Laboratório
Inteligência Computacional Aplicada

Pontifícia Universidade Católica
do Rio de Janeiro

REDES NEURAIS ARTIFICIAIS

Marley Maria Bernardes Rebuzzi Vellasco

2007

Sumário

1.1) Introdução

A expressão “rede neural” é motivada pela tentativa destes modelos imitarem a capacidade que o cérebro humano possui de reconhecer, associar e generalizar padrões. Trata-se de uma importante técnica estatística não-linear capaz de resolver uma gama de problemas complexos. Isso torna o método extremamente útil quando não é possível definir um modelo explícito ou uma lista de regras. Em geral, isso acontece em situações em que o ambiente dos dados muda muito. As principais áreas de atuação são para a classificação de padrões e previsão.

1.2) Histórico

O cérebro humano possui características desejáveis em qualquer sistema artificial. Como exemplo pode-se citar a sua capacidade para lidar com informações inconsistentes e/ou probabilística, alta flexibilidade para se adaptar a situações aparentemente pouco definidas, tolerância a falhas, entre outras. Todas estas características mencionadas despertaram o interesse de pesquisadores que, na década de 80, intensificaram suas linhas de estudo na área de inteligência artificial com o uso da computação intensiva.

No entanto, o aparecimento da neuro-computação ocorreu bem antes, na década de 40. Em 1943, Warren Mc Culloch, psiquiatra e neuroanatomista, e Walter Pitts, matemático, desenvolveram uma máquina inspirada no cérebro humano e um modelo matemático de neurônio biológico artificial denominado *Psychon*. Entretanto, este modelo não era capaz de desempenhar uma de suas principais tarefas: o aprendizado.

Em 1951, Marvin Minsky criou o primeiro neurocomputador chamado *Snark.* A partir de um ponto de partida, o *Snark*, operava bem, ajustando os seus pesos automaticamente. Apesar de não ter executado uma função de processamento de informação relevante serviu como “molde” para futuras estruturas.

Em 1958, Frank Rosenblatt e Charles Wightman junto com alguns outros estudiosos desenvolveram o primeiro neurocomputador bem sucedido. Estes pesquisadores são designados como os fundadores da neurocomputação devido a importância de seus trabalhos para essa linha de pesquisa muito próxima da forma como existe atualmente. Os seus estudo sustentaram os modelos do tipo *perceptron* (redes de um nível) e MLP (Perceptrons de múltiplas camadas). O objetivo inicial era aplicar a modelagem do tipo

Perceptron para o reconhecimento de padrões.

Os modelos baseados no *Perceptron* sofreram graves críticas. Na obra: *"Na Introduction to computacional geometry",* Minsky e Papert, mostraram matematicamente que estes modelos, na forma como estavam, não eram capazes de aprender a função lógica do "OU Exclusivo (Exclusive OR_XOR)". A função XOR possui padrões de valores de entrada e saída cuja associação não poderia ser aprendida pelos modelos baseados em Perceptrons. Esta constatação impactou negativamente as pesquisas que vinham sendo realizadas sobre este assunto nas décadas de 60 e 70.

A partir dos anos 80, os estudos com redes neurais tomaram um impulso revolucionário. Em 1982, John Hopfield, físico mundialmente conhecido, criou um tipo de rede diferente daquelas fundamentadas no *Perceptron.* Neste modelo a rede apresentava conexões recorrentes (sinal não se propaga exclusivamente para frente) e baseava-se num aprendizado não supervisionado com a competição entre os neurônios.

Em 1986, o reaparecimento das redes baseadas em *Perceptrons* foi possível graças a teoria de redes em multinível (MLP) treinadas com o algoritmo de aprendizado por retropropagação *(Backpropagation*) desenvolvida por Rumelhart, Hinton e Willians. Além disso, vale lembrar que a década de 80 foi marcada pelo desenvolvimento de computadores cada vez mais potentes e velozes que permitiram melhores simulações das redes neurais. Neste período também foram desenvolvidos modelos matemáticos que permitiam solucionar o problema do XOR.

A partir de então, um contexto favorável foi criado para o desenvolvimento das pesquisas em neurocomputação:

- (1987): acontece a primeira conferência de redes neurais, a IEEE *Internacional Conference on Neural Networks* em São Francisco. Criou-se ainda a INNS (*International Neural Networks Society).*
- (1989): fundação do INNS journal.
- (1990): criação do *Neural Computation* e do IEEE *Transactions on Neural Networks.*

1.3) Redes Neurais artificiais (RNA`s)

Uma rede neural é um sistema computacional constituído por unidades conhecidas como neurônios. Os neurônios são elementos processadores interligados, trabalhando em paralelo para desempenhar uma determinada tarefa. Os modelos RNA´s constituem uma importante técnica estatística não-linear capaz de resolver uma gama de problemas de grande complexidade. Por isso, são modelos úteis em situações que não é possível definir explicitamente uma lista de regras. Em geral, isso acontece quando o ambiente gerador dos dados muda constantemente. As principais áreas de atuação são para classificação de padrões e previsão.

Para começar a falar de redes neurais, o ponto de partida é definir o que são e como se constituem as suas unidades básicas. Nesta perspectiva, a figura 1 mostra a descrição funcional do k-ésimo neurônio de uma rede, ou seja, descreve o que se encontra no interior do neurônio.

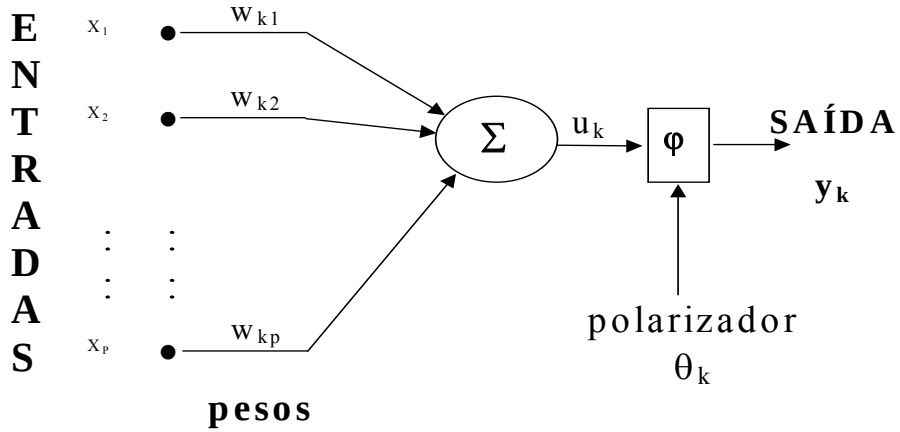

Figura 1: descrição do k-ésimo neurônio

As entradas estão representadas pelos x´s. Esses x´s não são os padrões da camada de entrada, mas a saída do neurônio da camada anterior.

