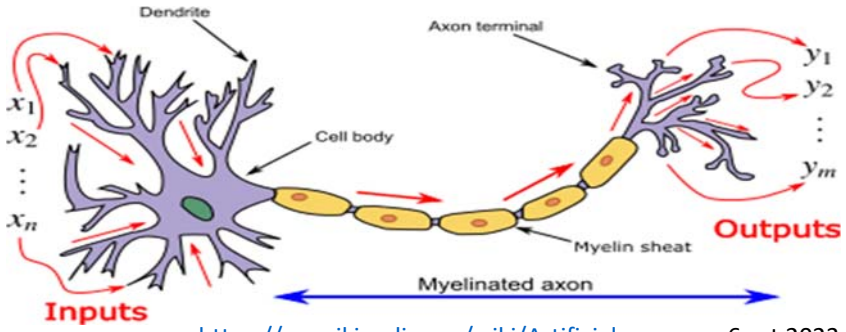
# Capítulo 4. Neurónios, Camadas, Redes Neuronais (Artificiais)

## 4.1 Introdução

Os neurónios biológicos têm uma configuração ilustrada na Fig. 4.1.1.




https://en.wikipedia.org/wiki/Artificial_neuron 6 set 2023

Figura 4.1.1. Um neurónio biológico.

A parte mais importante é o corpo celular (cell body, também conhecido por soma), que processa a informação. Ele recebe, através das dendrites, sinais em microvolts (entradas) de outros neurónios vizinhos. Quando no corpo celular a soma dessas tensões ultrapassa um certo patamar, ele dispara um impulso elétrico, também na ordem dos microvolts, que se propaga ao longo do axónio (Myelin sheat), até chegar aos terminais deste (saídas), que por sua vez entram em contacto com outros neurónios. Os pontos de contacto, quer os das entradas, quer os das saídas, chamam-se sinapses e podem ter intensidades variáveis. Esta estrutura foi descoberta em 1863 por Otto Friedrich Karl Deiters (1834-1863), e batizada com o nome de neurónio em 1891, por Heinrich Wilhelm von Waldeyer (https://cerebromente.org.br/n17/history/neurons2_p.htm 6 set 2023).

Na década de 1940, o neurologista Donald H. Hebb desenvolveu uma teoria de aprendizagem neuronal que descreveu na sua obra fundadora *The Organization of Behaviour* em 1949. Também nessa mesma década Warren McCulloch (neurocientista) e Walter Pitts (logiciano) inventaram (em 1943) o primeiro modelo matemático de um neurónio biológico capaz de executar as operações lógicas boleanas elementares AND, OR, NOR, NOT ((https://towardsdatascience.com/mcculloch-pitts-model-5fdf65ac5dd1 6 set 2023).

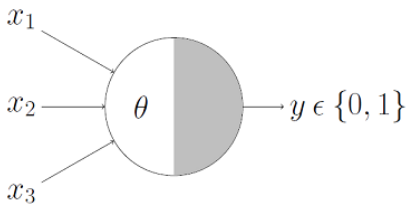



Figura 4.1.2. Modelo de McCulloch-Pitt (M-P)
(https://towardsdatascience.com/mcculloch-pitts-model-5fdf65ac5dd1 6 set 2023)

O termo **percetrão** foi proposto em 1958 por Frank Rosenblatt (no Cornell Aeronautical Laboratory) e generaliza o modelo de M-P para funções não booleanas e introduz pesos nas entradas que modelizam as sinapses dos neurónios biológicos.

Temos assim a figura 4.1.3 que ilustra as analogias entre os neurónios biológicos e os artificiais.

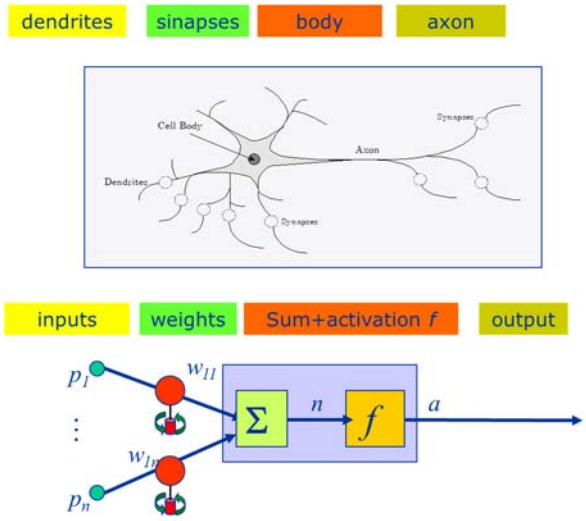



Figura 4.1.3. Analogia entre os neurónios biológicos e os artificiais.

Estava aberto o caminho para as redes neuronais artificiais que são compostas por neurónios (artificiais) associados em série e em paralelo.

O neurónio artificial tem entradas $p_1$, $p_2$, $p_n$. Cada entrada é multiplicada por um peso $w_{ij}$ , como as sinapses fazem no biológico. Depois faz-se a soma das entradas pesadas e o resultado passa por uma função *f*, chamada função de ativação, que produz a saída *a* do neurónio. Os pesos $w_{ij}$ são variáveis (como se fosse um botão rotativo) e é essa possibilidade de variar os pesos de acordo com os interesses do utilizador que está na base do potencial das redes neuronais, desde as mais simples às mais avançadas

E a analogia entre as redes neuronais biológicas e as artificiais é ilustrada na Fig. 4.1.4.

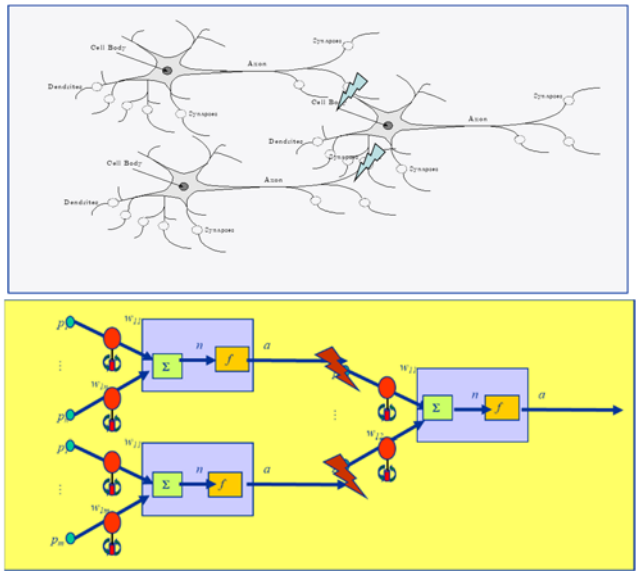


Figura 4.1.4. Rede neuronal com 3 neurónios, biológica e artificial.

Neste caso muito simples temos dois neurónios na primeira camada (camada de entrada) e um na segunda (camada de saída). Ao nosso dispor estão 6 pesos para variamos como quisermos, isto é, temos 6 graus de liberdade para "sintonizarmos" uma função que mapeia as 4 entradas na saída. Com 6 graus de liberdade é possível construir muitos mapeamentos.

Podemos aumentar o número de camadas, por exemplo para três, como na Fig. 4.1.5. Aqui cada neurónio, para além das entradas anteriores, tem uma entrada adicional, constante, também com um peso variável, chamada polarização. Veremos a sua utilidade.

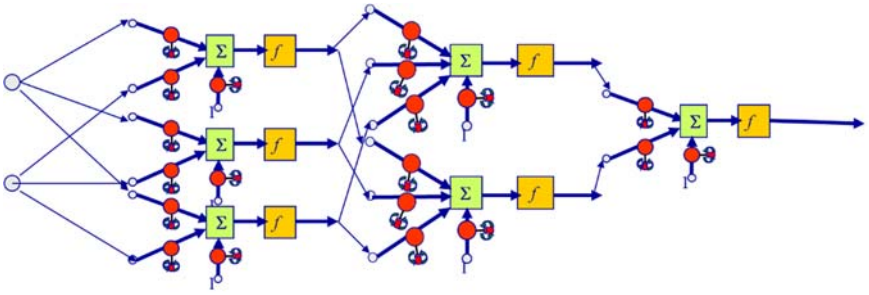


Figura 4. 1.5. Uma rede com três camadas.

A saída de um neurónio de uma camada é uma entrada de todos os neurónios da camada seguinte (diz-se em inglês que a rede é *fully connected* ou *dense*). Temos 20 graus de liberdade. Quer dizer que esta rede pode implementar funções, lineares ou não lineares, de 20 parâmetros, o que é já impressionante. Agora se imaginarmos dezenas ou centenas de neurónios por camada, alcançamos um potencial extraordinário. Está mesmo matematicamente provado que uma rede com três camadas pode aproximar qualquer tipo de função com qualquer precisão, desde que as duas primeiras camadas tenham o número de neurónios adequado. Colocando a rede dentro de uma caixa opaca, ficando os botões de fora, obtemos a Figura 4.1.6.

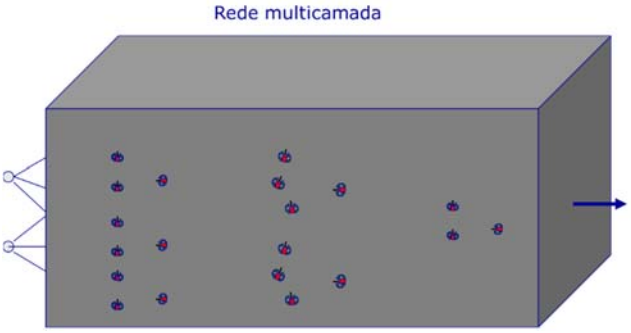
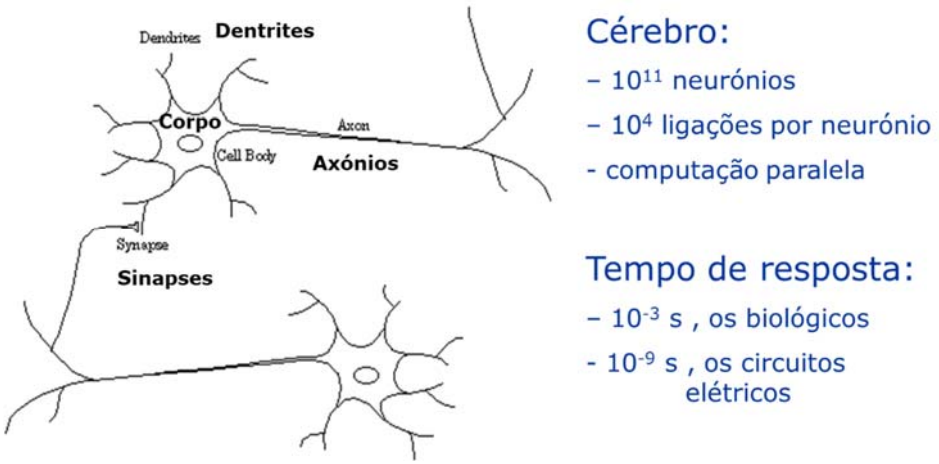



Figura 4.1.6. A rede da Fig. 4.1.5 "encaixotada". V

Vêm-se as entradas (no caso duas) e as saídas (no caso uma). A saída visível é da terceira camada, chamada camada de saída. As saídas das duas primeiras camadas não são visíveis, e por isso essas camadas chamam-se escondidas (*hidden*). Assim acontece com qualquer número de camadas: são todas escondidas exceto a última. Os pesos reguláveis nas paredes da caixa ilustram os nossos graus de liberdade para uma dada rede.

A Fig. 4.1.6 ilustra também uma característica das redes neuronais: elas são modelos de caixa negra (*black box*), isto é interpretamos as entradas e as saídas mas o que se passa lá dentro não tem interpretabilidade.

Se compararmos o poder computacional do cérebro com o de um computador digital (onde os neurónios articiciais são programados) temoS a Figura 4.1.7




Gigura 4.1.7. Comparação dos neurónios biológicos com os circuitos eletrónicos.

A vantagem do cérebro reside no número de neurónios e suas interligações. A vantagem dos neurónios eletrónicos reside no tempo de resposta (velocidade computacional).

## 4.2 Modelo matemático

Para uma melhor compreensão, vejamos o desenvolvimento matemático passo a passo.

### 4.2.1 Um só neurónio com uma entrada (de ora em diante referimo-nos apenas a neurónios e redes artificiais)

Consideremos s Fig. 4.2.1.

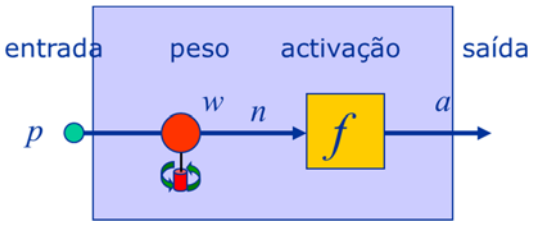



Figura 4.2.1. Um neurónio com uma entrada

Nela temos um neurónio com uma função de ativação *f*, e uma entrada *p* pesada pelo peso *w* (de *weight)*, resultando no argumento *n* de *f*. O peso é variável e a saída *a* varia quando o peso *w* varia, para a mesma entrada. Temos

$$n = wp$$
$$a = f(n) = f(wp) \qquad (4.1)$$

Se a entrada é nula, a saída será nula:

$$p = 0 \quad \Leftrightarrow \quad a = 0$$
$$\text{se } f(0) = 0$$

Ora podem existir situações em que queremos uma saída do neurónio não nula quando a entrada é nula. Para isso introduz-se uma entrada adicional constante e igual a 1, pesada por um peso *b* (de *bias*), chamada polarização. Temos assim a Fig. 4.2.2.

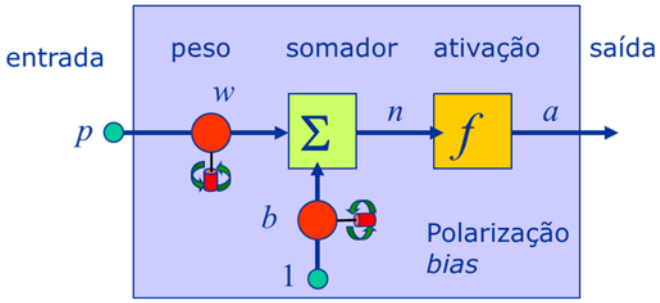



Figura 4.2.2. Um neurónio com uma entrada adicional de polarização.

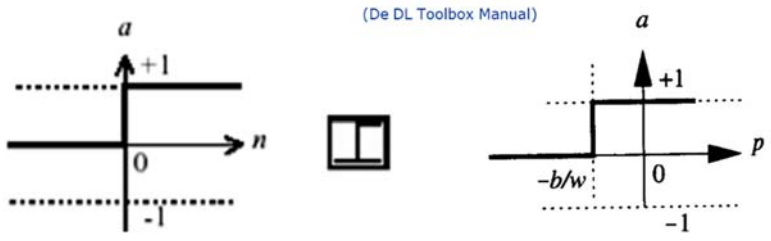
$$n = w \times p + b \times 1 = [w\ b] \times \begin{bmatrix} p \\ 1 \end{bmatrix} = w'\,p'$$
$$a = f(n) = f(wp + b) = f(w'\,p') \qquad (4.2)$$

N expressão (4.2) *p'* é um vetor de dois componentes, a entrada variável *p* e a entrada fixa de polarização *1,* e *w'* é um vetor composto pelo peso *w* e pela polarização *b*. A escrita vetorial é mais elegante e uniformiza as expressões matemáticas (4.1) e (4.2).

### 4.2.2. Funções de ativação

Poderemos usar muitos tipos de funções de ativação, conforme a aplicação em causa.

(i) Os primeiros neurónios, por analogia com os biológicos, tinham uma função de ativação binária, chamada *hardlim* por fixar um patamar rígido que leva à transição da saída de 0 para 1.




$a = \text{hardlim}(n)$ $\qquad$ $a = \text{hardlim}(wp + b)$

Figura 4.2.3. Função de ativação binária. Se *n*<0, a saída *a* é nula; se *n*>=0, a saída *a* é 1.

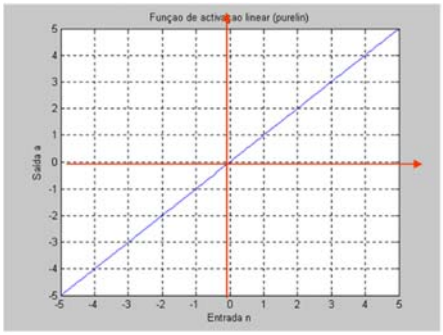
Na primeira parte da Fig. 4.2.3, não há polarização e portanto *n=p*, a transição dá-se em *n*=0 .Na segunda parte há polarização *b* e teremos *n=wp+b* (ver Fig. 4.7); para que a transição se dê em *n*=0, a entrada deverá ser tal que *n=wp+b*=0, ou seja *wp+b=0,ou seja,* resolvendo para *p,*

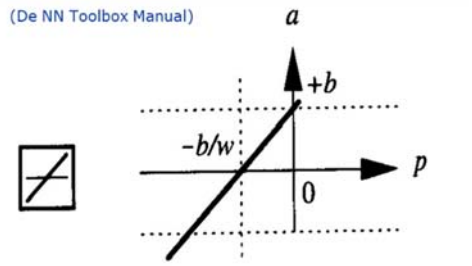
$wp=-b \Rightarrow p=-b/w$

como indicado na Figura. Assim a polarização provoca o deslocamento horizontal da função de ativação: se a polarização for positiva, para a esquerda, se negativa, para a direita (em relação à origem do sistema de eixos).

(ii) Funções de ativação contínuas lineares

As funções lineares representam-se no plano por retas com uma certa inclinação (dada pela derivada da função). Se a derivada é um (inclinação de 45º), chama-se *purline*.



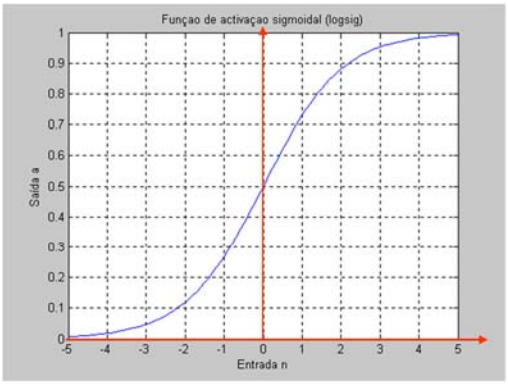

$$a = \text{purelin}(n)$$

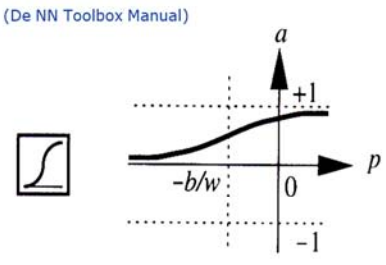
$$a = \text{purelin}(wp + b)$$

Figura 4.2.4. Função de ativação linear. Sem polarização dá a primeira parte da figura *a=n=p.* A transição de negativo para positivo dá-se em *n=0.* Na segunda parte, a polarização desloca horizontalmente a função de ativação, dado que *n= wp+b* e a transição de negativo para positivo dá-se de igual modo em *n=0.*

(iii) Funções de ativação contínuas não lineares

Para dar maior generalidade às redes neuronais, podem-se usar funções de ativação contínuas não lineares, diferenciáveis. Por exemplo a sigmóide unipolar ilustrada n Fig. 4.2.4.



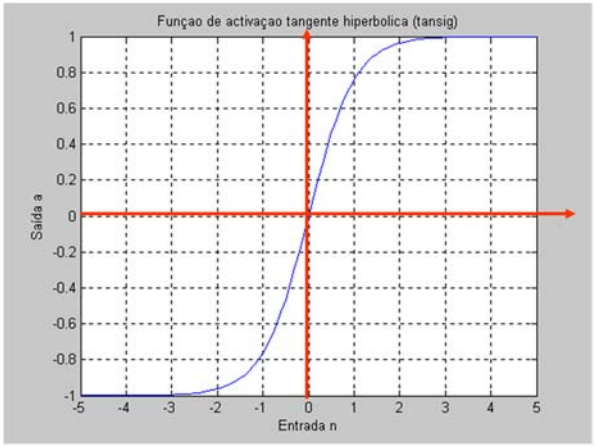

$$a = \text{logsig}(n) = \frac{1}{1+e^{-n}}$$

$$a = logsig(wp+b)$$

$$a = \frac{1}{1+e^{-(wp+b)}}$$

Figura 4.2.4. Sigmoide unipolar (curva a azul, é sempre positiva). Sem polarização para uma entrada nula dá uma saída 0,5. Com polarização, tal como anteriormente a função desloca-se horizontalmente e vale 0,5 em *-b/w*.

Podemos também usar a tangente hiperbólica, ou sigmoide bipolar, ilustrada na Fig. 4.2.5.



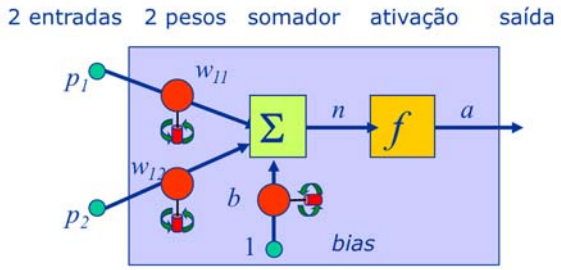

$$a = \text{tansig}(n) = \frac{e^{n} - e^{-n}}{e^{n} + e^{-n}}, \quad \text{no Matlab}$$

Figura 4.2.5. Tangente hiperbólica, ou sigmoide bipolar (por ser negativa ou positiva). Com polarização *b* desloca-se horizontalmente de modo que cruza o eixo horizontal no ponto *-b/w*.

As funções contínuas devem ser diferenciáveis para que a sua derivada exista em todos os pontos, o que tem a ver com o treino das redes neuronais que veremos posteriormente.

4.2.2 Neurónio com mais do que uma entrada

Seja agora um neurónio com duas entradas e polarização, Fig. 4.2.6.

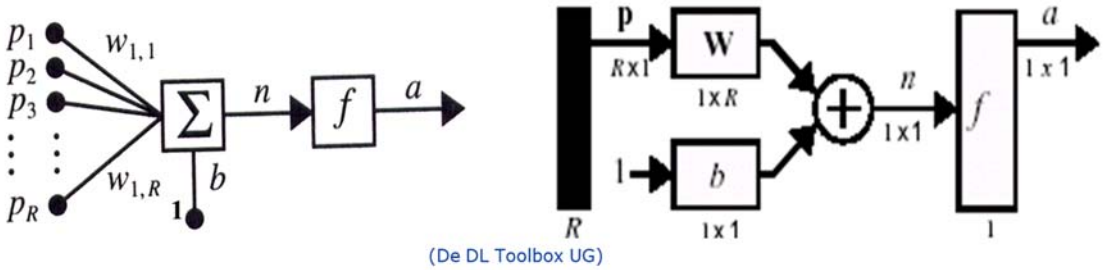



$$n = w_{11}p_1 + w_{12}p_2 + b = \begin{bmatrix} w_{11} & w_{12} \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \end{bmatrix} + b = \mathrm{W}p + b$$

$$a = f(n) = f(\mathrm{W}p + b)$$

Figura 4.2.6. Um neurónio com duas entradas e polarização e seu modelo matemático

Generalizando com *R* entradas e polarização, obtemos a Fig. 4.2.7.




$$n = w_{11}p_1 + w_{12}p_2 + \ldots + w_{1R}p_R + b \quad \Leftrightarrow \quad n = \mathrm{W}\,p + b$$

$$a = f(\mathrm{W}\,p + b)$$

Figura 4.2.7. Neurónio com *R* entradas mais polarização, e seu modelo matemática. A figura foi extraída do *Deep Learning Toolbox User's Guide* da Mathworks.

Como temos muitas entradas, teremos muitos pesos, e por isso é mais elegante usar representação compacta vetorial: *W* e *p* são vetores de dimensão *R*. *n* é um escalar, tal como a saída *a*. A segunda parte da Figura usa simbologia vetorial. Note-se que os índices dos pesos têm o seguinte significado:

$w_{1k}$ liga o neurónio 1 (neste caso o único) à entrada *k*.

## 4.3 Neurónio RBF

Existe um tipo especial de neurónio, chamado RBF de *Radial Basis Function* que funciona de um modo diferente.

A sua função de ativação não é monotonamente crescente: cresce até um máximo (geralmente 1) e depois decresce de um modo simétrico em relação ao seu eixo. As mais comuns são a função Gaussiana e a função triangular. A Figura 4.3.1 ilustra o caso da Gaussiana.

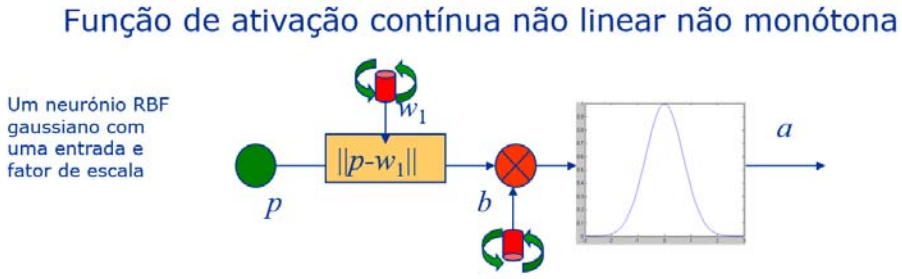



Figura 4.3.1. Um neurónio Gaussiano.

Além disso o peso $w_1$ e a polarização *b* têm um significado diferente.

O peso permite variar a distância da entrada em relação ao centro da gaussiana. Se o peso for nulo, o centro da gaussiana é a entrada. Se o peso for igual à entrada, o centro da gaussiana é zero. Assim

variando o peso move-se horizontalmente a Gaussiana, colocando-a onde seja mais conveniente O caso geral é que o centro da Gaussiana é dado pela distância Euclidiana entre a entrada e o peso, $\|p_1 - w_1\|$. Quer dizer que a dimensão do peso é igual ao número de entradas. Se tivermos duas entradas $p_1$ e $p_2$ a gaussiana será tridimensional, como um sino, assentando no plano $p_1$ - $p_2$ e o peso terá dois componentes que definem um ponto no plano $p_1$ - $p_2$ .

A polarização *b* no neurónio RBF é um fator de escala que multiplica a distância euclidiana anterior. Note-se que no neurónio anterior a polarização soma-se. A Gaussiana produz uma saída *a* dada por

$$a = e^{-(\|p-w_1\|\times b)^2} \qquad (4.3.1)$$

Se compararmos com a definição exata da Gaussiana, verificamos que *b* é o inverso de duas vezes a variância (sendo esta u quadrado do desvio padrão). Então um *b* pequeno corresponde a um desvio padrão grande, e vice-versa, como ilustrado na Figura 4.4.2.

