

# CAPÍTULO II

## SISTEMAS LINEARES E REDES NEURAIIS

### II.1- Introdução:

Apresentamos aqui um modelo de Redes Neurais voltado para a solução de sistemas lineares do tipo  $Ax=b$  baseado na minimização de uma função quadrática de erro que mede a variação entre os vetores  $Ax$  e  $b$ . Esta função será associada à função de energia utilizada em nossa rede neuronal. Para sistemas compatíveis esse erro será zero e uma solução exata será obtida. Sistemas lineares incompatíveis implicarão em uma função de erro estritamente positiva. Como veremos a função de erro será convexa e limitada inferiormente por zero e isto nos garantirá estabilidade de nossa rede já que um erro mínimo poderá sempre ser obtido.

Será apresentado um modelo de duas camadas altamente paralelizável que evolui continuamente no tempo e segue dinamicamente a trajetória obtida pelo gradiente da função de erro.

Como discutido em [1], apesar do número de neurônios crescer linearmente com a dimensão do sistema a velocidade de convergência implementada em uma máquina analógica dependerá principalmente da relação entre o maior e menor autovalor da hessiana  $A^T A$  da função de erro.

### II.2 - Função de Erro e Método Gradiente:

Seja  $Ax=b$  um sistema linear onde  $x \in \mathfrak{R}^n$ ,  $A \in \mathfrak{R}^{m \times n}$  e  $b \in \mathfrak{R}^m$ . Como o problema poderá admitir uma ou infinitas soluções ou mesmo não admitir nenhuma solução, é natural reescrever o problema utilizando uma função de erro definida por  $f(x) = \frac{1}{2} \|Ax - b\|^2$  onde  $\|\cdot\|$  representa uma norma qualquer desejada [2]. Em nosso caso trabalharemos com a norma euclideana obtendo dessa forma um problema de mínimos quadrados. Resumindo, podemos escrever o problema da seguinte forma:

$$\begin{aligned} \min f(x) &= \frac{1}{2} (Ax - b)^T (Ax - b) \\ \text{s.a} \quad x &\in \mathfrak{R}^n \end{aligned} \quad (1)$$

Como a função é convexa esperamos que  $\nabla f(x) = A^T (Ax - b)$  seja zero em uma solução ótima  $x^*$  (condição necessária de otimalidade). Note que se  $f(x^*)=0$  então  $x^*$  é uma solução ótima de nosso sistema linear, se  $f(x^*)>0$  o sistema será incompatível.

Podemos resolver o problema (1) acima utilizando o método gradiente (Luenberger [2]), que toma inicialmente um ponto  $x^0$  e gera uma sequência de aproximações  $x^1, x^2, \dots$  onde  $x^{k+1} = x^k - \alpha \nabla f(x)$ , sendo  $-\nabla f(x)$  a direção de descida e  $\alpha > 0$  o tamanho do passo. Quando  $\alpha \nabla f(x)$  é aproximadamente zero estaremos próximo da solução do problema. Calculando o gradiente mais explicitamente temos  $d_k = -\alpha \nabla f(x^k) = -\alpha A^T (Ax^k - b)$  ou ainda:

$$d_j^k = -\alpha \sum_{i=1}^m a_{ij} \left( \sum_{l=1}^n a_{il} x_l^k - b_i \right) \quad \text{onde } j=1,2,\dots,n. \quad (2)$$

A matriz hessiana  $\nabla^2 f(x) = A^T A$  é semi-definida positiva em todos os pontos de  $\mathfrak{R}^n$  (logo  $f(\cdot)$  é convexa). Calculando o maior e menor autovalores  $B$  e  $b$  da hessiana em  $x^*$  (solução ótima) temos que  $r = (B - b) / (B + b)^2$  nos dá uma estimativa para a velocidade de convergência da sequência gerada pelo método gradiente em uma vizinhança de  $x^*$ . Quanto menor o valor de  $r$  mais rápida será a convergência. Note que se  $r$  é pequeno  $B$  e  $b$  estarão muito próximos, assim

as curvas de nível se aproximam de bolas em uma vizinhança de  $x^*$  e o método gradiente será muito mais rápido (Figura II.1). Para maiores detalhes vide Luemberger [2].