Os w´s representam os pesos ou os parâmetros da rede. Estes pesos representam a "memória" da rede. Isto é, a experiência ganha como resultado das n-apresentações dos padrões. São os pesos que combinam a não-linearidade para que a mesma fique distribuída pela rede.

U_k (net_k) representa a combinação linear dos pesos. Corresponde a soma ponderada da entrada pelos pesos.

Y_k é a saída do k-ésimo neurônio que depende do nível de ativação aplicado ao neurônio pela função de ativação.

Ressalta-se que a função de ativação refere-se a parte não-linear de cada neurônio, sendo o único lugar que a não-linearidade se encontra.

θ_k é conhecido como termo polarizador. Indica o ponto em que a função se encontra em cima do eixo e, portanto, define o domínio dos valores de saída. Na modelagem recorre-se ao artifício de tratar este termo como mais um peso de modo que, durante o processo de otimização dos pesos, a ser realizado pelo algoritmo implementado, a atualização aconteça para todos os parâmetros, incluindo para o polarizador.

Em uma rede neural os parâmetros a serem estimados são os pesos e o polarizador. Como em cada neurônio chega a soma ponderada de todas as entradas, então o polarizador aparecerá associado a uma entrada fixa +1 ou -1.

1.4) Função de Ativação

Ainda sobre o que se encontra na estrutura interna de cada neurônio, sobre a função de ativação, que de acordo com a não-linearidade irá restringir a amplitude do intervalo de saída do neurônio, existem quatro tipos principais a serem destacados em redes neurais.

a) Função sigmóide ou logística: assume valores sempre positivos.

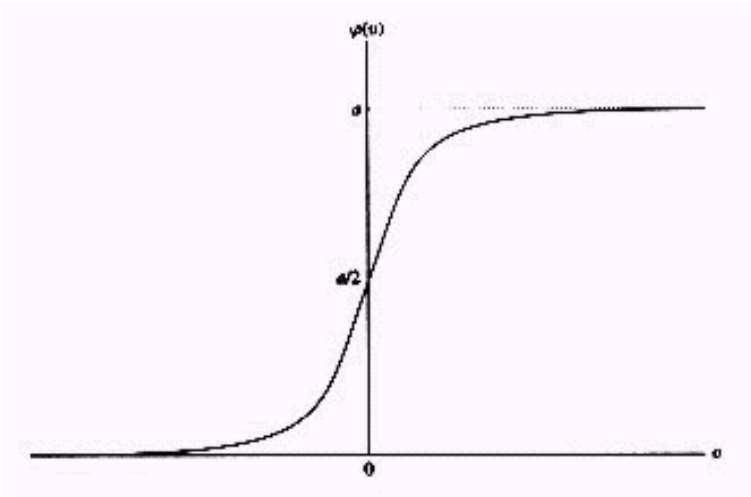

Figura 2: função sigmóide

$F(x) = 1 \ / \ 1+ e^{-x}$ ou (saída) $= 1/1+ e^{-uk}$

b) Função hiperbólica: a saída pode assumir valores positivos e negativos.

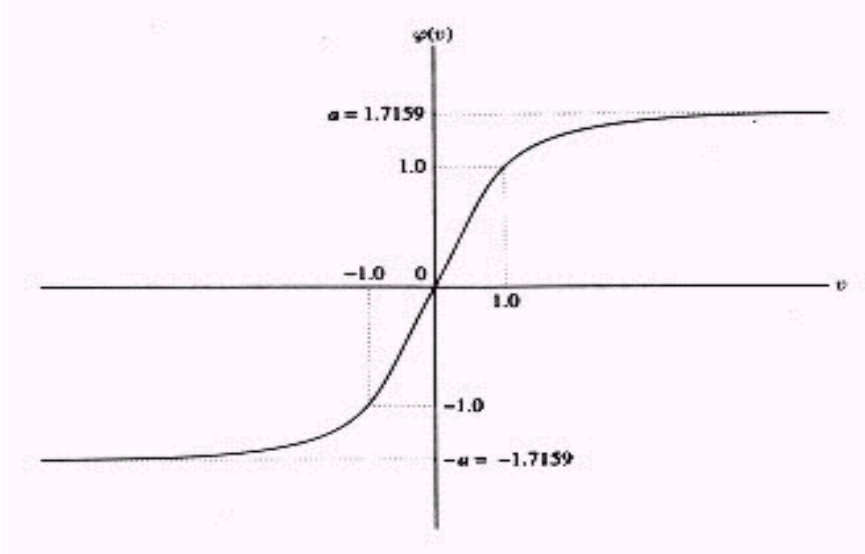

Figura 3: função hiperbólica

Quando usar uma ou outra mais ou menos tanto faz. Depende da aplicação a ser feita. Por exemplo, em séries de preço futuro de uma *commodity*, sabe-se que o valor previsto pela rede será um valor positivo, assim o bom senso tomará por mais coerente aplicar a função sigmóide na modelagem do problema.

Uma outra função muito utilizada é a <u>função linear.</u>

c) <u>Função linear</u>: usada principalmente em neurônios da camada de saída quando não é desejável o efeito de saturação das funções sigmóides e hiperbólicas. Este efeito de saturação está colocado na figura 4.

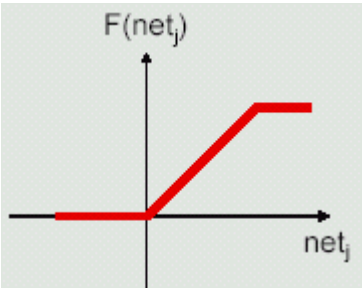

Figura 4: função linear

d) função degrau: valores rígidos.

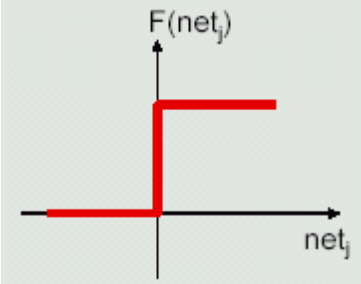

Figura 5: Função degrau.