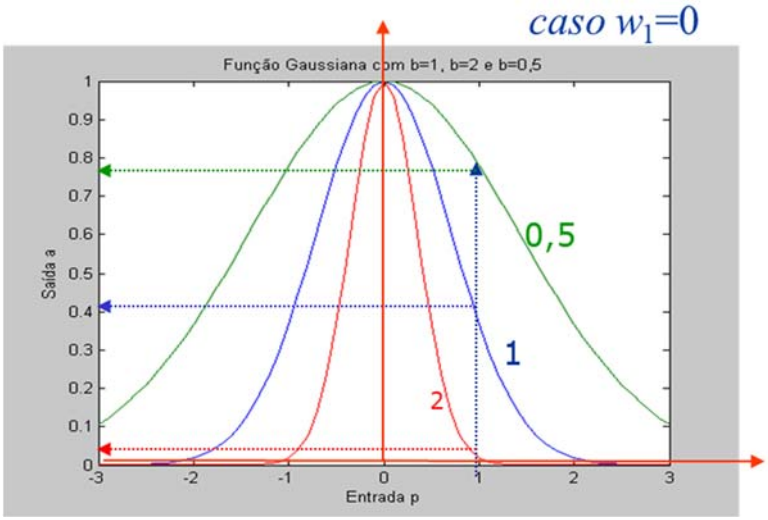



Figura 4.3.2. A polarização *b* (fator multiplicativo no RBF, e não aditivo como nos outros tipos de neurónios), permite abrir ou fechar a Gaussiana, aumentando ou diminuindo a sua propriedade de localidade.

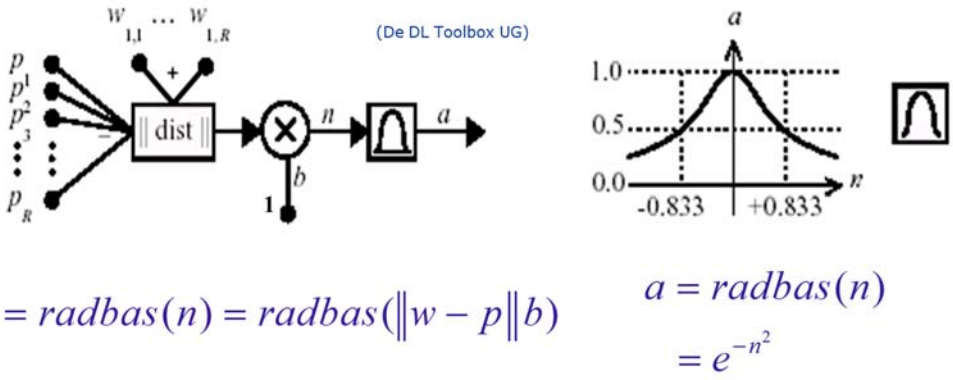
E como se calcula a saída do neurónio ? Aplicando a expressão (4.3.1). Graficamente é como está feito na Fig. 4.3.2, desenhada para o caso de $w_1$=0. Se a entrada é 1, a saída será 4,2 para *b*=1 , e 7,8 para *b*=0,5.

Portanto *w* e *b* permitem colocar a Gaussiana onde quisermos e com a abertura desejada. Se imaginarmos duas entradas, temos um sino que se move no plano, com abertura variável. A saída do neurónio para um valor concreto das duas entradas, que definem um ponto no plano, é o comprimento do segmento de reta vertical desde esse ponto à superfície da Gaussiana tridimensional, ou seja , do sino.

Com estes dois graus de liberdade, $w_1$ e *b* podemos colocar o neurónio centrado em qualquer ponto do espaço das entradas e com qualquer abertura. Diz-se por isso que o neurónio RBF tem a

propriedade de localidade porque só é ativado pelos pontos que fiquem abrangidos pela função radial; e essa localidade é maior ou menor conforme o coeficiente *b*. Assim se tivermos muitos neurónios distribuídos pelo espaço das entradas, para uma dada entrada só um neurónio é ativado se não houver sobreposição de neurónios. Se houver sobreposição podem ser ativados alguns, mas a maior parte ficará inativa.

No caso da DL Toolbox do Matlab temos a seguinte representação, para *R* entradas




$$a = radbas(n) = radbas(\|w - p\|b) \qquad a = radbas(n) = e^{-n^2}$$

Figura 4.3.3. Um neurónio RBF com *R* entradas.

Note-se que há também *R* pesos. Calculando a distância Euclidiana entre as *R* entradas e os *R* pesos, obtém-se um escalar, que é depois multiplicado por *b*. Por isso *n* é um escalar, tal como *a*, e a Gaussiana é sempre representada no plano qualquer que seja o valor de *R.* Por exemplo na pate direita da figura *n=0,833.*

4.4 Camadas de neurónios

Colocando dois neurónios em paralelo, havendo duas entradas, obtemos a Fig. 4.4.1

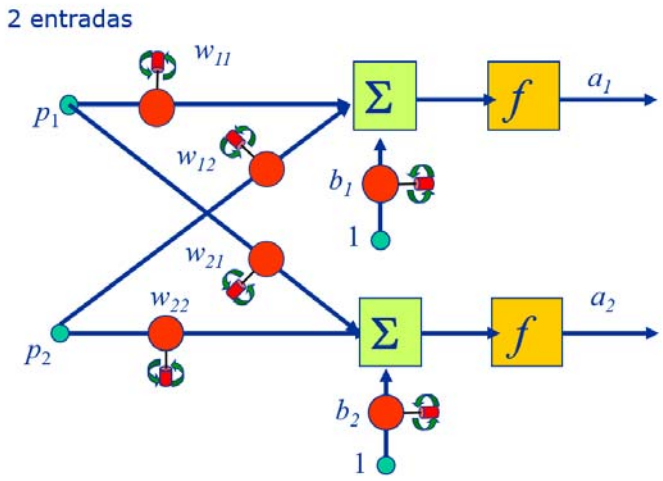



Figura 4.4.1. Dois neurónios em paralelo, ambos com duas entradas e polarização.

Se quisermos acrescentar uma terceira entrada, basta fazer como na Fig. 4.4.2.

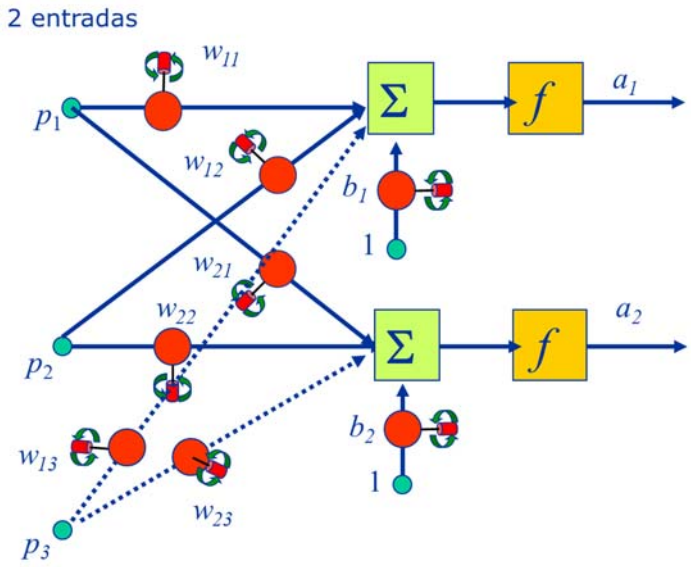


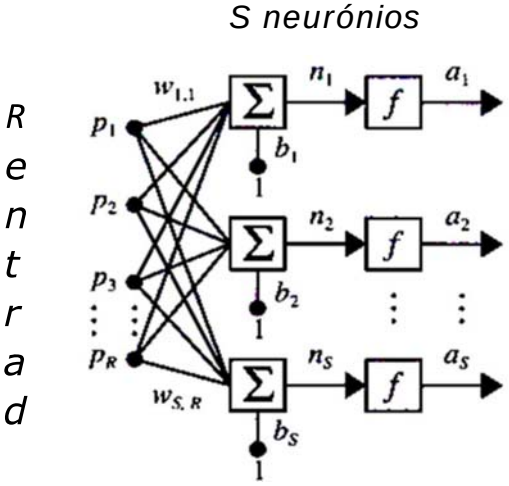

Figura 4.4.2. Dois neurónios com três entradas.

Agora a indexação dos pesos tem que ser feita com cuidado, dado que temos várias entradas e város neurónios. Adotaremos a seguinte convenção:

- peso $w_{ik}$ liga o neurónio *i* à entrada *k*.

Assim o primeiro índice é do neurónio e o segundo índice é da entrada. É importante fixar esta convenção, porque o desenvolvimento matemático depende dela.

Generalizando para um número qualquer de *R* entradas e de *S* neurónios (e portanto *S* saídas, temos a Fig. 4.4.3 e o seu modelo matemático.




$$a = f(Wp + b)$$

$$n_i = w_{i1}p_1 + w_{i2}p_2 + \ldots + w_{iR}p_R + b_i$$

$$a_i = f_i(n_i) \qquad i = 1, 2, \ldots, S$$

Figura 4.4.3. Camada de *S* neurónios, com *R* entradas e *S* polarizações.

Para cada neurónio *i* , *i=1, 2, …S,* temos as expressões $n_i$ e $a_i$ da figura; $f_i$ é a função de aivação do neurónio *i*. A notação começa a complicar-se, e convém simplificar. Para isso passa-se à notação matricial, tal como na Fig. 4.4.4.

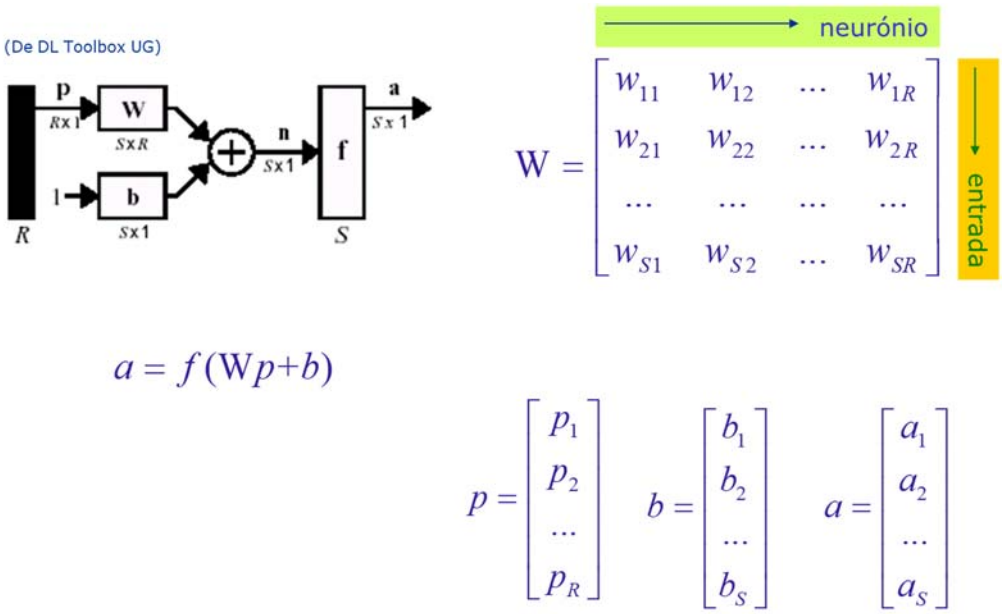



Figura 4.4.4. Notação matricial de uma camada de neurónios.

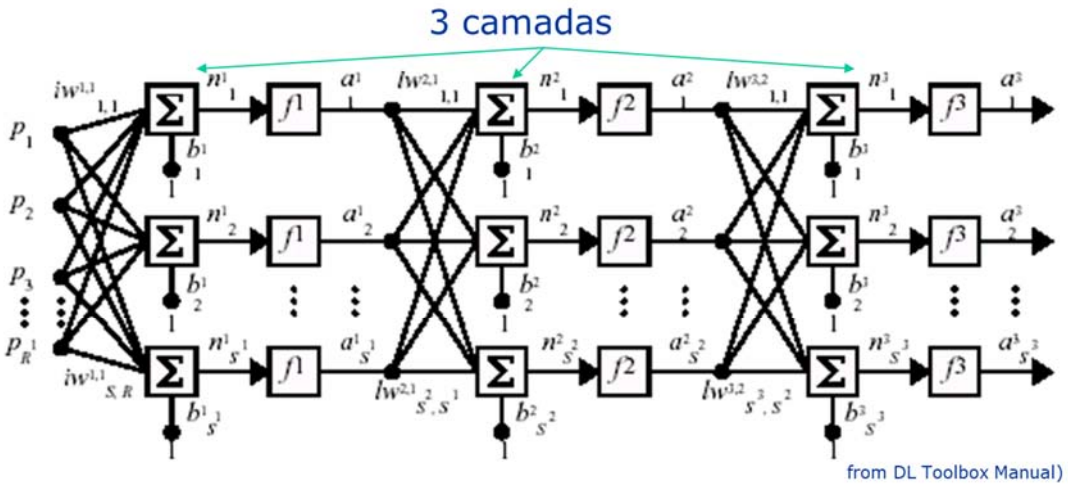
Agora temos a matriz de pesos W,de dimensões *SxR* em que cada linha correspondo aos pesos de um neurónio, e cada coluna aos pesos de uma entrada. Por exemplo a segunda linha contém os pesos que ligam o neurónio 2 a cada uma das *R* entradas, e e segunda coluna cotem os pesos que ligam cada uma das entradas ao neurónio 2. Usando os vetores *p*, *b*, e *a*, obtemos a expressão simples para a equação matemática da camada, simples e elegante.

$$a = f(Wp+b)$$

Mas é preciso não esquecer as convenções de indexação.

E se tivermos mais do que uma camada ?

As camadas ligam-se em série. No caso de três camadas teremos a Fig. 4.4.5.




$$a^1 = f^1(IW^{1,1}p+b^1) \qquad a^2 = f^2(LW^{2,1}a^1+b^2) \qquad a^3 = f^3(LW^{3,2}a^2+b^3)$$

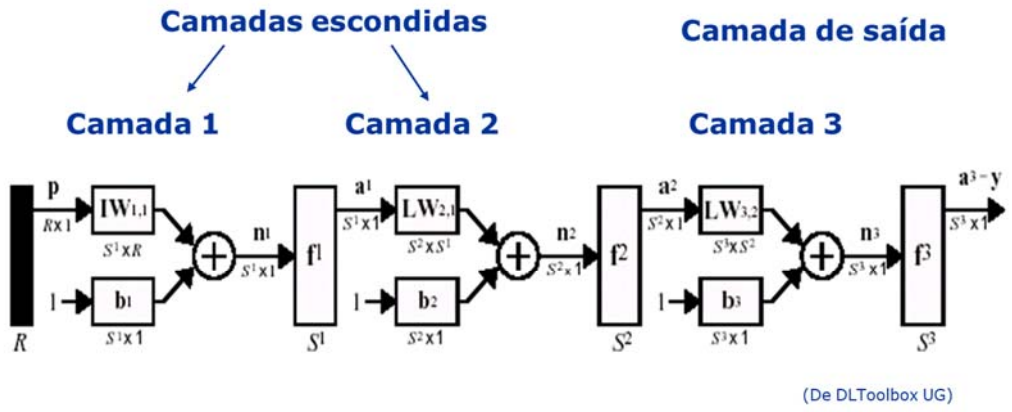
$$a^3 = f^3(LW^{3,2}f^2(LW^{2,1}f^1(IW^{1,1}p+b_1)+b_2)+b_3)$$

Figura 4.4.5. Uma rede de três camadas.

Agora é necessário mais um índice, o índice de camada, que se coloca como expoente nos *n* e nos *a*. Os pesos têm ainda mais índices: de entrada, de neurónio, de camada de origem e de camada de destino. Por exemplo

$$lw^{2,1}_{3,\,4}$$

Identifica o peso entre as camadas (layer, daí o *l*) 1 e 2, sendo 1 a origem a 2 a destino, ligando o neurónio *3* da camada 2 ao neurónio *4* da camada 1. É complicado mas toda esta informação é necessária para identificar os pesos. No caso da camada de entrada em vez de *lw* escreve-se *iw* , *i* para *input* (camada de entrada).




$$a^1 = f^1(IW^{1,1}p+b^1) \qquad a^2 = f^2(LW^{2,1}a^1+b^2) \qquad a^3 = f^3(LW^{3,2}a^2+b^3)$$

$$a^3 = y = f^3(LW^{3,2}f^2(LW^{2,1}f^1(IW^{1,1}p+b_1)+b_2)+b_3)$$

Figura 4.4.6. Representação compacta de uma rede com três camadas.

Camadas de neurónios RBF

No cado de neurónios RBF, uma camada é naturalmente composta por um certo número de neurónios em paralelo, dada um com a sua saída. Depois as saídas são todas somadas passando por uma camada de neurónios lineares (muitas vezes apenas um). Temos os pesos e polarizações dos neurónios RBF e os pesos e polarizações dos neurónios lineares,

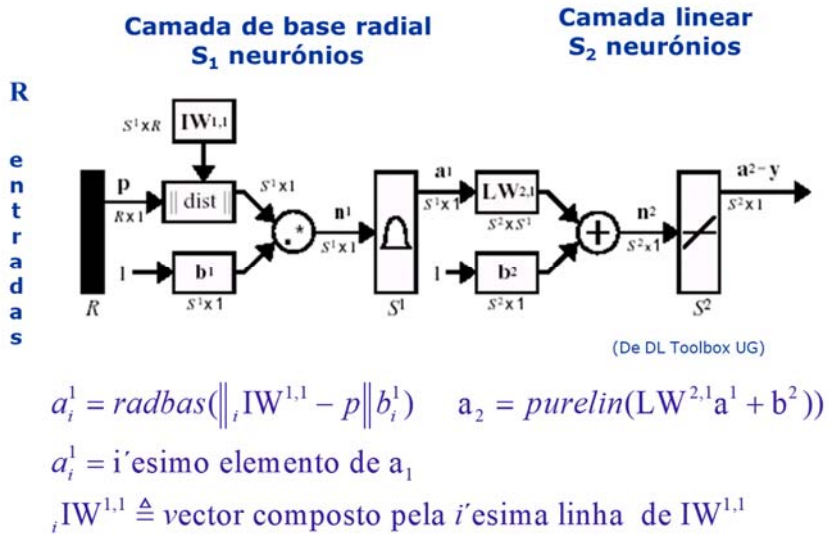



Figura 4.4.7. Rede RBF, com uma camada RBF seguida de uma linear.

4.5. Atrasos e redes dinâmicas ou redes recorrentes

Quando tratamos de sistemas dinâmicos, cuja saída atual depende de saídas passadas (o sistema tem memória), e queremos construir para eles modelos de redes neuronais, temos que introduzir elementos adicionais que exprimam essa memória. São os atrasos puros, ou simplesmente atrasos, representados na Fig. 4.5.5.

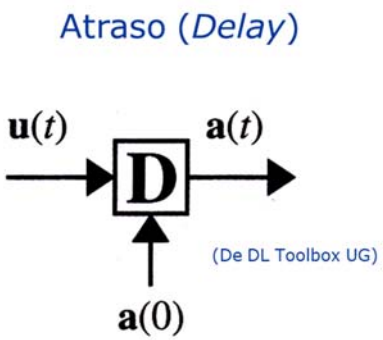



Figura 4.5.1 Um atraso puro

$a(t) = u(t-1)$

$a(0) \triangleq$ condição inicial

Agora para desenharmos uma rede dinâmica com um atraso, basta fazer como na Fig. 4.5.2

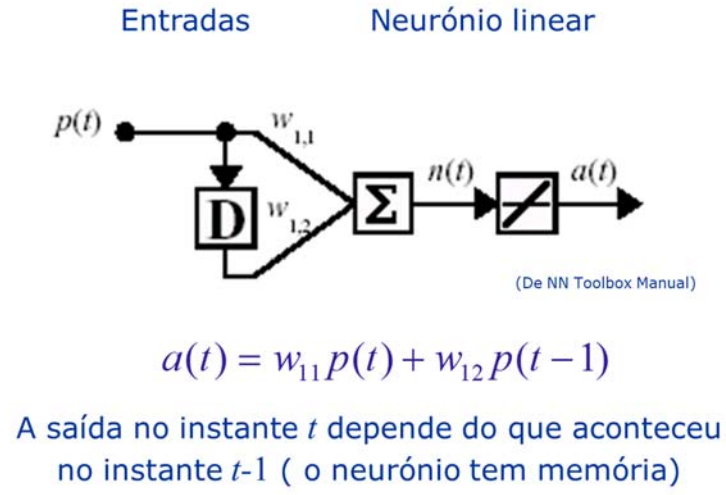



Figura 4.5.2. Rede dinâmica com um atraso.

A histórica rede recorrente de Elman, representada na Fig. 4.5.3, usa um atraso entre a saída da primeira camada (sigmoidal) e a sua própria entreda,

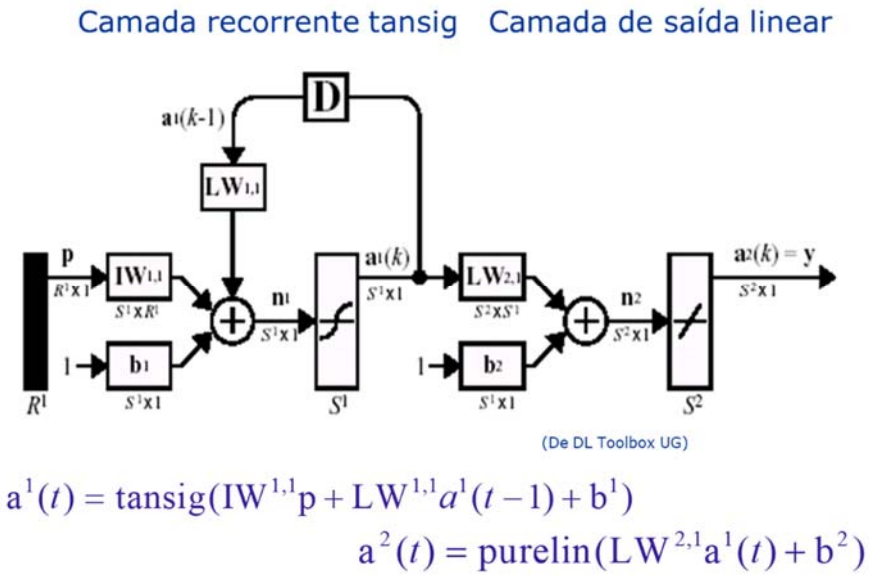



Figura 4.5.3. Rede de Elman. Tem uma camada sigmoidal seguida d e uma camada linear. Foi muito usada para modelizar sistemas dinâmicos.

4.6 Aprendizagem de redes neuronais: o caso do percetrão binário.

Nos parágrafos anteriores vimos que uma rede neuronal tem um conjunto de pesos e de polarizações que são os graus de liberdade de que dispomos para que a rede "reproduza" o comportamento de uma certa função. O processo de encontrar o melhor valor para esses pesos e polarizações á justamente o processo de aprendizagem (*learning*) da rede. Mexemos nos botões da caixa na figura 4.1.6 até que se obtenha o efeito desejado.

Mas mexemos como ? Por ordem ? À sorte ? Quanto tempo podemos demorar ?

Precisamos de um procedimento sistemático que pouco a pouco vá aproximando o comportamento da rede do desejado. E esse procedimento deve ser programado e resultar num código executável que seja eficiente e robusto. Isto é, precisamos de um algoritmo de aprendizagem.

Os algoritmos de aprendizagem são no essencial processos de otimização: partindo de um conjunto de valores iniciais, usando uma dada regra de iteração, vão evoluindo esses valores até que já não conseguem melhorar mais, e aí param e temos a solução ótima. Na realidade temos quase sempres uma solução subótima porque não existe ainda um algoritmo de otimização que vá iterando até alcançar a solução globalmente ótima. Em geral terminam num ótimo local (e não global). O problema dos ótimos locais constitui o maior drama de toda a otimização, e portanto também de todos os algoritmos de aprendizagem computacional.

Hoje em dia existem três abordagens fundamentais à aprendizagem computacional :

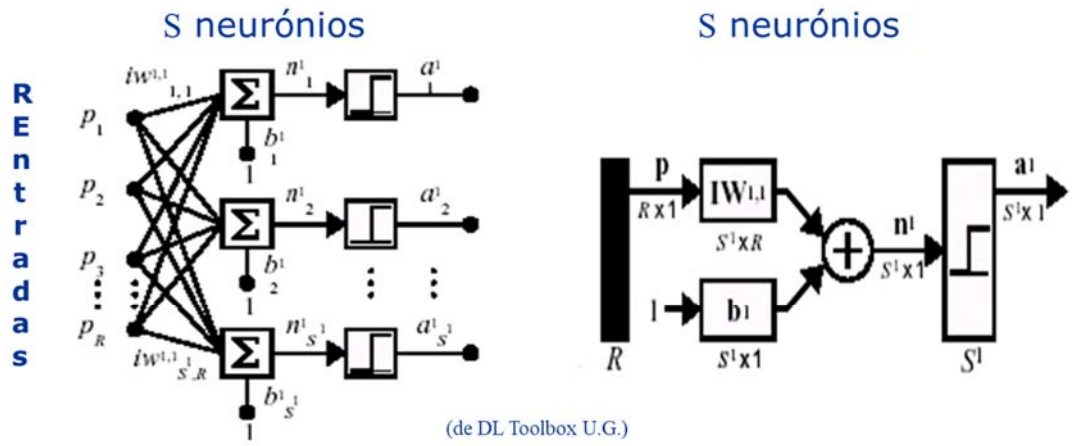
(i) Aprendizagem supervisionada, quando damos ao algoritmo de aprendizagem um alvo que ele deve alcançar, como se tivéssemos um professor a ensinar um aluno (*supervised learning*). Será esta a abordagem usada nesta disciplina. Fornece-se à RN um conjunto de *Q* de entradas *p*, e ao mesmo tempo para cada entrada fornece-se o valor desejado da saída, ou o alvo (*target, t*) para a saída da rede, por exemplo, no caso de *Q* valores de entrada, são fornecidos ao algoritmo de treni, qualquer que ele seja, *Q* pares ordenados

$$\left\{p^1,t_1\right\},\ \left\{p^2,t_2\right\},\ ...,\ \left\{p^Q,t_Q\right\}$$

(ii) aprendizagem não supervisionada, quando não se sabe à partida para onde se vai, mas é o próprio algoritmo que deve encontrar um caminho. Os mais conhecidos e usados são os algoritmos de agrupamento (*clustering*) que nos informam que grupos encontram.

(iii) aprendizagem por reforço (*reinforcement learning*), em que não se dá um alvo ao algoritmo, mas dá-se um estímulo positivo quando evolui bem, como se faz aos cavalos no circo com os cubos de açúcar. Nesta disciplina não abordaremos esta abordagem.

Os primeiros desenvolvimentos matemáticos de aprendizagem computacional foram feitos para o percetrão binário. O termo percetrão é usado na literatura com diferentes significados. Aqui chamaremos percetrão a uma rede com uma camada, e binário quando a função de ativação é a binária (0,1) ou (-1,1). É este, salvo melhor opinião, o significado inicial. Assim temos a Fig. 4.6.1 representando um percetrão binário com *R* entradas e *S* saídas.




$$a^1 = \text{hardlim}(IW^{1,1}p^1 + b^1)$$

Para uma só camada pode-se eliminar o indicativo ¹ de camada.

