Capítulo 3

O Perceptron

No capítulo anterior estudamos algoritmos de aprendizagem supervisionados, nos quais o aprendizado acontece através de um tutor. Em 1958 Rosenblatt propôs o Perceptron como o primeiro modelo para aprendizagem de RNAs por meio de um tutor.

O Perceptron é a forma mais simples de uma RNA usada para classificação de padrões linearmente separáveis; ou seja, padrões que estão em lados opostos de um hiperplano. Consiste basicamente de um único neurônio com pesos sinápticos ajustáveis e uma polarização (*bias*).

O algoritmo usado para ajustar os parâmetros livres desta RNA foi apresentado pela primeira vez no procedimento de aprendizagem desenvolvido por Rosenblatt, que provou que:

> *Se os padrões (vetores) usados para treinar o Perceptron são retirados de duas classes linearmente separáveis, então o algoritmo Perceptron converge e posiciona a superfície de decisão na forma de um hiperplano entre as duas classes.*

A prova de convergência do algoritmo é conhecida como Teorema de Convergência do Perceptron.

O Perceptron de um único neurônio é limitado a desempenhar classificação de padrões com apenas duas classes (duas hipóteses). Através da expansão da camada computacional de saída do Perceptron para incluir mais do que um neurônio, é possível classificar mais do que duas classes. Entretanto, as classes têm que ser linearmente separáveis para que o Perceptron tenha um desempenho adequado. Um ponto importante é

que a extensão da teoria básica do Perceptron a partir do caso de um neurônio para o caso de mais de um neurônio é trivial.

O neurônio único também forma a base de um filtro adaptativo, um bloco funcional que é básico nas aplicações concernentes a processamento de sinais. O desenvolvimento da filtragem adaptativa é devido grandemente ao clássico trabalho de Widrow e Hoff (1960) por apresentarem pela primeira vez o algoritmo *Least-Mean-Square* (LMS), também conhecido como a Regra Delta.

O algoritmo LMS e o Perceptron são relacionados e serão estudados ao longo deste capítulo. Primeiramente iremos abordar o problema da filtragem adaptativa e o algoritmo LMS para, após, tratarmos do Perceptron de Rosenblatt.

3.1 O Problema da Filtragem Adaptativa

Consideremos um sistema dinâmico $\mathbf{\Gamma}$ cuja caracterização matemática é **desconhecida**. O máximo de conhecimento que temos a respeito de $\mathbf{\Gamma}$ é um conjunto finito de dados, que é um subconjunto χ do universo Ω de todos os possíveis mapeamentos entrada-saída que podem ser gerados pelo sistema $\mathbf{\Gamma}$.

Suponhamos que os elementos do subconjunto χ sejam pares $(\underline{x}(i), d(i)) \in \chi$, onde :

$\underline{x}(i)$ é o *i*-ésimo vetor *M*-dimensional de χ aplicado na entrada de $\mathbf{\Gamma}$ e

$d(i)$ é a saída de $\mathbf{\Gamma}$ à entrada $\underline{x}(i)$, $i = 0,1,\cdots,N-1$, sendo

N o número de elementos de χ.

Especificamente, quando um estímulo real *M*-dimensional $\underline{x}(i) \in \Re^M$ é aplicado aos *M* nós de entrada do sistema $\mathbf{\Gamma}$, $\mathbf{\Gamma}$ responde gerando a saída escalar $d(i)$, como mostra a Figura 3.1(a).

A dimensão *M* dos vetores $\underline{x}(i)$ é usualmente referida como dimensionalidade do espaço de entrada.

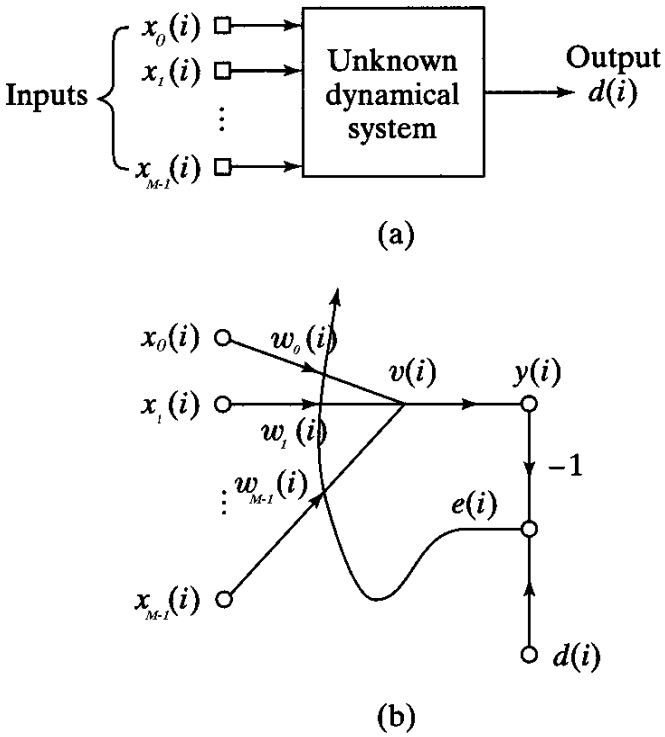

Figura 3.1: (a) Sistema dinâmico desconhecido $\mathbf{\Gamma}$. (b) Grafo de fluxo de sinal para o modelo adaptativo do sistema.

Portanto, o comportamento externo do sistema $\mathbf{\Gamma}$ é descrito pelo mapeamento

$$\Gamma : \underline{x}(i) \in \Re^M \rightarrow d(i) \in \Re, \quad i = 0,1,\cdots,N-1 \tag{3.1}$$

onde $\underline{x}(i)$ é o i-ésimo vetor de χ, definido por

$$\underline{x}(i) = [x_0(i) \quad x_1(i) \quad \cdots \quad x_{M-1}(i)]^T \tag{3.2}$$

Note que, na grande maioria dos casos, também não se conhece com precisão a distribuição de probabilidade dos elementos do conjunto χ, de modo que a tentativa de resolver um problema de filtragem através de uma abordagem estatística (através da matriz de correlação, por exemplo) não raro conduz a resultados não satisfatórios.

Um estímulo $\underline{x}(i)$ aplicado a um sistema $\mathbf{\Gamma}$ pode originar-se de dois cenários fundamentais, um espacial e outro temporal:

- Os M elementos do vetor $\underline{x}(i)$ originam-se de M distintas fontes de informação localizadas em diferentes pontos no espaço, sendo todos os M elementos obtidos no mesmo instante, de todas as M fontes.

- Os M elementos do vetor $\underline{x}(i)$ representam o valor presente e os $M-1$ valores passados de amostras seqüencialmente originadas de uma única fonte de informação.

Um problema clássico em filtragem adaptativa, conhecido como identificação de sistema, é determinar o modelo que rege o comportamento do sistema dinâmico desconhecido $\boldsymbol{\Gamma}$, caracterizado por (3.1), utilizando para tanto um único neurônio linear. O neurônio opera sob a influência de um algoritmo **A** que controla os ajustes necessários à transmitância (≡peso) de suas sinapses para que, à medida que os ajustes se sucedem, o mapeamento efetuado pelo neurônio tenda a aproximar o mapeamento efetuado pelo sistema $\boldsymbol{\Gamma}$. Este processo de ajustes sucessivos dos pesos sinápticos é efetuado observando as seguintes características:

- O algoritmo **A** inicia o processo de ajuste a partir de um conjunto de transmitâncias sinápticas (≡pesos sinápticos), com valor inicial arbitrário atribuído a cada uma delas.