Devido ao efeito de saturação, sempre se deve iniciar a rede em algum ponto da parte linear das funções (a) e (b). Um exemplo do efeito de saturação está colocado na figura 6.

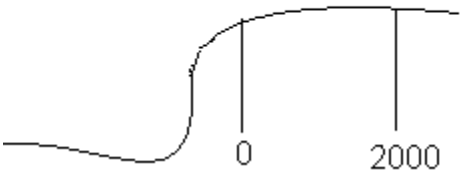

Figura 6: efeito de saturação

Como pode ser interpretado na figura 6, tudo o que não se quer é que na saída a rede forneça como resultado que 0 é igual a 2000 (efeito de saturação).

A descrição do neurônio permitiu compreender melhor o que são as unidades básicas que compõem uma RNA e agora o próximo passo é falar sobre o modelo da rede neural.

1.5) Etapas de modelagem de uma rede neural

As etapas de modelagem de uma rede neural envolvem essencialmente três passos:

1) Treinamento e Aprendizado: obtido pelo ambiente gerador dos dados.
2) Associação: reconhecimento de padrões distintos.
3) Generalização: relacionado a capacidade da rede de reconhecer com sucesso o ambiente que origina os dados e não propriamente os dados utilizados no treinamento.

A modelagem inicia-se escolhendo os exemplos a serem usados para o treinamento. Essas observações podem ser do tipo " rotuladas" ou não. Se forem rotuladas significa que haverá um conjunto de pares de entrada-saída desejável. Nesta etapa também devem ser selecionados os exemplos que servirão para validar o modelo no momento de testar a sua capacidade de generalização.

Como uma rede neural só aprende a partir de dados, a escolha das variáveis de entrada é de grande importância. Isso porque embora a rede neural tenha condições de modelar problemas difíceis de especificar, é preciso que existam dados e observações suficientes e representativas para o conhecimento ser extraído e para que o aprendizado da RNA aconteça com sucesso.

O conhecimento é passado para a rede por um algoritmo de treinamento e o aprendizado é transformado e armazenado em densidades de conexões que são os pesos. O aprendizado é o resultado das muitas apresentações de um determinado conjunto de exemplos de treinamento. Neste contexto, é válido destacar o conceito de época. Época significa uma apresentação completa de todo o conjunto de treinamento.

O treinamento da rede pode se dar de duas maneiras: batelada ou incremental.

1) batelada ou por ciclos: a atualização dos pesos acontece somente após a apresentação de todos os padrões. Cada padrão é avaliado na mesma configuração de pesos.
2) Padrão a Padrão ou incremental: o algoritmo faz a atualização dos pesos após a apresentação de cada novo padrão. Por isso mesmo, a freqüência das atualizações em um mesmo período tende a ser maior que no caso anterior.

Como por este critério o algoritmo tende a levar a rede a aprender melhor o último padrão apresentado, é interessante tornar a apresentação dos exemplos aleatória. A eficiência dos dois métodos depende do problema em questão.

O aprendizado é mantido de época em época até que os pesos e o polarizador se estabilizem e o erro médio quadrado sobre todo o conjunto de treinamento convirja para uma valor mínimo e o objetivo pretendido seja atingido.

Ressalva-se que minimizar o erro quadrado médio é um dos critérios para o ajuste de pesos, mas não o único. Existem outros critérios, como por exemplo, a maximização da informação mútua cujo objetivo é maximizar a informação entrada-saída.

1.6) Arquitetura da rede neural: redes recorrentes e redes não recorrentes

Voltando aos três passos da modelagem de dados por uma RNA (lembrando: passo1: seleção do conjunto de treinamento e validação), o próximo passo diz respeito a definição da topologia ou arquitetura da rede neural. Existem basicamente dois tipos de topologia:

1) redes não recorrentes
2) redes recorrentes

A arquitetura é determinante na capacidade de processamento de uma RNA. A escolha correta do número de conexões é decisiva para um treinamento bem sucedido.

As redes não recorrentes são aquelas que não possuem realimentação de suas saídas para as suas entradas e, por isso, são ditas "sem memória". A estrutura dessas redes pode ser formada por uma camada única ou por multi-camadas. No caso de redes em camadas existe um conjunto de neurônios de entrada, uma camada de saída e uma ou mais camadas intermediária ou oculta. Para alguns autores, as entradas não se constituem como uma camada da rede devido ao fato de apenas distribuir padrões.

Por se tratarem de redes não recorrentes, não existem conexões ligando um neurônio de uma camada a outro neurônio de uma camada anterior nem a um neurônio de uma mesma camada. Um exemplo desse tipo de arquitetura são as redes do tipo

"*feedfoward*", em que o sinal é sempre propagado para a frente, da entrada para a saída.

As redes *feedfoward*, necessariamente estão organizadas em camadas e são largamente utilizadas atualmente.

Chama-se a atenção para a distinção dos seguintes conceitos: sinal funcional e sinal de erro.

a) sinal funcional: é um sinal de entrada (estímulo que incide na entrada da rede) e se propaga para a frente (neurônio por neurônio) através das camadas da rede e termina na saída da rede como um sinal de saída. Este sinal é chamado como sinal funcional porque em cada neurônio da rede pelo qual o sinal passa, esse sinal é calculado como uma função das entradas pelos pesos associados aquele neurônio.

b) Sinal de erro: origina-se no neurônio de saída e se propaga para trás (camada por camada) através da rede.

As RNA´s recorrentes são redes mais gerais que contêm realimentação das saídas para as entradas, sendo suas saídas determinadas pelas entradas atuais e pelas saídas anteriores. Além disso, a sua estrutura não é obrigatoriamente organizada em camadas e se forem as redes podem apresentar interligações entre neurônios da mesma camada e entre camadas não consecutivas.

Por possuírem realimentação, as redes recorrentes respondem a estímulos dinamicamente, ou seja, após aplicar uma nova entrada, a saída é calculada e então realimentada para modificar a entrada. Por isso, essas redes são ditas "com memória".

Existe ainda o caso mais geral em que as redes são totalmente recorrentes (cada neurônio conectado a qualquer outro) e existe o caso de redes parcialmente recorrentes, como por exemplo a rede de Elman, em que os elos de realimentação ocorrem entre a saída e a entrada da primeira camada oculta. O elo de realimentação é feito através de uma unidade de contexto (camada extra), normalmente uma estrutura de atraso Z-1, que armazena a saída da primeira camada oculta por um passo de tempo. Por essa estrutura gerar padrões variáveis no tempo, este tipo de configuração neural pode ser útil em

aplicações de séries temporais porque ajudam a rede a memorizar as informações atuais no momento seguinte da seqüência investigada. A realimentação ocorre da saída de cada neurônio da camada oculta para todos os neurônios da mesma camada. Além dessa camada recorrente, a rede pode apresentar várias outras camadas do tipo MLP e pode também ter uma ou várias saídas.