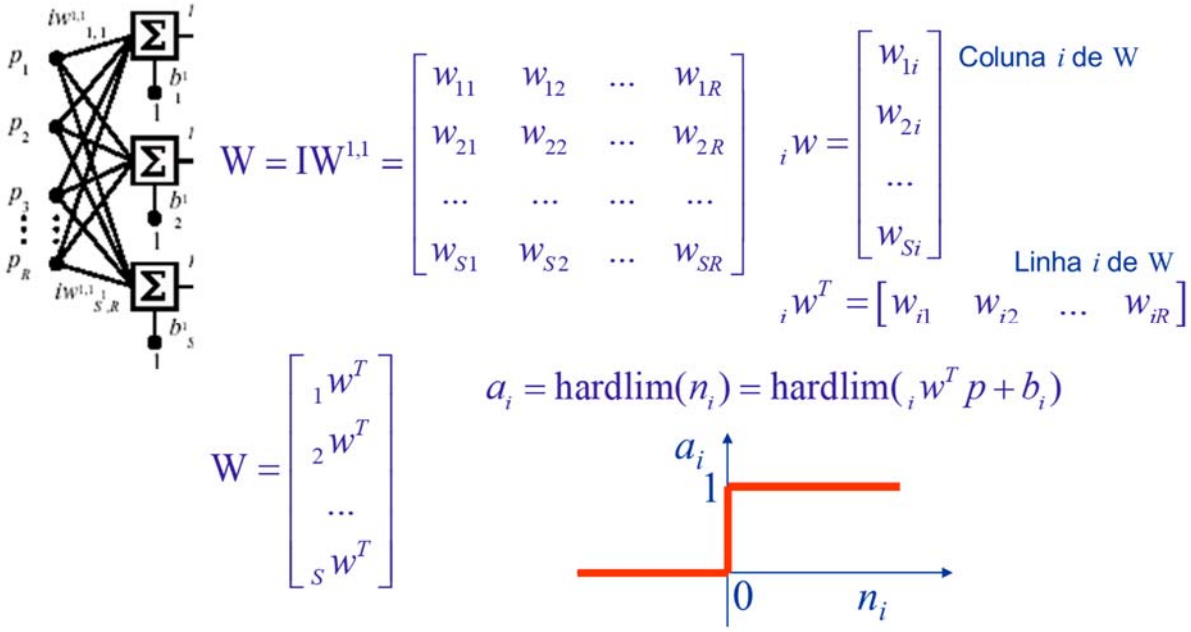



Figura 4.6.1. O percetrão binário e sua notação matemático. Note-se a convenção de indexação dos pesos definida anteriormente.

Cada neurónio divide o espaço de entradas em duas partes: aquela em que o $a_i$ vale 1 e aquela em que vale 0. Se fosse uma função binária bimodal (-1,1) aconteceria o mesmo.

Vejamos mais em pormenor no caso de um só neurónio binário com duas entradas, representado na Fig. 4.6.2.

No desenvolvimento matemático na Fig. 4.6.2, a fronteira de decisão é o conjunto de pontos em que a função de ativação transita de 0 para 1, ou seja, em que $n$=0. Calculando $n$ e resolvendo em ordem a $p_2$ obtém-se a equação de uma reta no plano, a vermelho na Fig, 4.6.2. Ora sendo duas as

entradas, o plano é o espaço das entradas, e o sistema de eixos tem como abcissa $p_1$ e como ordenada $p_2$. Portanto temos o plano dividido em dois semiplanos.

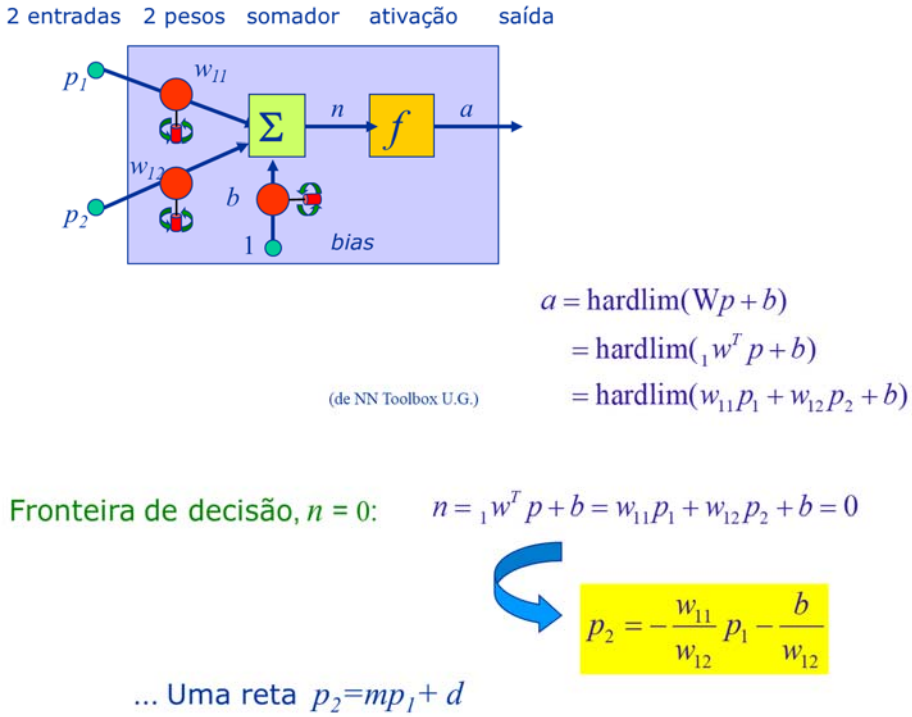



Figura 4.6.2. Um neurónio binário, com duas entradas e seu desenvolvimento matemático.

Admitindo o vetor de pesos $[-1\ 1]^T$, e a polarização $b$ =-1, desenhe-se a reta no plano das entradas.

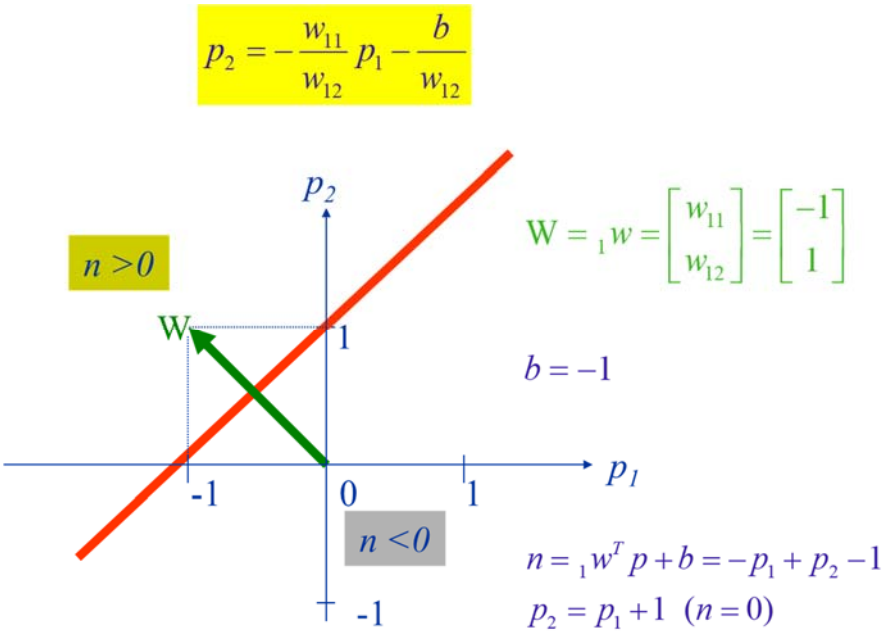



Figura 4.6.3 Análise geométrica do percetrão com duas entradas. A reta vermelha tem como equação $p_2 = p_1 + 1$, que se encontro a partir de $n$=0 e usando, como exemplo, os pesos $[-1\ 1\ ]^T$, que definem o vetor a verde no espaço das entradas.

Depois de encontrada a fronteira (reta a vermelho) do percetrão, é preciso ver qual dos dois semiplanos corresponde a *n*<0 e qual corresponde a n>=0.

Na Fig. 4.6.3 o vetor verde dos pesos é normal (perpendicular) à fronteira vermelha *n*=0. Será sempre assim ?.

Admita-se o caso *b*=0. Então,

$$n = w^T p = 0 = w_{11} p_1 + w_{12} p_2 \Rightarrow p_2 = -\frac{w_{11}}{w_{12}} p_1 \Rightarrow p_2 = m p_1$$

A fronteira passa pela origem do sistema de eixos. Sempre que a polarização é nula a fronteira passa pela origem.

Por outro lado, a fronteira é o produto interno do vetor de pesos pelo vetor de entradas que pertençam à fronteira. Se o produto interno de dois vetores é nulo, quer dizer que os vetores são perpendiculares. Por isso o vetor de pesos é perpendicular a qualquer vetor sobreposto à fronteira, ou seja, à própria fronteira. Daí o desenho na Fig. 4.6.3.

E se *b* não é nula ? Como vimos anteriormente, Fig. 4.2.3, a polarização desloca a fronteira horizontalmente, mas não altera o seu declive, ou seja, qualquer polarização não nula produz uma fronteira que é paralela à da polarização nula.

Pode-se assim concluir, e generalizando para qualquer dimensão, que o vetor de pesos é sempre normal à fronteira no espaço das entradas. Por exemplo com três entradas, o espaço das entradas é tridimensional, a fronteira é um plano, e o vetor de pesos é perpendicular a esse plano.

O vetor de pesos aponta para a região *n*>0. Pode-se verificar selecionando duas entradas e vendo o sinal de *n.* Na Fig. 4.6.3 todos os pontos acima da fronteira dão n>0, e todos os abaixo dão n<=0. Por isso podemos dizer que o vetor dos pesos aponta para a região *n*>0.

Se mudarmos os pesos para $[0\ 1]^T$ e mantendo b= -1, repetindo o raciocínio, se mantivermos a polarização nula, obtém-se a Fig. 4.6.4 .

$$p_2 = -\frac{w_{11}}{w_{12}} p_1 - \frac{b}{w_{12}}$$

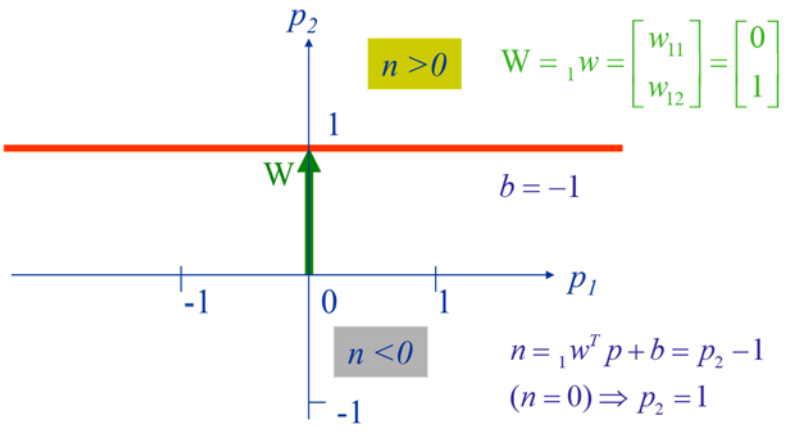


Figura 4.6.4. Mudando os pesos da Fig. 4.6.3, verifica-se que o vetor de pesos girou a fronteira, neste caso de modo que ficou horizontal.

E continuando a mudar os pesos, mantendo a polarização fixa em -1, de tal modo que o vetor respetivo gire no sentido dos ponteiros do relógio, obtêm-se as figuras seguintes.

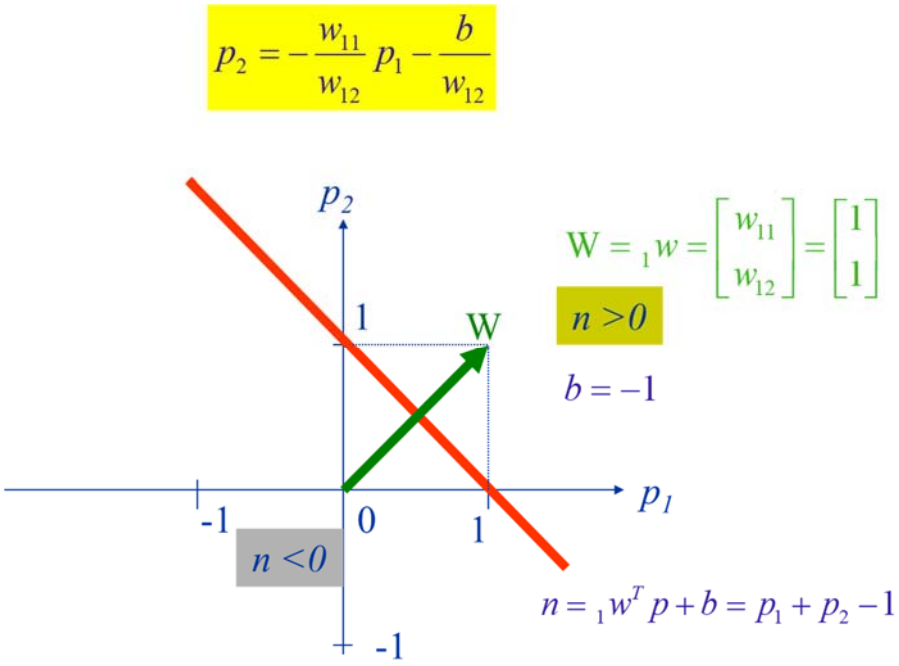



Figura 4.6.5. Girando os pesos mais para a direita, eles “levam” consigo a fronteira.

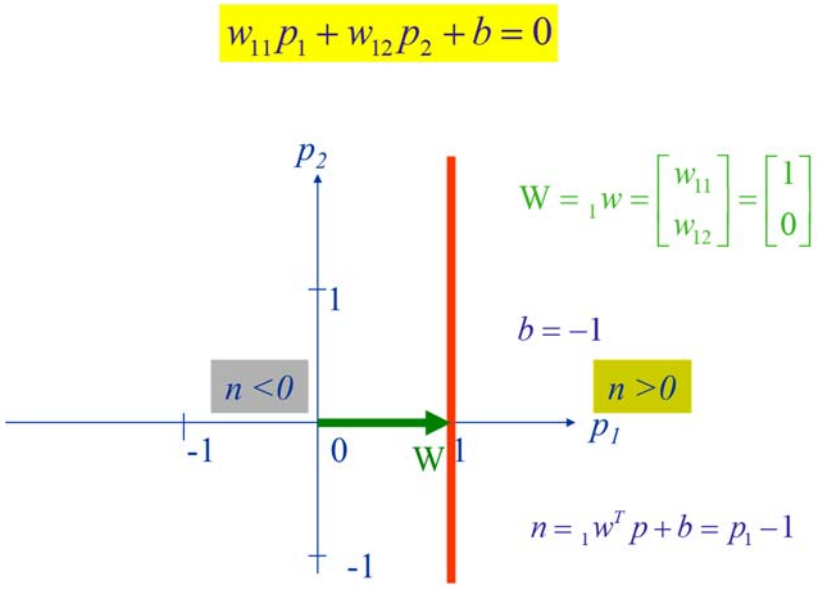



Figura 4.6.6.

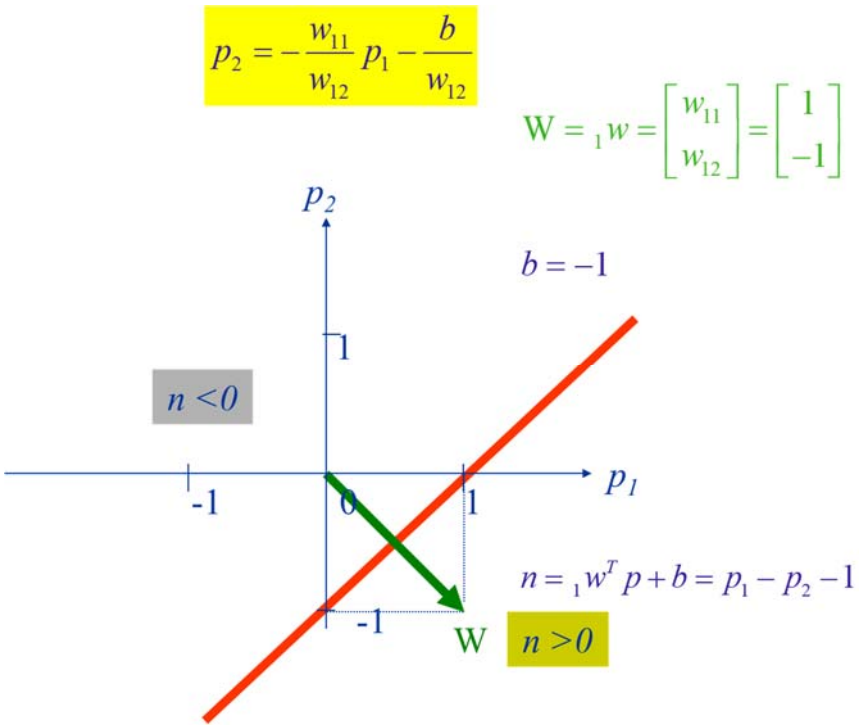



Figura 4.6.7.

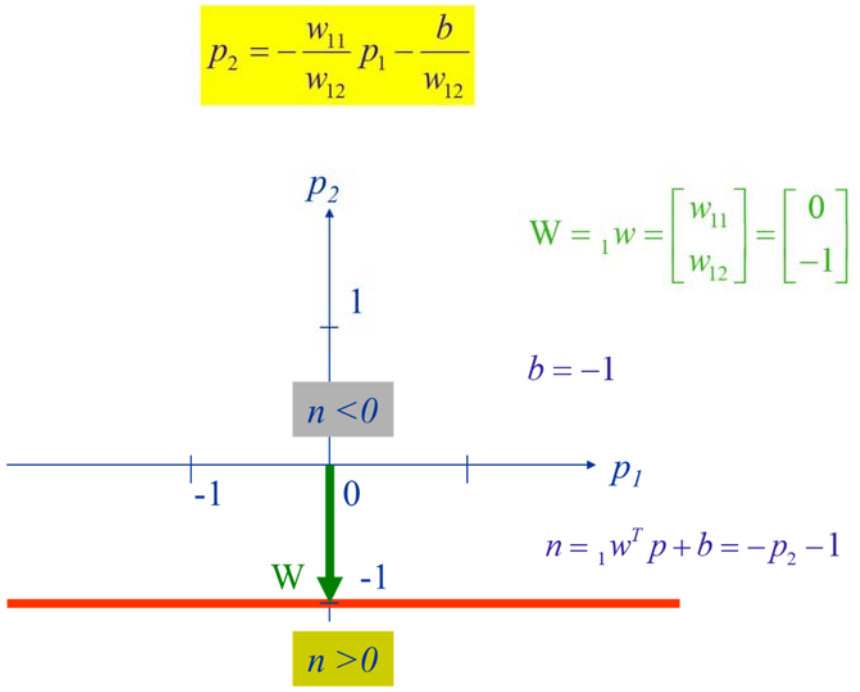



Figura 4.6.8

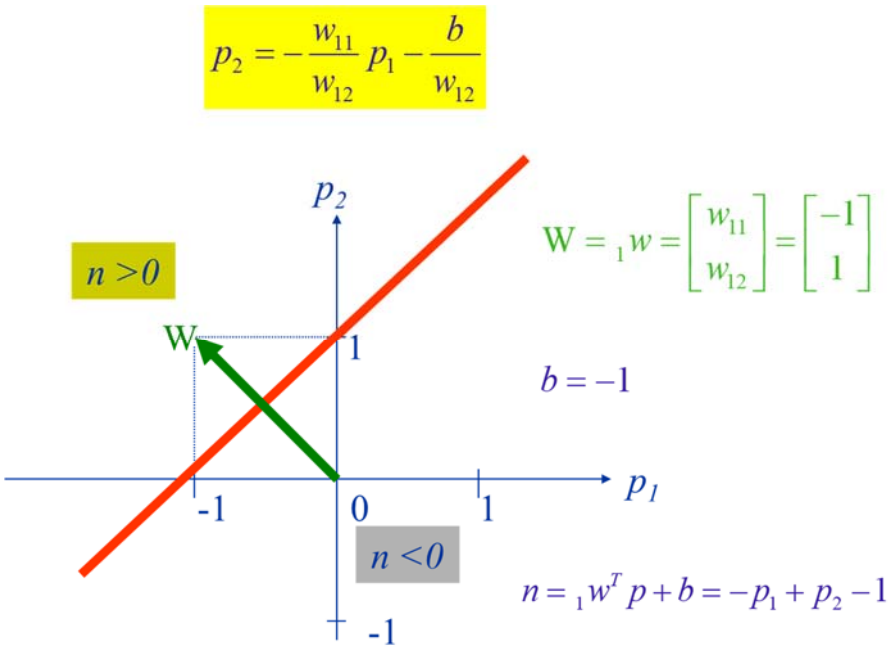



Figura 4.6.9.

Se o leitor interessado e paciente quiser repetir os desenhos com uma polarização diferente, verificará que se passa o mesmo, com a diferença de a fronteira se afastar ou se aproximar da origem. Por exemplo com *b*=0, a fronteira passa na origem.

Portanto variando o vetor de pesos e a polarização poderemos colocar a fronteira em qualquer região do espaço das entradas e com qualquer inclinação. Um simples neurónio binário tem já uma capacidade notável.

Num problema de classificação qualquer, no plano, isto é, com duas entradas e dua classes, pode-se proceder do seguinte modo para o treino manual e gráfico do neurónio:

1º- Traça-se em primeiro lugar e linha da fronteira, que deve separar os pontos nas duas classes, se isso for possível (se for impossível o neurónio não é capaz de classificar).

2º- Seleciona-se um vetor de pesos W perpendicular a essa linha, de qualquer amplitude (o que importa é a sua direção e o seu sentido).

3º- Calcula-se a polarização *b* necessária fazendo as contas para um ponto da fronteira (*n*=0)

4º- O vetor W aponta para a região *n* > 0 e temos assim a classificação definida: os pontos em que n>=0 dão a saída 1, e os pontos com *n*<0 dão a saída 0.

Exemplo do OR lógico

O Cálculo do valor lógico da disjunção de duas variáveis lógicas pode ser feito por um percetrão binário.

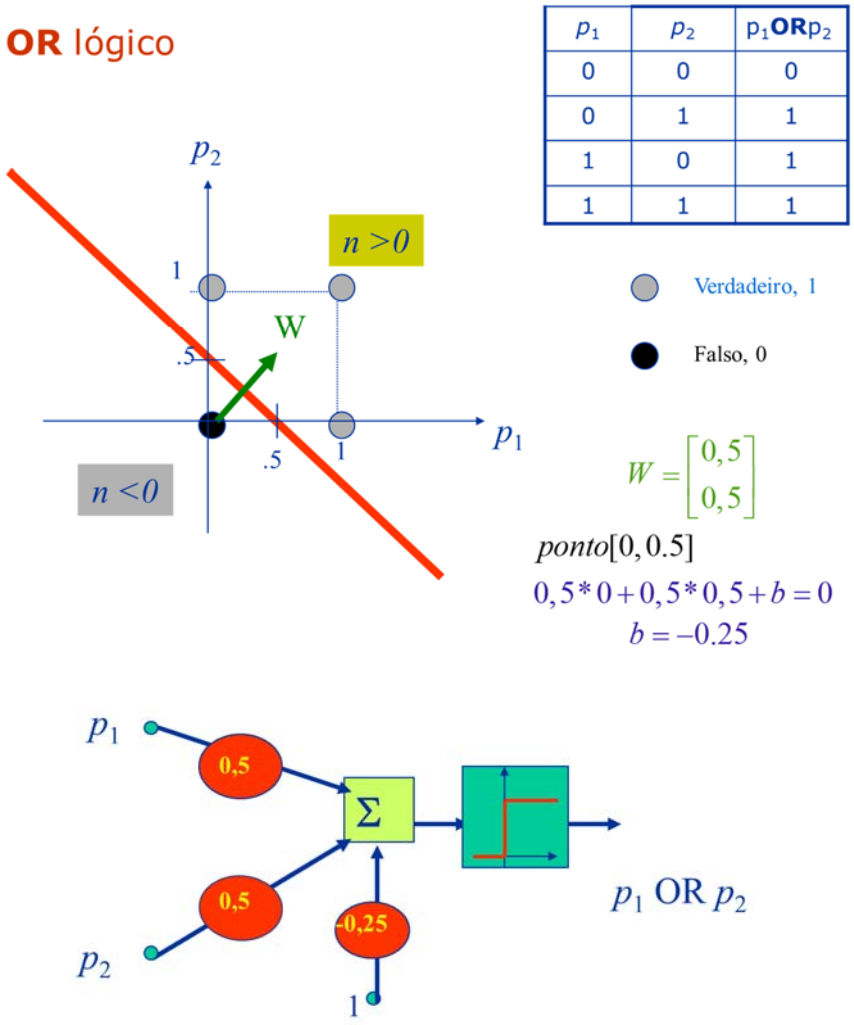



Figura 4.6.10. A operação OR implementada por um percetrão binário.

Um só neurónio permite classificar um conjunto de dados em duas classes. No caso de um percetrão com vários neurónios binários, podemos classificar um conjunto de dados em várias classes. Cada neurónio produz uma fronteira que divide o espaço de entradas em duas partes. Assim dois neurónios dividem o espaço das entradas em até 4 ($2^2$) zonas, e com *S* neurónios até $2^S$ zonas.

Exemplo 4.6.1

Considerem-se os 13 pontos seguintes pertencentes a 4 classes, cada uma com a sua cor.


Como temos 4 classes, precisamos de dois neurónios. O espaço das entradas, como são duas, é o plano. Desenhamos os 13 pontos no plano, traçamos as duas fronteiras que separam os pontos pela cor. Temos muitas soluções. Sempre que uma fronteira passa passar pela origem, é de aproveitar porque uma das polarizações será nula. Depois desenhamos dois vetores de pesos perpendiculares a essas fronteiras. Note-se que o vetor de pesos começa sempre na origem. Conforme o sentido dos

vetores de pesos, teremos $n_1$ e $n_2$ positivos ou negativos, como assinalado na figura. Note-se que N1 é a fronteira do neurónio 1 e N2 a do neurónio 2.

