4 PERCEPTRONS DE MÚLTIPLAS CAMADAS

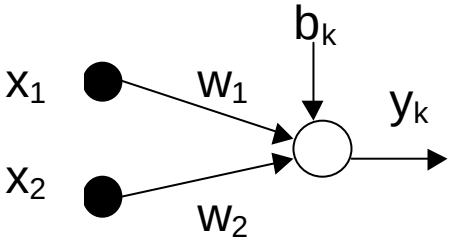
4.1-Limitações das redes monocamadas

Até agora vimos redes com uma só camada (além da camada de entrada) ou, no máximo, como é o caso das redes MADALINE, redes com duas camadas, na qual a camada de saída possui seus parâmetros (pesos e polarizações) fixos, funcionando como uma combinação lógica OU ou E dos nós da camada anterior.

O que não exploramos ainda, como foi mostrado por Minsky e Papert, são as limitações deste tipo de redes. O que estes autores provaram é que estas redes não resolvem, isto é, não são capazes de separar as classes, de uma infinidade de problemas reais.

Tomemos como exemplo o caso da operação lógica OU Exclusivo (XOR), cuja tabela verdade (bipolar) está abaixo:

x_1	x_2	y_k
-1	-1	-1
-1	1	1
1	-1	1
1	1	-1



Se usarmos uma rede com uma só camada e um só neurônio, com duas entradas e uma saída, teríamos como entrada líquida do neurônio o valor:

$$u_k = \sum_{i=1}^{n} w_{ki} x_i = w_{k1} x_1 + w_{k2} x_2 + b_k$$

Com uma função de ativação de limiar zero, os casos de valores de entrada líquida u_k que correspondem à saída zero estão na fronteira de separação das classes, correspondente a:

$$y_k = 0 \Rightarrow w_{k1} x_1 + w_{k2} x_2 + b_k = 0$$

Portanto, a fronteira de separação é dada por uma reta cuja equação é:

$$w_{k1}x_1 + w_{k2}x_2 = -b_k$$

Entretanto, os valores de w e b da equação acima, que definam uma reta que separe adequadamente os valores de saída da função XOR em duas classes distintas, não podem ser encontrados, haja vista que não há nenhuma reta que separe estes valores, como mostra a figura 12 (ao contrário de retas de separação para as funções AND e OR, que podem ser encontradas, como mostra a mesma figura).

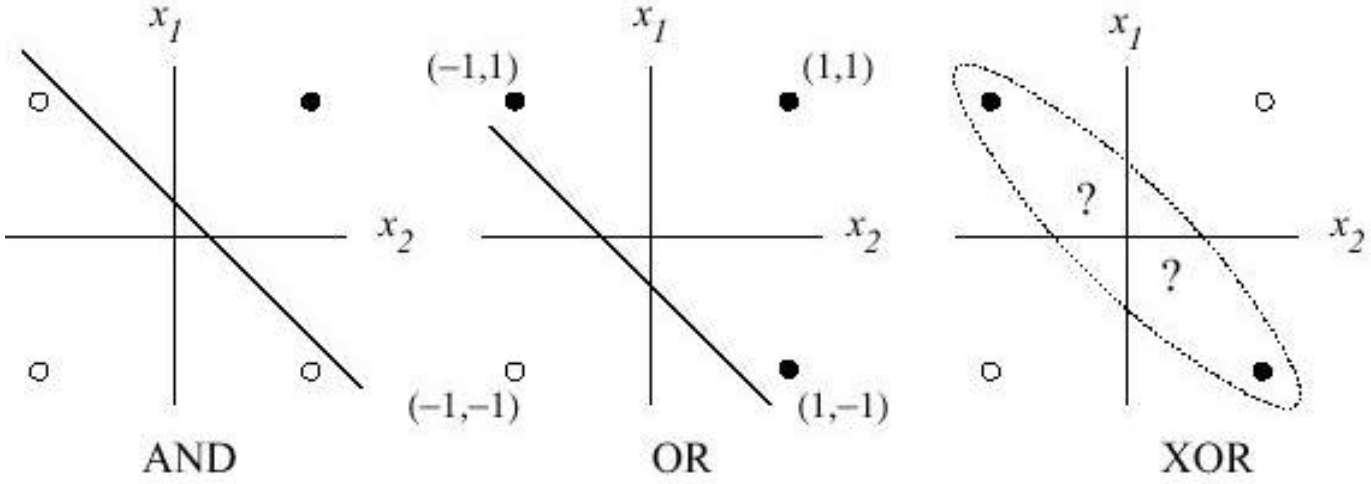


FIG. 12- RETAS DE SEPARAÇÃO DE CLASSES EM FUNÇÕES LÓGICAS

4.2-Possibilidades dos Perceptrons multicamadas

Como então resolver o problema? Minsky e Papert mostraram que esta limitação pode ser contornada acrescentando mais uma camada de nós, formando assim uma rede conhecida com Perceptron de Míltiplas Camadas. Para o exemplo específico citado (XOR), basta acrescentar uma outra camada, constituída de um único nó (situado portanto na saída do já existente), para que o problema passe a ter solução, isto é, para que a rede passe a ser capaz de separar os valores de entrada em duas classes de saída distintas. A arquitetura da rede está representada na figura 13a (os valores dentro dos

nós são os limiares) e a função de separação que esta arquitetura gera, é mostrada na figura 13b.