Neste tipo de rede o algoritmo utilizado para o treinamento é o de retropropagação do erro. Entretanto, é preciso tomar cuidado na implementação deste algoritmo uma vez que num dado instante de tempo a rede recebe não apenas as entradas externas, mas também as realimentadas da saída da primeira camada oculta obtidas no instante anterior.

Abaixo seguem nas figuras 7 e 8, 9 respectivamente, uma arquitetura de rede não recorrente e de rede recorrente. de Elman e no caso de rede recorrente geral.

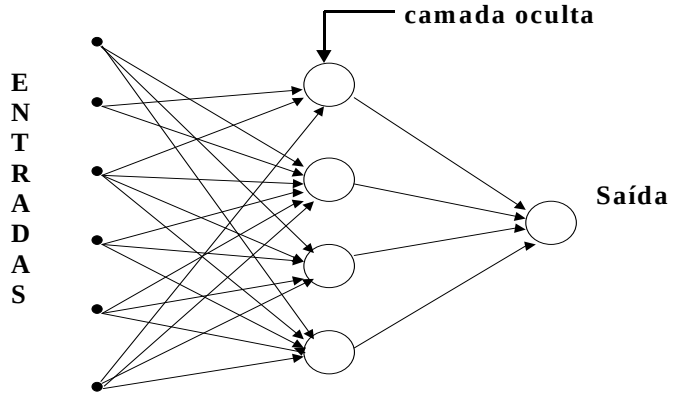

Figura 7: rede não recorrente_*feedfoward*

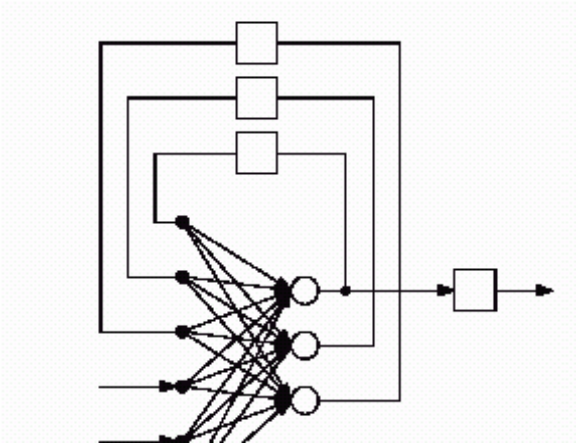

Figura 8: rede recorrente (caso redes Elman)

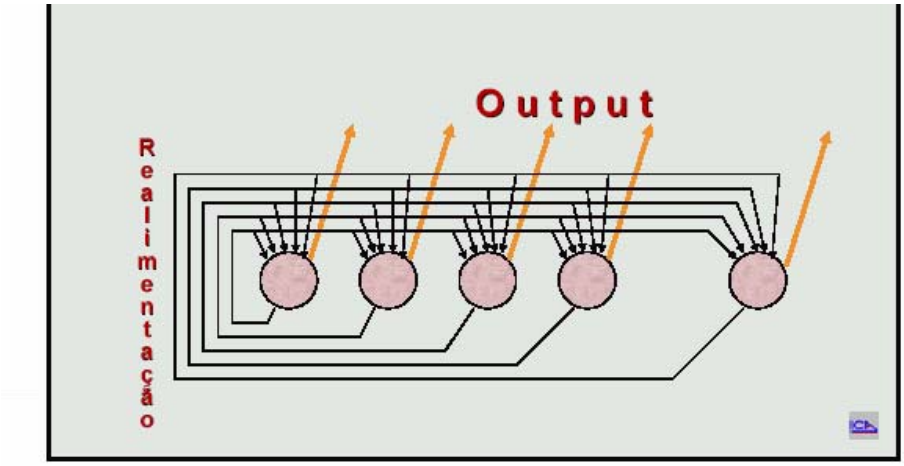

Figura 9: redes recorrentes - Hopfield

1.7) Treinamento das RNA´s: supervisionado e não supervisionado

Retornando a etapa do treinamento, o objetivo do treinamento de uma RNA é fazer com que a aplicação de um conjunto de entradas produza um conjunto de saídas desejado ou no mínimo consistentes. Durante o processo de treinamento, os pesos da rede gradualmente convergem para determinados valores, de modo que aplicação dos vetores de entrada produza as saídas necessárias.

O aprendizado relaciona-se a maneira pela qual a modificação dos parâmetros ocorre. Neste processo os parâmetros são ajustados através de um processo estimulado pelo ambiente no qual a rede está inserida. O algoritmo de aprendizagem refere-se a regras bem definidas para a solução de um problema de aprendizado. Os algoritmos diferem entre si pela forma como se dá o ajuste dos pesos.

Os procedimentos de treinamento e aprendizado podem ser classificados em dois tipos:

1) supervisionado
2) não supervisionado

Na aprendizagem supervisionada existem 3 componentes interrelacionados:

- Ambiente (estacionário): as características estatísticas não mudam com o tempo. Assim, em teoria, as estatísticas podem ser aprendidas pela rede por aprendizado supervisionado. Se o ambiente não é estacionário as estatísticas variam com o tempo. Neste caso, os métodos tradicionais podem ser inadequados porque a rede não está equipada para seguir as variações estatísticas do ambiente em que opera.

- O professor (resposta desejada): de certa forma, o aprendizado supervisionado funciona como se existisse um professor que conhece a resposta desejável (ação ótima a ser tomada pela rede).

- Algoritmo de aprendizado (ajusta os pesos para fazer o mapeamento entrada-saída).

O treinamento supervisionado acontece da seguinte forma: os exemplos utilizados para o treinamento são do tipo "rotulados", ou seja, existe um vetor de entrada e um vetor alvo que se deseja na saída. O processo funciona aplicando-se o vetor de entrada e, então, a saída fornecida pela rede é comparada com o vetor de resposta desejado. Dessa comparação obtém-se um sinal de erro. O erro encontrado é realimentado através da rede e os pesos são atualizados segundo um algoritmo determinado a fim de minimizar este erro. O algoritmo de aprendizado supervisionado mais utilizado é o ***Backpropagation*** (que será detalhado mais adiante). Portanto, o aprendizado supervisionado pressupõe um erro de saída.