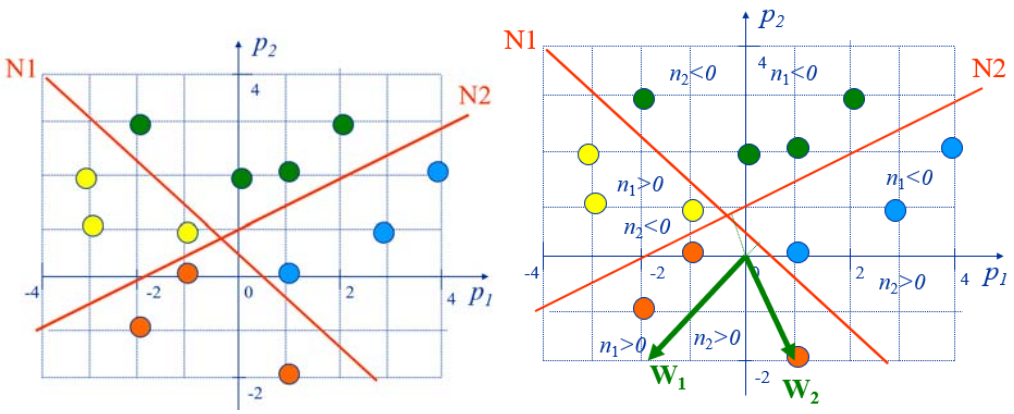



Figura 4.6.11. Resolução gráfica do Exemplo 4.6.1

Temos o exemplo quase resolvido. Basta ler as coordenadas dos pesos e calcular as polarizações, obtendo-se os seguintes resultados. O código das classes extrai-se dos sinais de $n_1$ e $n_2$ .

$$W_1 = \begin{bmatrix} -1,8 \\ -2 \end{bmatrix} \qquad W_2 = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$$

$$b_1=1 \qquad b_2=2$$

$$\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$n_1<0$ $n_1>0$ $n_1<0$ $n_1>0$
$n_2<0$ $n_2<0$ $n_2>0$ $n_2>0$

E o resultado final é ilustrado pela Figura 4.6.12. Se dermos a entrada $[-2\ 3]^T$, o primeiro ponto da classe 1, a saída do neurónio será $[0\ 0]^T$.

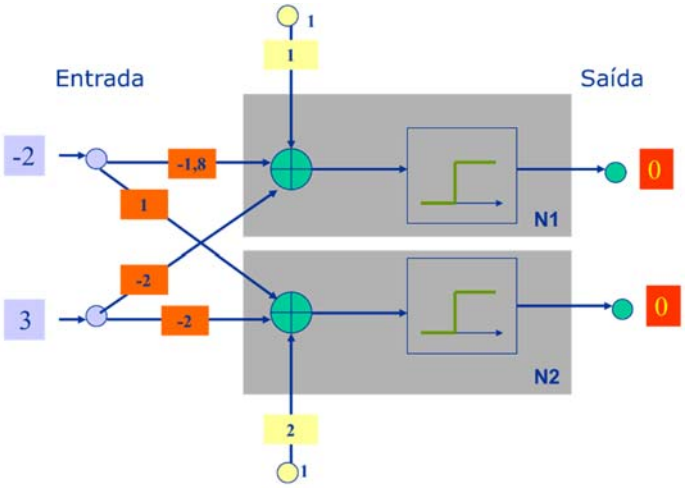



Figura 4.6.12. Neurónio para o exemplo 4.6.1.

Esta construção gráfica é muito interessante, permite-nos apreender características importantes do neurónio, mas não serve para problemas práticos.

Na prática temos sempre uma grande quantidade de dados a classificar, com entradas de muitas dimensões, e por isso é preciso um processo analítico, programável, isto é, precisamos de ume regra

de aprendizagem que funcione para qualquer número de dados de qualquer dimensão, para um qualquer número de classes.

Para o seu desenvolvimento podemos usar um método gráfico, mais fácil do que o analítico. Vejamos através de um exemplo simples, mas ilustrativo (de Hagan et coll.).

Exemplo 4.6.2

Consideremos três entradas e três alvos

$$\left\{p^1=\begin{bmatrix}1\\2\end{bmatrix},t_1=1\right\},\ \left\{p^2=\begin{bmatrix}-1\\2\end{bmatrix},t_2=0\right\},\left\{p^3=\begin{bmatrix}0\\-1\end{bmatrix},t_3=0\right\}$$

Desenhando-os no plano, com o código de cores indicado. Como a cada entrada corresponde um alvo escalar, basta-nos um neurónio binário, e, portanto, teremos apenas uma fronteira, que desconhecemos de momento. São também necessários dois pesos (um para cada componente da entrada), e uma polarização.

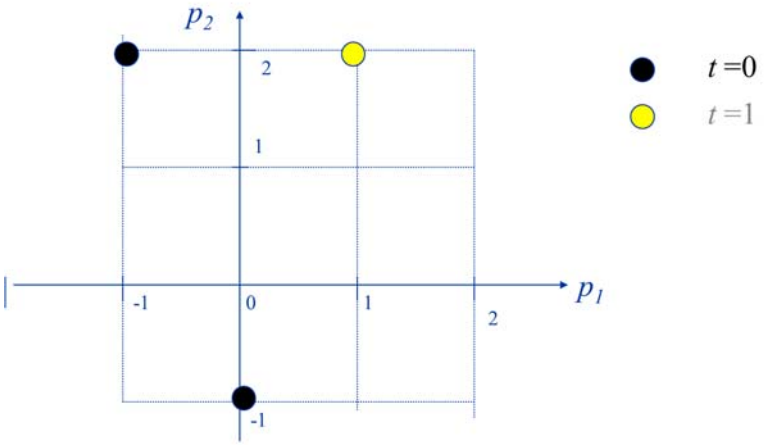



Figura 4.6.13.

Faça-se uma inicialização aleatória dos pesos, obtendo-se $[1,0\ \ -0,8]^T$. Sem perda de generalidade, admitamos que a polarização é nula e em consequência a fronteira passará pela origem. A fronteira correspondente será perpendicular ao vetor de pesos, conforme ilustrado na Fig. 4.6.14. A fonteira é a linha vermelha sólida. Atendendo ao sentido de $W^{(0)}$, pesos iniciais, ou da iteração zero, os pontos à direita devem ter alvos iguais a 1, e os situados à esquerda alvos iguais a zero. Vendo o que se obteve, o ponto $p_1$ está bem classificado, $p_2$ está bem classificado, *e* $p_3$ está mal classificado (pouca sorte …).

Portanto analisando os pontos de 1 a *n*, o primeiro que encontramos mal classificado é neste caso precisamente o primeiro. Não é necessário prosseguir para os próximos. Podemos já alterar a fronteira de modo a colocar o $p_1$ na região certa (a região dos amarelos). Como $t_1$ =1 e dá $t_1$=0, temos que subir este alvo. Para isso é necessário rodar a fonteira no sentido contrário ao dos ponteiros de um relógio, até por exemplo à linha tracejada. Para calcular os pesos que produzem essa fronteira tracejada, basta desenhar um vetor de pesos que lhe seja perpendicular.

Mas para se obter um procedimento mais algorítmico, some-se o vetor da entrada $p_1$ ao vetor de pesos $w^{(0)}$ , obtendo-se o vetor $W^{(1)}$ na Figura

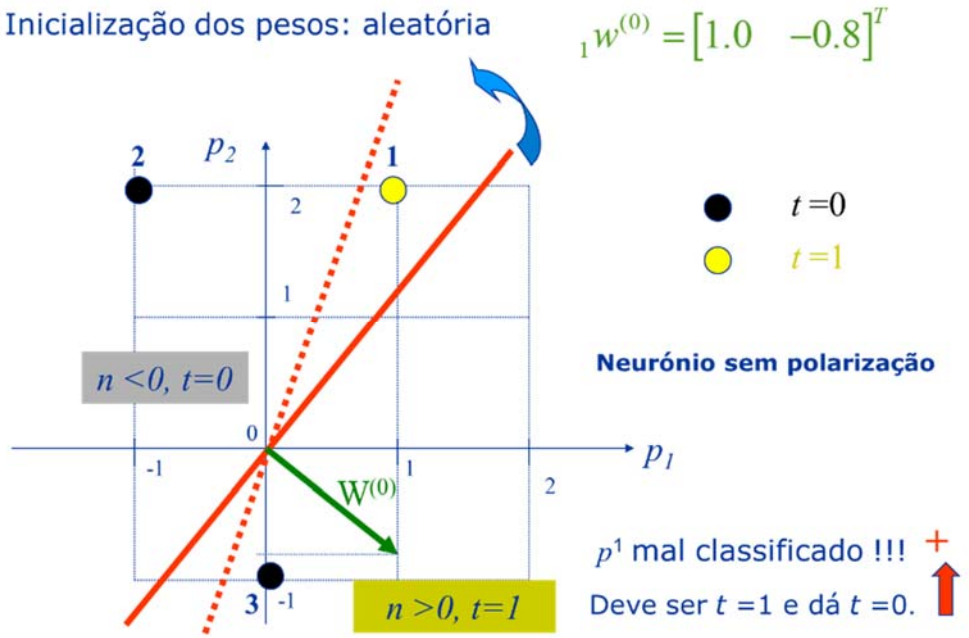



Figura 4.6.14. Primeira iteração do algoritmo de treino do percetrão.

Soma-se $p_1$ (sinal + a vermelho) porque tem que se subir o alvo.

$$_1w^{(1)} = {}_1w^{(0)} + p_1$$

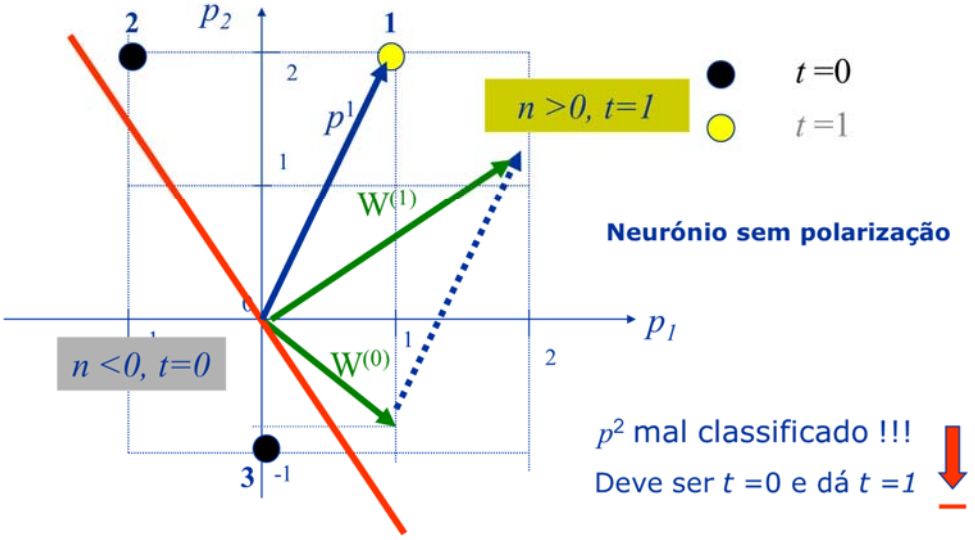



Figura 4.6.15. Primeira iteração do algoritmo.

Obtém-se agora a situação da Fig. 4.6.15 com a nova fronteira linha vermelha contínua. O ponto $p_1$ ficou bem classificado, mas em contrapartida o $p_2$, que estava bem, ficou mal …. Portanto esta solução não serve, o $t$ de $p_2$ tem que diminuir, passar de 1 a 0.

Sendo assim, e seguindo a abordagem da iteração 1, subtrai-se $p_2$ a $W^{(1)}$, obtendo-se a Fig.4.6.15

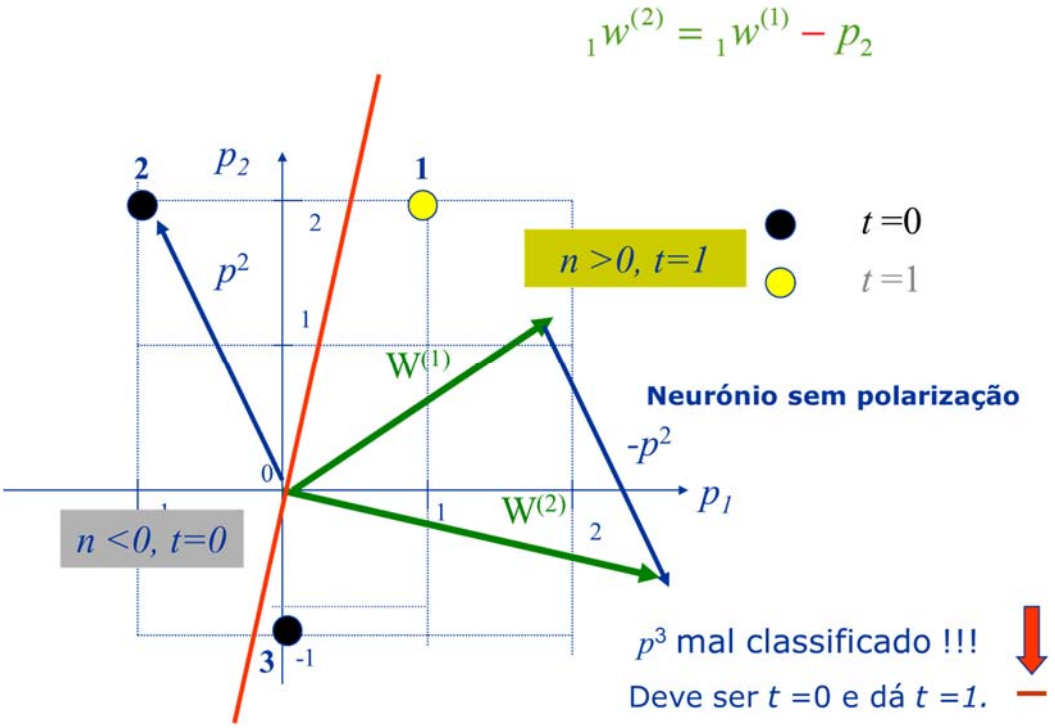



Figura 4.6.13. Segunda iteração do algoritmo.

Resolveu-se o problema de $p_2$ mas criou-se o problema de $p_3$ . Subtraindo $p_3$ a $W^{(2)}$, como na Fig. 4.6.14, obtém-se finalmente uma solução em que todos os pontos estão bem classificados.

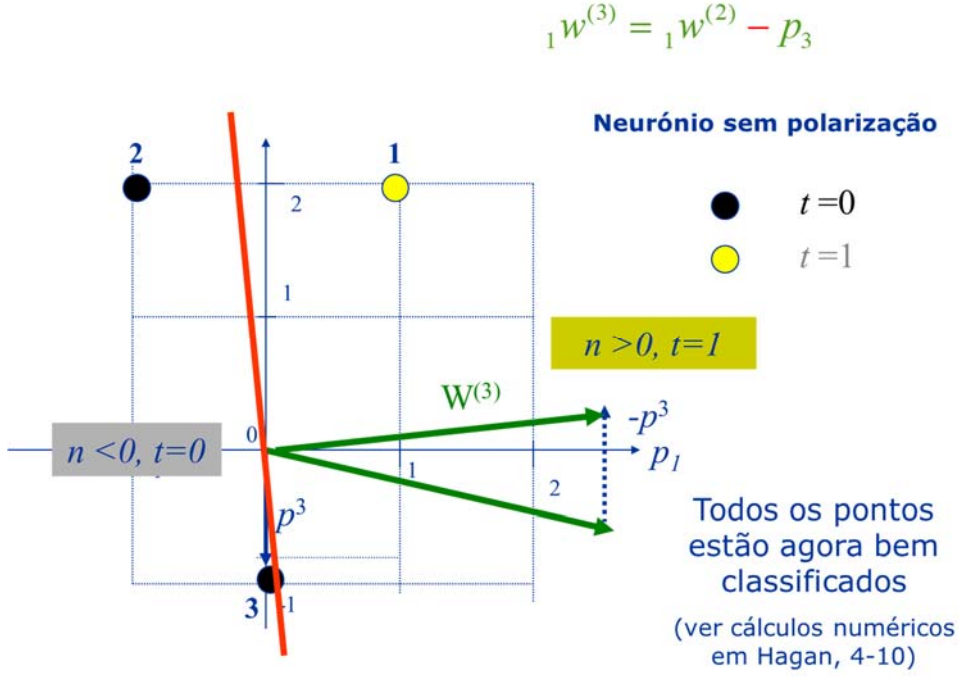



Figura 4.6.14. Terceira e última iteração do algoritmo.

O algoritmo consiste afinal numa sequência de rotações da fronteira, através de somas ou subtrações da primeira entrada mal classificada aos pesos atuais, conforme ilustrado pela Fig. 4.6.15.

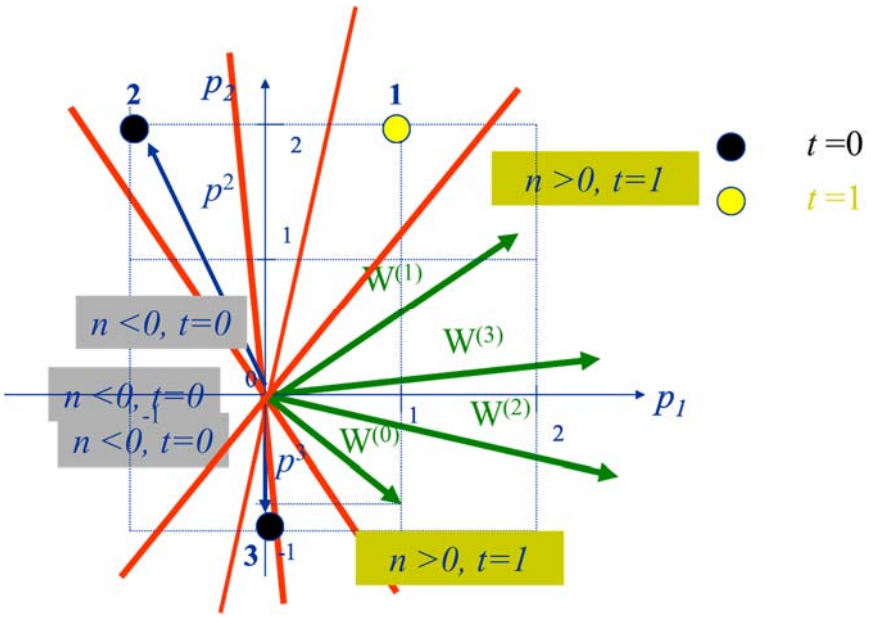



Figura 4.6.15. Todas as iterações juntas

O algoritmo analisa os dados ponto a ponto, começando pelo primeiro. Se um ponto está bem, nada faz e prossegue. Se um ponto está mal,

Se $t=1$ e $a=0$, então fazer $\quad {}_1w^{(new)} = {}_1w^{(old)} + p \qquad e=+1$

Se $t=0$ e $a=1$, então fazer $\quad {}_1w^{(new)} = {}_1w^{(old)} - p \qquad e=-1$

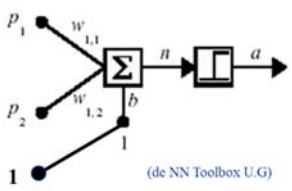
Se $t=a$ então fazer $\quad {}_1w^{(new)} = {}_1w^{(old)} \qquad e=0$

Definindo agora $e=t-a$, temos finalmente o algoritmo matematicamente expresso:

$$ {}_1w^{(new)} = {}_1w^{(old)} + e.p $$

Os cálculos anteriores foram feitos para um neurónio sem polarização (a fronteira passa pela origem). Mas isso é possível em poucos problemas.

Se um neurónio tem polarização, considere-se esta como uma entrada adicional e faça-se como na Fig. 4.6.16.




A polarização é um peso de uma entrada constante igual a 1 !

$$ {}_1\theta = \begin{bmatrix} {}_1w \\ b \end{bmatrix} \qquad z = \begin{bmatrix} p \\ 1 \end{bmatrix} $$

$$ a = \text{hardlim}({}_1\theta^T.z) $$

Figura 4.6.16. A polarização é uma entrada adicional.

O vetor de pesos tem mais um elemento *b* e o vetor de entradas mais um elemento constante igual a 1.

Usando a notação da Fig.4.6.16 pode-se escrever a famosa regra (de aprendizagem) do percetrão

$$ {}_1\theta^{(new)} = {}_1\theta^{(old)} + e.z $$

Se tivermos uma camada de *S* neurónios, portanto com *S* saídas e alvos, para o neurónio *i, i* =1, ..., *S*

$$ e_i = t_i - a_i \qquad z_i = \begin{bmatrix} p \\ 1 \end{bmatrix} \qquad {}_i\theta = \begin{bmatrix} {}_i w \\ b_i \end{bmatrix} $$

cria-se o vetor $z_i$ e o vetor $\theta_i$

$$ {}_i\theta^{(new)} = {}_i\theta^{(old)} + e_i.z_i \qquad {}_i\theta^{(new)T} = {}_i\theta^{(old)T} + e_i.z_i^T $$

$$ \Theta = \begin{bmatrix} {}_1\theta^T \\ \dots \\ {}_S\theta^T \end{bmatrix} \qquad z^T = \begin{bmatrix} z_1^T \\ \dots \\ z_S^T \end{bmatrix} \qquad e = \begin{bmatrix} e_1 & e_2 & \dots & e_S \end{bmatrix} $$

E obtém-se a regra do percetrão para uma camada

$$ \Theta^{new} = \Theta^{old} + e.z^T $$

Como vimos pela análise gráfica é um processo surpreendentemente simples.

Converge sempre num número finito de iterações ? Sim, se houver uma solução. Ver a demonstração analítica por exemplo em Hagan at Coll 4-15.

Quando um conjunto de dados tem entradas de dimensões muito diversas, algumas anormais (*outliers*), convém normalizar o processo para que todas as entradas tenham a mesma importância, fazendo

$$ \Theta^{new} = \Theta^{old} + e.\frac{z^T}{\|z\|} $$

E quando há solução ? Todo o processo de aprendizagem se baseia no traçado de fronteiras lineares (retas, planos, hiperplanos ) de tal modo que cada classe tenha a sua zona bem definida pelas fronteiras. Isso é possível se os dados foram linearmente separáveis. No plano, isso quer dizer que uma reta (função linear) cumpre a função. No entanto é fácil encontrar conjuntos de dados que não sejam linearmente separáveis.

Um caso clássico, e que na altura levantou muita polémica sobre redes neuronais, é o do XOR, ou OR exclusivo, representado na Fig.4.6.17.

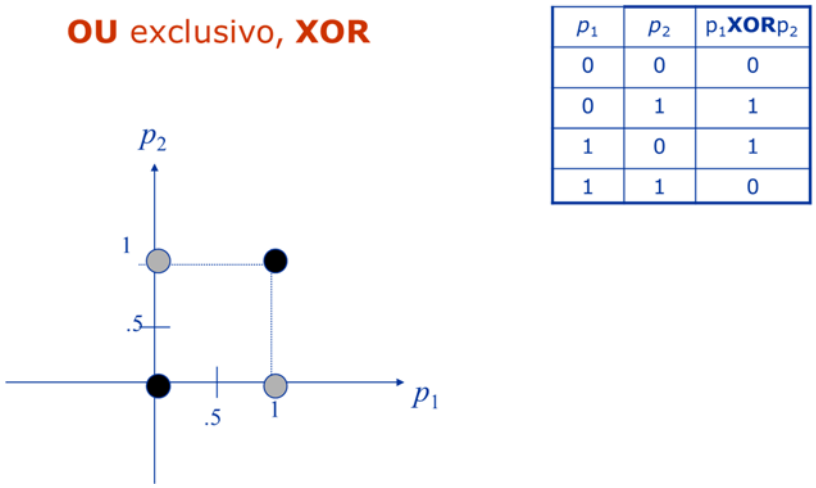



Figura 4.6.17. O caso do XOR. Não é possível traçar uma linha reta que separe os pontos cinzentos (verdade) dos pretos (falso)

Conclusão:

A regra de aprendizagem do percetrão é supervisionada, porque depende do erro e este depende do alvo (target).

É uma regra simples, mas poderosa. Resolve problemas com muitos dados, num número não muito grande de iterações, se os dados forem linearmente separáveis.

Os problemas não linearmente separáveis só podem ser resolvidos com redes multicamada, que veremos posteriormente.

4.7. Redes genéricas de uma camada: ADALINE e Memória Associativa

4.7.1 Treino em diferido (*batch*) da ADALINE

O percetrão que vimos em 4.6 é binário, dada a sua função de ativação. No entanto uma camada de neurónios pode ter outras funções de ativação (lineares, sigmoidais, etc.) Nestes casos não se pode aplicar a regra do percetrão.

Se por exemplo a função de ativação for linear (*purlin*), a rede chama-se ADALINE (*Adaptive Linear Network*). Está ilustrada na Fig. 4.7.1.