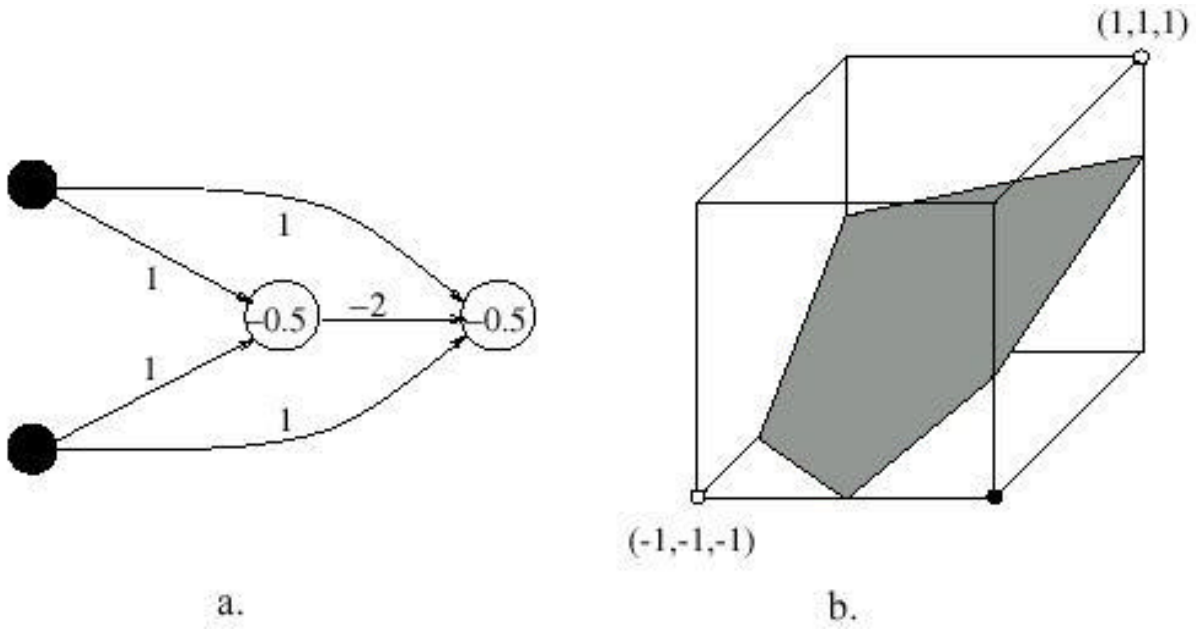


FIG. 13- ARQUITETURA E FUNÇÃO SEPARADORA DO XOR

Na verdade, o que a rede achou agora foi uma superfície separadora (plano separador) que separou os resultados da combinação de entradas possíveis, que agora se encontram em um plano tridimensional, haja vista que as entradas do segundo nó agora são três.

O que os autores mostraram, em um plano mais geral, é que para qualquer entrada N dimensional é possível achar uma superfície separadora para as classes do problema usando uma rede de múltiplas camadas. Entretanto, encontrar os valores dos pesos e polarizações necessários ou mesmo qual é a exata arquitetura necessária, já é um outro problema. A dificuldade é que não sabemos o valor desejado na(s) saída(s) do(s) nó(s) da camada oculta e, portanto, não podemos usar a regra de atualização do PERCEPTRON para atualizar os pesos que ligam a camada de entrada à camada oculta. Trataremos deste problema mais à frente no próximo item.

4.3-Treinamento dos Perceptrons multicamadas

Apesar das descobertas interessantes de Minsky e Papert em 1969 quanto à aplicabilidade das redes, os autores não sugeriram nenhum método que fosse capaz de encontrar os parâmetros da rede. Tal algoritmo só surgiu a partir das pesquisas independentes de Paul Werbos (1974) e Rumelhart, Willians e Hinton (1986). O que estes autores propuseram foi um método de propagar o erro da camada de saída (que conhecemos) para a(s) camada(s) oculta(s). Desta forma, tornou-se novamente possível alterar os parâmetros de todas as camadas da rede, a partir do erro na saída, ou seja, procurando-se, como nas redes de uma só camada, qual é a alteração dos parâmetros que minimiza o erro na saída da rede.

O algoritmo de treinamento ficou conhecido como Backpropagation e consiste basicamente nos seguintes passos:

1. Apresentar um padrão à rede;
2. Calcular o erro na saída da rede;
3. Retropropagar o erro na rede calculando de que forma as mudanças nos pesos afetam o erro;
4. Modificar os pesos de forma a minimizar o erro médio;
5. Retornar ao passo 1 enquanto ainda houverem padrões que não foram apresentados nesta iteração;
6. Se um erro máximo desejado não tiver sido atingido, retornar ao passo 1 para a próxima iteração (apresentação de todos os padrões novamente)

Com o objetivo de minimizar o erro médio para todos os padrões apresentados, são feitas, portanto, modificações nos pesos a cada padrão.

O algoritmo de treinamento trata de forma diferenciada os neurônios da camada de saída o os das camadas ocultas, baseado na rede genérica vista na figura 14.