É importante que os valores alvo (resposta desejada) sejam escolhidos dentro do intervalo da função de ativação. Isto é dj (resposta desejada para o neurônio 'j' na camada de saída da rede deve ser deslocada uma quantidade ξ afastada do valor limite da função de ativação. Caso contrário, o algoritmo de retropropagação tende a levar os parâmetros livres da rede para o infinito, reduzindo a velocidade do processo de treinamento podendo levar os neurônios ocultos a saturação.

Exemplo:

Considerando a função de ativação hiperbólica com valores limites +a ou -a. A resposta desejada deve ser colocada como: $dj = a - \xi$ e $dj = -a - \xi$

Quanto ao treinamento não supervisionado, não existe um vetor de resposta desejada, logo, não existem comparações que forneçam um sinal de erro. Os exemplos são "não rotulados". Nesta situação, são fornecidas a rede condições para realizar uma medida independente da tarefa que deve ser aprendida e os parâmetros livres da rede são otimizados em relação a esta medida.

Uma vez que a rede esteja ajustada às regularidades estatísticas dos dados de entrada, a rede desenvolve a habilidade de formar representações internas para codificar as características de entrada e criar automaticamente novas classes. Para esse tipo de treinamento pode-se utilizar a regra de aprendizagem competitiva. A rede de Kohonem (será vista mais adiante) é um exemplo de rede com treinamento não supervisionado.

A rede neural reconhece um padrão passando por uma seção de treinamento, durante a qual se apresenta repetidamente à rede um conjunto de padrões de entrada junto com a categoria a qual cada um pertence. Em seguida, apresenta-se à rede um padrão que nunca foi visto, mas que pertence a população de padrões utilizados para o treinamento e a rede é capaz de identificar a categoria correta daquele padrão particular por causa da informação extraída no aprendizado. Essa capacidade de associação, isto é, as condições de reconhecer padrões distintos, relaciona-se ao passo_2 das etapas de modelagem (passo_1: treinamento e aprendizagem). O passo 3 refere-se a capacidade de generalização.

Uma rede tem boa capacidade de generalização quando consegue fazer um mapeamento entrada-saída correto mesmo se a entrada é um pouco diferente dos exemplos apresentados. Isso deve ser analisado através dos resultados fornecidos pela rede *in-sample* (dados do conjunto para o treinamento) e *out-of-sample* (dados selecionados para validar o modelo).

Quando a rede aprende um número excessivo de exemplos, acaba memorizando os dados do treinamento, ou seja, uma rede treinada em excesso perde a capacidade de generalização dos padrões de entrada-saída similares (há excesso de ajuste: *overfitting*). Neste caso, obtém-se um excelente resultado *in-sample* e um péssimo ajuste *out-of-sample.*

Também quando se utiliza mais neurônios ocultos que o necessário, isso faz com que contribuições indesejáveis da entrada fiquem armazenadas nos pesos e a rede aprende ruídos.

A generalização é influenciada por 3 fatores:

1) O tamanho do conjunto de treinamento e o quanto os exemplos são representativos do ambiente de interesse.
2) A arquitetura da rede neural.
3) A complexidade física do problema (sobre este fator não se tem controle).

Se por um lado o excesso de complexidade do modelo prejudica o resultado *out-of-sample*. Por outro lado, a falta de complexidade pode ser observada na análise *in-sample*.

As figuras 10, 11 e 12 permitem ilustra melhor a questão da complexidade do modelo. Suponha que o modelo adequado para o problema representado na figura 10 seja uma parábola . Os x´s e as bolinhas representam dois padrões distintos e da forma como estão distribuídos não é possível traçar uma reta para tratar o problema linearmente, separando os padrões x's de um lado e bolinhas de outro. Assim, essa curvinha na parábola é necessária para executar a separação padrões eficientemente sendo uma alternativa não-linear para resolver o problema.

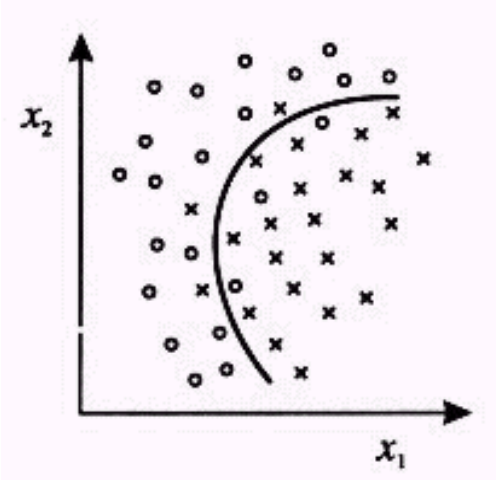

Figura 10: modelo "adequado" para o problema desenhado

Primeiramente será relatado uma situação em que o modelo de rede neural foi construído com mais complexidade que a necessária. O excesso de complexidade está retratado na figura 11. Supondo que o modelo adequado para o problema é uma parábola

(figura 10) e o problema foi modelado por uma curva cheia de vales e picos, portanto, um polinômio acima de grau 2 (figura 11), neste caso, pode-se obter um excelente resultado com os exemplos do treinamento, mas os resultados com os dados de validação possivelmente não serão bons.

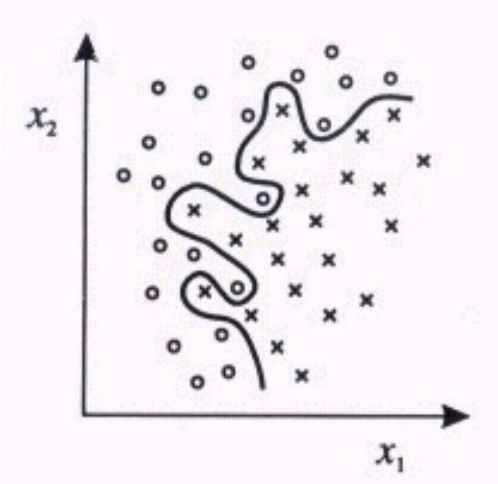

Figura 11: modelo com excesso de complexidade para o problema

Agora, imagine que o modelo de rede desenvolvido para o resolver o problema apresentado na figura 10 esteja mais simples do que a modelagem necessária para uma solução satisfatória do problema. Conforme a figura 12, percebe-se que a questão está mal representada pela falta de complexidade aplicada ao modelo. Ou seja, a modelagem considerada adequada é utilizar uma parábola (figura 10) e se está usando uma reta (equação de 1º grau) para a representação, então, o desempenho da rede no conjunto de treinamento e de validação ficarão abaixo das expectativas.