- O algoritmo **A** ajusta as transmitâncias sinápticas do neurônio continuamente ao longo do intervalo de operação do sistema $\boldsymbol{\Gamma}$, para permitir que eventuais variações no padrão de comportamento de $\boldsymbol{\Gamma}$ (variações na estatística do comportamento de $\boldsymbol{\Gamma}$) também possam influenciar o processo de ajuste.

- Para cada valor de i, o algoritmo **A** deve ser rápido o suficiente para ajustar todas as M transmitâncias sinápticas do neurônio dentro do intervalo de tempo que transcorre entre a ocorrência das entradas $\underline{x}(i)$ e $\underline{x}(i+1)$.

A Figura 3.1 (b) mostra o grafo de fluxo de sinal de um filtro adaptativo de neurônio único, aplicado ao contexto de identificação do sistema desconhecido **Γ**. A operação do filtro consiste de dois processos continuamente executados em seqüência:

1. Processo de Filtragem, o qual envolve o cômputo de dois sinais :

   1.1. Uma saída, denotada por $y(i)$, que é produzida em resposta ao vetor estímulo $\underline{x}(i)$.

   1.2. Um sinal de erro, denotado por $e(i)$, que é obtido pela comparação da saída $y(i)$ com a saída desejada $d(i)$ correspondente, sendo $d(i)$ produzida por **Γ** quando o estímulo $\underline{x}(i)$ é aplicado à sua entrada. Em outras palavras, $d(i)$ constitui a resposta desejada ou o sinal alvo (*target signal*).

2. Processo de Adaptação, o qual envolve o ajuste automático dos pesos sinápticos do neurônio através de um algoritmo **A**, tendo como base o sinal de erro $e(i)$.

Desta maneira, a combinação destes dois processos (operando em conjunto) constitui um elo de realimentação (*feedback loop*) na operação do neurônio. Uma vez tendo sido aplicados todos os N vetores $\underline{x}(i)$ à entrada do neurônio e tendo sido executados todos os N ajustes através do algoritmo **A**, repete-se novamente as etapas 1 e 2 até que $e(n)$ seja suficientemente pequeno, onde $e(n)$ é o valor do sinal de erro e em um instante n qualquer da operação do filtro.

Uma vez que o neurônio é linear, a saída $y(i)$ é idêntica ao potencial de ativação (≡nível de ativação) $v(i)$, isto é,

$$y(i) = v(i) = \sum_{k=0}^{M-1} w_k(i) x_k(i) \tag{3.3}$$

onde $w_k(i)$ é o valor da k-ésima transmitância sináptica medida no instante discreto i. Em forma vetorial, podemos expressar $y(i)$ como o produto interno entre os vetores $\underline{x}(i)$ e $\underline{w}(i)$, conforme segue:

$$y(i) = \underline{x}^T(i)\underline{w}(i) \tag{3.4}$$

onde

$$\underline{w}(i) = [w_0(i) \quad w_1(i) \quad \cdots \quad w_{M-1}(i)]^T \tag{3.5}$$

A saída $y(i)$ do neurônio é comparada com a saída $d(i)$ do sistema desconhecido **Γ** no instante discreto *i*. Tipicamente, a comparação é estabelecida pela diferença entre $d(i)$ e $y(i)$, portanto o processo de comparação define o sinal de erro $e(i)$ dado por

$$e(i) = d(i) - y(i) \tag{3.6}$$

Observe de (3.4) e (3.6) que o sinal de erro $e(i)$ depende do vetor $\underline{w}(i)$. Note também que $\underline{w}(i)$ é o parâmetro livre do neurônio que será sucessivamente ajustado pelo algoritmo **A,** objetivando minimizar $e(i)$. Portanto, para que se possa medir a ineficiência do processo de ajuste de $\underline{w}$, e, em função disto adotar as correções necessárias, é útil definir uma função $\mathrm{J}(e)$ (ou $\mathrm{J}(\underline{w})$, já que *e* depende de $\underline{w}$) que defina da maneira o mais inequívoca possível o "grau de incompetência" do neurônio em aproximar sua saída $y(i)$ de $d(i)$.

A função $\mathrm{J}(\underline{w})$, cujo valor resultante é uma grandeza escalar real, é denominada de <u>função de custo</u>. A definição de $\mathrm{J}(\underline{w})$ deve ser tal que meça o quanto o processo de ajuste está sendo incapaz de reduzir o erro $e(i)$ entre $d(i)$ e $y(i)$. Por exemplo, uma popular definição de J é $\mathrm{J} = \mathrm{J}(e) = \frac{1}{2}e^2$. Em especial, o algoritmo **A** e a função de custo J idealmente devem ser tais que $\mathrm{J}(\underline{w}(n+1)) < \mathrm{J}(\underline{w}(n))$, onde *n* é um instante qualquer do processo de ajuste.

3.1.1 O Processo de Minimização da Função de Custo

Consideraremos neste estudo o denominado Algoritmo de Descida Mais Íngreme (SD – *Steepest Descent*), por ser um dos mais utilizados, e de baixo custo computacional.

Existem, no entanto, outros algoritmos, como o Método de Newton e o Método de Gauss-Newton, que são descritos em [4].

No algoritmo SD os sucessivos ajustes aplicados à $\underline{w}$ estão na direção da descida mais íngreme da superfície $S = \mathrm{H}(w_0, w_1, \ldots, w_{M-1})$ formada pelos valores escalares H do conjunto imagem de $\mathrm{J}(\underline{w})$ em função do domínio M-dimensional $\underline{w} = [w_0 \quad w_1 \quad \cdots \quad w_{M-1}]^T$, isto é, $\mathrm{H} = \mathrm{J}(\underline{w})$. Em outras palavras, os sucessivos ajustes aplicados à $\underline{w}$ estão na direção oposta do vetor gradiente $\underline{\nabla}\,\mathrm{J}(\underline{w})$ da superfície formada por $\mathrm{J}(\underline{w})$.

Uma interpretação intuitiva do método SD é imaginarmos um observador míope que enxergue apenas a distância de um passo ao seu redor, caminhando sobre a superfície $\mathrm{J}(\underline{w})$, e cujo objetivo é chegar ao ponto de cota mínima de $\mathrm{J}(\underline{w})$ o mais rapidamente possível. No instante n o observador, localizado na coordenada $\underline{w}(n)$, olha ao redor e localiza a direção $\underline{\nabla}\,\mathrm{J}(\underline{w}(n))$ de subida mais íngreme em $\mathrm{J}(\underline{w})$. A seguir o observador dá um passo na direção contrária à $\underline{\nabla}\,\mathrm{J}(\underline{w}(n))$ de tamanho proporcional à declividade $\left|\underline{\nabla}\,\mathrm{J}(\underline{w}(n))\right|$ encontrada na coordenada $\underline{w}(n)$ e desloca-se para a nova coordenada $\underline{w}(n+1)$. Supondo que não existam mínimos locais (buracos e/ou depressões) na superfície $\mathrm{J}(\underline{w})$ de diâmetro algo maior que o passo do observador, o mesmo atingirá a cota mínima $\mathrm{J}(\underline{w}^*)$ na coordenada $\underline{w}^*$ após repetir este procedimento um número suficiente de vezes.

Formalmente, o algoritmo SD é descrito por

$$\underline{w}(n+1) = \underline{w}(n) - \eta\,\underline{\nabla}\,\mathrm{J}(\underline{w}(n)) \tag{3.7}$$

onde $\eta > 0$ é chamado passo de adaptação (*stepsize*) ou razão de aprendizado (*learning rate*).

Para a função de custo $\mathrm{J}(n) = \mathrm{J}(\underline{w}(n)) = \frac{1}{2}e^2(n)$, a superfície $\mathrm{J}(\underline{w})$ é um parabolóide M+1-dimensional (i.e., uma "tigela" em $\Re^{M+1}$, não necessariamente de boca circular), e, portanto, apresenta um mínimo global mas não apresenta mínimos locais (qualquer função quadrática possui um e somente um mínimo). Por isto, para esta função de custo, o

algoritmo SD converge para $\underline{w}^*$ de modo lento mas seguro desde que η não seja demasiadamente grande (caso em que o observador míope pularia fora da "tigela").