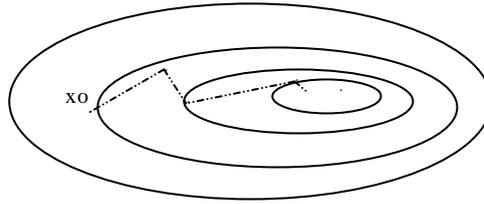


Figura II.1: Iterações do Método Gradiente

Esta análise dos autovalores (como mencionado em [1]), também será verdadeira para implementações em máquinas analógicas via Redes Neurais.

### II3 - A Rede Neural:

Na abordagem analógica a obtenção da sequência de pontos não é obtida discretamente como acima. Supondo um instante  $t$  qualquer temos que a equação diferencial  $x'(t) = -\alpha A^T A x(t) + A^T b$  juntamente com uma condição inicial  $x^o = x(t_0)$  define a trajetória a ser percorrida a partir de  $x^o$ . Se desejarmos obter a solução correspondente a um instante  $t_1$  basta calcular  $x(t_1) = \int_0^{t_1} x'(t) dt$ .

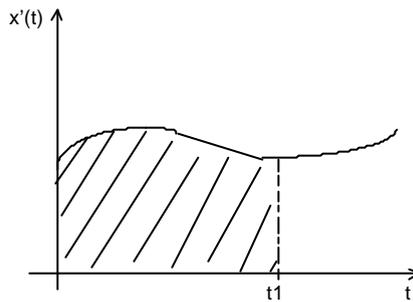


Figura II.2: Obtenção do Estado de Ativação do Neurônio

O modelo de neurônio para sistemas lineares não é exatamente igual ao modelo de Hopfield como descrito anteriormente. Trabalhamos com uma função de ativação linear definida por  $x_j(t) = \alpha u_j(t)$  onde  $x_j(t)$  representa o estado de ativação e  $u_j(t)$  a energia potencial do neurônio  $j$ . Faremos uma associação da variável  $x$  de nosso sistema linear com  $x_j(t)$  para  $j=1, \dots, n$  (estados de ativação dos neurônios). Note que como teremos soluções do sistema em  $\mathfrak{R}^n$  seria inconveniente trabalharmos com os estados de ativação de neurônios restritos a um hipercubo contido em  $\mathfrak{R}^n$  (vide modelo de Hopfield).

A regra de propagação será idêntica ao modelo de Hopfield, ou seja:

$$C_j \frac{dx_j}{dt} = \sum_{i=1}^m w_{ji} x_i - \frac{x_j}{R_j} \quad (3)$$

Note que substituímos  $u_j(t)$  por  $x_j(t)$  já que trabalhamos com a função de ativação linear  $x_j(t) = \alpha u_j(t)$ . A constante  $\alpha$  será incorporada no valor de  $C_j$  e  $R_j$ .

Para fins de simplificação supomos que  $C_j = C$  e  $R_j = R$  para todo neurônio  $j$ .

A idéias apresentadas em [1] consistem basicamente em uma associação do gradiente da função de erro com a regra de propagação de nossos neurônios. Assim, comparando as equações (2) e (3) temos:

$$\frac{dx_j}{dt} = \frac{1}{C} \sum_{i=1}^m w_{ji} x_i - \frac{x_j}{R}$$

onde  $x_i = \sum_{j=1}^n a_{ij} x_j - b_i$ ,  $w_{ij} = -a_{ij}$  e  $\mathbf{a} = 1/C$ . Logo:

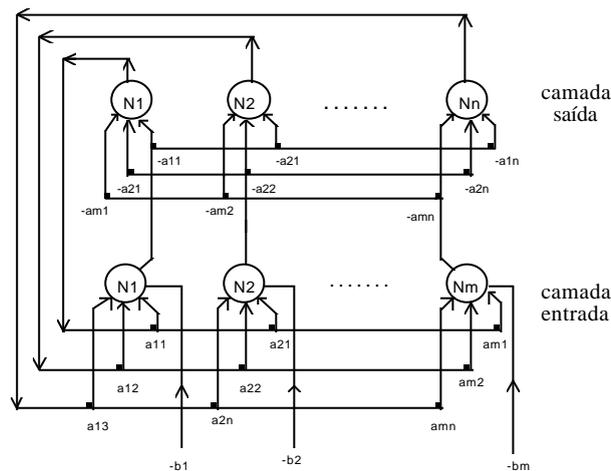
$$\frac{dx_j}{dt} = -\frac{1}{C} \sum_{i=1}^m a_{ij} \left( \sum_{l=1}^n a_{il} x_l - b_i \right) - \frac{x_j}{R}$$