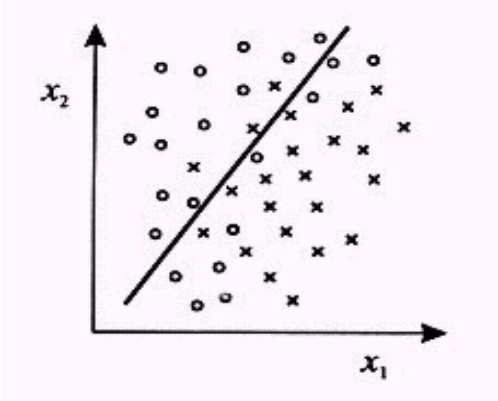

Figura 9: modelo com falta de complexidade para o problema

Obviamente as figuras anteriores é um exemplo bastante simplista que, porém, permite esclarecer a idéia sobre a importância de se escolher um modelo adequado para representar um determinado problema e, conseqüentemente, obter resultados satisfatórios. As próximas figuras (13 e 14) também retratam a importância de uma representatividade adequada do problema, com relação a complexidade, para se obter um bom desempenho do modelo.

Figura 13: exemplo 2 (excesso de complexidade)

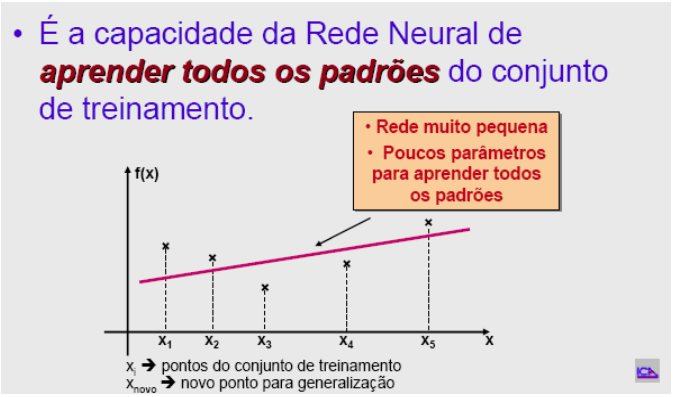

Figura 14: exemplo 2 falta de complexidade)

Ainda com relação a complexidade do modelo, o número de camadas ocultas e de unidades nestas camadas é de extrema importância. Quanto maior o número de neurônios, mais pesos para ajustar e mais complexa será a rede neural. É necessário mexer na complexidade do modelo até que os dados estejam bem ajustados considerando o conjunto de treinamento e de validação.

Um método utilizado com sucesso para especificar adequadamente o modelo a ser adotado é o da Regularidade Bayesiana (implementado no matlab). Este método consiste

em realizar várias tentativas de redes jogando em determinados momentos os parâmetros para zero (“matando neurônios”). Assim, várias redes são geradas. Por este critério a meta não é propriamente minimizar o erro que está sendo cometido dentro e/ou fora da amostra, o objetivo da técnica é definir onde colocar mais linearidade ou não. Isto é, não se deve responder com mais não-linearidade quando se tem linearidade. A regularidade do mapeamento está fortemente associada a uma boa generalização.

Assim, em resumo, o número de entradas tem haver com o número de atributos necessários para o treinamento. As entradas devem ser variáveis informativas e descorrelatadas. Do contrário, podem atrapalhar a rede na tarefa de aprendizagem. Ainda em relação aos dados de entrada é útil realizar uma normalização dos padrões. As variáveis de entrada devem ser pre-processadas de modo que seu valor médio calculado sobre todo o conjunto de treinamento seja próximo de zero, ou seja, pequeno quanto ao todo do conjunto.

Também para acelerar o treinamento as variáveis descorrelatadas podem ser escaladas para que suas covariâncias sejam aproximadamente iguais. Assim, os pesos da rede aprendem aproximadamente a uma mesma velocidade.

Para exemplificar a questão da normalização, no caso de uma série temporal com tendência e sazonalidade, é pertinente primeiramente identificar e excluir este comportamento da série (que representam padrões lineares). Desta forma, a rede poderá melhor identificar os padrões não-lineares, deixando para os métodos clássicos o tratamento das componentes lineares. Ainda é preciso verificar a presença se *outlies* (pontos discrepantes).

Quanto ao número de neurônios na camada de saída, dependerá do modelo.

Para as camadas ocultas, como foi demonstrado, sua complexidade dependerá dos resultados com os dados de treinamento e de validação. A figura 15 ilustra esse ponto da questão.

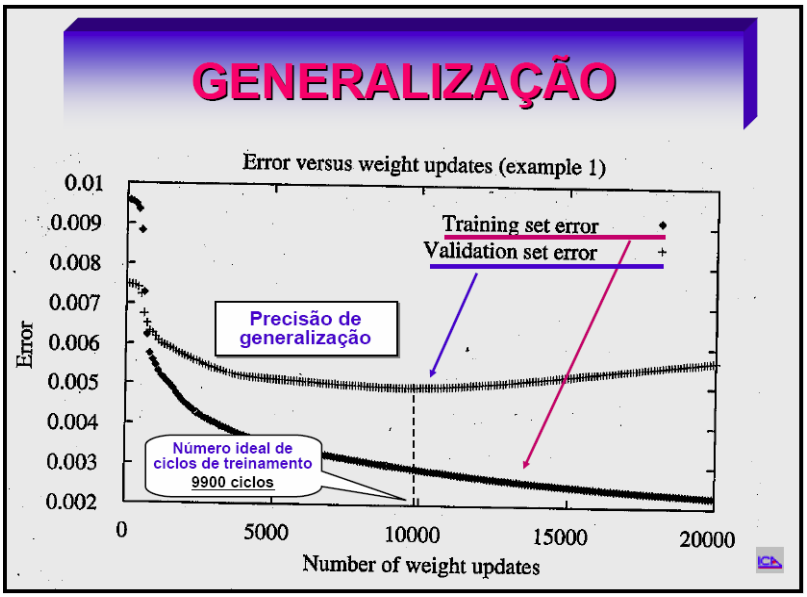

Figura 15: resultados de treinamento e de validação

Verifica-se que o erro de treinamento cai a cada nova época. Porém, o erro do conjunto de validação cresce quando o número de épocas aumenta, além do necessário para treinar a rede. Ou seja, o excesso de treinamento faz a rede perder a sua capacidade de generalização e o erro aumenta fora da amostra decorrente deste fato.