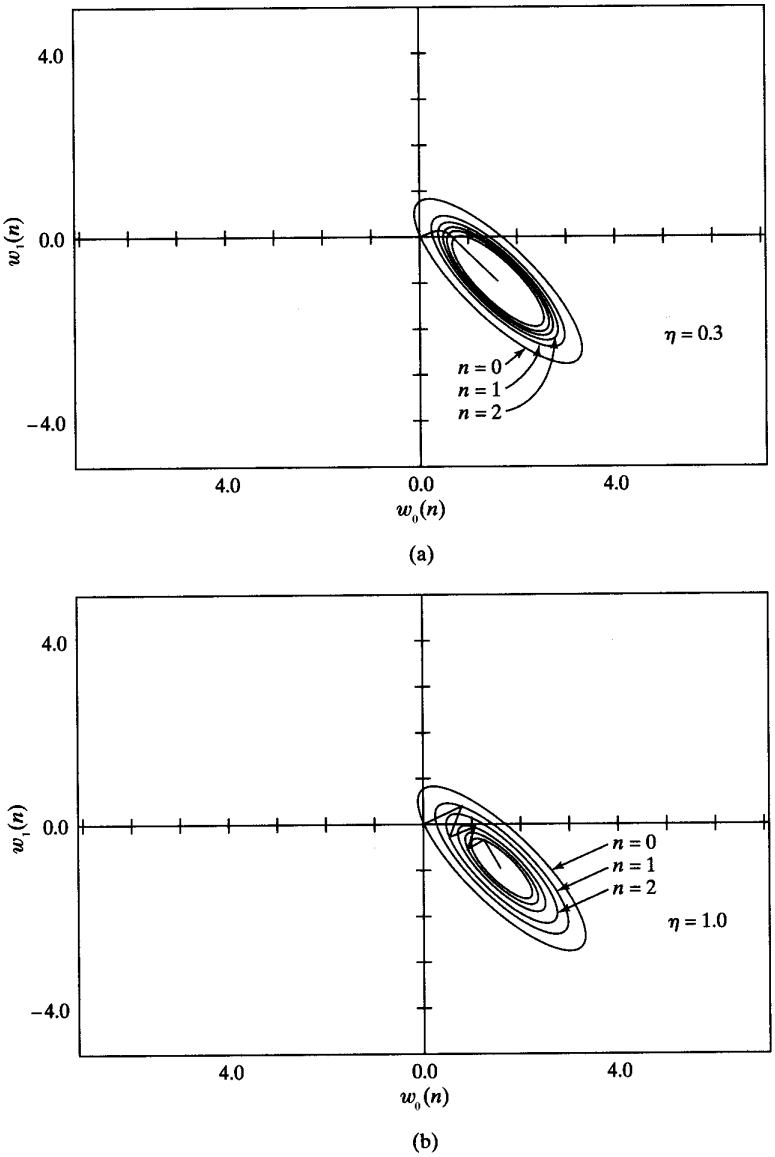

Figura 3.2: Trajetória do método de Descida Mais Íngreme (*steepest descent*) em um espaço bi-dimensional para dois valores diferentes de parâmetros razão de aprendizado: (a) $\eta = 0.3$, (b) $\eta = 1.0$. As coordenadas w_1 e w_2 são elementos do vetor de pesos $\underline{w}$.

➡ É importante observar que o passo de adaptação η tem profunda influência na trajetória do "observador míope" até a convergência para $\underline{w}^*$, e, não raro, o valor de η é alterado convenientemente ao longo do processo de minimização de J para que η se adeqüe às exigências da coordenada instantânea da trajetória. Para filtros cuja função de custo é $\mathrm{J}(n) = \mathrm{J}(\underline{w}(n)) = \frac{1}{2}e^2(n)$ são válidas as seguintes observações:

- Para η pequeno, a resposta transiente do algoritmo SD é super-amortecida (*overdamped*) e a trajetória percorrida por $\underline{w}(n)$ é uma curva suave em $\Re^M$, conforme mostrado na Figura 3.2(a).

- Para η grande, a resposta transiente do algoritmo SD é sub-amortecida (*underdamped*) e a trajetória percorrida por $\underline{w}(n)$ é uma curva em zig-zag (oscilatória) em $\Re^M$, conforme mostrado na Figura 3.2(b).

- Para η acima de um determinado valor crítico, o algoritmo SD torna-se instável e termina divergindo.

3.2 O Algoritmo LMS

O Algoritmo LMS (*Least Mean Square*) procura minimizar uma função de custo J definida por $\mathrm{J} = \mathrm{J}(e) = \frac{1}{2}e^2$ com base nos valores instantâneos da mesma, isto é,

$$\mathrm{J} = \mathrm{J}(e(n)) = \frac{1}{2}e^2(n) \tag{3.8}$$

onde $e(n)$ é o sinal de erro medido em um instante *n* qualquer do processo de minimização de J.

Nota: Diferentemente do algoritmo **LMS**, apenas como exemplo comparativo, o algoritmo **RLS** (*Recursive Least Squares*) baseia-se em uma função de custo J definida por uma soma ponderada do erro quadrático $e^2(n)$ do instante atual *n* com os erros quadráticos ocorridos anteriormente a *n*, isto é, $\mathrm{J}(n)=\beta_0 e^2(n)+\beta_1 e^2(n-1)+\beta_2 e^2(n-2)+\cdots$, onde $0<\beta_k\leq 1$ são os coeficientes de ponderação. Os coeficientes β_k são tais que $\beta_k>\beta_{k+1}$, de forma que erros ocorridos em um passado distante sejam "esquecidos" por J objetivando minimizar sua influência sobre ela. Assim, se o conjunto χ de $(\underline{x}(n),d(n))\in\chi$ (entradas, saídas desejadas) não for um processo estacionário (i.e., os parâmetros estatísticos de χ variam com o tempo), o "esquecer do passado" auxilia a melhorar a velocidade de convergência. No entanto, como é fácil perceber, o custo computacional do algoritmo RLS é maior que o do algoritmo LMS, o que o torna inadequado para certas aplicações que requeiram alta velocidade de processamento, como por exemplo, em equalização de canal para um *link* de microondas com alta taxa de transmissão.

O gradiente $\underline{\nabla}\,\mathrm{J}(\underline{w}(n))$ da superfície $\mathrm{J}(n)=\mathrm{J}(\underline{w}(n))=\frac{1}{2}e^2(n)$ no instante *n* é obtido através da variação de $\mathrm{J}(\underline{w}(n))$ em resposta a uma variação infinitesimal na coordenada $\underline{w}(n)$, isto é,

$$\underline{\nabla}\,\mathrm{J}(\underline{w}(n))=\frac{\partial\,\mathrm{J}(\underline{w}(n))}{\partial\,\underline{w}(n)} \tag{3.9}$$

mas, visto que $\mathrm{J}(\underline{w}(n))=\frac{1}{2}e^2(n)$, temos

$$\underline{\nabla}\,\mathrm{J}(\underline{w}(n))=\frac{\partial\left\{\frac{1}{2}e^2(n)\right\}}{\partial\,\underline{w}(n)}=e(n)\frac{\partial e(n)}{\partial\,\underline{w}(n)} \tag{3.10}$$

Vimos que

$$e(n)=d(n)-y(n)=d(n)-\underline{x}^T(n)\underline{w}(n) \tag{3.11}$$

e como $d(n)$ não depende de $\underline{w}(n)$, temos que

$$\frac{\partial e(n)}{\partial\,\underline{w}(n)}=-\underline{x}(n) \tag{3.12}$$

De (3.12) e (3.10) temos

$$\underline{\nabla}\,\mathrm{J}(\underline{w}(n)) = -e(n)\underline{x}(n) \tag{3.13}$$

e, substituindo (3.13) em (3.7), encontraremos para $\underline{w}(n+1)$,

$$\underline{w}(n+1) = \underline{w}(n) + \eta\, e(n)\underline{x}(n) \tag{3.14}$$

onde η é o passo de adaptação ou razão de aprendizado.