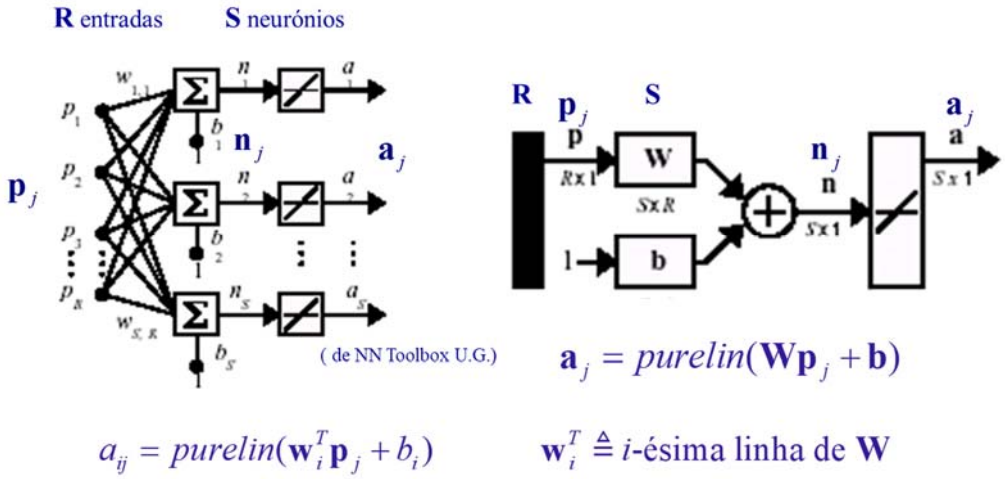



Figura 4.7.1 ADALINE: uma camada de neurónios lineares

Os desenvolvimentos começam a complicar-se, principalmente por causa da notação que é necessário usar. Assim, para uma camada temos a seguinte notação:

$$\mathbf{P} = \begin{bmatrix} \mathbf{p}_1 & \mathbf{p}_2 & \dots & \mathbf{p}_2 \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} & \dots & p_{1Q} \\ p_{21} & p_{22} & \dots & p_{2Q} \\ \dots & \dots & \dots & \dots \\ p_{R1} & p_{R2} & \dots & p_{RQ} \end{bmatrix}_{RxQ} \quad \mathbf{P}^T = \begin{bmatrix} p_{11} & p_{21} & \dots & p_{R1} \\ p_{12} & p_{22} & \dots & p_{R2} \\ \dots & \dots & \dots & \dots \\ p_{1Q} & p_{2Q} & \dots & p_{RQ} \end{bmatrix}_{Q\times R} = \begin{bmatrix} \mathbf{p}_1^T \\ \mathbf{p}_2^T \\ \dots \\ \mathbf{p}_Q^T \end{bmatrix}_{Q\times R}$$

$$\mathbf{Z} = \begin{bmatrix} \mathbf{z}_1 & \mathbf{z}_2 & \dots & \mathbf{z}_2 \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} & \dots & p_{1Q} \\ \dots & \dots & \dots & \dots \\ p_{R1} & p_{R2} & \dots & p_{RQ} \\ 1 & 1 & \dots & 1 \end{bmatrix}_{(R+1)xQ} = \begin{bmatrix} \mathbf{P} \\ \mathbf{1} \end{bmatrix}$$

$$\mathbf{W} = \begin{bmatrix} w_{11} & w_{12} & \dots & w_{1R} \\ w_{21} & w_{22} & \dots & w_{2R} \\ w_{31} & w_{32} & \dots & w_{3R} \\ w_{S1} & w_{S2} & \dots & w_{SR} \end{bmatrix}_{S\times R} = \begin{bmatrix} \mathbf{w}_1^T \\ \mathbf{w}_2^T \\ \dots \\ \mathbf{w}_S^T \end{bmatrix}_{S\times R} \qquad \Theta = \begin{bmatrix} w_{11} & \dots & w_{1R} & b_1 \\ w_{21} & \dots & w_{2R} & b_2 \\ \dots & \dots & \dots & \dots \\ w_{S1} & \dots & w_{SR} & b_S \end{bmatrix}_{S\times(R+1)} = \begin{bmatrix} \boldsymbol{\theta}_1^T \\ \boldsymbol{\theta}_2^T \\ \dots \\ \boldsymbol{\theta}_S^T \end{bmatrix} = \begin{bmatrix} \mathbf{W} & \mathbf{b} \end{bmatrix}$$

$$\mathbf{T} = \begin{bmatrix} t_{11} & t_{12} & \dots & t_{1Q} \\ t_{21} & t_{22} & \dots & t_{2Q} \\ \dots & \dots & \dots & \dots \\ t_{S1} & t_{S2} & \dots & t_{SQ} \end{bmatrix}_{S\times Q} = \begin{bmatrix} \mathbf{t}_1 & \mathbf{t}_2 & \dots & \mathbf{t}_Q \end{bmatrix}_{S\times Q} = \begin{bmatrix} \mathbf{T}_1^T \\ \mathbf{T}_2^T \\ \dots \\ \mathbf{T}_S^T \end{bmatrix}$$

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1Q} \\ a_{21} & a_{22} & \dots & a_{2Q} \\ \dots & \dots & \dots & \dots \\ a_{S1} & a_{S2} & \dots & a_{SQ} \end{bmatrix} = \begin{bmatrix} \mathbf{A}_1^T \\ \mathbf{A}_2^T \\ \dots \\ \mathbf{A}_S^T \end{bmatrix} = \begin{bmatrix} \mathbf{a}_1 & \mathbf{a}_2 & \dots & \mathbf{a}_Q \end{bmatrix}_{S\times Q}$$

$\mathbf{p}_j \triangleq$ $j$-ésimo vector de entrada de dimensão $R$

$p_{ij} \triangleq$ componente $i$ do $j-$ésimo vector de entrada,

$\mathbf{a}_j \triangleq$ $j$-ésimo vector de saída obtida com $j$-ésima entrada $\mathbf{p}_j$

$\mathbf{A}_i^T \triangleq$ vector linha das saídas do neurónio $i$ para todas as entradas de 1 a $Q$

$a_{ij} \triangleq$ componente $i$ da $j$-ésima saída $\mathbf{a}_j$ , saída do $i$-ésimo neurónio

$\mathbf{t}_j \triangleq$ $j$-ésima saída desejada (*target*), quando a entrada $\mathbf{p}_j$ é aplicada

$\mathbf{T}_i^T \triangleq$ vector linha dos alvos do neurónio $i$ para todas as entradas de 1 a $Q$

$t_{ij} \triangleq$ componente $i$ da $j$-ésima saída desejada (alvo) $\mathbf{t}_j$

$w_{ij} \triangleq$ peso entre o neurónio $i$ e o componente $j$ do vector de entrada

Considere-se o exemplo simples de um neurónio linear com duas entradas, o espaço das entradas é o plano. Existe uma fronteira, composta pelo conjunto de pontos em que $n$=0, que é formada por uma reta perpendicular ao vector de pesos, apontando este para a zona positiva.

Partilha algumas das propriedades do percetrão binário, nomeadamente o facto de poder classificar classes linearmente separáveis. Mas tem uma vantagem: o seu algoritmo de treino otimiza a posição da fronteira em relação aos padrões (pontos) de treino.

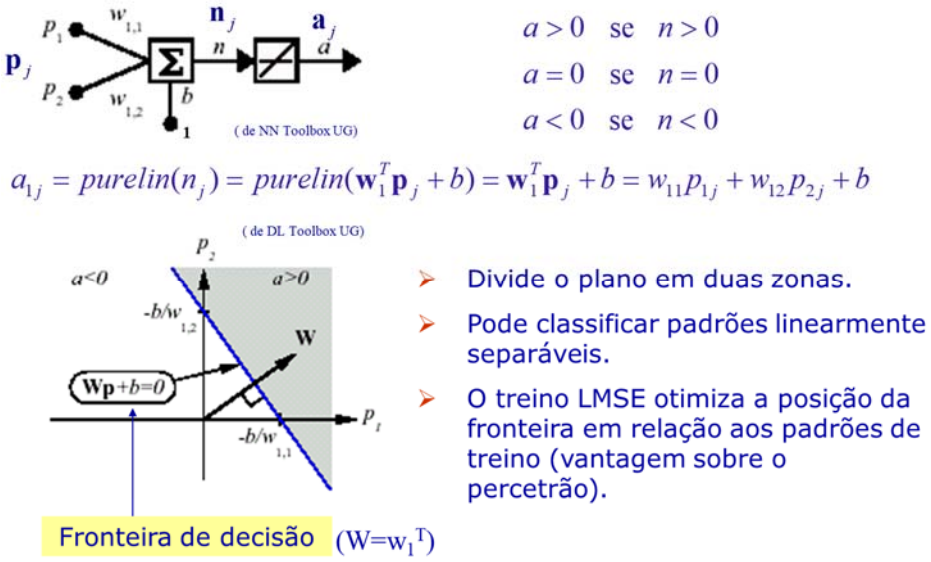
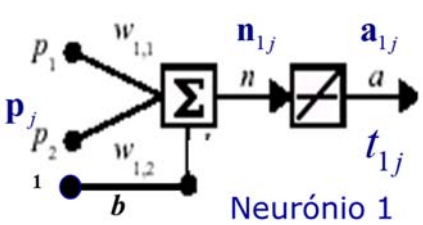



Figura 4.7.2. Exemplo de um neurónio linear.

Para facilitar o desenvolvimento, e como se fez anteriormente, considere-se a polarização como uma entrada constante adicional, obtendo-se a Figura 4.7.3.

Escrevendo a expressão da saída, obtém-se uma equação a três incógnitas. Para a resolver, pecisamos de três equações. E como obtê-las ? Muito simplesmente usando três entradas $z_1$, $z_2$, e $z_3$




$$\boldsymbol{\theta}_1 = \begin{bmatrix} \mathbf{w}_1 \\ b \end{bmatrix} = \begin{bmatrix} w_{11} \\ w_{12} \\ b \end{bmatrix} \qquad \mathbf{z}_j = \begin{bmatrix} p_{1j} \\ p_{2j} \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{p}_j \\ 1 \end{bmatrix}$$

$$a_{1j} = {}_1\mathbf{w}^T \mathbf{p}_j + b = w_{11}p_{1j} + w_{12}p_{2j} + b = \begin{bmatrix} p_{1j} & p_{2j} & 1 \end{bmatrix} \begin{bmatrix} w_{11} \\ w_{12} \\ b \end{bmatrix} = \mathbf{z}_j{}^T \boldsymbol{\theta}_1 =$$

3 incógnitas
x

$$= \begin{bmatrix} w_{11} & w_{12} & b \end{bmatrix} \begin{bmatrix} p_{1j} \\ p_{2j} \\ 1 \end{bmatrix} = \boldsymbol{\theta}_1{}^T \mathbf{z}_j$$

Figura 4.7.3. Equações do neurónio linear, usando a notação estabelecida.

Assim as três equações a três incógnitas serão uma para cada entrada $z_1$, $z_2$, e $z_3$:

1ª $$a_{11} = t_{11} = w_{11}p_{11} + w_{12}p_{21} + b = \begin{bmatrix} w_{11} & w_{12} & b \end{bmatrix} \begin{bmatrix} p_{11} \\ p_{21} \\ 1 \end{bmatrix} = \boldsymbol{\theta}_1{}^T \mathbf{z}_1$$

2ª $$a_{12} = t_{12} = w_{11}p_{12} + w_{12}p_{22} + b = \begin{bmatrix} w_{11} & w_{12} & b \end{bmatrix} \begin{bmatrix} p_{12} \\ p_{22} \\ 1 \end{bmatrix} = \boldsymbol{\theta}_1{}^T \mathbf{z}_2$$

3ª $$a_{13} = t_{13} = w_{11}p_{13} + w_{12}p_{23} + b = \begin{bmatrix} w_{11} & w_{12} & b \end{bmatrix} \begin{bmatrix} p_{13} \\ p_{23} \\ 1 \end{bmatrix} = \boldsymbol{\theta}_1{}^T \mathbf{z}_3$$

Concatenando a três equações na horizontal, obtém-se , em forma matricial,

$$\begin{matrix} \text{1ª} & \text{2ª} & \text{3ª} \\ \begin{bmatrix} a_{11} & a_{12} & a_{13} \end{bmatrix} \end{matrix} = \begin{bmatrix} t_{11} & t_{12} & t_{13} \end{bmatrix} = \begin{bmatrix} w_{11} & w_{12} & b \end{bmatrix} \begin{bmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ 1 & 1 & 1 \end{bmatrix}$$

$$= \boldsymbol{\theta}_1{}^T \begin{bmatrix} \mathbf{z}_1 & \mathbf{z}_2 & \mathbf{z}_3 \end{bmatrix} = \boldsymbol{\theta}_1{}^T \mathbf{Z}$$

$$\mathrm{A}_1^T = T_1^T = \boldsymbol{\theta}_1{}^T \mathbf{Z}$$ (tr. supervisionado, a saída A deve ser igual ao alvo T)

Havendo um alvo (*target*), obrigam-se as saídas a serem iguais ao seu alvo, e obém-se a equação

$$\mathbf{T}_1^T = \boldsymbol{\theta}_1{}^T \mathbf{Z}$$

As incógnitas são os pesos e as polarizações, contidos no vetor $\theta_1$. Para o extrair da equqção anterior, multiplica-se à direita ambos os lados da equação pela inversa de *Z*

$$\mathbf{T}_1^T = \boldsymbol{\theta}_1^T \mathbf{Z} \Rightarrow \mathbf{T}_1^T \mathbf{Z}^{-1} = \boldsymbol{\theta}_1^T \mathbf{Z}\mathbf{Z}^{-1} \Rightarrow \mathbf{T}_1^T \mathbf{Z}^{-1} = \boldsymbol{\theta}_1^T (\mathbf{Z}\mathbf{Z}^{-1}) \Rightarrow \mathbf{T}_1^T \mathbf{Z}^{-1} = \boldsymbol{\theta}_1^T$$

e temos a solução encontrada. O problema é o da existência da inversa de Z. Z é a matriz das entradas e da polarização. A sua inversa existe se as colunas são liearmente independentes. Ora não se pode garantir que tal acontece. Só muito raramente temos essa situação ideal.

Então como resolver ?

Podemos usar mais entradas, obtendo mais equações do que incógnitas e aproveitando a propriedade dos sistemas lineares que estabelece que se temos mais equações do que incógnitas, poderemos não ter uma solução exata mas muitas soluções aproximadas. Uma delas é obtida pela pseudo-inversa de Z de seguida.

Tomando

$$\boldsymbol{\theta}_1^T \mathbf{Z} = \mathbf{T}_1^T$$

multiplicam-se ambos o lados, à direita, pela transposta de Z.

$$\boldsymbol{\theta}_1^T \mathbf{Z}\mathbf{Z}^T = \mathbf{T}_1^T \mathbf{Z}^T$$

multiplicam-se ambos os lados, novamente à direita, por $(ZZ^T)^{-1}$

$$\boldsymbol{\theta}_1^T (\mathbf{Z}\mathbf{Z}^T)(\mathbf{Z}\mathbf{Z}^T)^{-1} = \mathbf{T}_1^T \mathbf{Z}^T (\mathbf{Z}\mathbf{Z}^T)^{-1}.$$

E agora na esquerda temos a matriz identidade a multiplicar teta, ou seja,

$$\boldsymbol{\theta}_1^T \mathrm{I} = \boldsymbol{\theta}_1^T = \mathbf{T}_1^T \mathbf{Z}^T (\mathbf{Z}\mathbf{Z}^T)^{-1}$$

encontrámos a solução.

A $\mathbf{Z}^T (\mathbf{Z}\mathbf{Z}^T)^{-1}$ chama-se pseudo inversa de Moore - Penrose de Z. Usa-se quando temos matrizes retangulates e precisamos de qualquer coisa que funcione como a sua inversa. No Matlab está inplementada pela função *pinv(Z).* Esta solução é para a situação em que temos mais exemplos de treino do que dimensões das entradas, por exemplo 4 entradas de três dimensões (note-se que na nossa notação a polarização é tratada como uma entrada normal, só que é constante, e portanto é uma dimensão adicional, em vez de termos entradasde dimensão *R*, temos entradas de dimensão *R+1*.

Se tomarmos os pesos obtidos e simularmos a rede para as entradas usadas, não se obtém sempre erro nulo, ou pode mesmo nunca se obter erro nulo, mas podem-se obter erros pequenos.

Levanta-se aqui novamente a questão de saber se existe sempre a pseudo-inversa, e se existir qual o seu custo computacional ( a inversão de matrizes é uma das operações mais delicadas, computcionalmente falando).

Por isso, sobretudo quando temos muitas entradas, isto é, muitos padrões de treino, usam-se preferencialmente algoritmos iterativos que tratam uma entrada de cada vez.

Minimização do erro quadrático médio (LMS-Least Mean Square Error)

No treino supervisionado dá-se à rede um conjunto *Q* de exemplos de treino. Cada exemplo tem duas partes, uma entrada p, e o alvo correspondente t, isto é, para um neurónio *i* teremos

$$\{\mathbf{p}_1, t_{i1}\}, \{\mathbf{p}_2, t_{i2}\}, \ldots, \{\mathbf{p}_Q, t_{iQ}\}$$

Dado um conjunto de pesos e de polarizações, passa-se cada entrada na rede e compara-se a saída $a_{ik}$ obtida com a saída desejada (o alvo) $t_{ik}$ , obtendo-se o erro

$$e_{ik}=t_{ik} - a_{ik}\ ,\ k=1,\ \ldots,\ Q$$

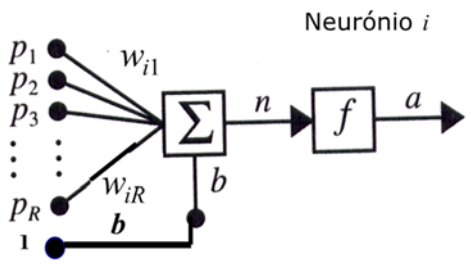
Depois de passar todas as entradas pela rede, somam-se os quadrados dos erros, divide-se pelo número Q de entradas e obtém-se um critério chamado Média da Soma dos Quadrados dos Erros, ou do Inglês *MSE-Mean Square Error,*

$$mse = \frac{1}{Q}\sum_{k=1}^{Q} e_{ik}^{\ 2} = \frac{1}{Q}\sum_{k=1}^{Q} (t_{ik} - a_{ik})^2$$

Elevando os errros ao quadrado, passam a contar tanto os positivos como os negativos. Se se tomasse a soma simples dos erros, poder-se-ia obter uma soma nula com erros muito grandes, uns positivos outros negativos.

Se conseguirmos encontrar o menor valor do *mse* , teremos a aolução ótima do probema de treino da rede, isto é, encontrámos o comjunto ótimo de pesos e de polarizações para o conjunto de daos de treino. Por isso resolvemos o problema de minimização do *mse*. Trata-se portanto de um problema de otimização no sentido mormal do termo. A aprendizagem computacional, como veremos, reduz-se no essencial à resolução de um problema de otimização.

Vejamos mais em pormenor como se faz.




$$\boldsymbol{\theta}_i = \begin{bmatrix} \mathbf{w}_i \\ b \end{bmatrix} = \begin{bmatrix} w_{i1} \\ w_{i2} \\ \ldots \\ w_{iR} \\ b \end{bmatrix}_{R+1} \qquad \mathbf{z}_j = \begin{bmatrix} \mathbf{p}_j \\ 1 \end{bmatrix} = \begin{bmatrix} p_{1j} \\ p_{2j} \\ \ldots \\ p_{Rj} \\ 1 \end{bmatrix}$$

$$a_{ij} = \mathbf{w}_i^T p_j + b = w_{i1}p_{1j} + w_{i2}p_{2j} + \ldots + w_{iR}p_{Rj} + 1b = \boldsymbol{\theta}_i^T \mathbf{z}_j = \mathbf{z}_j^T \boldsymbol{\theta}_i$$

$$e_{ij}^{\ 2} = (t_{ij} - a_{ij})^2 = (t_{ij} - \mathbf{z}_j^{\ T}\boldsymbol{\theta}_i)^2 = (t_{ij} - \mathbf{z}_j^{\ T}\boldsymbol{\theta}_i)^T (t_{ij} - \mathbf{z}_j^{\ T}\boldsymbol{\theta}_i)$$
$$= (t_{ij}^{\ T} - \boldsymbol{\theta}_i^{\ T}\mathbf{z}_j)(t_{ij} - \mathbf{z}_j^{\ T}\boldsymbol{\theta}_i) = t_{ij}^{\ 2} - 2t_{ij}\mathbf{z}_j^{\ T}\boldsymbol{\theta}_i + \boldsymbol{\theta}_i^{\ T}\mathbf{z}_j\mathbf{z}_j^{\ T}\boldsymbol{\theta}_i$$

Para um conjunto de Q pares de treino, a soma de todos os erros quadráticos será

$$\sum_{k=1}^{Q} e_{ik}^{\ 2} = e_{i1}^{\ 2} + e_{i2}^{\ 2} + \ldots + e_{i3}^{\ 2} = \begin{bmatrix} e_{i1} & e_{i2} & \ldots & e_{iQ} \end{bmatrix} \begin{bmatrix} e_{i1} \\ e_{i2} \\ \ldots \\ e_{iQ} \end{bmatrix} = \mathbf{e}_i^{\ T}\mathbf{e}_i$$

Sejam os vectores concatenados das entradas e das saídas

$$\mathbf{Z}=\begin{bmatrix}\mathbf{z}_1 & \mathbf{z}_2 & \dots & \mathbf{z}_Q\end{bmatrix}=\begin{bmatrix}p_{11} & p_{12} & \dots & p_{1Q}\\ p_{22} & p_{22} & \dots & p_{2Q}\\ \dots & \dots & \dots & \dots\\ p_{R1} & p_{R2} & \dots & p_{RQ}\\ 1 & 1 & \dots & 1\end{bmatrix}_{(R+1)\times Q} \qquad \mathbf{Z}^T=\begin{bmatrix}\mathbf{z}_1^T\\ \mathbf{z}_2^T\\ \dots\\ \mathbf{z}_Q^T\end{bmatrix}_{Q\times(R+1)} \qquad \boldsymbol{\theta}_i=\begin{bmatrix}w_{i1}\\ w_{i2}\\ \dots\\ w_{iR}\\ b\end{bmatrix}_{(R+1)}$$

$$\mathbf{A}_i^T=\begin{bmatrix}a_{i1} & a_{i2} & \dots & a_{iQ}\end{bmatrix}_{1\times Q} \qquad \mathbf{T}_i^T=\begin{bmatrix}t_{i1} & t_{i2} & \dots & t_{iQ}\end{bmatrix}_{1\times Q}$$

$$\mathbf{A}_i^T=\boldsymbol{\theta}_i^{\mathbf{T}}\mathbf{Z}$$

$Q \geq R+1$

(Mais equações que incógnitas)

$$\mathbf{e}_i^T=\mathbf{T}_i^T-\mathbf{A}_i^T=\mathbf{T}_i^T-\boldsymbol{\theta}_i^{\mathbf{T}}\mathbf{Z}$$

Agora $$\begin{aligned}F(\boldsymbol{\theta}_i)=\mathbf{e}_i^T\mathbf{e}_i&=(\mathbf{T}_i^T-\mathbf{A}_i^T)(\mathbf{T}_i-\mathbf{A}_i)=(\mathbf{T}_i^T-\boldsymbol{\theta}_i^T\mathbf{Z})(\mathbf{T}_i-\mathbf{Z}^T\boldsymbol{\theta}_i)\\&=\mathbf{T}_i^T\mathbf{T}_i-\mathbf{T}_i^T\mathbf{Z}^T\boldsymbol{\theta}_i-\boldsymbol{\theta}_i^T\mathbf{Z}\mathbf{T}_i+\boldsymbol{\theta}_i^T\mathbf{Z}\mathbf{Z}^T\boldsymbol{\theta}_i\\&=\mathbf{T}_i^T\mathbf{T}_i-2\mathbf{T}_i^T\mathbf{Z}^T\boldsymbol{\theta}_i+\boldsymbol{\theta}_i^T\mathbf{Z}\mathbf{Z}^T\boldsymbol{\theta}_i\end{aligned}$$

Pretende-se minimizar esta expressão em ordem a $\boldsymbol{\theta}_i$:

gradiente: $$\begin{aligned}\nabla F(\boldsymbol{\theta}_i)&=\nabla(\mathbf{T}_i^T\mathbf{T}_i-2\mathbf{T}_i^T\mathbf{Z}^T\boldsymbol{\theta}_i+\mathbf{x}_i^T\mathbf{Z}\mathbf{Z}^T\boldsymbol{\theta}_i)=\\&=\nabla(\mathbf{T}_i^T\mathbf{T}_i)-2\nabla(\mathbf{T}_i^T\mathbf{Z}^T\boldsymbol{\theta}_i)+\nabla(\boldsymbol{\theta}_i^T\mathbf{Z}\mathbf{Z}^T\boldsymbol{\theta}_i)=\\&=-2\mathbf{Z}\mathbf{T}_i+2\mathbf{Z}\mathbf{Z}^T\boldsymbol{\theta}_i\end{aligned}$$

$$-2\mathbf{Z}\mathbf{T}_i+2\mathbf{Z}\mathbf{Z}^T\boldsymbol{\theta}_i=0 \iff \mathbf{Z}\mathbf{Z}^T\boldsymbol{\theta}_i=\mathbf{Z}\mathbf{T}_i \iff$$

$$\boldsymbol{\theta}_i=(\mathbf{Z}\mathbf{Z}^T)^{-1}\mathbf{Z}\mathbf{T}_i^T=(\mathbf{Z}^T)^{+}\mathbf{T}_i$$

$$\Downarrow$$

$$((R+1)\times Q)\times(Q\times(R+1)=(R+1)\times(R+1)$$

Nota:

$\nabla(\mathbf{a}^T\mathbf{x})=\nabla(\mathbf{x}^T\mathbf{a})=\mathbf{a}$

$\nabla(\mathbf{x}^T\mathbf{A}\mathbf{x})=\mathbf{A}\mathbf{x}+\mathbf{A}^T\mathbf{x}=2\mathbf{A}\mathbf{x}$

( se A simétrica)

Será mínimo? Será máximo? Será ponto cela?

Segunda derivada

$$\frac{\partial^2(\mathbf{e}_i^T\mathbf{e}_i)}{\partial\boldsymbol{\theta}_i^{\,2}} = \frac{\partial(-2\mathbf{Z}\mathbf{T}_i + 2\mathbf{Z}\mathbf{Z}^T\boldsymbol{\theta}_i)}{\partial\boldsymbol{\theta}_i} = \mathbf{Z}\mathbf{Z}^T$$

$\mathbf{Z}\mathbf{Z}^T > \mathbf{0}$, tem um mínimo global único

$\mathbf{Z}\mathbf{Z}^T \geq \mathbf{0}$, tem um mínimo global fraco ou não tem ponto de estacionaridade.

Interpretação da condição de mínimo:

$$F(\mathbf{x}_i) = \mathbf{T}_i^T\mathbf{T}_i - 2\mathbf{T}_i^T\mathbf{Z}^T\boldsymbol{\theta}_i + \mathbf{x}_i^T\mathbf{Z}\mathbf{Z}^T\boldsymbol{\theta}_i$$

Se as entradas da rede forem vectores aleatórios, o erro será aleatório e então a função objectivo a minimizar será

$$F(\boldsymbol{\theta}_i) = E[\mathbf{T}_i^T\mathbf{T}_i - 2\mathbf{T}_i^T\mathbf{Z}^T\boldsymbol{\theta}_i + \boldsymbol{\theta}_i^T\mathbf{Z}\mathbf{Z}^T\boldsymbol{\theta}_i]$$

$$F(\boldsymbol{\theta}_i) = E[\mathbf{T}_i^T\mathbf{T}_i] - 2E[\mathbf{T}_i^T\mathbf{Z}^T]\boldsymbol{\theta}_i + \boldsymbol{\theta}_i^T E[\mathbf{Z}\mathbf{Z}^T]\boldsymbol{\theta}_i$$

Matriz de correlação entre as entradas e as saídas desejadas

Matriz de auto-correlação das entradas

Uma matriz de correlação ou é positiva definida (>0) ou é semidefinida positiva (≥0).

Se as entradas aplicadas à rede não forem correlacionadas, a matriz de correlação $\mathbf{ZZ}^T$ é diagonal, sendo os elementos da diagonal os quadrados das entradas. Nestas condições a matriz de correlação é positiva definida e o mínimo global existe e é único.

São as características do conjunto das entradas de treino que determinam a existência ou não de um mínimo global único.