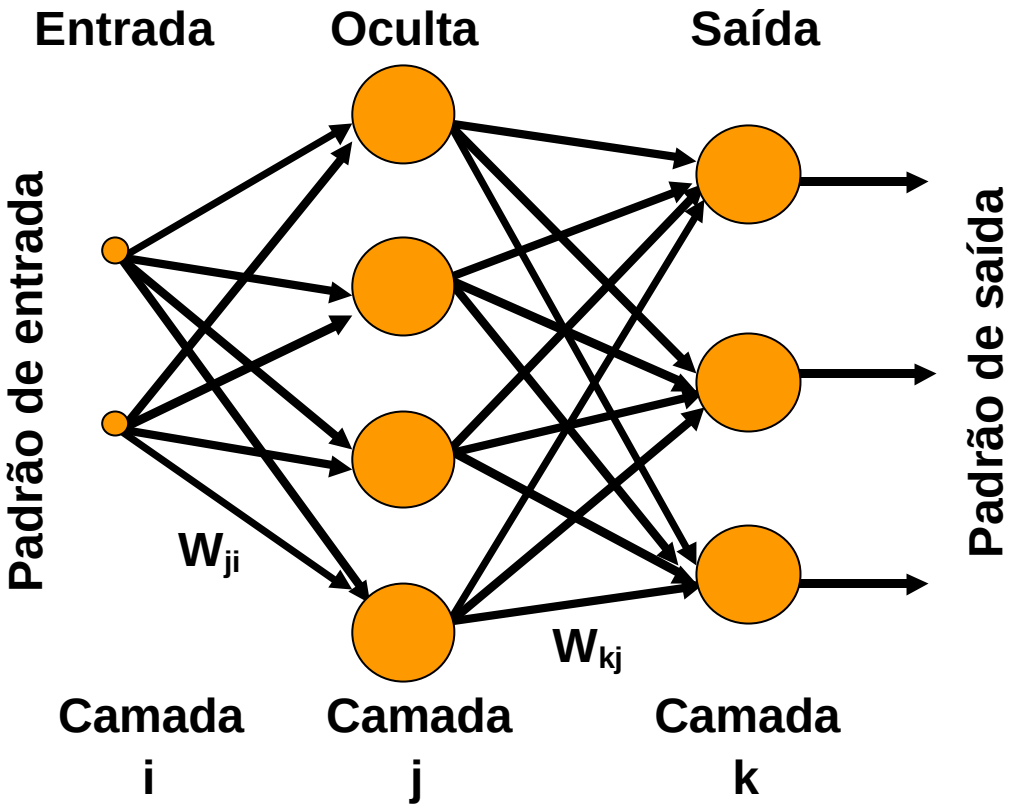


FIG. 14- ARQUITETURA DO PERCEPTRON DE MÚLTIPLAS CAMADAS

Em um neurônio k da camada de saída, o erro da n-ésima iteração é dado por:

$$e_k(n) = d_k(n) - y_k(n) \qquad (4.1)$$

A soma dos erros quadráticos na camada de saída pode ser escrita como:

$$x(n) = \frac{1}{2}\sum_k e_k^2(n) \qquad (4.2)$$

Onde k varia de 1 até o número de elementos da camada de saída. O erro médio para os N padrões apresentados no treinamento é dado por:

$$x_{md} = \frac{1}{N}\sum_{n=1}^{N} x(n) \qquad (4.3)$$

A entrada líquida do neurônio k é:

$$u_k(n) = \sum_{j=0}^{p} w_{kj}(n) y_j(n) \qquad (4.4)$$

Onde p é o número de elementos da camada oculta. Quando j=0 temos a polarização do nó k.
A saída do neurônio k na iteração n é:

$$y_k(n) = j\left[u_k(n)\right] \qquad (4.5)$$

A idéia é fazer modificações nos pesos Wkj proporcionais ao gradiente do erro em relação aos pesos, isto é, queremos encontrar:

$$\frac{\partial x(n)}{\partial w_{kj}(n)} \qquad (4.6)$$

De acordo com a regra da cadeia temos:

$$\frac{\partial x(n)}{\partial w_{kj}(n)} = \frac{\partial x(n)}{\partial e_k(n)} \frac{\partial e_k(n)}{\partial y_k(n)} \frac{\partial y_k(n)}{\partial u_k(n)} \frac{\partial u_k(n)}{\partial w_{kj}(n)} \qquad (4.7)$$

Para obter o primeiro fator de (4.7), vamos derivar os dois lados de (4.2):

$$\frac{\partial x(n)}{\partial e_k(n)} = e_k(n) \qquad (4.8)$$

Derivando (4.1) em relação a y_k, temos:

$$\frac{\partial e_k(n)}{\partial y_k(n)} = -1 \qquad (4.9)$$

Derivando (4.5) em relação a u_k, temos:

$$\frac{\partial y_k(n)}{\partial u_k(n)} = j'_k(u_k(n)) \qquad (4.10)$$

Derivando (4.4) em relação a w_{kj}, temos:

$$\frac{\partial u_k(n)}{\partial w_{kj}(n)} = y_j(n) \qquad (4.11)$$

Substituindo (4.8), (4.9), (4.10), e (4.11) em (4.7):

$$\frac{\partial x(n)}{\partial w_{kj}(n)} = -e_k(n)j'_k(u_k(n))y_j(n) \qquad (4.12)$$