A Equação (3.14) define o processo de ajuste do vetor de pesos $\underline{w}$ de um neurônio linear objetivando minimizar J através do algoritmo LMS.

É instrutivo comparar os algoritmos SD e LMS utilizando a alegoria do "observador míope", cujo objetivo é atingir o mais rapidamente possível a coordenada $\underline{w}^*$, a qual define a coordenada da cota mínima da superfície $\mathrm{J}(\underline{w})$.

No algoritmo SD, o observador localizado na coordenada $\underline{w}(n)$ olha ao redor, localiza a direção $\underline{\nabla}\,\mathrm{J}(\underline{w}(n))$ de subida mais íngreme na superfície $\mathrm{J}(\underline{w})$ e dá um passo em direção contrária à ela, conforme já discutido. O ato de "olhar ao redor" significa matematicamente ter o conhecimento da

- matriz de correlação **R** do conjunto de vetores de entrada $\underline{x}$, e
- do vetor de correlação cruzada $\underline{p}$ entre o conjunto de saídas desejadas *d* e o conjunto de vetores $\underline{x}$.

O conhecimento destes elementos é necessário porque, no algoritmo SD, o gradiente no instante n é calculado através de $\underline{\nabla}\,\mathrm{J}(\underline{w}(n)) = -2\underline{p} + 2\mathbf{R}\underline{w}(n)$ (conforme S. Haykin em *Adaptive Filter Theory*, referenciado em [3]).

No algoritmo LMS, o observador não é somente míope como também é totalmente cego. O observador, localizado na coordenada $\underline{w}(n)$, consegue "observar" sua posição relativa porque segura em sua mão um cordão infinitamente elástico cuja outra extremidade encontra-se fixa na coordenada $\underline{w}^*$. A cada instante *n*, o observador dá um passo na direção em que ele percebe a maior redução na tensão τ do elástico (diminuição do valor absoluto

do erro $e(n)$), com tamanho de passo proporcional à redução de τ. Como não existem mínimos locais na superfície $J(\underline{w})$, porque ela é quadrática, o observador se aproximará da cota mínima $J(\underline{w}^*)$ na coordenada $\underline{w}^*$ após repetir este procedimento um número suficiente de vezes. Note que, como o tamanho e sentido do passo do observador dependem da redução na tensão τ do elástico, quando o observador chegar próximo à coordenada $\underline{w}^*$ ele ficará eternamente "pulando" sobre e ao redor dela a menos que, por um raro golpe de sorte, a coordenada resultante do último passo do observador coincida com $\underline{w}^*$ (situação que ocorrerá para um valor bastante particular e crítico de η e para uma bastante particular coordenada inicial $\underline{w}^0$ da trajetória do observador). Apesar disto, o algoritmo LMS tem a vantagem de não necessitar do conhecimento de **R** e de $\underline{p}$, ao contrário do algoritmo SD.

Em suma, no algoritmo SD o vetor $\underline{w}(n)$ segue uma trajetória bem definida no espaço de pesos sinápticos, para um valor não excessivo de η. Em contraste, no algoritmo LMS o vetor $\underline{w}(n)$ segue uma trajetória aleatória, especialmente nas vizinhanças de $\underline{w}^*$.

A Tabela 3.1 apresenta um sumário do procedimento do algoritmo LMS.

Conjunto de Treino:	Sinal de entrada em forma vetorial = $\underline{x}(n)$ Sinal resposta desejada escalar = $d(n)$
Parâmetro ajustável pelo usuário:	η
Inicialização do vetor $\underline{w}$:	$\underline{w}^0 = \underline{w}(0) = \underline{0}$
Procedimento Computacional:	Para $n = 0,1,\cdots$ computar $e(n) = d(n) - \underline{x}^T(n)\underline{w}(n)$ $\underline{w}(n+1) = \underline{w}(n) + \eta\, e(n)\underline{x}(n)$

Tabela 3.1: Sumário do algoritmo LMS. O Procedimento Computacional é executado até que a média de $e^2(n)$ atinja um patamar suficientemente baixo para a solução do problema em questão ou estabilize em um valor constante.

3.2.1 Considerações quanto à Convergência do LMS

Combinando (3.11) e (3.14) podemos expressar a evolução do vetor $\underline{w}$ através de

$$\begin{aligned}\underline{w}(n+1) &= \underline{w}(n)+\eta\underline{x}(n)[d(n)-\underline{x}^T(n)\underline{w}(n)] = \underline{w}(n)+\eta\underline{x}(n)d(n)-\eta\underline{x}(n)\underline{x}^T(n)\underline{w}(n) = \\ &= \left[\mathbf{I}-\eta\underline{x}(n)\underline{x}^T(n)\right]\underline{w}(n)+\eta\underline{x}(n)d(n)\end{aligned} \tag{3.15}$$

onde **I** é a matriz identidade.

O processo de ajuste do vetor $\underline{w}(n)$ é uma operação iterativa indexada pela variável inteira *n*. Em função disto, podemos então reconhecer que o valor de $\underline{w}(n+1)$ será o valor de $\underline{w}(n)$ quando a variável *n* for incrementada de 1 na próxima iteração. Em outras palavras, o valor obtido de (3.15) para $\underline{w}(n+1)$ no instante *n* é armazenado em uma posição de memória para ser utilizado como o valor de $\underline{w}(n)$ em (3.15) no instante $n+1$.

No domínio z, esta relação entre $\underline{w}(n)$ e $\underline{w}(n+1)$ é expressa por

$$\mathbf{Z}\{\underline{w}(n)\} = z^{-1}\mathbf{Z}\{\underline{w}(n+1)\} \tag{3.16}$$

onde $\mathbf{Z}\{\cdot\}$ é o operador Transformada Z e z^{-1} é o operador atraso unitário (*unit delay*). A partir das equações (3.15) e (3.16) podemos representar o algoritmo LMS através do grafo de fluxo de sinal mostrado na Figura 3.3.

A Figura 3.3 revela que o algoritmo LMS pode ser considerado como um sistema realimentado, já que existem dois *loops* de *feedback*, um superior e outro inferior. A presença de realimentação exerce um profundo impacto no comportamento do algoritmo LMS, visto que os parâmetros dos *loops* definem a estabilidade da trajetória dos estados de qualquer sistema realimentado.

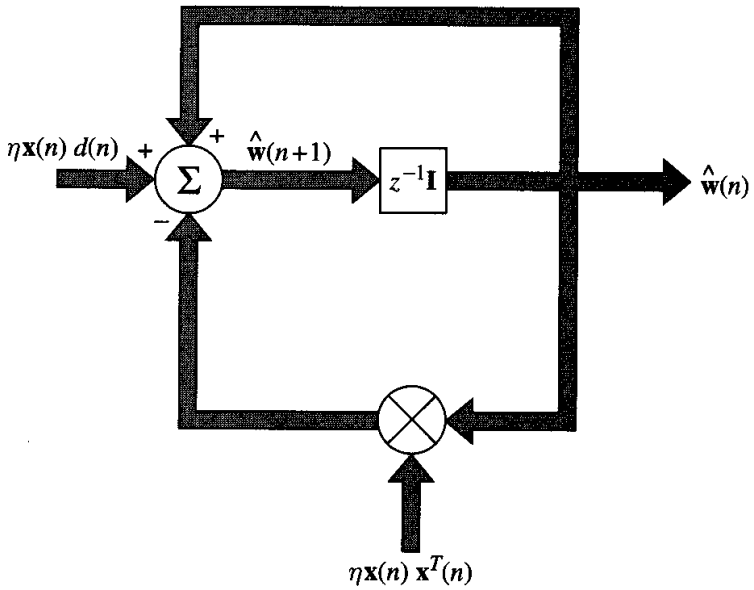

Figura 3.3: Grafo de fluxo de sinal representativo do algoritmo LMS.