Para uma rede com S neurónios,

$$\Theta = \begin{bmatrix} \boldsymbol{\theta}_1^T \\ \dots \\ \boldsymbol{\theta}_S^T \end{bmatrix} = \begin{bmatrix} w_{11} & \dots & w_{1R} & b_1 \\ w_{21} & \dots & w_{2R} & b_2 \\ \dots & \dots & \dots & \dots \\ w_{S1} & \dots & ..w_{SR}. & b_S \\ & & \dots & \end{bmatrix}_{S\times(R+1)}$$

$$\mathbf{T} = \begin{bmatrix} \mathbf{t}_1 & \mathbf{t}_2 & \dots & \mathbf{t}_Q \end{bmatrix} \xrightarrow{\text{neurónio}} \begin{bmatrix} t_{11} & t_{12} & \dots & t_{1Q} \\ t_{21} & t_{22} & \dots & t_{2Q} \\ \dots & \dots & \dots & \dots \\ t_{S1} & t_{S2} & \dots & t_{SQ} \end{bmatrix}_{S\times Q} = \begin{bmatrix} T_1^T \\ T_2^T \\ T_S^T \end{bmatrix}$$

$$\mathbf{T}^T = \begin{bmatrix} T_1^T \\ T_2^T \\ T_S^T \end{bmatrix}^T = \begin{bmatrix} T_1 & T_2 & \dots & T_S \end{bmatrix} \qquad (Q > (R+1))$$

$$\boldsymbol{\theta}_i = (\mathbf{Z}\mathbf{Z}^T)^{-1}\mathbf{Z}\mathbf{T}_i = (\mathbf{Z}^T)^{+}\mathbf{T}_i$$

$$\begin{aligned}\boldsymbol{\Theta}^T &= \left[\boldsymbol{\theta}_1 \quad \boldsymbol{\theta}_2 \quad \ldots \quad \boldsymbol{\theta}_S\right] = \left[(\mathbf{Z}\mathbf{Z}^T)^{-1}\mathbf{Z}\mathbf{T}_1 \quad (\mathbf{Z}\mathbf{Z}^T)^{-1}\mathbf{Z}\mathbf{T}_2 \quad \ldots \quad (\mathbf{Z}\mathbf{Z}^T)^{-1}\mathbf{Z}\mathbf{T}_S\right] = \\ &= (\mathbf{Z}\mathbf{Z}^T)^{-1}\mathbf{Z}\left[\mathbf{T}_1 \quad \mathbf{T}_2 \quad \ldots \quad \mathbf{T}_S\right] \\ &= (\mathbf{Z}\mathbf{Z}^T)^{-1}\mathbf{Z}\mathbf{T}^T\end{aligned}$$

$$\boldsymbol{\Theta} = ((\mathbf{Z}\mathbf{Z}^T)^{-1}\mathbf{Z}\mathbf{T}^T)^T = \mathbf{T}\mathbf{Z}^+$$

Possível? Existe $(\mathbf{Z}\mathbf{Z}^T)^{-1}$ ? Peso computacional ?

O cálculo da inversa, para um grande número *R* de entradas, é difícil. Não haverá um algoritmo computacionalmente mais simples ?

A técnica mais básica de otimização é a usada para o cálculo dos mínimos (e máximos) de funções: o método do gradiente (ou da derivada). Aqui leva ao algoritmo de treino LMSE (*Least Mean Square Error*) que é simplesmente a minimização do *mse.*

4.7.2. Treino iterativo da ADALINE

Um algoritmo iterativo inicializa-se num conjunto de pesos e polarizações, escolhido ou com base no conhecimento do problema, ou aleatoriamente (por um gerador de números aleatórios).

Sendo *k* o índice de iteração, teremos na inicialização

$$k=0 \quad w_{ij}^{k} = w_{ij}^{0}$$

Agora inicia-se o processo iterativo:

Para *k* de 1 a *Q* fazer (caso do neurónio *i*) *:*

Para cada par de treino $( a_{ik}, t_{ik})$

1º calcular o erro $e_{ik}^2=(t_{ik}-a_{ik}))^2$

2º calcular o gradiente $\nabla$ do quadrado do erro em relação aos pesos e à polarização

$$\left[\nabla e_{ik}{}^2\right] = \frac{\partial e_{ik}{}^2}{\partial w_{ij}{}^k} = 2e_{ik}\frac{\partial e_{ik}}{\partial w_{ij}{}^k} \quad \text{para os pesos} \quad j = 1,2,...,R$$

$$\left[\nabla e_{ik}{}^2\right]_{R+1} = \frac{\partial e_{ik}{}^2}{\partial b} = 2e_{ik}\frac{\partial e_{ik}}{\partial b} \quad \text{para a polarização}$$

Calculando as derivadas pelas regras usuais, tendo em atenção que agora temos vetores e matrizes,

$$\frac{\partial e_{ik}}{\partial w_{ij}{}^k} = \frac{\partial [t_{ik} - a_{ik}]}{\partial w_{ij}{}^k} = \frac{\partial}{\partial w_{ij}{}^k}[t_{ik} - (\mathbf{w}_i{}^{\mathbf{T}k}\mathbf{p}_k + b)]$$

$$= \frac{\partial}{\partial w_{ij}{}^k}\left[t_{ik} - \left(\sum_{j=1}^{R} w_{ij}{}^k p_{jk} + b\right)\right] = -p_{jk} \qquad j = 1,2,...R$$

$$\frac{\partial e_{ik}}{\partial b} = -1, \quad \text{para a polarização } b$$

Teremos finalmente,

$$\left[\nabla e_{ik}{}^2\right] = -2e_{ik}\mathbf{z}_k$$

em que como anteriormente

$$\mathbf{z}_k = \begin{bmatrix} p_{1k} \\ p_{2k} \\ ... \\ p_{Rk} \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{p}_k \\ 1 \end{bmatrix}$$

Note-se que no desenvolvimento anterior supôs-se que a função de ativação é a linear, cuja derivada é 1 (*a*=*n*, se bem se lembra o leitor). Se fosse outra, por exemplo a sigmoide unipolar, sig, teria que se calcular a sua derivada ,dado que neste caso a=sig(n) e o erro seria $e_{ik}$=$t_{ik}$ – sig( $\mathbf{w}_i{}^{\mathbf{T}k}\mathbf{p}_k + b$ ).

3º Aplicar o método do gradiente para minimizar $F(\boldsymbol{\theta}_i)$

$$\boldsymbol{\theta}_i{}^{k+1} = \boldsymbol{\theta}_i{}^k - \alpha\nabla F(\boldsymbol{\theta}_i{}^k)$$

$$\boldsymbol{\theta}_i = \begin{bmatrix} \mathbf{w}_i \\ b \end{bmatrix}$$

$$F(\boldsymbol{\theta}_i{}^k) = e_{ik}{}^2$$

Resultando em

$$\left[\nabla e_{ik}{}^{2}\right] = -2e_{ik}\mathbf{z}_k = \text{-}2e_{ik}\begin{bmatrix}\mathbf{p}_k \\ 1\end{bmatrix}$$

E temos finalmente o processo iterativo, para o caso do neurónio *i*,

$$\mathbf{w}_i{}^{k+1} = \mathbf{w}_i{}^{k} + 2\alpha e_{ik}\mathbf{p}_k$$

$$b_i^{k+1} = b_i^{k} + 2\alpha e_{ik}$$

Mas

$$\mathbf{w}_i^{k+1} = \mathbf{w}_i{}^{k} + 2\alpha e_{ik}\mathbf{p}_k \Rightarrow (\mathbf{w}_i^T)^{k+1} = (\mathbf{w}_i{}^T)^k + 2\alpha e_{ik}\mathbf{p}_k^T$$

Para uma camada S de neurónios, teremos

$$\begin{bmatrix}\mathbf{w}_1^T \\ \dots \\ \mathbf{w}_S^T\end{bmatrix}^{k+1} = \begin{bmatrix}\mathbf{w}_1^T \\ \dots \\ \mathbf{w}_S^T\end{bmatrix}^{k} + 2\alpha\begin{bmatrix}e_{1k} \\ \dots \\ e_{sk}\end{bmatrix}\mathbf{p}_k^T \qquad \begin{bmatrix}b_1 \\ \dots \\ b_S\end{bmatrix}^{k+1} = \begin{bmatrix}b_1 \\ \dots \\ b_S\end{bmatrix}^{k} + 2\alpha\begin{bmatrix}e_{1k} \\ \dots \\ e_{sk}\end{bmatrix}$$

Ou seja,

$$\mathbf{W}^{k+1} = \mathbf{W}^{k} + 2\alpha\mathbf{e}_k\mathbf{p}_k{}^T$$

$$\mathbf{b}^{k+1} = \mathbf{b}^{k} + 2\alpha\mathbf{e}_k$$

Ou ainda se definirmos

$$\Theta = \begin{bmatrix}\mathbf{W} & \mathbf{b}\end{bmatrix} \quad (\mathbf{z}_k)^T = \begin{bmatrix}(\mathbf{p}_k)^T & 1\end{bmatrix}$$

Poderemos escrever a forma mais compacta

$$\Theta^{k+1} = \Theta^{k} + 2\alpha\mathbf{e}_k\mathbf{z}_k^T$$

Este algoritmo RLMS (*Recursive Mean Least Squares*) para funções de ativação lineares é conhecido também como a regra de Widrow-Hoff ( ver em Hagan and Col.).

O RLMS é uma aproximação do método do gradiente, dado que o gradiente calculado em cada iteração é uma aproximação do verdadeiro gradiente. A sua convergência depende de $\alpha$, chamado coeficiente de aprendizagem.

A qualidade dos dados de entrada também influencia a convergência. Se os sucessivos vetores de entrada foram estatisticamente independentes uns dos outros, e se se $\theta(k)$ for estatisticamente independente de **z**(k), então converge para um mínimo global.

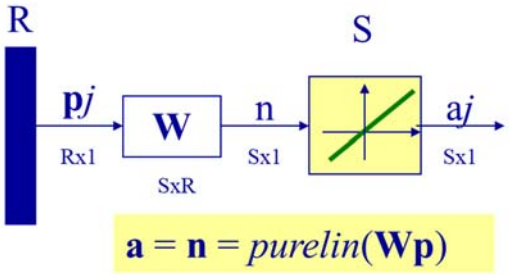
Um dos problemas com que se defronta o projetista é a fixação do valor de $\alpha$. Infelizmente não há nenhuma regra exata, e tem que se usar a técnica da tentativa e erro, até se encontrar um $\alpha$ que dê bons resultados, ou pelo menos resultados aceitáveis no contexto do nosso problema. Há uma regra empírica que diz que o $\alpha$ deve sitiar-se no intervalo

$$0 < \alpha < \frac{1}{\lambda_{max}}$$

Em que $\lambda_{max}$ é o máximo valor próprio da matriz de correlação das entradas $R=ZZ^T$. Ver mais em Hagan 10-9.

### 4.7.3. A Memória Associativa

Chama-se Memória Associativa a uma camada de neurónios com função de ativação linear, e sem polarização, ilustrada na Fig. 4.7.3.1.




$$\mathbf{a}_j = \mathbf{W}\mathbf{p}_j$$

$$a_{ij} = \sum_{k=1}^{R} w_{ik} p_{kj}$$

Treina-se com Q pares de vetores protótipo de entrada/saída ,

$$\{\mathbf{p}_1, \mathbf{t}_1\}, \{\mathbf{p}_2, \mathbf{t}_2\}, \ldots, \{\mathbf{p}_Q, \mathbf{t}_Q\}$$

Depois dando-lhe um dos protótipos de entrada, a saída é a correta. Dando-lhe uma entrada próxima de um protótipo, a saída será também próxima da saída correspondente: um pequeno desvio na entrada produz um pequeno desvio na saída. Pode-se dizer que se lembra de uma entrada e associa-lhe uma entrada parecida: um pequeno desvio na entrada produz um pequeno desvio na saída.

Trata-se de treino supervisionado em que WP=A e se deseja que A=T ; então substitua-se A por T na equação, ficando, WP=T

Se **R > Q** (mais entradas do que protótipos), **P** é retangular, com mais linhas que colunas . Temos um sistema com mais incógnitas do que equações.

Se **P** é de característica máxima ( se as suas colunas forem linearmente independentes), podemos usar a pseudo inversa para encontrar uma solução **exata** para o sistema de equações, usando a pseudo inversa de Moore-Penrose $P^+$.

$$\mathbf{W}\,\mathbf{P} = \mathbf{T} \Rightarrow$$

Usando a técnica que vimos anteriormente

$$\mathbf{W} = \mathbf{T}\mathbf{P}^{+} = \mathbf{T}(\mathbf{P}^{T}\mathbf{P})^{-1}\mathbf{P}^{T}$$

Se **R=Q** , o número *Q* de protótipos for igual ao número *R* de entradas da rede, e se os protótipos forem linearmente independentes então **P** é invertível.

Resolvendo em ordem a **W**, usando $\mathbf{P}^{-1}$:

$$\mathbf{W}\mathbf{P} = \mathbf{T} \Leftrightarrow \mathbf{W}\mathbf{P}\mathbf{P}^{-1} = \mathbf{T}\mathbf{P}^{-1} \Leftrightarrow \mathbf{W}(\mathbf{P}\mathbf{P}^{-1}) = \mathbf{T}\mathbf{P}^{-1}$$
$$\Rightarrow \mathbf{W} = \mathbf{T}\mathbf{P}^{-1}, \qquad \text{se } \mathbf{P} \text{ for invertível}$$

Se **R < Q** (mais protótipos do que entradas), **P** é retangular, com mais colunas que linhas . Temos um sistema com mais equações do que incógnitas.

Não há em geral uma solução exata. Pode-se apenas encontrar uma solução aproximada, que pode ser encontrada pela pseudo-inversa de Moore-Penrose e que minimiza a soma dos erros quadráticos:

$$\mathbf{W}\,\mathbf{P} = \mathbf{T} \Rightarrow$$

$$\mathbf{W} = \mathbf{T}\mathbf{P}^{+} = \mathbf{T}\mathbf{P}^{T}(\mathbf{P}\mathbf{P}^{T})^{-1}$$

Note-se que aqui $\mathbf{P}^{+}$ não se calcula do mesmo modo que no caso anterior R > Q , dadas as diferentes retangularidades das matrizes. Dá uma solução aproximada que minimiza a soma dos quadrados dos erros.

Note-se que as fórmulas da memória associativa são um caso particular das da Adaline, mas aqui como não há polarização em vez do $\Theta$ da Adaline temos o W da memória associativa.

A versão recursiva será a mesma da Adaline.

Há um algoritmo histórico, **a regra de Hebb**, que tem uma forma diferente

$$\mathbf{W}^{k+1} = \mathbf{W}^{k} + \alpha \cdot \mathbf{t}_{k} \cdot \mathbf{p}_{k}^{T}, \; k = 1, 2, \ldots, Q$$

A regra Adaline advém do desenvolvimento matemático para minimização do erro quadrático médio (LMSE). A regra de Hebb resulta de um princípio empírico enunciado pelo neurobiologista Hebb na sua obra *The Organization of Behavior*, 1949.

4.8 Podemos agora comparar as regras de aprendizagem para uma camada de neurónios.

## Tipos de problemas

- Classificação (reconhecimento de padrões)
  - Número de protótipos não superior ao número de entradas Q≤R, : regra de Hebb ou da pseudo-inversa
  - Número de protótipos superior ao número de entradas, Q>R: pseudo-inversa (LMS), ou a sua versão recursiva
- Aproximação de funções: LMSE (Widrow-Hoff)

1. **Percetrão**: funções de ativação saturadas (hardlim, hardlims, satlin, satlins), sendo Θ a matriz de pesos:

$$\Theta^{k+1} = \Theta^{k} + \alpha \mathbf{e}_k \mathbf{z}_k^T$$

2. **Memórias associativas** lineares: funções de ativação lineares, sem polarização ( Θ é aqui apenas W e Z apenas P ; o número de padrões de treino não é superior ao número de características (entradas), Q<R.

$$\Theta^{k+1} = \Theta^{k} + \alpha \cdot \mathbf{t}_{k+1} \cdot \mathbf{z}_{k+1}^T, \quad k < Q$$

Regra de Hebb iterativa

$$\Theta = \mathbf{T}\mathbf{Z}^T$$

(Hebb não iterativa, se os padrões de treino são ortonormais)

$$\Theta = \mathbf{T}(\mathbf{Z}^T\mathbf{Z})^{-1}\mathbf{Z}^T = \mathbf{T}\mathbf{Z}^+ \quad \text{se existir } (\mathbf{Z}^T\mathbf{Z})^{-1}$$

Pseudo-inversa : se os padrões de treino são linearmente independentes

3. **ADALINE**: neurónios lineares, com polarização. O número de dados de treino é superior ao número de entradas da rede (Q>R).

$$\Theta^{k+1} = \Theta^{k} + 2\alpha \mathbf{e}_k \mathbf{z}_k^T$$

Algoritmo LMSE recursivo

$$\Theta = \mathbf{T}\mathbf{Z}^T(\mathbf{Z}\mathbf{Z}^T)^{-1} = \mathbf{T}\mathbf{Z}^+ \quad \text{se existir } (\mathbf{Z}\mathbf{Z}^T)^{-1}$$

Algoritmo LMSE não recursivo

## The general Gradient Method

Iterative minimization of a general function g($x$) with respect to $x$ :

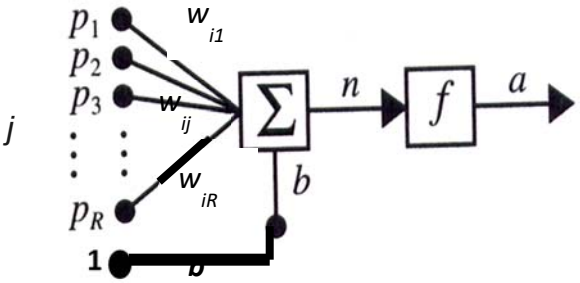
At iteration $k$,

$$x^{k+1} = x^k - \alpha \frac{\partial g}{\partial x}\Big|_{x^k} = x^k - \alpha \nabla g(x)\Big|_{x^k}$$

$\alpha$ is a constant to be fixed by the user.

This is the **gradient method** and is the base of many ML learning algorithms.

### 4.10. One layer with any activation function




$j$

$t$ target

$$a = f(n)$$
$$e = t - a$$
$$F = e^2 \quad \text{to be minimized}$$

For the weight $w_{ij}$ between neuron $i$ and input $p_j$

$$\frac{\partial F}{\partial w_{ij}} = \nabla e^2\Big|_{w_{ij}} = \frac{\partial F}{\partial e}\frac{\partial e}{\partial a}\frac{\partial a}{\partial n}\frac{\partial n}{\partial w_{ij}} =$$

$$= 2e.(-1).\dot{f}.p_j = -2e.\dot{f}.p_j$$

For the bias b

$$\frac{\partial F}{\partial b} = \nabla e^2\Big|_{b} = \frac{\partial F}{\partial e}\frac{\partial e}{\partial a}\frac{\partial a}{\partial n}\frac{\partial n}{\partial b} =$$

$$= 2e.(-1).\dot{f}.1 = -2e.\dot{f}.1$$

for the input $\mathbf{p}_q$ *q=1, ...., **Q***

$$\frac{\partial F}{\partial w_{ij}} = \nabla e^2 \Big|_{w_{ij}} = \frac{\partial F}{\partial e}\frac{\partial e}{\partial a}\frac{\partial a}{\partial n}\frac{\partial n}{\partial w_{ij}}$$

at iteration *k :*

$$\frac{\partial e_{iq}}{\partial w_{ij}^{\,k}} = \frac{\partial [t_{iq} - a_{iq}]}{\partial w_{ij}^{\,k}} = \frac{\partial}{\partial w_{ij}^{\,k}}[t_{iq} - f(n^k)]$$

chain rule

$$= \frac{\partial}{\partial w_{ij}^{\,k}}\left[t_{iq} - f\left(\sum_{j=1}^{R} w_{ij}^{\,k} p_{jq} + b^k\right)\right] = -\frac{\partial f}{\partial n_q}.\frac{\partial n_q^k}{\partial w_{ij}^k} \qquad j = 1, 2, \ldots R$$

$$= -\dot{f}\,.p_{jq} \qquad j = 1, 2, \ldots R$$

$$\frac{\partial e_{iq}}{\partial b} = -\dot{f}\,.1, \quad \text{for the bias } b$$

The gradient of the squared error, for all weights and bias, will come

$$\left[\nabla e_{iq}^{\,2}\right] = -2e_{iq}.\dot{f}.\mathbf{z}_q = -2e_{iq}.\dot{f}.\begin{bmatrix}\mathbf{p}_q \\ 1\end{bmatrix}$$

and the update formula will be, for a layer of neurons,

$$\boldsymbol{\Theta}^{k+1} = \boldsymbol{\Theta}^k + 2\alpha \mathbf{e}_q.\dot{f}.\mathbf{z}_q^T$$

This is the general iterative gradient method for one layer, also known as the LMSE (least minimum squared error).

The Widraw-Hoff algorithm is the particular case of the gradient method when the activation function is linear, and so its derivative is one.

After passing through all the inputs (1.... Q) we say that an epoch of training is complete.

The process restarts with the actual parameters and again for the inputs (1 ... Q), the second epoch is ended.

And so on, until the convergence criteria is reached or a fixed maximum number of epochs is attained.

- **A aprendizagem de Widrow-Hoff (WH) é usada em RN Adaline (neurónios lineares)**

- **A principal vantagem de WH reside na sua implementação recursiva**

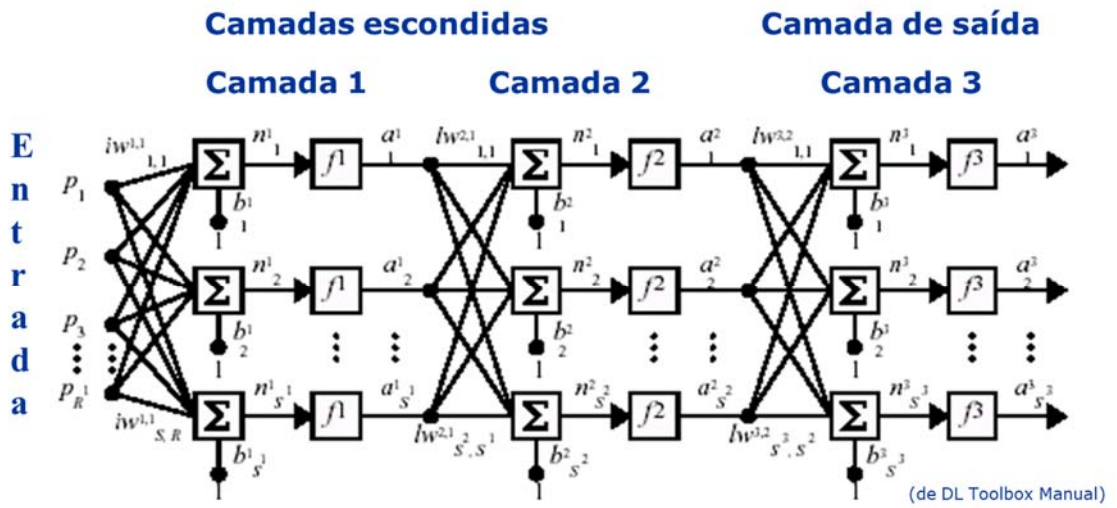
- **É necessário que o conjunto de dados de treino seja preparado com cuidado: os seus elementos devem ser estatisticamente independentes.**

- **É muito adequada para aproximação de funções e para a aprendizagem de modelos de sistemas dinâmicos.**

- **Em classificação, a ADLINE só resolve problemas linearmente separáveis.**

### 4.8. Redes multicamada

As redes multicamada mais usuais, em que os sinais se propagam sempre para a frente (MLFNN- *Multi Layer Feedforward Neural Network*) têm a topografia e a notação que já vimos, no caso de três camadas ilustrada pela Fig. 4.8.1.



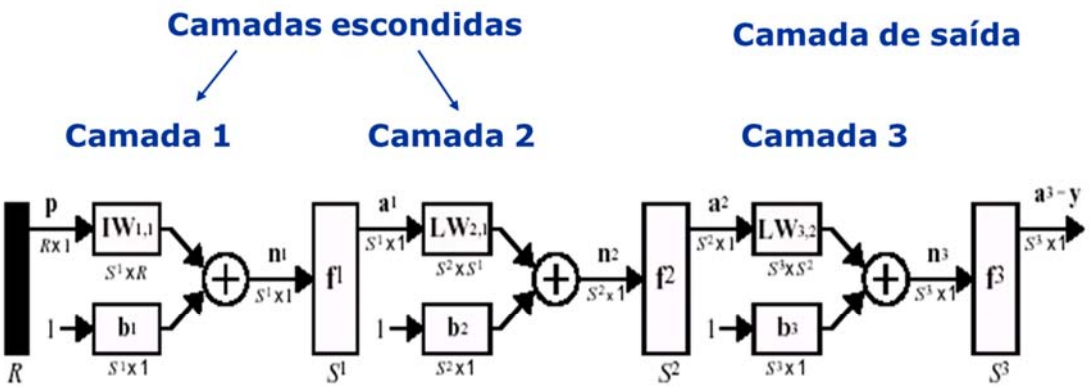

$$\mathbf{a}^1 = \mathbf{f}^1(\mathbf{IW}^{1,1}\mathbf{p} + \mathbf{b}^1) \qquad \mathbf{a}^2 = \mathbf{f}^2(\mathbf{LW}^{2,1}\mathbf{a}^1 + \mathbf{b}^2) \qquad \mathbf{a}^3 = \mathbf{f}^3(\mathbf{LW}^{3,2}\mathbf{a}^2 + \mathbf{b}^3)$$

$$\mathbf{a}^3 = \mathbf{f}^3(\mathbf{LW}^{3,2}\mathbf{f}^2(\mathbf{LW}^{2,1}\mathbf{f}^1(\mathbf{IW}^{1,1}\mathbf{p} + \mathbf{b}_1) + \mathbf{b}_2 + \mathbf{b}_3)$$

Figura 4.8.1.Rede multicamada para a frente

Dá-se-lhe uma entrada e os seus sinais vão passando pelos pesos e pelas funções de ativação até chegarem à última camada que pode ter um número qualquer de neurónios. Posteriormente veremos que há redes em que os sinais se propagam também para trás

através de caminhos de retroação (feedback). Mas aqui interessam-nos apenas as para a frente, que numa notação mais compacta podem ser representadas pela Fig. 4.8.2.




$$\mathbf{a}^1 = \mathbf{f}^1(\mathbf{IW}^{1,1}\mathbf{p}+\mathbf{b}^1) \qquad \mathbf{a}^2 = \mathbf{f}^2(\mathbf{LW}^{2,1}\mathbf{a}^1+\mathbf{b}^2) \qquad \mathbf{a}^3 = \mathbf{f}^3(\mathbf{LW}^{3,2}\mathbf{a}^2+\mathbf{b}^3)$$

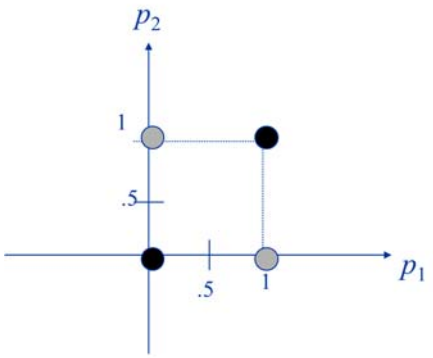
$$\mathbf{a}^3 = \mathbf{y} = \mathbf{f}^3(\mathbf{LW}^{3,2}\mathbf{f}^2(\mathbf{LW}^{2,1}\mathbf{f}^1(\mathbf{IW}^{1,1}\mathbf{p}+\mathbf{b}_1)+\mathbf{b}_2+\mathbf{b}_3)$$

Figura 4.8.2. Rede com três camadas com representação com vetores e matrizes.