O gradiente nos dá a direção de máximo crescimento do erro em relação à variação dos pesos. Assim, queremos que a variação dos pesos nos leve na direção contrária (regra delta):

$$\Delta w_{kj}(n) = -h\frac{\partial x(n)}{\partial w_{kj}(n)} \qquad (4.13)$$

Substituindo (4.12) em (4.13), temos:

$$\Delta w_{kj}(n) = hd_k(n)y_j(n) \qquad (4.14)$$

Onde:

$$d_k(n) = e_k(n)j'_k(u_k(n)) \qquad (4.15)$$

Ou seja, se k é um nó da camada de saída, sabemos que devemos atualizar os pesos que o ligam à camada anterior usando as fórmulas (4.14) e (4.15).

Para um nó j da camada oculta, o problema é que não conhecemos $e_j(n)$.

Queremos agora encontrar:

$$\frac{\partial x(n)}{\partial w_{ji}(n)} \qquad (4.16)$$

Usando novamente a regra da cadeia:

$$\frac{\partial x(n)}{\partial w_{ji}(n)} = \frac{\partial x(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial u_j(n)} \frac{\partial u_j(n)}{\partial w_{ji}(n)} \qquad (4.17)$$

Mas:
$$\frac{\partial y_j(n)}{\partial u_j(n)} = j'_j(u_j(n)) \qquad (4.18)$$

Derivando o erro (2), temos:

$$\frac{\partial x(n)}{\partial y_j(n)} = \sum_k e_k(n) \frac{\partial e_k(n)}{\partial y_j(n)} \qquad (4.19)$$

Expandindo (4.19) pela regra da cadeia:

$$\frac{\partial x(n)}{\partial y_j(n)} = \sum_k e_k(n) \frac{\partial e_k(n)}{\partial u_k(n)} \frac{\partial u_k(n)}{\partial y_j(n)} \qquad (4.20)$$

Mas:

$$e_k(n) = d_k(n) - y_k(n) = d_k(n) - j'_k(u_k(n)) \qquad (4.21)$$

Derivando (4.21):
$$\frac{\partial e_k(n)}{\partial u_k(n)} = -j'_k(u_k(n)) \qquad (4.22)$$

Por outro lado, diferenciando (4.4):
$$\frac{\partial(u_k(n))}{\partial(y_j(n))} = w_{kj}(n) \qquad (4.23)$$

Substituindo (4.22) e (4.23) em (4.20), temos:

$$\frac{\partial x(n)}{\partial y_j(n)} = -\sum_k e_k(n)\, j'_k(u_k(n)) w_{kj}(n) = -\sum_k d_k(n) w_{kj}(n) \quad (4.24)$$

Como em (4.4) temos que: $u_j(n) = \sum_{i=0}^{q} w_{ji}(n) y_i(n)$ (4.25)

Onde q é o número de elementos da camada de entrada. Quando i=0 temos a polarização do nó j.
Da mesma forma que em (4.11), de (4.25):

$$\frac{\partial u_j(n)}{\partial w_{ji}(n)} = y_i(n) \quad (4.26)$$

Substituindo (4.18), (4.24) e (4.26) em (4.17):

$$\frac{\partial x(n)}{\partial w_{ji}(n)} = \left(-\sum_k d_k(n) w_{kj}(n) \right) j'_j(u_j(n)) y_i(n) \quad (4.27)$$

Como em (4.13), queremos: $\Delta w_{ij}(n) = -h \frac{\partial x(n)}{\partial w_{ji}(n)}$ (4.28)

Juntando (4.28) e (4.27), como em (4.14) e (4.15):

$$\Delta w_{ji}(n) = h d_j(n) y_i(n) \quad (4.29)$$

Onde:

$$d_j(n) = j'_j(u_j(n)) \sum_k d_k(n) w_{kj}(n) \quad (4.30)$$

Ou seja, se j é um nó da camada oculta, devemos atualizar os pesos que o ligam à camada de entrada usando as fórmulas (4.29) e (4.30).

Os autores do algoritmo de *Backpropagation* mostraram também que estas contas são extensíveis para N camadas.

Ou seja, detalhando um pouco melhor o algoritmo no que tange ao que deve ser feito a cada no padrão apresentado, poderiamos enumerar os seguintes passos:

1. Apresentar um padrão de entrada;
2. Propagar o sinal para a frente, calculando as saídas;
3. Calcular os δ's para os neurônios da camada de saída;
4. Retropropagar o erro da saída calculando os δ's das camadas ocultas;
5. Atualizar os pesos, apresentar novo padrão, e retornar ao passo 1.

Alguns aspectos relevantes sobre a função de ativação a ser empregada, dadas as características do algoritmo:

1. A função de ativação precisa ser diferenciável em todo o domínio;
2. A função de ativação precisa ser não decrescente para que a sua derivada não troque de sinal comprometendo a convergência do algoritmo;

Por estes motivos e também pela facilidade de obtenção da derivada (que pode ser expressa em termos da própria função), é comum que sejam empregadas funções de ativação semi-lineares, tais como a função Sigmóide (figura 15) e a Tangente Hiperbólica (cuja única diferença é que varia entre –1 e 1).