Observe na Figura 3.3 que o *loop* inferior impõe variabilidade ao comportamento do LMS, particularmente porque a transmitância deste *loop* é controlada pela matriz $\eta\,\underline{x}(n)\underline{x}^{T}(n)$, a qual depende do vetor de entrada $\underline{x}(n)$, com parâmetro de controle dado pela razão de aprendizado η. Infere-se, portanto, que a estabilidade da trajetória de $\underline{w}(n)$ é influenciada pelas características estatísticas do conjunto de vetores de entrada $\underline{x}$ e pelo valor da razão de aprendizado η.

Expressando este fato de outro modo, para um dado conjunto de vetores de entrada $\underline{x}$ deve-se escolher η tal que a trajetória de $\underline{w}(n)$ seja estável o suficiente para permitir a convergência para as vizinhanças de $\underline{w}^{*}$. A convergência da trajetória de $\underline{w}(n)$ para as vizinhanças de $\underline{w}^{*}$ é caracterizada por uma constância no valor médio de $e^{2}(n)$.

Como regra geral, a razão de aprendizado η deve obedecer à relação:

$$0 < \eta < \frac{2}{\frac{1}{N}\sum_{i=0}^{N-1} \underline{x}^T(i)\underline{x}(i)} \tag{3.17}$$

onde N é o número total de vetores no conjunto de vetores de entrada $\underline{x}$.

3.3 O Perceptron

Enquanto que o algoritmo LMS, descrito na Seção 3.2, é construído em torno de um neurônio linear, o Perceptron é construído ao redor de um neurônio não-linear, que é o neurônio descrito pelo modelo de McCulloch-Pitts.

Conforme vimos no Capítulo 1, este modelo de neurônio consiste de um combinador linear seguido de um limitador, desempenhando a função signum, conforme mostrado na Figura 3.4.

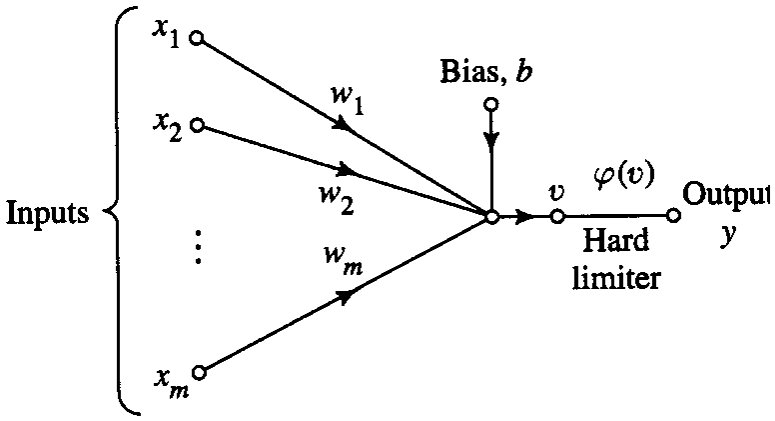

Figura 3.4: Grafo de fluxo de sinal do Perceptron.

O nó somador do modelo neural mostrado na Figura 3.4 computa uma combinação linear das entradas aplicadas a suas sinapses com os pesos sinápticos associados, e também incorpora uma polarização externamente aplicada. A soma resultante (que é o potencial de ativação v) é aplicada a um limitador, representado por $\varphi(v)$, que implementa a função

signum. Desta forma, o neurônio produz uma saída igual a (+1) se a entrada do limitador é positiva, e (-1) se é negativa.

No grafo de fluxo de sinal mostrado na Figura 3.4, os pesos sinápticos do Perceptron são denotados por $w_1, w_2, \cdots, w_m$. De forma correspondente, as entradas aplicadas ao Perceptron são denotadas por $x_1, x_2, \cdots, x_m$. A polarização (ou *bias*) é aplicada externamente e denotada por *b*. A partir do modelo verifica-se que a entrada do limitador, ou o potencial de ativação v do neurônio, é:

$$v = \sum_{i=1}^{m} w_i x_i = b \tag{3.18}$$

O objetivo do Perceptron é classificar corretamente o conjunto de estímulos externos aplicados $x_1, x_2, \cdots, x_m$ em uma de duas classes, C_1 ou C_2. A regra de decisão para a classificação é atribuir o ponto representado pelas entradas $x_1, x_2, \cdots, x_m$ à classe C_1 se a saída y do Perceptron for (+1) e à classe C_2 se for (-1).

Para compreender o comportamento de um classificador de padrões, costuma-se plotar um mapa das regiões de decisão no espaço de sinal *m*-dimensional gerado pelas m variáveis de entrada $x_1, x_2, \cdots, x_m$. Na forma mais simples do Perceptron há duas regiões de decisão separadas por um hiperplano definido por

$$\sum_{i=1}^{m} w_i x_i + b = 0 \tag{3.19}$$

conforme ilustrado na Figura 3.5 para o caso de duas variáveis de entrada x_1 e x_2, para as quais o limite de decisão assume a forma de uma linha reta. Um ponto (x_1, x_2) que esteja acima da linha limítrofe é atribuído à classe C_1 e um ponto (x_1, x_2) que esteja abaixo da linha limítrofe é atribuído à classe C_2. O efeito da polarização (ou *bias*) é simplesmente deslocar o limite de decisão para longe da origem.

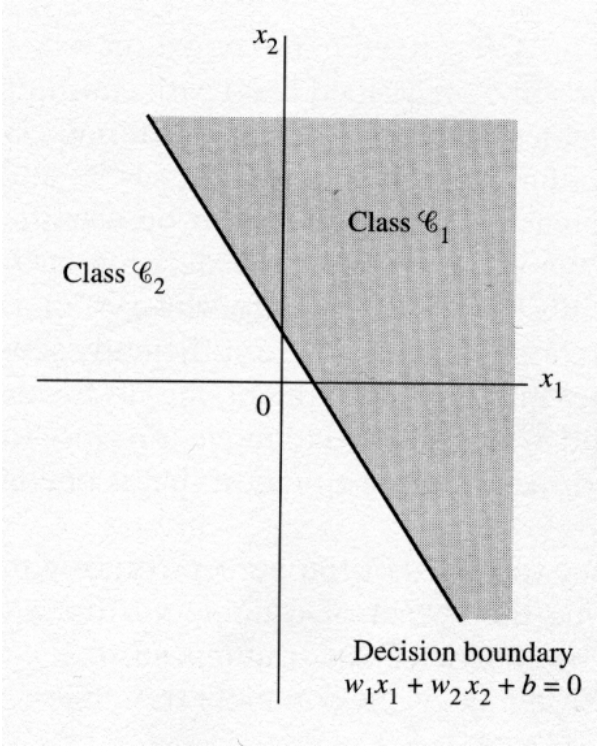

Figura 3.5: Ilustração do hiperplano (neste caso, uma linha reta) como limite de decisão para um problema de classificação de padrões de duas classes (bi-dimensional).