A validação cruzada é uma alternativa eficiente para se averiguar o poder de generalização do modelo obtido. A validação cruzada consiste em se dividir a amostra em partes. Por exemplo, para 150 observações, faz-se 10 grupos de 15. Pega-se 9 para treinar a rede e 1 para validar o modelo. A idéia do procedimento consiste em criar grupos e combiná-los várias vezes.

I
II
III
IV

Cada retângulo da tabela representa um pedaço da amostra. Pela validação cruzada, faz-se a combinação N vezes desses retângulos e encontra uma média que servirá para detectar o potencial do modelo desenvolvido.

1.8) Perceptron

A rede do tipo "perceptron" consiste basicamente de um modelo como apresentado a seguir.

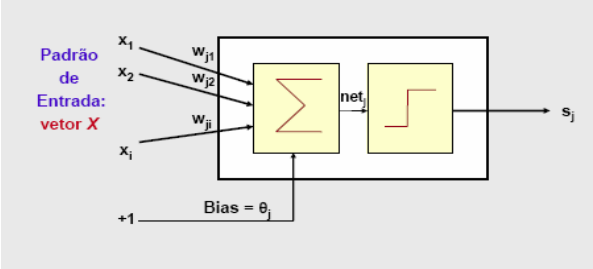

Figura 16:modelo simplificado do perceptron

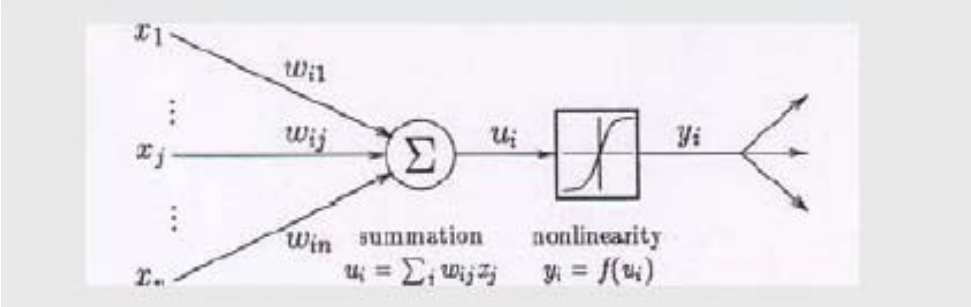

Figura 17: estrutura interna do perceptron

De acordo com as gravuras 16 e 17, observa-se que o perceptron se trata de uma rede de uma camada. Neste caso, geralmente, a rede é constituída por um único neurônio e um polarizador. Uma rede neural sem camada oculta só consegue classificar padrões que sejam linearmente separáveis (ou seja, padrões que se encontram em lados opostos de um hiperplano).

Características básicas do perceptron:

- função de ativação: degrau
- topologia: uma única camada de neurônios
- algoritmo de aprendizado: supervisionado.
- valores entrada/saída: binários [-1, +1].

Para essa rede funcionar adequadamente, as duas classes C1 e C2 devem ser linearmente separáveis, isto é, os padrões a serem classificados devem estar suficientemente distantes entre si para assegurar que a superfície de decisão consiste em um hiperplano. Se as duas classes se aproximarem demais, tornam-se não-linearmente separáveis, uma situação que está além da capacidade do neurônio. Isto está mostrado nas figuras 18, 19 e 20.

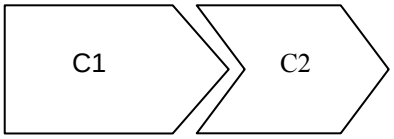

Figura 18: padrões não-linearmente separável

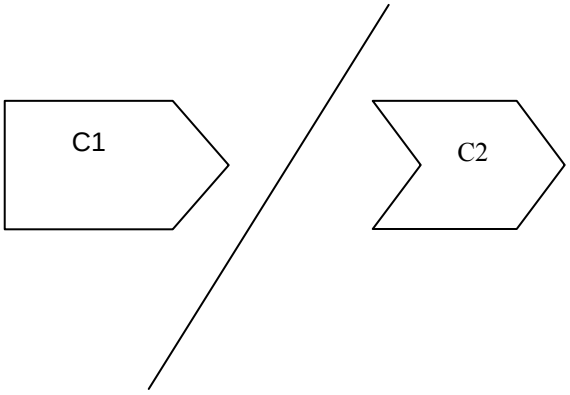

Figura 19: padrões linearmente separáveis

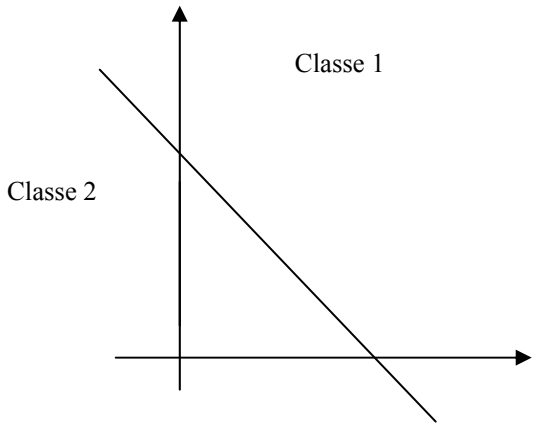

Figura 20: (problema linearmente separável)

Assim, não há como separar linearmente por uma reta os padrões A0 e A3 de um lado e os padrões A1 e A2 de outro lado. Este fato está ilustrado nas figuras 21 e 22;