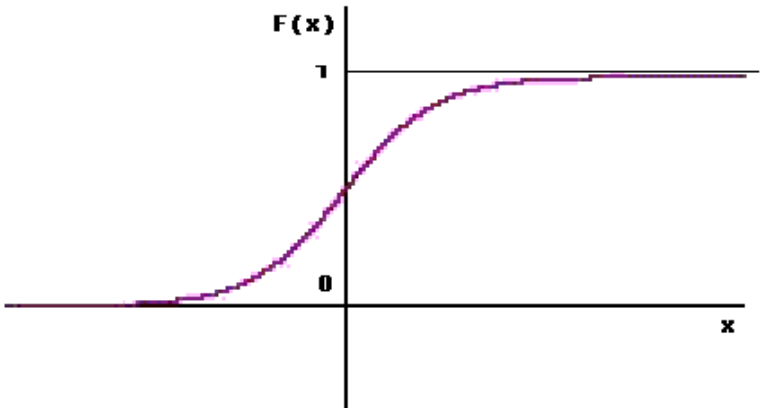


FIG. 15- FUNÇÃO SEMI-LINEAR SIGMÓIDE

A função Sigmóide possui a seguinte fórmula:

$$y_j = j\left(u_j\right) = \frac{1}{1+e^{-l\,u_j}} \quad \text{, onde } l > 0$$

Diferenciando ambos os lados: $$\frac{dy_j}{du_j} = j'\left(u_j\right) = \frac{e^{-l\,u_j}}{\left(1+e^{-l\,u_j}\right)^2}$$

Ou ainda: $$j'\left(u_j\right) = y_j(n)\left[1 - y_j(n)\right]$$

4.4-Características práticas do backpropagation

Quanto ao instante de atualização dos pesos - Se a atualização dos pesos, proposta nas equações (4.14) e (4.29) for realizada a cada apresentação de um novo padrão, ao final de uma iteração é razoável imaginar que a atualização dos pesos corresponda mais às alterações feitas pelos últimos padrões apresentados do que aos primeiros. Se a ordem de apresentação é a mesma a cada iteração, então a alteração será tendenciosa.

Portanto, caso desejemos alterar os pesos a cada apresentação de um novo padrão, é conveniente que variemos a ordem de apresentação dos padrões a cada iteração, por exemplo, sorteando essa ordem. A este esquema de apresentação dos padrões, chamamos atualização iterativa.

Uma outra opção é não alterar o peso a cada apresentação de padrões e sim, após o cálculo das variações desejadas pelo padrão (dadas pelas equações 4.14 e 4.29), acumular a alteração deste padrão com as alterações determinadas pelos demais padrões (isto é, fazer um somatório dos Δw) e realizar a

alteração nos pesos somente ao final da iteração. A este método chamamos atualização por batelada.

Quanto à parada do treinamento – o treinamento geralmente é encerrado quando ocorre um dos seguintes casos:

1. erro atingiu um valor pré determinado;
2. erro está caindo a uma taxa suficientemente pequena de acordo com um parâmetro pré determinado; ou,
3. A atualização está prejudicando a generalização (medida com um conjunto de validação). Isto é, o erro está crescendo para o conjunto de validação, apesar de possivelmente estar caindo para o conjunto de treinamento.

Quanto à escolha dos pesos iniciais – devemos também tomar alguns cuidados ao escolher os valores iniciais para os pesos da rede. Rumelhart (1986) mostrou que a rede pode nunca conseguir aprender se todos os pesos forem iniciados com um mesmo valor, uma vez que todos os pesos seriam alterados de uma mesma maneira. Os pesos são mudados proporcionalmente à derivada da função de ativação, portanto, quanto mais próximos do centro da função estivermos maior será a modificação. Nas extremidades da função de ativação (regiões de saturação) as derivadas são próximas de zero e, consequentemente, há pouca variação nos pesos, mesmo que o erro seja significativo. Ele sugere que o processo de aprendizado deva ser iniciado utilizando-se valores de pesos de pequena amplitude, que sejam aleatoriamente gerados a partir de uma distribuição uniforme. Sugere-se, por exemplo, tomar os valores iniciais dos pesos e polarizações a partir de uma distribuição uniforme no intervalo [-0.5;0.5].

4.5-Modificações na Retropropagação básica

O algoritmo de retropropagação possui algumas características que devem ser levadas em consideração para acelerar a convergência, evitar oscilações ou mesmo para impedir que o treinamento da rede fracasse (não convirja).

Introdução do termo de momento – Na prática, uma forma de aumentar a taxa de aprendizado sem causar oscilações é modificar a equação (4.29) para incluir um termo chamado termo de momento (o mesmo ocorre com a equação (4.14)). Esta alteração consiste da utilização da parcela de atualização dos pesos (D_w) da iteração anterior, no processo de atualização dos pesos da iteração corrente. Observe a alteração:

$$\Delta w_{ji}(n) = a\Delta w_{ji}(n-1) + hd_j(n)y_i(n)$$

O índice *n* indica a iteração (também chamada de etapa de treinamento ou época) e o coeficiente **a** (taxa de momento) determina o efeito de alterações anteriores na direção do movimento no espaço de pesos. O efeito desta alteração é a aceleração do treinamento por um fator de 1/(1-**a**).