Os pesos sinápticos $w_1, w_2, \cdots, w_m$ do Perceptron podem ser adaptados de iteração a iteração. Para a adaptação pode-se usar a regra de correção de erro conhecida como algoritmo de convergência do Perceptron.

3.3.1 O Teorema de Convergência do Perceptron

Para derivar o algoritmo de aprendizagem por correção de erro para o Perceptron, consideremos o modelo do grafo de fluxo de sinal modificado mostrado na Figura 3.6. neste modelo, equivalente ao da Figura 3.4, a polarização $b(n)$ é tratada como um peso sináptico cuja entrada é fixa em +1 (conforme vimos no Capítulo 1).

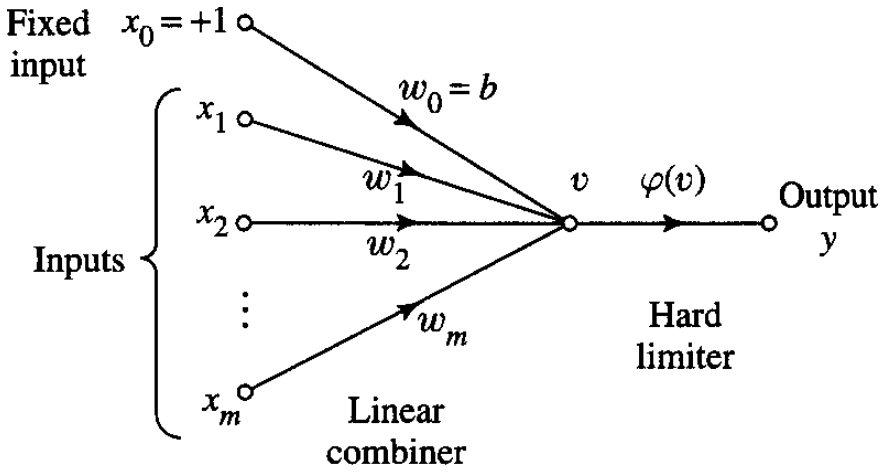

Figura 3.6: Grafo de fluxo de sinal equivalente do Perceptron (a dependência do tempo foi omitida por questões de clareza).

Pode-se, então, definir o vetor de entrada $[(m+1)\times 1]$-dimensional como

$$\underline{x}(n) = [+1 \quad x_1(n) \quad x_2(n) \quad \cdots \quad x_m(n)]^T \tag{3.20}$$

onde *n* denota o passo da iteração do algoritmo. De forma correspondente, podemos definir o vetor de pesos $[(m+1)\times 1]$-dimensional como

$$\underline{w}(n) = [b(n) \quad w_1(n) \quad w_2(n) \cdots \quad w_m(n)]^T \tag{3.21}$$

da mesma forma, a saída do combinador linear pode ser escrita na forma compacta,

$$v(n) = \sum_{i=0}^{m} w_i(n)\, x_i(n) = \underline{w}^T(n)\,\underline{x}(n) \tag{3.22}$$

onde $w_0(n)$ representa a polarização $b(n)$. Para *n* fixo, a equação $\underline{w}^T\,\underline{x} = 0$, plotada em um espaço *m*-dimensional (e para algum *bias* prescrito) com coordenadas $x_1, x_2, \cdots, x_m$, define um hiperplano como a superfície de decisão entre duas diferentes classes de entradas (vide Figura 3.5).

Para que o Perceptron funcione adequadamente, as duas classes C_1 e C_2 precisam ser linearmente separáveis, o que significa dizer que os padrões a serem classificados devem ser suficientemente separados uns dos outros para garantir que a superfície de decisão consista de um hiperplano.

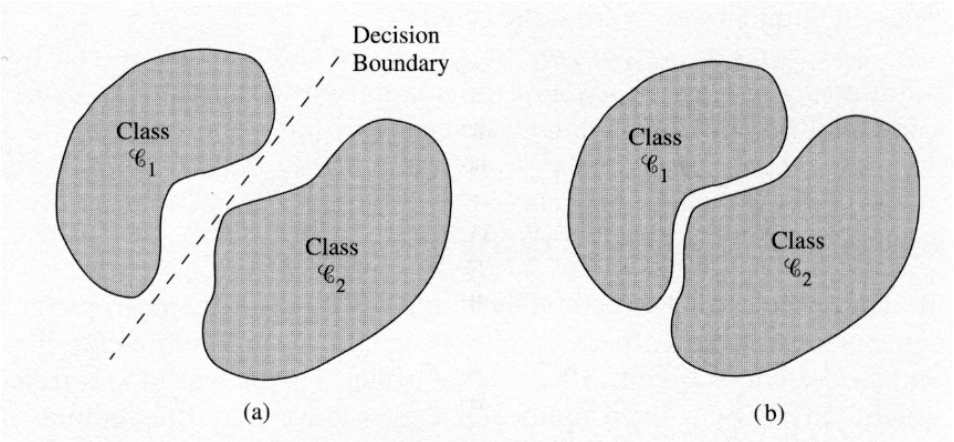

Figura 3.7: (a) Um par de padrões linearmente separáveis. (b) Um par de padrões não-linearmente separáveis.

Este requerimento é ilustrado na Figura 3.7 para o caso de um Perceptron bi-dimensional. Na Figura 3.7(a), as duas classes C_1 e C_2 são suficientemente separáveis uma da outra, de tal forma que é possível desenhar um hiperplano (neste caso uma linha reta) como limite de decisão. Se, entretanto, as duas classes C_1 e C_2 tivessem se aproximado tanto uma da outra (como mostrado na Figura 3.7(b)) teriam se tornado não-linearmente separáveis, uma situação que está além da capacidade computacional do Perceptron.

Suponhamos então que as variáveis de entrada do Perceptron tenham se originado de duas classes linearmente separáveis. Seja X_1 o sub-conjunto de vetores de treino $\underline{x}_1(1), \underline{x}_1(2), \ldots$ que pertençam à classe C_1, e seja X_2 o sub-conjunto de vetores de treino $\underline{x}_2(1), \underline{x}_2(2), \ldots$ que pertençam à classe C_2. A união de X_1 e X_2 é o conjunto de treino completo X.

Dados os conjuntos de vetores X_1 e X_2 para treinar o classificador, o processo de treino envolve o ajuste do vetor de pesos $\underline{w}$, de tal forma que as duas classes C_1 e C_2 sejam linearmente separáveis. Ou seja, exista um vetor de pesos $\underline{w}$ tal que possamos afirmar:

$$\begin{aligned} &\underline{w}^T \underline{x} > 0 \quad \text{para cada vetor de entrada } \underline{x} \text{ pertencente à classe } C_1 \\ &\underline{w}^T \underline{x} \leq 0 \quad \text{para cada vetor de entrada } \underline{x} \text{ pertencente à classe } C_2 \end{aligned} \tag{3.23}$$

Observe que, na segunda linha da Equação (23), foi escolhido arbitrariamente que o vetor de entrada $\underline{x}$ pertencesse à classe C_2 se $\underline{w}^T \underline{x} = 0$.

Dados os sub-conjuntos de vetores de treino X_1 e X_2, o problema de treinamento para o Perceptron elementar é, então, encontrar um vetor de pesos $\underline{w}$ tal que as duas inigualdades da Equação (23) sejam satisfeitas.