4.8.1. AS MLNN em reconhecimento de padrões.

Vimos anteriormente que o percetrão não é capaz de resolver o OU exclusivo (XOR). Este facto foi um dos incentivos à invenção das redes com várias camadas. Como estamos a tratar de variáveis lógicas, temo sum problema de reconhecimento de padrões (o que não quer dizer que o reconhecimento de padrões só contemple casos com variáveis lógicas).

Vejamos o caso do XOR, cuja tabela de verdade é dada na Fig. 4.8.3.




| $p_1$ | $p_2$ | $p_1$**XOR**$p_2$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

$p$ $t$

Nota: $p_1$**XOR**$p_2$ =($p_1$**OR**$p_2$)**AND**($p_1$**NAND**$p_2$)

Figura 4.8.3. Exemplo do XOR

A solução para o XOR é uma rede com duas camadas e neurónios binários. A primeira camada tem dois neurónios e a segunda um.

Os da camada um devem dividir o plano de tal modo que em cada parte existam pontos de apenas uma classe. Isto obriga a três zonas, criadas pelos neurónios seguintes.

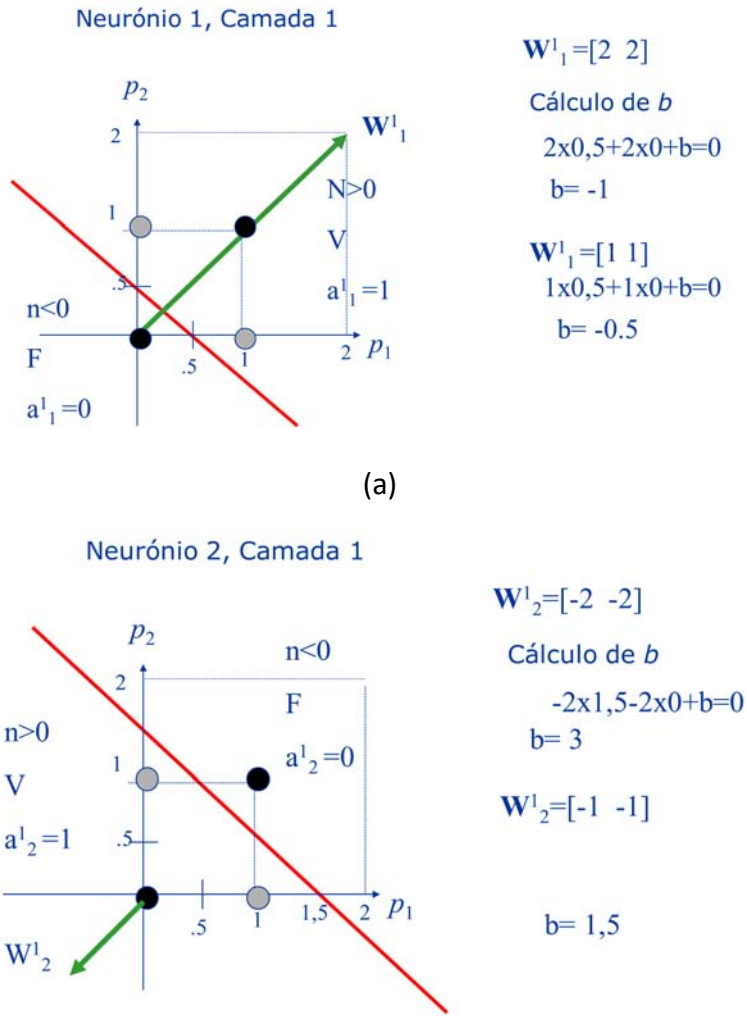



Figura 4.8.4. Os neurónios da camada 1. As fronteiras foram escolhidas empiricamente.

Juntando o efeito dos dois neurónios obtém-se a Fig. 4.8.5

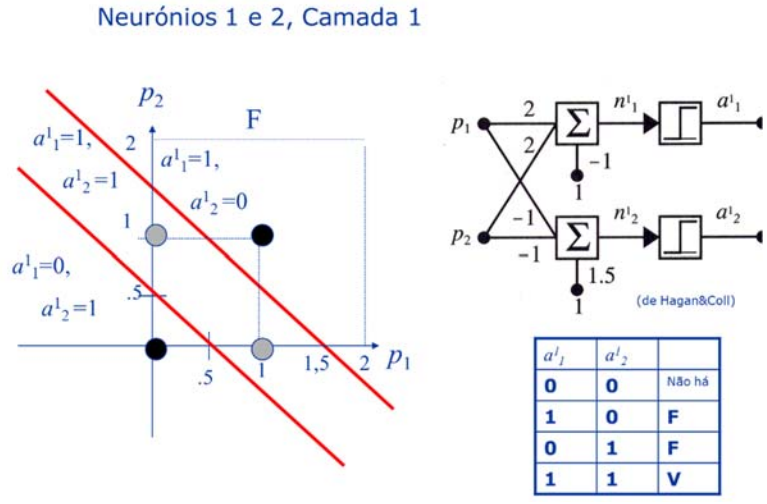



Figura 4.8.5. As três zonas criadas no plano não têm mistura de classes.

Na Fig. 4.8.5 existem três zonas, falta a zona (0,0). Mas isso não é problema como veremos.

Construída a primeira camada é necessário acrescentar o neurónio da segunda, que terá como entradas as saídas dos dois neurónios da primeira camada. Existem muitas soluções, uma delas ilustrada na Fig. 4.8.6. Os pesos da 2ª camada são unitários, e depois ajusta-se a polarização para que o resultado dê certo.

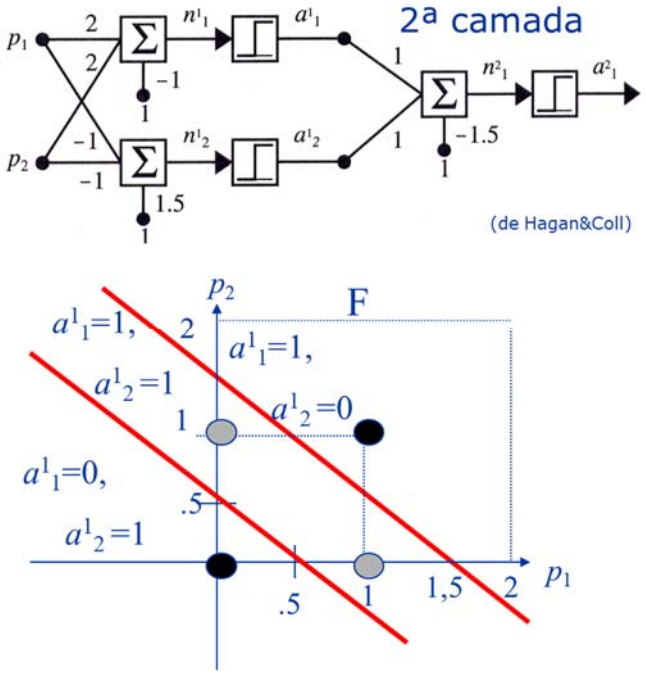



Figura 4.8.6. Rede completa para o XOR. Note-se que se as entradas são (0,0), a saída será 0, concordante com a tabela lógica do XOR.

4.8.2. As MLNN em aproximação de funções

As redes multicamada têm um grande potencial para aproximar funções. Vejamos um exemplo ilustrativo.

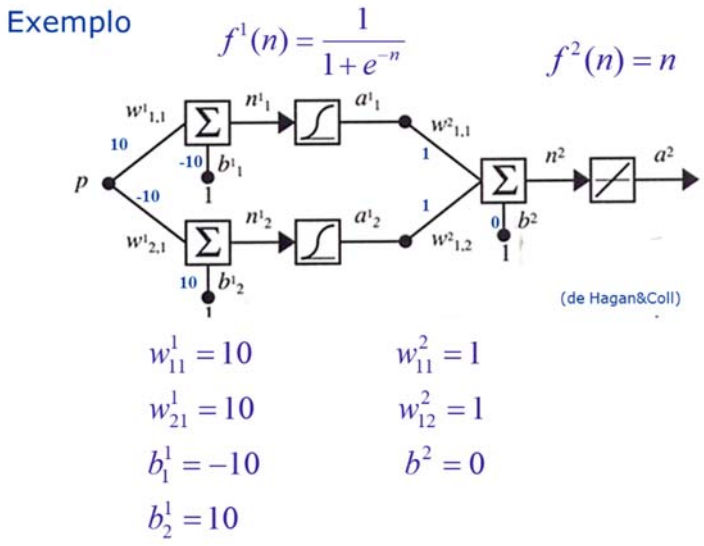



Figura 4.8.7. Exemplo de aproximação de uma função univariável.

Seja uma função univariável $a_2=g(p)$ implementada pela rede da Fig.4.8.7. NA primeira camada temos dois neurónios com função de ativação sigmoidal unipolar a segunda camada com um neurónio com função de ativação linear. Os pesos e as polarizações estão indicados na figura.

A função que a rede implementa é definida por

$$a^2 = n^2 = w_{11}^2 a_1^1 + w_{12}^2 a_2^1 + b^2$$

$$= w_{11}^2 \left(\frac{1}{1+e^{-n_1^1}}\right) + w_{12}^2 \left(\frac{1}{1+e^{-n_2^1}}\right) + b^2 =$$

$$= w_{11}^2 \left(\frac{1}{1+e^{-(w_{11}^1 p + b_1^1)}}\right) + w_{12}^2 \left(\frac{1}{1+e^{-(w_{21}^1 p + b_2^1)}}\right) + b^2$$

Repare-se na complexa não linearidade da função. A entrada (variável independente) está nos expoentes das exponenciais dos denominadores.

Se dermos uma entrada linear, como ilustrado na Fig. 4.8.8, como será a saída ?

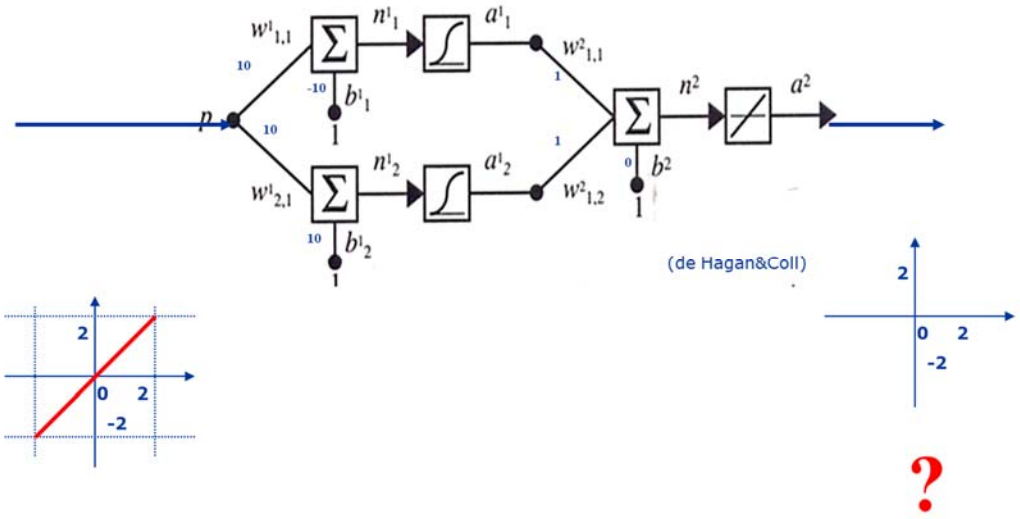



Figura 4.8.8. Uma entrada linear que saída produz ?

Usando os pesos e as polarizações dados na Fig. 4.8.7. e fazendo os cálculos separados para os dois neurónios da primeira camada, somando-os depois, obtém-se a Fig. 4.8.9.

É interessante vermos agora a influência na saída de cada peso e cada polarização.

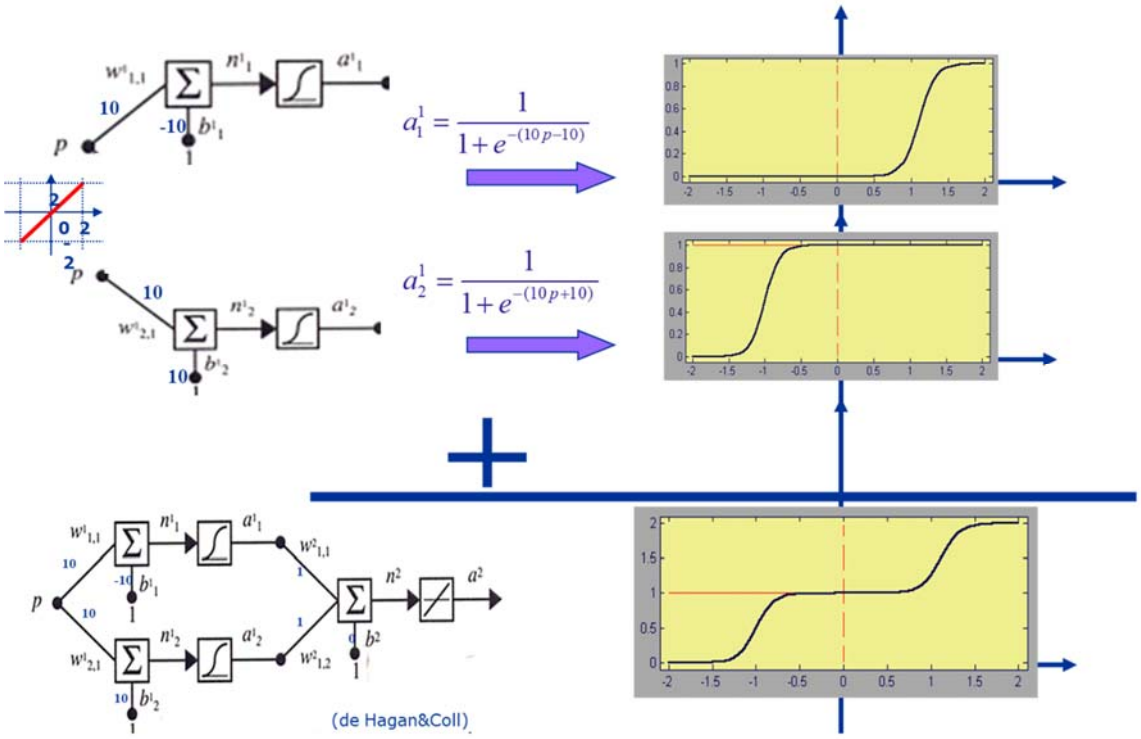



Figura 4.8.9. Cálculo da saída para a entrada linear

Por exemplo variando $b^1_2$ , $w^2_{1,1}$ , $w^2_{1,2}$ , $b^2$, obtêm-se as curvas da Figura 4.8.10.

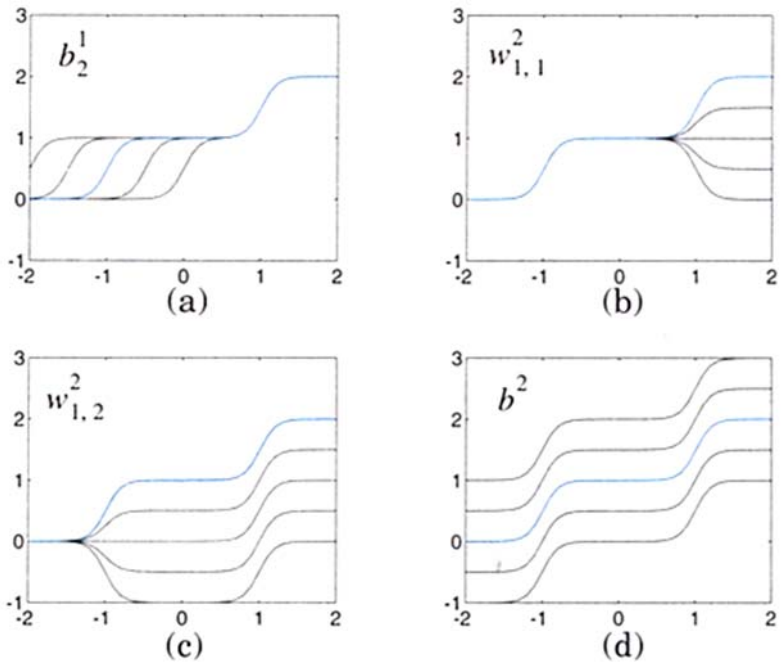



Figura 4.8.10. Influência na saída de diversos parâmetros da rede (Hagan).

Observe-se o elevado número e tipo de funções que é possível obter apenas com estes quatro graus de liberdade. Na prática temos centenas de graus de liberdade, e por isso uma flexibilidade extraordinária.

Pode-se provar que uma camada sigmoidal escondida seguida de uma camada linear de saída, pode aproximar qualquer função de interesse com qualquer precisão, desde que a camada escondida seja provida com um número suficiente de neurónios.

4.9. Treino de uma rede multicamada: o algoritmo de retropropagação (*backpropagation*)

Quando queremos treinar uma rede para aproximar o modelo de um processo, uma função, etc., damos à rede as mesmas entradas do processo, e depois comparamos as saídas. Queremos que q rede imite o processo, e por isso o alvo (*target*) da rede é a saída do processo (para a mesma entrada), como ilustrado na Fig. 4.9.1.

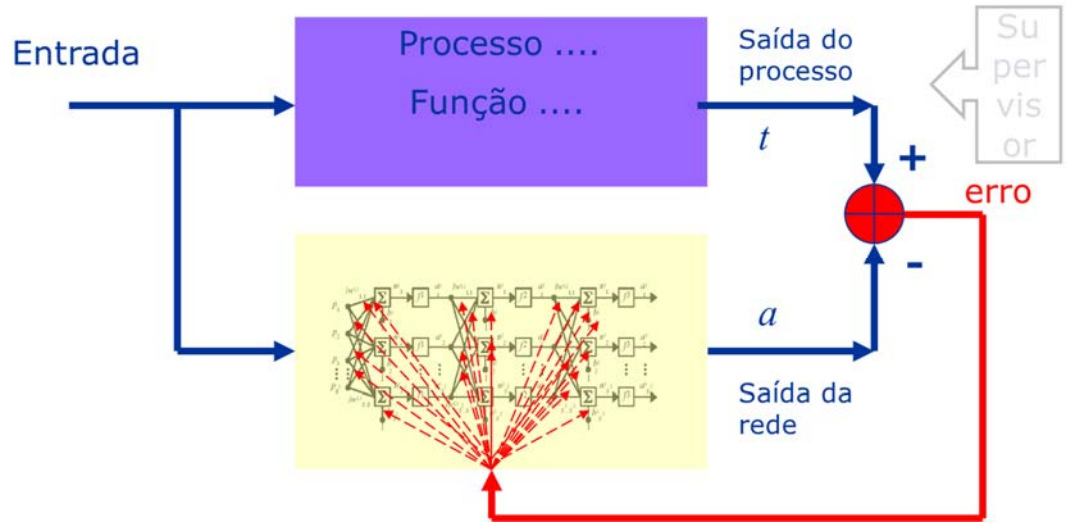



Figura 4.9.1. Treino supervisionado de uma rede multicamada.

Mas a saída da rede nem sempre é igual à do processo, havendo por isso um erro dado por

$$e = t\text{-}a.$$

Para diminuir o erro, mexem-se os pesos e as polarizações segundo alguma estratégia, isto é, segundo algum algoritmo.

Pretende-se minimizar o erro. Para isso aplica-se a mesma técnica do gradiente que vimos para uma camada, através do algoritmo LMSE iterativo. Os pesos são atualizados, de iteração em iteração, segundo s conhecida fórmula

$$w^{New} = w^{Old} - \alpha \bullet \begin{pmatrix} \text{derivada do} \\ \text{critério em} \\ \text{relação ao peso} \end{pmatrix}$$

Em que $\alpha$ é uma constante a fixar, e é o coeficiente de aprendizagem.

O desenvolvimento matemático é aqui um pouco mais complexo porque trabalha com vetores e matrizes.

Suponhamos que a rede tem *M* camadas, sendo *M* a última. Então a soma dos quadrados dos erros é dado por $\mathbf{e}(k)^T\mathbf{e}(k)$ e portanto a função objetivo *F*, ou critério, a minimizar, é dada por

$$F(\mathbf{W},\mathbf{b}) = (\mathbf{t}_k - \mathbf{a}^M(k))^T(\mathbf{t}_k - \mathbf{a}^M(k)) = \mathbf{e}(k)^T\mathbf{e}(k)$$

$k$, índice de iteração.

$\mathbf{W}$, matriz dos pesos

$\boldsymbol{b}$, matriz das polarizações

$$\mathbf{W} = \begin{bmatrix} w_{11} & w_{12} & \dots & w_{1R} \\ w_{21} & w_{22} & \dots & w_{2R} \\ \dots & \dots & \dots & \dots \\ w_{S1} & w_{S2} & \dots & w_{SR} \end{bmatrix}_{S\times R}$$

$$\mathbf{a}^i = \begin{bmatrix} a_1^i & a_2^i & \dots & a_{S^i}^i \end{bmatrix}^T$$

$\mathbf{a}^M \triangleq$ saída da última camada

Em cada iteração $k$,

$$w_{ij}^m(k+1) = w_{ij}^m(k) - \alpha \frac{\partial F}{\partial w_{ij}^m}(k)$$

$$b_j^m(k+1) = b_j^m(k) - \alpha \frac{\partial F}{\partial b_j^m}(k)$$

Como calcular as derivadas ???

Calculadas com os valores dos pesos e polarizações da iteração $k$

com o coeficiente de aprendizagem $\alpha$.

Para calcular as derivadas em relação aos pesos e polarizações ao longo da rede, usa-se a técnica da retropropagação, que parte da função *F* até ao parâmetro em causa, como ilustrado na Fig. 4.9.2 para o peso $w_{21}^1$. Neste exemplo trata-se de uma rede com três camadas, dois neurónios nas camadas escondidas e um neurónio na camada de saída.

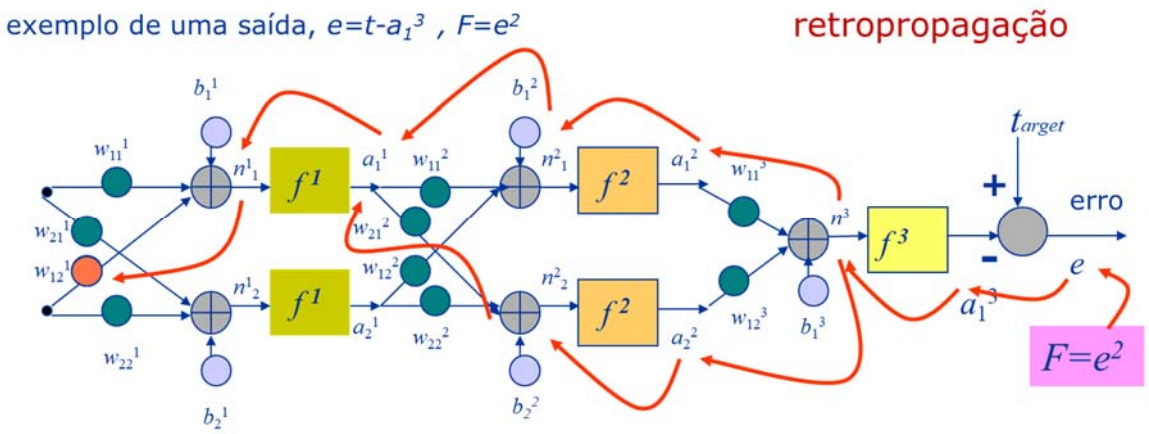



Figura 4.9.2. Esquema da retropropagação, desde *F* até $w_{12}^1$ .

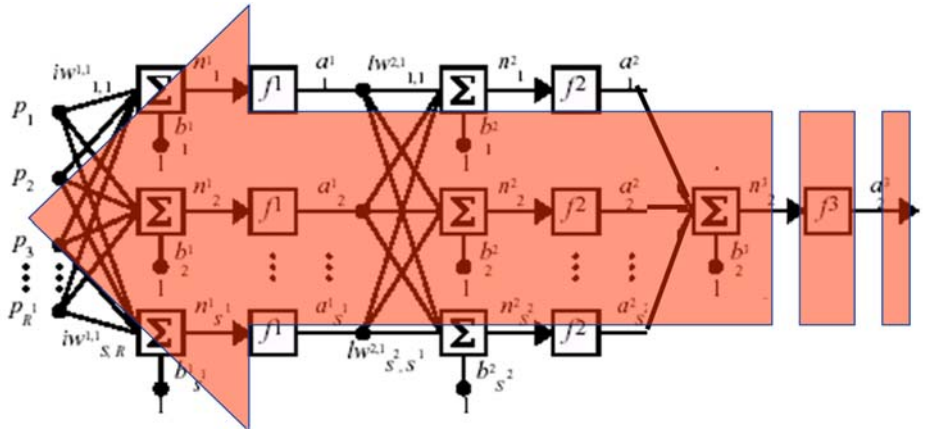


Figura 4.9.3. A essência da retropropagação

Para calcular $\frac{dF}{dw_{12}^1}$ usa-se a regra das derivadas em cadeia. Em primeiro lugar têm que se identificar todos os caminhos que levam de *F* ao peso em causa, no exemplo são dois assinalados pelas setas em laranja. Depois, caminho a caminho, ir calculando as derivadas das funções que vão aparecendo em relação às suas entradas (das funções), até se chegar ao peso em causa. Na figura cada seta indica uma derivada. Finalmente tem que se somar a influência de cada caminho. No caso

$$F(\mathbf{W},\mathbf{b}) = (t_k - a_1^3(k))^T (t_k - a_1^3(k)) = e(k)^T e(k),$$

$$\text{se a saída é escalar } F = e(k)^2 \text{ para a entrada } p^k = [\,p_1^k \;\; p_2^k\,]^T$$

Calculando a derivada pela técnica descrita das derivadas em cadeia,

$$\frac{dF}{dw_{12}^1} = \frac{\partial F}{\partial e}\cdot\frac{\partial e}{\partial a_1^3}\cdot\frac{\partial a_1^3}{\partial n^3}\cdot\left(\frac{\partial n^3}{\partial a_2^2}\cdot\frac{\partial a_2^2}{\partial n_2^2}\cdot\frac{\partial n_2^2}{\partial a_1^1}\cdot\frac{\partial a_1^1}{\partial n_1^1}\cdot\frac{\partial n_1^1}{\partial w_{12}^1} + \frac{\partial n^3}{\partial a_1^2}\cdot\frac{\partial a_1^2}{\partial n_1^2}\cdot\frac{\partial n_1^2}{\partial a_1^1}\cdot\frac{\partial a_1^1}{\partial n_1^1}\cdot\frac{\partial n_1^1}{\partial w_{12}^1}\right)$$

$$= 2e.(-1).\dot{f}^3.(w_{12}^3.\dot{f}^2.w_{21}^2.\dot{f}^1.p_2 + w_{11}^3.\dot{f}^2.w_{11}^2.\dot{f}^1.p_2)$$

Para calcular esta expressão, precisamos do valor do erro, dos pesos e polarizações na iteração atual, das derivadas das funções de ativação, e da entrada.