Para  $R \gg 0$  obtemos:

$$\frac{dx_j}{dt} \approx -\frac{1}{C} \sum_{i=1}^m a_{ij} \left( \sum_{l=1}^n a_{il} x_l - b_i \right) \quad (4)$$

onde  $j=1,2,\dots,n$ .

A partir de (4) definimos uma topologia de rede para nosso problema (figura II.3) onde trabalhamos com uma rede de duas camadas. A solução de nosso sistema será dada obviamente pelos neurônios da camada de saída.



**Figura II.3: Topologia da Rede**

Note que  $x_i = \sum_{j=1}^n a_{ij} x_j - b_i$  para  $i=1,2,\dots,m$  representa os estados de ativação da camada de entrada e é uma função das sinapses  $w_{ij} = a_{ij}$  e estados de ativação  $x_j$  da camada de saída. Observe também que  $-b_i$  onde  $i=1,\dots,m$  são adicionados na camada de entrada.

Os estados de ativação da camada de entrada são utilizados para a obtenção de uma outra camada oculta com  $n$  neurônios e com estados de ativação dados por  $\frac{dx_j}{dt}$ ,  $j=1,\dots,n$ .

Em uma simulação em máquina digital não teríamos o tempo variando continuamente. Dessa forma um incremento fixo de tempo  $\Delta t$  seria necessário para obtenção da camada de saída. Ou seja:

$$x_j^{k+1} = \Delta t \cdot \frac{dx_j^k}{dt} + x_j^k \quad \text{onde } j = 1, \dots, n$$

Note que  $x_j^k$  e  $x_j^{k+1}$  representam os estados de ativação dos neurônios da camada de entrada e saída respectivamente.

Abaixo apresentamos um algoritmo onde fazemos uma simulação de nossa rede neural:

**Algoritmo:** (Sistemas Lineares-Simulação);

{Constantes:  $C = 10^{-5}$ ,  $\mathbf{e} = 10^{-2}$ ,  $\Delta t = 10^{-6}$ }

**Início**

- leia(A,b);

- para j:=1 até n faça

$x_2(j) \leftarrow 0;$  {obtem ponto inicial}

$\frac{dx_j}{dt} \leftarrow 0;$  {inicializa derivada}

**fim para;**

- repita

- para j:=1 até n faça

$x_2(j) \leftarrow \Delta t \cdot \frac{dx_2(j)}{dt} + x_2(j);$  {obtem camada de saída}

- para i:=1 até m faça

$x_1(i) \leftarrow \sum_{j=1}^n a(i,j) \cdot x_2(j) - b(i);$  {obtem camada de entrada}

- para j:=1 até n faça

$\frac{dx_2(j)}{dt} = -\frac{1}{C} \sum_{i=1}^m a(i,j) \cdot x_1(i);$  {obtem camada oculta}

- calcula:  $\left\| \frac{dx_2(j)}{dt} \right\|;$

- Até que  $\left\| \frac{dx_2(j)}{dt} \right\| < \mathbf{e}$  {condição de parada}

- Imprime camada de saída;

**fim algoritmo.**

As constantes expressas acima são sugeridas com base em simulações apresentadas em [1], aonde são apresentados problemas com até 150 linhas e 180 colunas. Obviamente as dimensões do problema terão influência negativa em simulações de máquinas digitais.

Inquestionavelmente o modelo aqui descrito tem inúmeras aplicações em outros problemas da matemática de grande interesse. Por exemplo, a inversa de uma matriz quadrada pode ser calculada resolvendo-se um conjunto independente de sistemas lineares. Implementando-se cada um desses sistemas em processadores distintos de uma máquina paralela teríamos a inversão de uma matriz tão rapidamente quanto a solução de um sistema linear simples.