como se diz no jargão da área, depende de saídas desejadas $d_i(t)$, fornecida por um “supervisor” externo a fim de guiar o processo de ajuste dos parâmetros, conforme ilustrado na Figura 4. Por este motivo, este tipo de aprendizado é chamado também de *aprendizado supervisionado*.

O desempenho efetivo do Perceptron Simples como classificador é avaliado após a apresentação de um conjunto de vetores selecionados aleatoriamente do conjunto total de vetores disponíveis. Os vetores aleatoriamente selecionados constituem o *conjunto de treinamento*. Várias reapresentações do conjunto de treinamento podem ser necessárias até que a rede aprenda a representar adequadamente o conjunto de dados. A seguir são dadas algumas dicas para treinar e avaliar o modelo Perceptron simples.

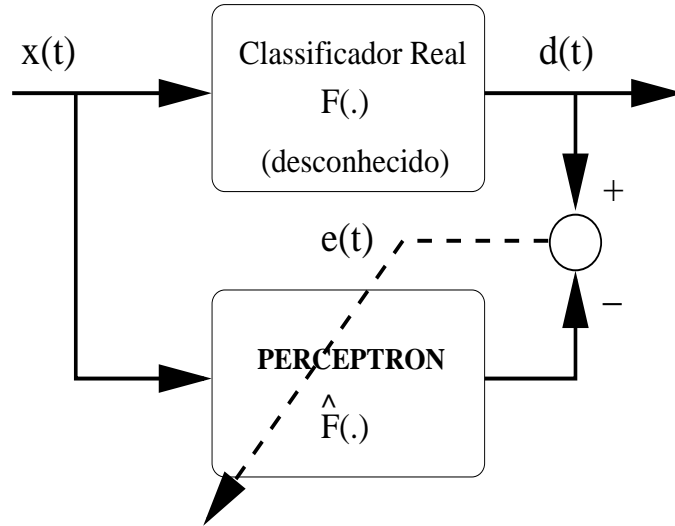


Figura 4: Aprendizado supervisionado.

Fase de Treinamento - Nesta etapa, um determinado número $N_1 < N$ de pares entrada-saída $\{\mathbf{x}_\mu, \mathbf{d}_\mu\}$ são escolhidos aleatoriamente para formar o *conjunto de dados de treinamento*. Não existe regra definida para a escolha de N_1 ; em geral, faz-se $N_1/N \approx 0,75$ ou $0,8$. Os N_1 pares são então apresentados, um por vez, para o modelo Perceptron, que tem seus pesos modificados segundo a Equação (18).

Toda vez que se apresenta todos os N_1 vetores de treinamento, diz-se que uma *época de treinamento* foi completada. Em geral, o conjunto de treinamento é apresentado mais de uma época ao modelo Perceptron, sendo que a ordem de apresentação dos pares entrada-saída deve ser mudada a cada época.

O treinamento deve ser interrompido quando o aprendizado convergir. Isto equivale a verificar se os pesos deixaram de variar, ou seja

$$\|\Delta \mathbf{W}\|_{frob} = \|\mathbf{W}(n) - \mathbf{W}(n-1)\|_{frob} \approx 0. \quad (22)$$

em que $\|\mathbf{A}\|_{frob}$ denota a norma de Frobenius⁴ da matriz \mathbf{A} , a matriz \mathbf{W} está definida na Eq. (21) e n corresponde à época de treinamento atual. Para monitorar a convergência época-a-época, pode-se fazer o gráfico da norma de Frobenius da matriz $\mathbf{W}(n)$ em função de n . Este gráfico, em geral, apresenta uma tendência de queda exponencial até atingir um patamar próximo de zero.

Outro critério de parada para o treinamento envolve simplesmente a especificação de um número máximo ($K_{max} \gg 1$) de épocas de treinamento. Normalmente, os dois critérios são usados simultaneamente.

Fase de Teste - Para validar o modelo Perceptron treinado, ou seja, dizer que ele está pronto para ser utilizado, é importante testar a sua resposta para dados de entrada diferentes daqueles vistos durante o treinamento. Durante esta fase os pesos da rede não são ajustados.

Os pares entrada-saída de teste podem ser obtidos através de novas medições, o que nem sempre é possível de se realizar. A opção viável então consiste na utilização dos $N_2 = N - N_1$ pares entrada-saída restantes, chamados de *conjunto de teste*. A avaliação, em geral, é feita também com base no valor de J_N . O valor de J_N calculado para o conjunto de teste é chamado

⁴ $\|\mathbf{A}\|_{frob} = \sqrt{\text{tr}(\mathbf{A}^T \mathbf{A})}$, em que $\text{tr}(\mathbf{B})$ é o traço da matriz \mathbf{B} .

de *erro de generalização* do modelo Perceptron, pois testa a capacidade deste em “extrapolar” o conhecimento aprendido durante o treinamento para novos dados.

Como o modelo Perceptron é usado como classificador de padrões, o seu desempenho é também comumente avaliado pela *taxa de acerto na classificação*, definida como:

$$P_{\text{epoca}} = \frac{\text{Número de vetores classificados corretamente}}{\text{Número de total de vetores de teste}} \quad (23)$$

Neste caso, é prática estabelecida representar o vetor de saídas desejadas $\mathbf{d}(t)$ como um vetor que tem apenas uma componente com valor igual a $+1$, as outras possuem valor -1 . A saída desejada com valor $+1$ representa a classe do vetor de entrada correspondente.

Assim, o número de neurônios é sempre igual ao número de classes existentes. Por exemplo, para um problema de classificação com três classes, teríamos três neurônios de saída, sendo que os respectivos vetores de saídas desejadas, representando os rótulos das classes, seriam representados por:

$$\text{Classe 1} \Rightarrow \mathbf{d}(t) = \begin{bmatrix} +1 \\ -1 \\ -1 \end{bmatrix}, \quad \text{Classe 2} \Rightarrow \mathbf{d}(t) = \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix} \quad \text{e} \quad \text{Classe 3} \Rightarrow \mathbf{d}(t) = \begin{bmatrix} -1 \\ -1 \\ 1 \end{bmatrix} \quad (24)$$

Exercício 3 - Usar o banco de dados sobre dermatologia disponíveis no repositório UCI para avaliar o modelo Perceptron em um problema de diagnóstico automático de doenças. Pede-se determinar a curva de aprendizagem do modelo Perceptron, a taxa de acerto média e o desvio-padrão correspondente obtidos no teste e a matriz de confusão.

Referências

- [1] A. Webb. *Statistical Pattern Recognition*. John Wiley & Sons, 2nd edition, 2002.