Todos esses valores se obtêm através de um passo para frente: dada uma entrada, dado um conjunto de pesos e polarizações na iteração atual, vão-se fazendo as contas sempre para a frente (*forward pass*), calculando-se todos os valores intermédios dos sinais até à saída. Sabendo o alvo, calcula-se o erro e o seu quadrado (a função objetivo)

A retropropagação pode-se agora fazer, considerando os valores intermédios calculados no passo para a frente, e os pesos e polarizações são atualizados pela regra do gradiente.

O processo repete-se com a próxima entrada, usando os pesos atualizados com a entrada anterior, até se chagar à última entrada.

A este conjunto de atualizações dos pesos e polarizações usando sucessivamente todas as entradas, chama-se época de treino.

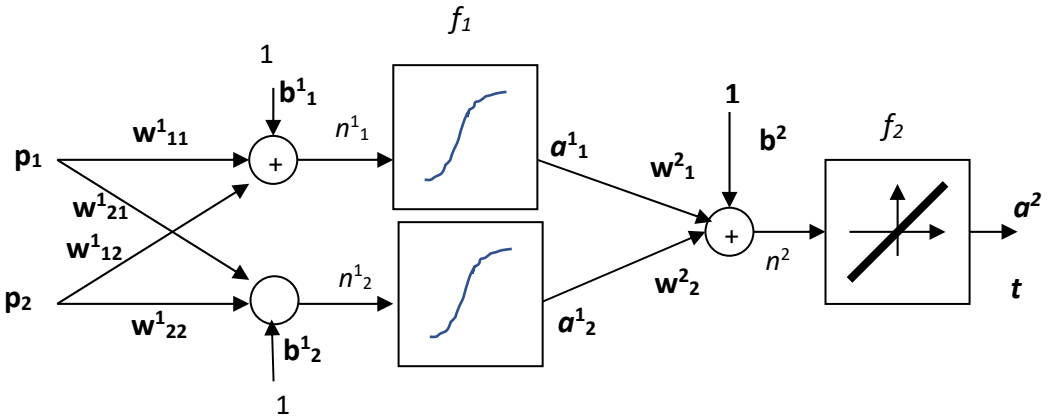
Há uma questão importante: quando se inicializa o processo iterativo, quais os valores dos pesos e polarizações para se fazer o primeiro passo para a frente ? Ou seja, como se inicializam ?

Não havendo conhecimento específico que nos dê alguma luz sobre os valores dos pesos e das polarizações, o mais natural é inicializá-los arbitrariamente.

Para o caso geral de uma rede com qualquer número de camadas e de entradas e para quaisquer funções de ativação, o desenvolvimento analítico é um pouco mais complexo.

Exemplo 4.9.1. Retropropagação numa rede de duas camadas

Seja a rede da Fig. seguinte




$$e = t - a^2$$

$$F = (e)^2$$

Para uma entrada dada, a saída $a^3$ deve ser igual ao target *t.* As funções de ativação podem ser quaisquer, e representam-se por $f_1$ e $f_2$ respetivamente na primeira e na segunda camadas.

Calculando o erro, $e=t-a^2$

$$\begin{aligned} e = t - a^2 &= t - f^2(n^2) = t - f^2(w_1^2 a_1^1 + w_2^2 a_2^1 + b^2) = \\ &= t - f^2\left[ w_1^2 f^1(n_1^1) + w_2^2 f^1(n_2^1) + b^2 \right] = \\ &= t - f^2\left[ w_1^2 f^1(w_{11}^1 p_1 + w_{12}^1 p_2 + b_1^1) + w_2^2 f^1(w_{21}^1 p_1 + w_{22}^1 p_2 + b_2^1) + b^2 \right] \end{aligned}$$

Ou seja, a função objetivo, dada pelo quadrado do erro, uma função de 9 argumentos

$$F = (e)^2 = \mathit{função}(w_1^2, w_2^2, w_{11}^1, w_{12}^1, w_{21}^1, w_{22}^1, b_1^1, b_2^1, b^2)$$

Que são precisamente os parâmetros que queremos encontrar.

Para minimizar o erro, tem que se minimizar F em relação a cada um dos seus 9 argumentos, seguindo o método do gradiente: para minimizar uma função qualquer f(x) em ordem a x, começa-se num ponto x(0) e depois itera-se segundo a fórmula

$$x(k+1) = x(k) - \alpha \left. \frac{\partial f}{\partial x} \right|_{x(k)}$$

Até se convergir para o mínimo, em que a derivada será nula e, portanto, x(k+1)=x(k) a partir daí. Isto em teoria, porque na prática nunca se consegue anular a derivada, e por isso adota-se o critério de parar quando a derivada for inferior a um certo valor pré-estabelecido.

Assim, por exemplo para $w_{11}^1$, teremos que calcular as derivadas parciais em cadeia desde *F* até $w_{11}^1$:

$$\frac{\partial F}{\partial w_{11}^1} = \frac{\partial F}{?} \; \frac{?}{?} \; \frac{?}{?} \; \frac{?}{?} \; \frac{?}{?} \; \frac{?}{\partial w_{11}^1}$$

Para substituirmos os ?? voltemos à figura. Temos de percorrer o caminho desde *F* até $w_{11}^1$ . Obtemos

$$\frac{\partial F}{\partial w_{11}^1} = \frac{\partial F}{\partial e} \; \frac{\partial e}{\partial a^2} \; \frac{\partial a^2}{\partial n^2} \; \frac{\partial n^2}{\partial a_1^1} \; \frac{\partial a_1^1}{\partial n_1^1} \; \frac{\partial n_1^1}{\partial w_{11}^1}$$

Atendendo agora às relações concretas entre as diversas variáveis, teremos

$$\frac{\partial F}{\partial w_{11}^1} = 2e\ .(-1).\ \dot{f}^2(n_2).\ w_1^2.\ \dot{f}^1(n_1^1).\ p_1$$

Para $w_{12}^1$, de modo análogo,

$$\frac{\partial F}{\partial w_{12}^1} = \frac{\partial F}{\partial e}\ \frac{\partial e}{\partial a^2}\ \frac{\partial a^2}{\partial n^2}\ \frac{\partial n^2}{\partial a_1^1}\ \frac{\partial a_1^1}{\partial n_1^1}\ \frac{\partial n_1^1}{\partial w_{12}^1}$$

Fazendo as contas,

$$\frac{\partial F}{\partial w_{12}^1} = 2e\ .(-1).\ \dot{f}^2(n_2).\ w_1^2.\ \dot{f}^1(n_1^1).\ p_2$$

E para a polarização $b_1^1$,

$$\frac{\partial F}{\partial b_1^1} = \frac{\partial F}{\partial e}\ \frac{\partial e}{\partial a^2}\ \frac{\partial a^2}{\partial n^2}\ \frac{\partial n^2}{\partial a_1^1}\ \frac{\partial a_1^1}{\partial n_1^1}\ \frac{\partial n_1^1}{\partial b_1^1} =$$

$$= 2e\ .(-1).\ \dot{f}^2(n_2).\ w_1^2.\ \dot{f}^1(n_1^1).1$$

Constata-se que a cadeia de derivadas é igual para os dois pesos e a polarização do neurónio 1 da camada 1, exceto a última derivada. Essa parte comum exprime o modo como $n_1^1$ influencia o critério *F.* Pode-se por isso chamar sensibilidade do critério em relação a $n_1^1$, ou seja, sensibilidade em relação à ativação do neurónio 1 da camada 1. Chame-se-lhe por isso $s_1^1$. Assim teremos simplesmente

$$\frac{\partial F}{\partial w_{11}^1} = s_1^1 p_1 \qquad \frac{\partial F}{\partial w_{21}^1} = s_1^1 p_2 \qquad \frac{\partial F}{\partial b_1^1} = s_1^1.1$$

Para $w_{21}^1$, e aplicando a mesma definição de sensibilidade,

$$\frac{\partial F}{\partial w_{21}^1} = \frac{\partial F}{\partial e}\ \frac{\partial e}{\partial a^2}\ \frac{\partial a^2}{\partial n^2}\ \frac{\partial n^2}{\partial a_2^1}\ \frac{\partial a_2^1}{\partial n_2^1}\ \frac{\partial n_1^1}{\partial w_{12}^1} = \frac{\partial F}{\partial n_2^1}\ \frac{\partial n_2^1}{\partial w_{12}^1}$$

$$= s_2^1 p_1$$

Para $w^1{}_{22}$ e para $b^1{}_2$ ,

$$\frac{\partial F}{\partial w_{22}^1} = s_2^1 p_2 \qquad \frac{\partial F}{\partial b_2^1} = s_2^1 .1 = s_2^1$$

Reorganizando para a primeira camada obtemos s expressões vetoriais

$$\begin{bmatrix} \frac{\partial F}{\partial w_{11}^1} \\ \frac{\partial F}{\partial w_{12}^1} \\ \frac{\partial F}{\partial b_1^1} \end{bmatrix} = s_1^1 \begin{bmatrix} p_1 \\ p_2 \\ 1 \end{bmatrix} \qquad \begin{bmatrix} \frac{\partial F}{\partial w_{21}^1} \\ \frac{\partial F}{\partial w_{22}^1} \\ \frac{\partial F}{\partial b_2^1} \end{bmatrix} = s_2^1 \begin{bmatrix} p_1 \\ p_2 \\ 1 \end{bmatrix}$$

Ou ainda mais compactamente,

$$\begin{bmatrix} \frac{\partial F}{\partial w_{11}^1} & \frac{\partial F}{\partial w_{21}^1} \\ \frac{\partial F}{\partial w_{12}^1} & \frac{\partial F}{\partial w_{22}^1} \\ \frac{\partial F}{\partial b_1^1} & \frac{\partial F}{\partial b_2^1} \end{bmatrix} = \begin{bmatrix} p_1 \\ p_2 \\ 1 \end{bmatrix} \begin{bmatrix} s_1^1 & s_2^1 \end{bmatrix}$$

Para a segunda camada teremos que calcular as derivadas em relação a $w^2_1$, $w_2^2$, e $b^2$

$$\frac{\partial F}{\partial w_1^2} = \frac{\partial F}{\partial e} \frac{\partial e}{\partial a^2} \frac{\partial a^2}{\partial n^2} \frac{\partial n^2}{\partial w_1^2} = 2.e.(-1). \dot{f}^2(n^2).a_1^1$$

$$\frac{\partial F}{\partial w_2^2} = \frac{\partial F}{\partial e} \frac{\partial e}{\partial a^2} \frac{\partial a^2}{\partial n^2} \frac{\partial n^2}{\partial w_2^2} = 2.e.(-1). \dot{f}^2(n^2).a_2^1$$

$$\frac{\partial F}{\partial b^2} = \frac{\partial F}{\partial e} \frac{\partial e}{\partial a^2} \frac{\partial a^2}{\partial n^2} \frac{\partial n^2}{\partial b^2} = 2.e.(-1). \dot{f}^2(n^2).1$$

Chamando $s^2$ à sensibilidade de $F$ à ativação $n_2$,

$$s^2 = \frac{\partial F}{\partial e}\ \frac{\partial e}{\partial a^2}\ \frac{\partial a^2}{\partial n^2}$$

Pode-se escrever

$$\begin{bmatrix} \frac{\partial F}{\partial w_1^2} \\ \frac{\partial F}{\partial w_2^2} \\ \frac{\partial F}{\partial b^2} \end{bmatrix} = s^2 \begin{bmatrix} a_1^1 \\ a_2^1 \\ 1 \end{bmatrix}$$

Se repararmos bem esta equação é análoga às da primeira camada, mas agora as entradas da 2ª camada são as saídas da primeira,

Comparando as expressões das sensibilidades $s_1^1$, $s_2^1$, e $s^2$ verifica-se

$$s_1^1 = \dot{f}^1(n_1^1).w_1^2.s^2$$

$$s_2^1 = \dot{f}^1(n_2^1).w_2^2.s^2$$

Se definirmos

$$s^1 = \begin{bmatrix} s_1^1 \\ s_2^1 \end{bmatrix}$$

Virá

$$s^1 = \begin{bmatrix} \dot{f}^1(n_1^1).w_1^2 \\ \dot{f}^1(n_2^1).w_2^2 \end{bmatrix} s^2$$

Isto é, para calcular $s^1$ retropropagamos $s_2$ através dos neurónios da primeira camada. Esta é a essência do algoritmo de retropropagação.

Depois de calcular todas as derivadas parciais, faz-se a atualização dos pesos e das polarizações . Claro que temos que inicializar os seus valores nalgum ponto. Vejamos um exemplo para este caso.

Sejam P e T

$$P = \begin{bmatrix} 1 & 4 \\ 2 & 2,5 \end{bmatrix} \qquad T = \begin{bmatrix} 0,5 & 0,5 \end{bmatrix}$$

Inicializamos os pesos e polarizações em

$$\begin{array}{lll} w_{11}^1 = 0,5 & w_{12}^1 = 1,5 & \mathrm{b}_1^1 = 0,3 \\ w_{21}^1 = -0,4 & w_{22}^1 = 3,7 & \mathrm{b}_2^1 = -0,8 \\ w_1^2 = 1 & w_2^2 = -3,7 & \mathrm{b}^2 = 1,7 \end{array}$$

Agora fornecemos à rede primeira entrada $p1 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$

A rede calcula para a frente:

$$\begin{array}{ll} n_1^1 = 1 \times 0,5 + 2 \times 2,5 + 0,3 = 3,8 & a_1^1 = \log sig(3,8) = 0,98 \\ n_2^1 = 1 \times (-0,4) + 2 \times 3,7 + (-0,8) = 6,2 & a_2^1 = \log sig(6,2) = 1,0 \\ n^2 = 1 \times 0,98 + 1,0 \times (-3,7) + 1,7 = -1,02 & a^2 = purlin(-1,02) = -1,02 \end{array}$$

$$e = 0,5 - (-1,02) = 0,5 + 1,02 = 1,52$$

Concluiu-se assim uma etapa para a frente, desde a entrada até ao erro na saída.

Façamos agora uma etapa para trás para calcular as derivadas parciais e atualizar os pesos e as polarizações. Usamos as sensibilidades definidas anteriormente.

$$s^2 = \frac{\partial F}{\partial e}\frac{\partial e}{\partial a^2}\frac{\partial a^2}{\partial n^2} = 2.e.(-1).1 = -2e = -2 \times 1,52 = -3,04$$

Note-se que os valores das derivadas sucessivas se calculam com os valores atuais dos pesos e das polarizações.

Derivada em relação a $w^1_{11}$

$$\frac{\partial F}{\partial w_{11}^1} = s_1^1 . p_1 \qquad s_1^1 = \dot{f}^1(n_1^1).w_1^2.s^2$$

Sendo

$$f^1(n) = \log sig(n) = \frac{1}{1+e^{-n}}$$

$$\frac{\partial f^1}{\partial n} = \frac{-1(-1.e^{-n})}{(1+e^{-n})^2} = \frac{e^{-n}}{(1+e^{-n})^2}$$

e sendo

$$n_1^1 = 3,8 \text{ logo } \dot{f}^1(n_1^1) = \frac{e^{-3,8}}{(1+e^{-3,8})^2} = 0,0214$$

Temos tudo para calcular $s^1_1$

$$s_1^1 = 0,0214 \times 1 \times (-3,04) = -0,0651$$

e finalmente

$$\frac{\partial F}{\partial w_{11}^1} = -0,0651 \times 1 = -0,0651$$

Podemos agora atualizar o peso seguindo a regra do gradiente

$$w_{11}^1(1) = w_{11}^1(0) - \alpha \left.\frac{\partial F}{\partial w_{11}^1}\right|_{(0)} = 0,5 - \alpha(-0,0651)$$

Chamámos a $\alpha$ o coeficiente de aprendizagem (*learning coeficient*). Vemos que ele influencia a amplitude da variação do peso de uma etapa para a outra. Geralmente não interessa uma variação muito grande. Se fizermos $\alpha$=1 obtemos

$$w_{11}^1(1) = w_{11}^1(0) + 0,0651 = 0,5 + 0,0651 = 0,561$$

$$\Delta w_{11}^1 = 0,0651$$

Derivada em relação a $w^1_{12}$

$$\frac{\partial F}{\partial w_{12}^1} = s_1^1 . p_2 = -0,0651 \times 2 = -0,1302$$

E a atualização do peso

$$w_{12}^1(1) = w_{12}^1(0) - \alpha(-0,1302) = 1,5 + 0,1302 = 1,6302 \text{ com } \alpha = 1$$

$$\Delta w_{11}^1 = 0,1302$$

Derivada em relação a $b^1_1$

$$\frac{\partial F}{\partial b_1^1} = s_1^1 \qquad b_1^1(1) = b_1^1(0) - \alpha(-0,0652) = 0,3 + 0,0651 = 0,3651 \quad \text{com} \quad \alpha = 1$$

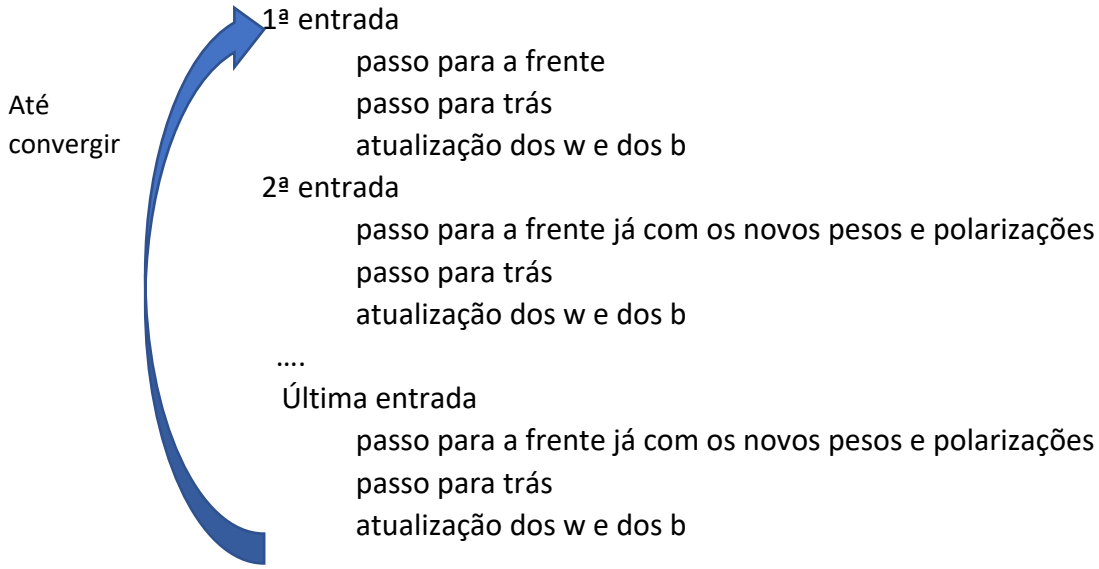
$$\Delta b_1^1 = 0,0651$$

Procede-se de modo análogo para todos os outros pesos e polarizações. Por exemplo para $w^2_1$

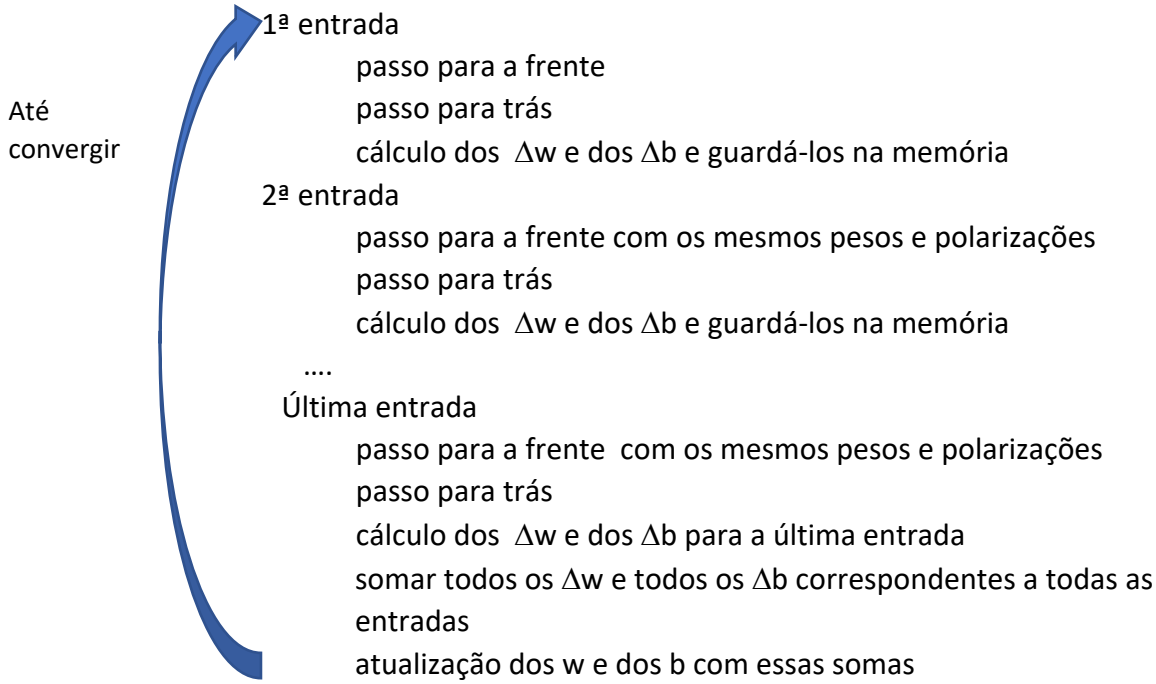
$$w_1^2(1) = w_1^2(0) - \alpha.s^2.a_1^1(0) = 1 - \alpha.(-3,04).0,98 = 1 + 2,98 = 3,98$$
$$\Delta w_1^2 = 2,98$$

Depois de todos estes cálculos para a primeira entrada podemos adotar uma de duas estratégias de treino:

i) Treino iterativo (no Matlab *adapt*): uma época de treino




ii) Treino em diferido (no Matlab *train*): uma época de treino



No treino em diferido, em cada época faz-se apenas uma atualização dos pesos e polarizações. No treino iterativo faz-se uma atualização por entrada. A evolução dos pesos e polarizações será diferente. Qual a que converge melhor? Depende do problema. Em teoria o treino em diferido aproxima-se mais do verdadeiro gradiente, mas requer mais memória. O treino iterativo é mais rápido, e é usado frequentemente em aplicações com um elevado número de entradas. Mas só a experiência ditará a melhor escolha.

O algoritmo de retropropagação ainda é o grande cavalo de batalha do treino de redes neuronais, incluindo as redes profundas (*deep learning*) por isso é importante a sua compreensão.

Generalizando o desenvolvimento do exemplo de duas camadas para um número qualquer de M camadas, pode resumir-se pela Fig.4.9.3. Em Hagan and Coll. encontra-se um desenvolvimento mais detalhado .

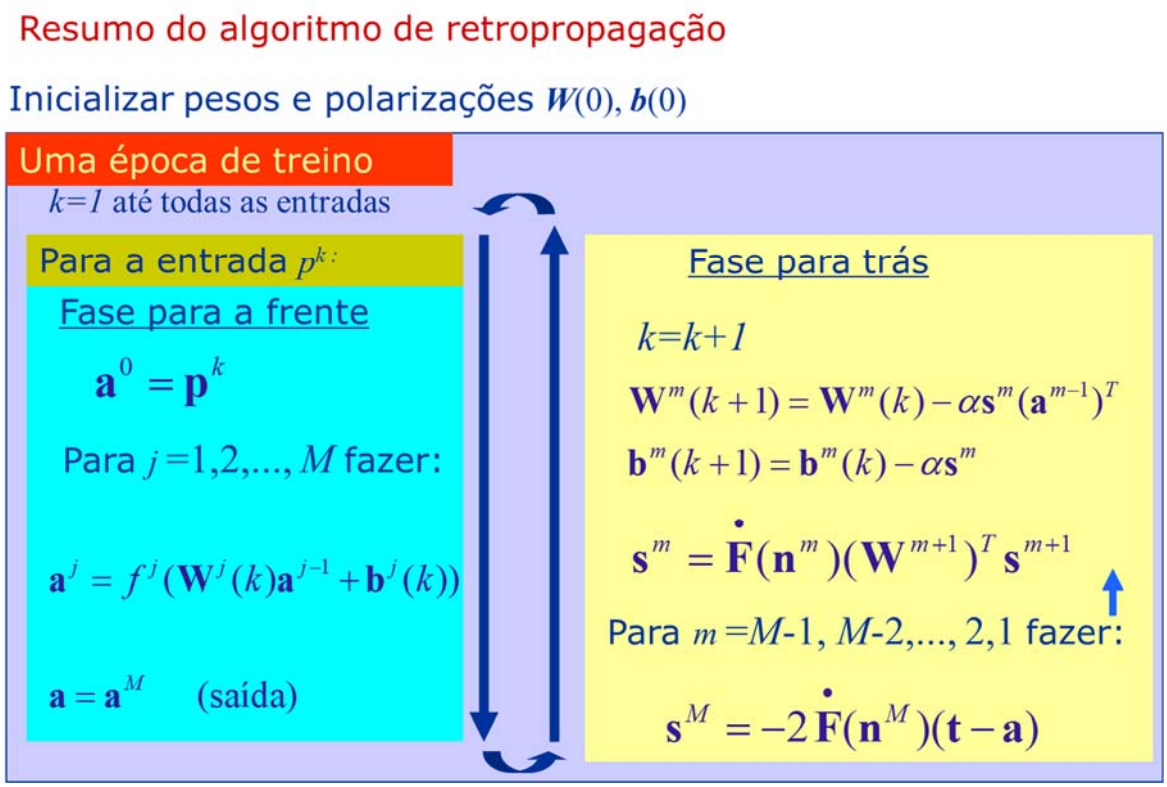



Figura 4.9.3. Resumo do treino por retropropagação (iterativo).

Tal como no exemplo a sensibilidade da segunda camada foi retropropagada para calcular a sensibilidade da primeira camada,

$$s_1 \leftarrow s_2$$

No caso de *M* camadas, teremos

$$\mathbf{s}^1 \leftarrow \mathbf{s}^2 \leftarrow \ldots \leftarrow \mathbf{s}^{M-1} \leftarrow \mathbf{s}^M$$

A sensibilidade da última camada calcula-se por (em notação vetorial)

$$s^M = -2\dot{\mathbf{F}}(\mathbf{n}^M)(\mathbf{t} - \mathbf{a}^M) = -2\dot{\mathbf{F}}(\mathbf{n}^M)\mathbf{e}$$

Para melhorar a convergência há variações do método do gradiente (que só por si é na maior parte das vezes de convergência difícil). Técnicas de combinação de gradientes em diversos instantes levam a uma família de métodos chamados de gradiente conjugado. Para mais detalhes ver Hagan and Coll Cap. 9-17, 9-18, 9-22.

Bibliografia:


Hagan, M.T., H.B. Demuth, M. Beale, Neural Network Design, 2nd ed., ebook, 2013. Freely downloadable from hagan.okstate.edu/nnd.html

Hassoun. M. H., Fundamentals of Artificial Neural Networks, MIT Press, 1994.

Deep Learning Toolbox Users´s Guide, The Mathworks, 2022.