Mínimos locais – outro problema que pode afetar o treinamento são os mínimos locais. O algoritmo do gradiente decrescente pode encontrar mínimos locais na busca da solução do problema. Isto normalmente está ligado à amplitude do valor inicial dos pesos e ao fator de normalização. Se os valores estiverem concentrados nos extremos da função semi linear (perto de 0 ou 1 para a função logística ou de -1 ou 1 para a função tangente hiperbólica), o processo pode ficar parado ou cair em um mínimo local.

Uma maneira de minimizar este problema é escolher como faixa de trabalho uma porção mais central da função semi-linear, onde a função apresenta um comportamento mais linear. Outra forma de se contornar o problema dos mínimos locais é estabelecer uma tolerância satisfatória para a solução do problema. O processo de aprendizado é interrompido quando a tolerância é atingida. Assim, o fato do mínimo ser global ou local torna-se irrelevante, uma vez que uma solução satisfatória foi encontrada. Se a solução não é satisfatória, pode-se mudar os pesos iniciais e executar novamente a

simulação, com outras taxa de aprendizado e/ou taxa de momento.

Funções de erro alteradas – alguns autores sugerem a alteração da função de erro a ser minimizada como forma de agilização do processo de aprendizado da rede. Franzini (1978), por exemplo, afirma que a utilização da função de erro descrita na equação abaixo, agiliza o processo em 50%, contra a tradicional função de erro médio quadrático.

$$E(W) = -\sum_{i,p} \ln(1 - (D_i - O_i)^2)$$

Alterações no algoritmo de minimização – alguns autores propõem a substituição do algoritmo de minimização, como forma de agilização do processo Waltros (1987) e White (1990), entre outros, utilizaram o *algoritmo de Newton*, enquanto Nowlan & Hinton(1992) utilizaram *o algoritmo de gradiente conjugado*. O problema na utilização do primeiro método é que o algoritmo requer o cálculo da inversa da matriz Hessiana, o que torna praticamente impossível seu emprego em redes de grandes dimensões, enquanto o segundo método apresenta algumas vantagens em relação ao modelo de gradiente decrescente, principalmente em funções quadráticas.

Atualização dinâmica dos parâmetros da rede: Existem autores que sugerem regras empíricas para atualização dos valores da taxa de aprendizado e da taxa de momento. Jacobs(1988) sugere a utilização de uma taxa de aprendizado por peso, enquanto outros (Rigler-1991 e Zeng-1989) sugerem a utilização de uma taxa variável ao longo do treinamento mas única para todos os pesos. Nenhuma destas técnicas se mostrou superior as demais, visto que o desempenho de todas estas implementações estão relacionados ao problema estudado.

4.6-Algoritmos de poda em redes feedforward

Um dos principais problemas quando se usa treinamento supervisionado é a capacidade de generalização, isto é, a capacidade da rede de responder adequadamente a padrões que não fizeram parte do conjunto de treinamento.

Os pesos das ligações são modificados durante o treinamento de forma a minimizar o erro para os padrões apresentados. Se a quantidade de padrões apresentados é suficientemente representativa da função, a tendência durante o treinamento com o algoritmo de *Back-propagation* é que o erro, tanto do conjunto de padrões de treinamento quanto do conjunto de padrões de validação, vá diminuindo à medida que a rede vá generalizando o aprendizado, como pode ser visto na figura 16. Entretanto, após um certo tempo o erro do conjunto de validação atinge um mínimo e começa a crescer, enquanto que o erro do conjunto de treinamento continua a cair, indicando uma tendência da rede de incorporar informações que são específicas do conjunto de treinamento, tal como, por exemplo, os erros de medidas daquele conjunto.

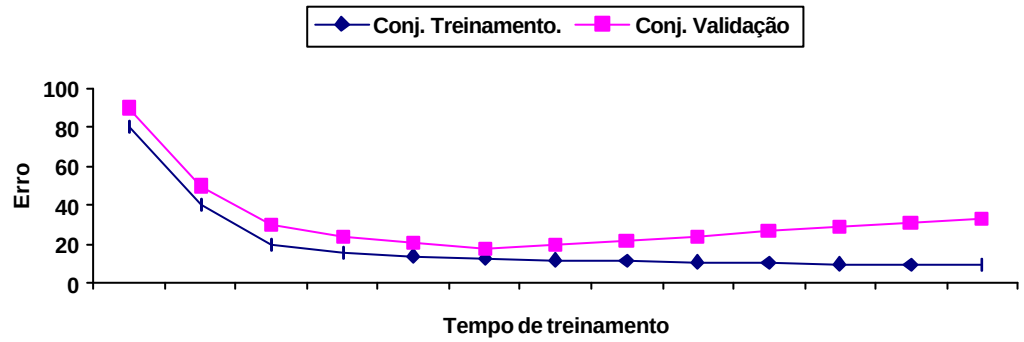


FIG. 16- COMPORTAMENTO TÍPICO DO TREINAMENTO BACK-PROPAGATION

Uma solução é parar o treinamento quando a tendência de erro do conjunto de validação começa a divergir da tendência do conjunto de treinamento. Dessa forma, o conjunto de treinamento é usado para modificar os pesos enquanto que o conjunto de validação é usado para estimar a capacidade de generalização.