O algoritmo para adaptar o vetor de pesos do Perceptron elementar pode ser agora formulado conforme segue:

1. Se o *n-ésimo* membro do conjunto de treino, $\underline{x}(n)$, é corretamente classificado pelo vetor de pesos $\underline{w}(n)$ computado na *n-ésima* iteração do algoritmo, nenhuma correção é feita no vetor de pesos do Perceptron de acordo com a regra:

$$\begin{aligned} &\underline{w}(n+1) = \underline{w}(n) \quad \text{se} \quad \underline{w}^T(n)\underline{x}(n) > 0 \quad \text{e} \quad \underline{x}(n) \text{ pertence à classe } C_1 \\ &\underline{w}(n+1) = \underline{w}(n) \quad \text{se} \quad \underline{w}^T(n)\underline{x}(n) \leq 0 \quad \text{e} \quad \underline{x}(n) \text{ pertence à classe } C_2 \end{aligned} \tag{3.24}$$

2. Em caso contrário, o vetor de pesos do Perceptron é atualizado de acordo com a regra:

$$\begin{aligned} &\underline{w}(n+1) = \underline{w}(n) - \eta(n)\underline{x}(n) \quad \text{se} \quad \underline{w}^T(n)\underline{x}(n) > 0 \quad \text{e} \quad \underline{x}(n) \text{ pertence à classe } C_2 \\ &\underline{w}(n+1) = \underline{w}(n) + \eta(n)\underline{x}(n) \quad \text{se} \quad \underline{w}^T(n)\underline{x}(n) \leq 0 \quad \text{e} \quad \underline{x}(n) \text{ pertence à classe } C_1 \end{aligned} \tag{3.25}$$

onde o parâmetro razão de aprendizado $\eta(n)$ controla o ajuste aplicado ao vetor de pesos na iteração *n*.

Para o caso particular em que $\eta(n) = \eta > 0$ (onde η é uma constante independente do número da iteração *n*), temos uma regra de adaptação de incrementos fixos para o Perceptron.

Desejamos primeiro provar a convergência de uma regra de adaptação de incrementos fixos, com $\eta = 1$. Claramente o valor de η não é importante, enquanto for positivo. Um valor de $\eta \neq 1$ simplesmente escala os vetores sem afetar sua separabilidade. O caso de uma razão de aprendizado $\eta(n)$ variável será considerado posteriormente.

Convergência da Regra de Adaptação de Incremento Fixo (Razão de Aprendizado η Fixa)

A prova é apresentada para a condição inicial $\underline{w}(0) = \underline{0}$.

Suponha que $\underline{w}^T(n)\underline{x}(n) < 0$ para $n = 1, 2, \ldots$, e o vetor de entrada $\underline{x}(n)$ pertença ao sub-conjunto X_1.

Ou seja, nesta condição, o Perceptron classificou de forma incorreta os vetores $\underline{x}(1), \underline{x}(2), \ldots$, desde que a segunda condição (dada pela Equação 23) foi violada.

Então, com a constante $\eta(n) = 1$, podemos usar a segunda linha da Equação 3.25 para escrever

$$\underline{w}(n+1) = \underline{w}(n) + \underline{x}(n) \qquad \text{para } \underline{x}(n) \text{ pertencente à classe } C_1 \tag{3.26}$$

Dada a condição inicial $\underline{w}(0) = \underline{0}$, podemos iterativamente resolver esta equação para $\underline{w}(n+1)$, obtendo o resultado

$$\underline{w}(n+1) = \underline{x}(1) + \underline{x}(2) + \cdots + \underline{x}(n) \tag{3.27}$$

Desde que as classes C_1 e C_2 são assumidas linearmente separáveis, existe uma solução $\underline{w}_0$ para a qual $\underline{w}^T \underline{x}(n) > 0$ para os vetores $\underline{x}(1), \underline{x}(2), \ldots \underline{x}(n)$ pertencentes ao sub-conjunto X_1. Para uma solução fixa $\underline{w}_0$, podemos então definir um número positivo α como

$$\alpha = \min_{\underline{x}(n) \in X_1} \underline{w}_0^T \underline{x}(n) \tag{3.28}$$

Multiplicando ambos os lados da Equação (3.27) pelo vetor linha $\underline{w}_0^T$ teremos

$$\underline{w}_0^T \underline{w}(n+1) = \underline{w}_0^T \underline{x}(1) + \underline{w}_0^T \underline{x}(2) + \cdots + \underline{w}_0^T \underline{x}(n) \tag{3.29}$$

De acordo com a definição dada na Equação (3.28), teremos

$$\underline{w}_0^T \underline{w}(n+1) \geq n\alpha \tag{3.30}$$

Dados dois vetores $\underline{w}_0$ e $\underline{w}(n+1)$, a inigualdade de Cauchy-Schwarz, afirma que

$$\|\underline{w}_0\|^2 \, \|\underline{w}(n+1)\|^2 \geq \left[\underline{w}_0^T \underline{w}(n+1)\right]^2 \tag{3.31}$$

onde $\|\cdot\|$ denota a norma Euclidiana do vetor argumento, e o produto interno $\underline{w}_0^T \underline{w}(n+1)$ é uma quantidade escalar.

A partir da Equação (3.30) observa-se que $\left[\underline{w}_0^T \underline{w}(n+1)\right]^2$ é igual ou maior que $n^2\alpha^2$. A partir da Equação (3.31) observa-se que $\|\underline{w}_0\|^2 \, \|\underline{w}(n+1)\|^2$ é igual ou maior que $\left[\underline{w}_0^T \underline{w}(n+1)\right]^2$. Segue, portanto, que

$$\|\underline{w}_0\|^2 \, \|\underline{w}(n+1)\|^2 \geq n^2\alpha^2 \tag{3.32}$$

ou equivalentemente,

$$\|\underline{w}(n+1)\|^2 \geq \frac{n^2\alpha^2}{\|\underline{w}_0\|^2} \tag{3.33}$$

Seguindo, agora, uma nova rota de desenvolvimento, rescreveremos a Equação (3.26) sob a forma

$$\underline{w}(k+1) = \underline{w}(k) + \underline{x}(k) \qquad \text{para } k = 1, \ldots, n \text{ e } \underline{x}(k) \in X_1 \tag{3.34}$$

Tomando o quadrado da norma Euclidiana de ambos os lados da Equação (3.34), obteremos

$$\|\underline{w}(k+1)\|^2 = \|\underline{w}(k)\|^2 + \|\underline{x}(k)\|^2 + 2\underline{w}^T(k)\underline{x}(k) \tag{3.35}$$

Mas, tendo sido assumido que o Perceptron classifica incorretamente um vetor de entrada $\underline{x}(k)$ pertencente ao sub-conjunto X_1, teremos que $\underline{w}^T(k)\underline{x}(k) < 0$. Portanto, pode-se deduzir, a partir da Equação (3.35) que

$$\|\underline{w}(k+1)\|^2 \leq \|\underline{w}(k)\|^2 + \|\underline{x}(k)\|^2 \tag{3.36}$$

ou, de forma equivalente,

$$\|\underline{w}(k+1)\|^2 - \|\underline{w}(k)\|^2 \leq \|\underline{x}(k)\|^2, \; k = 1, \ldots, n \tag{3.37}$$

Adicionando estas inigualdades para $k = 1, \ldots, n$ e invocando a condição inicial assumida $\underline{w}(0) = \underline{0}$, chegamos à seguinte inigualdade:

$$\|\underline{w}(n+1)\|^2 \leq \sum_{k=1}^{n} \|\underline{x}(k)\|^2 \leq n\beta \tag{3.38}$$

onde

$$\beta = \max_{\underline{x}(k) \in X_1} \|\underline{x}(k)\|^2 \tag{3.39}$$

A Equação (3.38) afirma que o quadrado da a norma Euclidiana do vetor de pesos $\underline{w}(n+1)$ cresce no máximo linearmente com o número de iterações *n*.

O segundo resultado da Equação (3.38) está claramente em conflito com o resultado anterior da Equação (3.33) para valores de *n* suficientemente grandes.

