Perceptron Simples

Prof. Dr. Guilherme de Alencar Barreto

JUL/2020

Departmento 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:

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

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

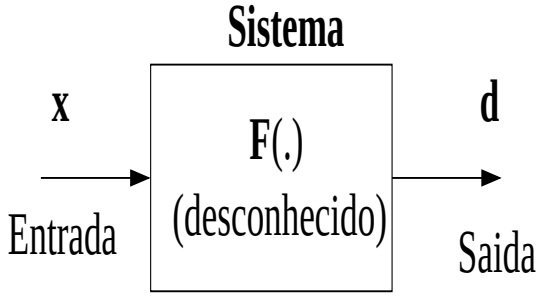

Figura 1: Representação simplificada de um mapeamento entrada-saída genérico.

O mapeamento $\mathbf{F}(\cdot)$ pode 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 desja 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 númerica 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

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

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

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

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

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

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

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:

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

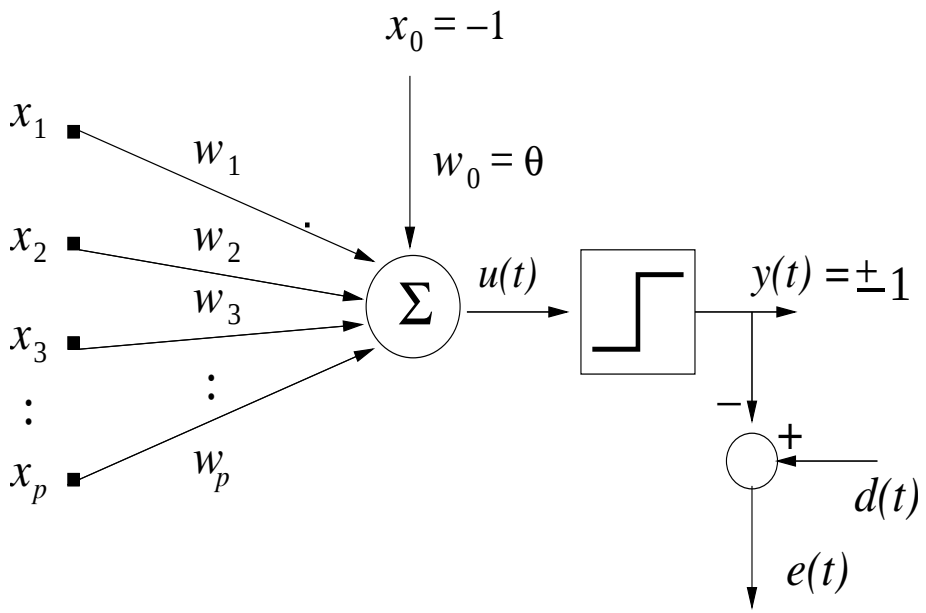

Figura 2: Arquitetura do neurônio da rede neural Perceptron.

W. S. McCullogh 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

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

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

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

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

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:

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

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 sináticos 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:

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

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 graus[2] (Figura 3a).

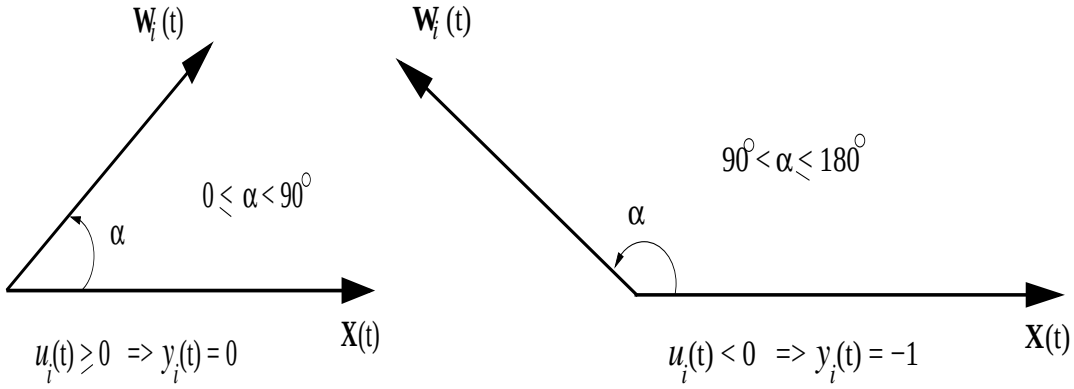

Figura 3: Produto interno entre o vetor de entrada atual $\mathbf{x}(t)$ e o vetor de pesos $\mathbf{w}_i(t)$ do i-ésimo neurônio da rede Perceptron.

[2]Por 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:

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

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

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

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]:

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

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 descendente[3] para derivar uma regra de ajuste recursivo do vetor de pesos $\mathbf{w}$. Assim, temos que

$$\mathbf{w}^{novo} = \mathbf{w}^{atual} + \Delta\mathbf{w} \tag{12}$$

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

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

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

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

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

O tipo de treinamento que usa a regra de aprendizagem monstrada 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

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

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:

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

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

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

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

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

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

[3] Também 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

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

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:

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

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 extremamanete 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.

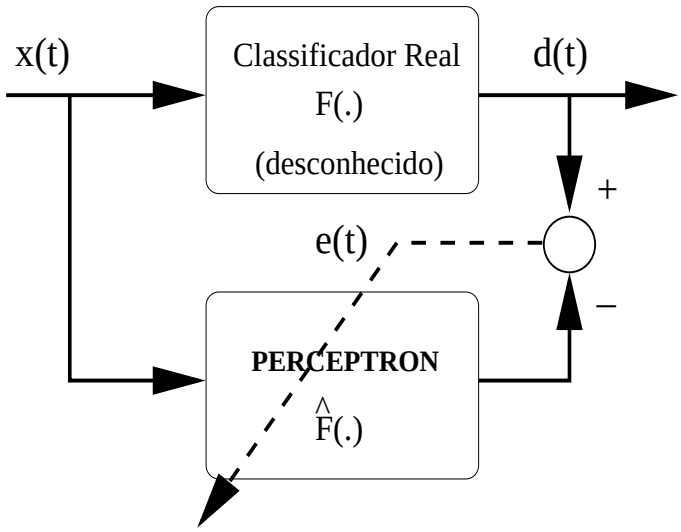

Figura 4: Aprendizado supervisionado.

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

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

em que $\|\mathbf{A}\|_{frob}$ denota a norma de Frobenius[4] 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] $\|\mathbf{A}\|_{frob} = \sqrt{\mathrm{tr}(\mathbf{A}^T\mathbf{A})}$, em que $\mathrm{tr}(\mathbf{B})$ 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:

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

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:

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

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.