Entretanto, isto pode não ser possível se o conjunto de dados disponíveis for pequeno, na medida que a divisão em dois conjuntos pode agravar a falta de representatividade do conjunto de padrões.

Uma outra abordagem para evitar o excesso de treinamento é limitar a capacidade da rede de absorver as correlações espúrias entre os dados de entrada. Isto acontece basicamente quando a rede possui mais graus de liberdade (que são proporcionais ao número de ligações) do que o número de padrões de treinamento. Assim, o problema da generalização está diretamente ligado ao problema do dimensionamento da rede. Ou seja, basicamente a idéia é conseguir a menor rede que consiga acomodar as informações contidas no conjunto de dados de treinamento. O problema é justamente estimar qual é esse tamanho mínimo que permite que a rede ainda aprenda sem incorporar os dados espúrios do conjunto de treinamento.

Estudos no sentido de estimar esse tamanho, relacionam a complexidade do sistema com o número de exemplos de treinamento necessários para aprender uma determinada função. Se o número de exemplos de treinamento for pequeno em relação ao tamanho da rede, o erro de generalização é alto.

A abordagem mais simples, embora bastante ineficiente, é simplesmente executar várias redes de diversos tamanhos e verificar qual é a rede mínima que consegue aprender os padrões de entrada. Mesmo que seja possível determinar a menor rede por este processo, ainda se fica muito susceptível aos parâmetros de treinamento, de forma que pode ser muito difícil determinar se a rede é muito pequena para aprender os padrões, se ela simplesmente aprende lentamente ou se devido a valores de iniciação ou parâmetros de treinamento mal escolhidos ela caiu em algum mínimo local.

A abordagem escolhida por vários autores, parte do princípio comum de iniciar o treinamento com uma rede maior do que o necessário e ir podando (removendo) partes da rede que não sejam necessárias. Como inicialmente a rede é grande, possui

graus de liberdade suficientes para acomodar rapidamente as características gerais dos dados de entrada de uma forma pouco sensível às condições iniciais e aos mínimos locais. Após a acomodação inicial então a rede pode ser podada de forma a eliminar características específicas do conjunto de treinamento, favorecendo portanto os critérios desejáveis de generalidade.

4.6.1 Eliminação de conexões

Uma idéia básica para simplificar a rede é cortar as conexões entre os nós. Para tanto uma primeira abordagem pode ser a de simplesmente zerar o peso de uma conexão e verificar qual a influência disso no erro. Se o erro aumentar muito então restaura-se o peso cortado, se não houver mudança significativa, remove-se definitivamente o peso e espera-se um determinado tempo para que as demais conexões absorvam as características das entradas que estavam representadas no peso cortado. Se a rede possui uma quantidade X de pesos o tempo para cada propagação é da ordem de O(X). Como a operação deve ser executada para cada peso e para cada um dos Y padrões de entrada, então cada passo de poda é da ordem de $O(Y.X^2)$, sem contar o tempo de acomodação. Caso se queira adotar um procedimento mais cauteloso que analise a influência no erro de cada peso para cada padrão e se remova apenas o peso de menor influência, repetindo-se o procedimento até que a menor alteração de erro atinja um limiar de parada, então o tempo é da ordem de $O(YX^3)$. Naturalmente que esses são procedimentos demasiadamente lentos, principalmente para redes de maior porte. Assim, os algoritmos de poda que veremos a seguir adotam outras abordagens.

4.6.2 Métodos de cálculo de sensibilidade

Existem alguns métodos que procuram estimar a sensibilidade da função de erro frente à remoção de uma ligação e então remover as ligações com menor influência. Em

geral esses métodos atuam após o treinamento da rede. Ou seja, primeiramente a rede é treinada para um tamanho maior que o necessário, depois é estimada a sensibilidade de cada peso e então os de menor sensibilidade são retirados.
O problema com esses métodos é que eles não levam em consideração a correlação entre os pesos da rede. A sensibilidade de cada peso w_{ij} é calculada como se ele fosse o único candidato a ser retirado ou como se o nó i fosse o único candidato a ser podado. Quando um dos pesos é retirado, as demais sensibilidades calculadas não continuam necessariamente válidas para a nova configuração da rede.
Isto pode ser facilmente compreendido se pensarmos em um exemplo com dois nós que se cancelem mutuamente na contribuição para a saída. Vistos como um par eles não têm contribuição para a saída mas, vistos individualmente, cada um tem uma forte contribuição e, portanto, provavelmente não serão cortados. Com nós que estejam parcialmente correlacionados acontece o mesmo e eles são bastante comuns na maioria das redes.

4.6.3 Métodos com termos de penalização

A idéia central do algoritmo de *back-propagation* é minimizar uma função de custo, no caso, o erro na saída da rede. Como a hipótese dos algoritmos de poda é de que a menor rede, que seja capaz de responder aos padrões de treinamento, é a que apresentará a melhor generalização, surgiu a idéia de dividir a função de custo em uma soma de dois termos, um representando o custo devido ao erro e outro representado o custo devido à complexidade da rede. Esse termo adicional é chamado de termo de penalização, na medida que representa uma penalização que a função de custo sofre devido à complexidade da rede. Os métodos que introduzem esse termo, procuram minimizar a nova função de custo (que inclui um termo de custo de complexidade). Assim, a tendência é que haja uma redução na complexidade da rede, forçada pela

tendência de que pesos sejam levados para zero para atender a minimização do custo de complexidade.