Na verdade, pode-se afirmar que *n* não pode ser maior do que algum valor $n_{\max}$ para o qual as Equações (3.33) e (3.38) são ambas satisfeitas com o sinal de igualdade. Ou seja, $n_{\max}$ é a solução da equação

$$\frac{n_{\max}^2 \alpha^2}{\left\| \underline{w}_0 \right\|^2} = n_{\max} \beta \tag{3.40}$$

Resolvendo para $n_{\max}$, dado um vetor solução $\underline{w}_0$,encontraremos

$$n_{\max} = \frac{\beta \left\| \underline{w}_0 \right\|^2}{\alpha^2} \tag{3.41}$$

Temos, assim, provado que para $\eta(n)=1$ para todo *n*, $\underline{w}(0)=\underline{0}$ e dado que existe um vetor solução $\underline{w}_0$, a regra para adaptação dos pesos sinápticos do Perceptron deve terminar após, no máximo, $n_{\max}$ iterações. Note também a partir das Equações (3.28), (3.39) e (3.41) que não há uma única solução para $\underline{w}_0$ ou $n_{\max}$.

Podemos, agora, afirmar que o teorema da convergência da regra de adaptação de incremento fixo para o Perceptron como segue:

- Sejam os sub-conjuntos de vetores de treino X_1 e X_2 linearmente separáveis;
- Sejam as entradas apresentadas ao Perceptron originadas destes dois sub-conjuntos;

$\Rightarrow$ O Perceptron converge após algumas iterações n_0, no sentido de que $\underline{w}(n_0)=\underline{w}(n_0+1)=\underline{w}(n_0+2)=\cdots$ é um vetor solução para $n_0 \le n_{\max}$.

Convergência da Regra de Adaptação de Incremento Variável (Razão de Aprendizado $\eta(n)$ Variável)

Consideremos agora o procedimento de correção de erro absoluto para a adaptação de um Perceptron de uma única camada, para o qual $\eta(n)$ é variável. Em particular, seja $\eta(n)$ o menor inteiro para o qual

$$\eta(n)\underline{x}^T(n)\underline{x}(n) > \left|\underline{w}^T(n)\underline{x}(n)\right| \tag{3.42}$$

Com este procedimento podemos afirmar que: se o produto interno $\underline{w}^T(n)\underline{x}(n)$ na iteração *n* tem um sinal incorreto, então $\underline{w}^T(n+1)\underline{x}(n)$ na iteração $n+1$ pode ter o sinal correto. Isto sugere que, se $\underline{w}^T(n)\underline{x}(n)$ tem um sinal incorreto, podemos modificar a seqüência de treino na iteração $n+1$ fazendo $x(n+1)=x(n)$.

Em outras palavras, cada padrão é apresentado repetidamente ao Perceptron até que o padrão seja classificado corretamente.

Note também que o uso de um valor inicial $\underline{w}(0)$ diferente de zero meramente resulta no decréscimo ou acréscimo do número de iterações requeridas para convergência dependendo de como $\underline{w}(0)$ se relaciona com a solução $\underline{w}_0$. Indiferentemente do valor atribuído a $\underline{w}(0)$, o Perceptron tem sua convergência garantida.

Na Tabela 3.2 é apresentado um sumário do algoritmo de convergência do Perceptron. O símbolo $\text{sgn}(\cdot)$, usado no passo 3 da tabela para computar a resposta atual do Perceptron, representa a função signum, descrita no Capítulo 1 deste texto.

Podemos, então, expressar a resposta quantizada $y(n)$ do Perceptron na forma compacta:

$$y(n) = \text{sgn}\left(\underline{w}^T(n)\underline{x}(n)\right) \tag{3.43}$$

	Variáveis e Parâmetros: Vetor de entrada $\underline{x}(n)$ de dimensão $[(m+1)\times 1]$; $\underline{x}(n)=[+1\ x_1(n)\ x_2(n)\cdots\ x_m(n)]^T$ Vetor de pesos $\underline{w}(n)$ de dimensão $[(m+1)\times 1]$; $\underline{w}(n)=[b(\mathrm{n})\ w_1(n)\ w_2(n)\cdots\ w_m(n)]^T$ *Bias* = $b(n)$ Resposta atual (quantizada) = $y(n)$ Resposta desejada = $d(n)$ Parâmetro razão de aprendizado (constante positiva <1) = η
1.	**Inicialização:** Faça $\underline{w}(0)=\underline{0}$. Então execute as etapas seguintes do algoritmo para os instantes de tempo $n=1,2,\ldots$
2.	**Ativação:** No instante de tempo *n* ative o Perceptron aplicando o vetor de entrada $\underline{x}(n)$ e a resposta desejada $d(n)$.
3.	**Cômputo da Resposta Atual:** Compute a resposta atual do Perceptron através de $y(n)=\mathrm{sgn}\left(\underline{w}^T(n)\underline{x}(n)\right)$, onde $\mathrm{sgn}(\cdot)$ é a função signum.
4.	**Adaptação do Vetor de Pesos:** Atualize o vetor de pesos do Perceptron através de $\underline{w}(n+1)=\underline{w}(n)+\eta[d(n)-y(n)]\underline{x}(n)$ onde $d(n)=\begin{cases}+1 & \text{se } \underline{x}(n) \text{ pertence à classe } C_1\\ -1 & \text{se } \underline{x}(n) \text{ pertence à classe } C_2\end{cases}$
5.	**Continuação:** Fazer $n=n+1$ e voltar à etapa 2.

Tabela 3.2 Sumário do Algoritmo de Convergência do Perceptron

Note que o vetor de entrada $\underline{x}(n)$ é um vetor $[(m+1)\times 1]$, cujo primeiro elemento é fixo em (+1) ao longo de todo o processo computacional. De forma correspondente, o vetor de pesos $\underline{w}(n)$ é um vetor $[(m+1)\times 1]$, cujo primeiro elemento é igual ao *bias* $b(n)$. Outro ponto a salientar na Tabela 3.2 é a introdução de uma resposta desejada quantizada $d(n)$, definida por

$$d(n)=\begin{cases}+1 & \text{se } \underline{x}(n) \text{ pertence à classe } C_1\\ -1 & \text{se } \underline{x}(n) \text{ pertence à classe } C_2\end{cases} \tag{3.44}$$

Então, a adaptação do vetor de pesos $\underline{w}(n)$ pode ser sumarizada na forma da regra de aprendizado por correção de erro:

$$\underline{w}(n+1)=\underline{w}(n)+\eta[d(n)-y(n)]\underline{x}(n) \tag{3.45}$$

onde η é o parâmetro razão de aprendizado, e a diferença $d(n)-y(n)$ representa um sinal de erro. O parâmetro razão de aprendizado é uma constante positiva limitada ao intervalo $0<\eta\leq 1$. Na escolha de um valor para η, dentro deste intervalo, é preciso considerar dois requisitos conflitantes:

- Manter a estabilidade da trajetória (estimativas estáveis para os pesos) requer valores pequenos para η;
- Adaptação rápida com respeito às mudanças reais nas distribuições subjacentes do processo responsável pela geração do vetor de entrada $\underline{x}$ requer valores grandes para η.

3.4 Referências Bibliográficas do Capítulo 3:

[1] M. H. Hassoun, *Fundamentals of Artificial Neural Networks*, MIT Press, Massachusetts, 1995.

[2] R. D. Strum e D. E. Kirk, *First Principles of Discrete Systems and Digital Signal Processing*, Addison-Wesley, 1989.

[3] S. Haykin, *Adaptive Filter Theory*, 3rd ed., Prentice Hall, Upper Saddle River, New Jersey, 1996.

[4] S. Haykin, *Neural Networks*, 2nd ed., Prentice Hall, Upper Saddle River, New Jersey, 1999.

[5] Z.L.Kovács, *Redes Neurais Artificiais*, Editora Acadêmica São Paulo, São Paulo, 1996.