Decaimento de pesos – Alguns dos métodos que usam termo de penalização incluem termos que provocam um decaimento dos pesos durante o processo de treinamento. Chauvin (1990) adicionou um termo à equação de custo, com esse objetivo:

$$C = m_{er}\sum_{j}^{P}\sum_{i}^{O}(d_{ij} - o_{ip})^2 + m_{\mathrm{w}}\sum_{\mathrm{ij}}^{\mathrm{W}}\mathrm{w}_{\mathrm{ij}}^2$$

A derivada do segundo termo em relação a w_{ij} é $2m_w w_{ij}$, o que introduz efetivamente um termo de decaimento dos pesos na equação do *back-propagation*. Pesos que não sejam essenciais à solução tendem a decair para zero e podem ser retirados. Este método é conhecido como *Weigth Decay*.

Ishikawa (1990) propôs uma outra função de custo da forma:

$$C = \sum_{k\in T}(t_k - o_k)^2 + l\sum_{i,j}\left|w_{ij}\right|$$

O segundo termo introduz uma mudança na regra de atualização dos pesos que passa agora a ter um termo da forma: $-l\ sgn(w_{ij})$. Se $w_{ij} > 0$, o peso é decrementado de l, caso contrário($w_{ij} < 0$), o peso é incrementado de l.

O método de Weigth-elimination – O método com esse nome foi proposto por Weigend (1991) e consiste na adição de um termo de penalização da forma do segundo termo abaixo:

$$\sum_{k\in P}\left(D_k - O_k\right)^2 + l\sum_{i\in C}\frac{w_i^2 / w_0^2}{1 + w_i^2 / w_0^2}$$

Onde C é o conjunto de todas as conexões e w_0 é a escala para as conexões dos pesos. O primeiro termo desta equação mede a performance da rede, enquanto o segundo termo avalia o tamanho (complexidade) da rede. O parâmetro λ representa

a importância relativa do termo de complexidade com relação ao termo de performance.
Weigend mostra o custo de complexidade como função de w_i/w_0 (figura 17). As regiões onde o valores absolutos da conexões $|w_i|$ são muito grandes ou muito pequenos são facilmente interpretadas. Para $||w_i|| >> w_0$, o custo de uma conexão aproxima-se de 1 (multiplicado por λ). Isto justifica a interpretação do termo de complexidade como um contador de conexões com magnitude significativa. Para $|w_i| << w_0$, o custo se aproxima de zero. Grande e pequeno são definidos com relação à escala w_0, que é um parâmetro livre do procedimento para eliminação das conexões. Segundo Weigend a escolha de w_0 de ordem unitária é bom para ativações de ordem unitária.

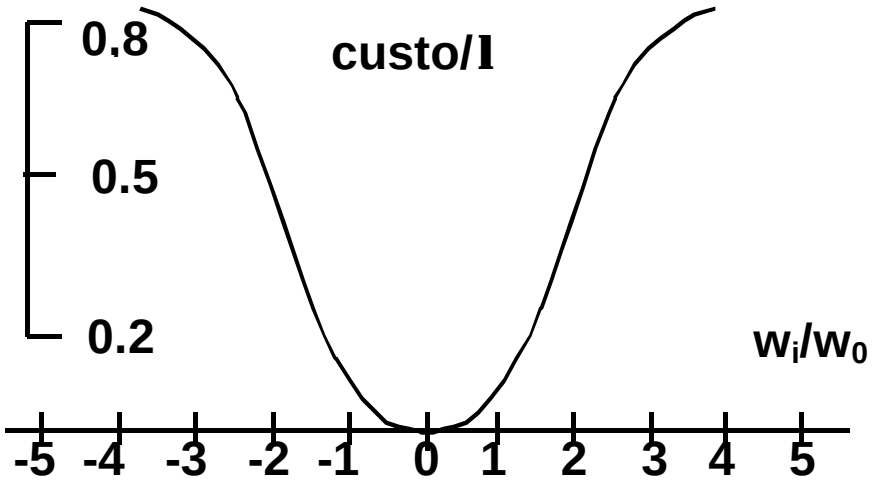


FIG. 17- CUSTO DE COMPLEXIDADE EM FUNÇÃO DO TAMANHO DO PESO

O parâmetro λ é ajustado dinamicamente durante o treinamento. O método de Chauvin (1990) (Weigth Decay - decaimento proporcional ao quadrado do peso) tende a dar preferência a uma grande quantidade de pesos pequenos, em detrimento a poucos pesos com valores altos. O método de Weight-elimination (*Weigend, 1991*) tende a preservar poucos pesos grandes em relação à média dos pesos, eliminando os pesos menores. O parâmetro de escala w_0 permite expressar a preferência por muitos pesos pequenos (w_0 grande) ou poucos pesos com valores de pesos grandes (w_0 pequeno).