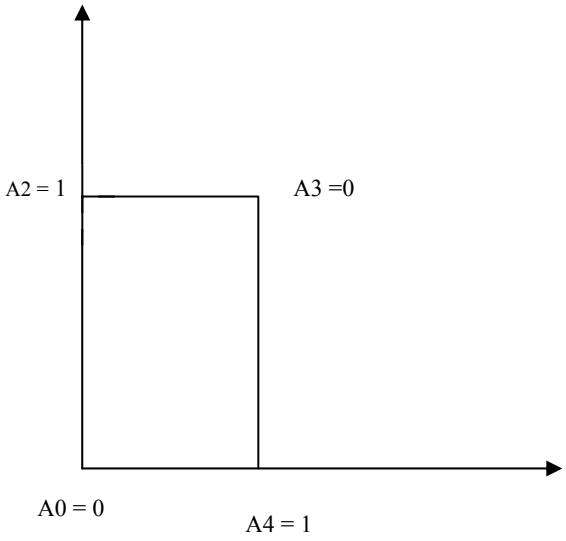

Figura 21: representação 1 problema XOR

O problema OU-EXCLUSIVO

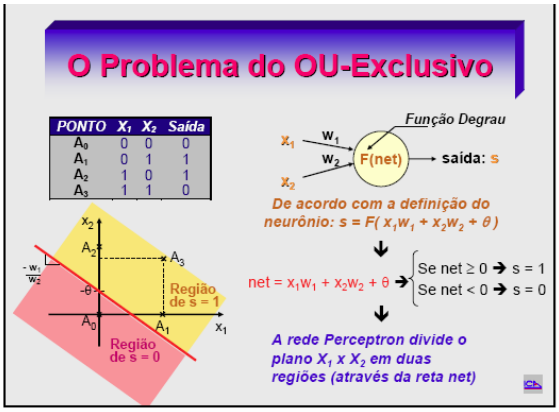

Figura 22: representação 2 problema XOR

1.9) MLP (redes de múltiplas camadas) e o algoritmo de retropropagação

As redes em camada são tipicamente constituídas por uma camada de entrada, uma ou mais camadas ocultas e uma camada de saída. O sinal se propaga sempre para a frente, camada por camada. A rede MLP tem sido aplicada a problemas através de seu treinamento

de forma supervisionada com o algoritmo de retropropagação do erro. Este algoritmo é baseado na regra de correção de erro. A figura 23 desenha um exemplo genérico de redes em camadas.

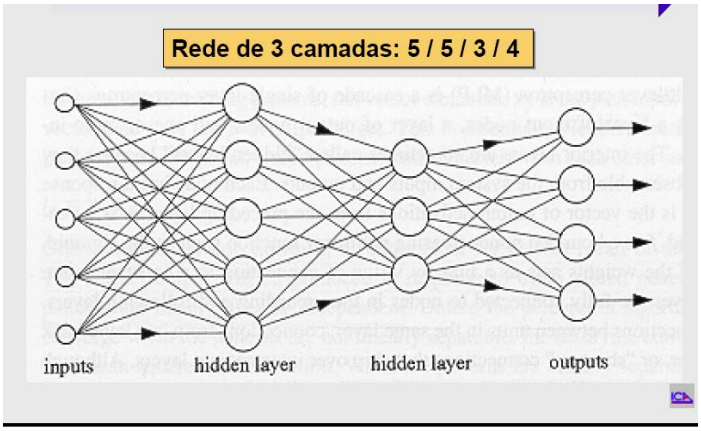

Figura 23: redes em múltiplas camadas

Existem três características básicas das redes em camadas:

a) o modelo de cada neurônio da rede inclui uma função de ativação não-linear. Além disso, a não-linearidade é do tipo suave (diferenciável em qualquer ponto). Uma forma de não-linearidade normalmente usada que satisfaz esta exigência é uma não-linearidade sigmóide. A não-linearidade é importante, pois, do contrário, a relação de entrada-saída da rede seria reduzida a forma existente em uma rede de camada única.

b) A rede contém uma ou mais camadas de neurônios ocultos que não são parte da entrada e nem da saída da rede. Estes neurônios da camada oculta capacitam a rede a aprender tarefas complexas, extraindo progressivamente as características mais significativas dos padrões de entrada.

c) A rede exibe um alto grau de conectividade determinado pelos seus pesos. Uma modificação na conectividade da rede requer modificações nos pesos.

Estas três características juntamente com a habilidade de aprender da experiência do treinamento são responsáveis pelo alto poder computacional da rede e também por suas deficiências a respeito do conhecimento sobre o comportamento da rede. A saber sobre essas deficiências mencionadas:

a) a presença de uma forma distribuída de não-linearidade e a alta conectividade da rede torna difícil a análise teórica de uma MLP.

b) a utilização de neurônios ocultos torna o processo mais difícil de ser visualizado. Em um sentido implícito, o processo de aprendizagem deve decidir quais as características dos padrões de entrada serão representados por neurônios ocultos.

Em redes de múltiplas camadas identifica-se dois tipos de sinais: sinal funcional e sinal de erro e na aplicação do algoritmo de retropropagação distinguem-se dois passos: a propagação e a retropropagação.

O algoritmo de retropropagação do erro é o algoritmo de treinamento supervisionado mais difundido. Este algoritmo juntamente com as redes em camada foram os responsáveis pelo ressurgimento do interesse de pesquisas na área de neurocomputação (detalhes na seção 1.1). As redes que não possuem camada oculta (perceptron) são apenas capazes de resolver problemas que são linearmente separáveis. Neste caso, quando os padrões são não-linearmente separáveis surge o problema do Ou-Exclusivo (XOR). O problema do XOR pode ser resolvido utilizando uma única camada oculta com dois neurônios como é mostrado na figura 24.

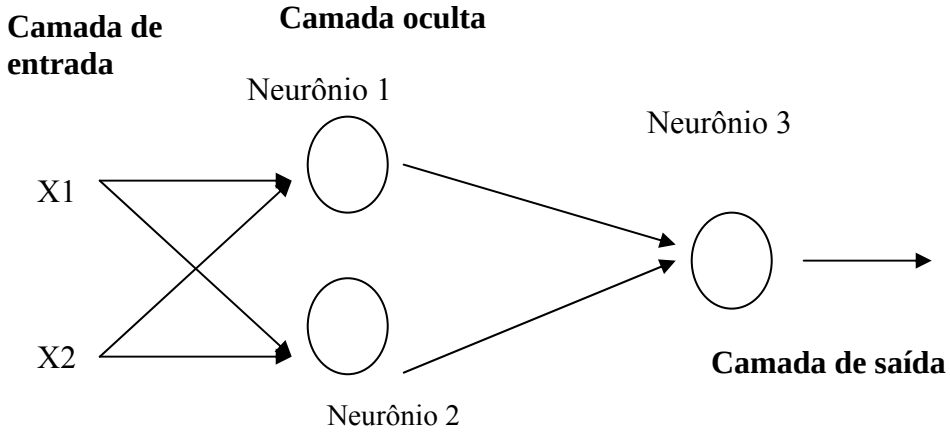

Figura 24: rede neural para resolver o problema XOR

O algoritmo de retropropagação aparece como uma alternativa para treinar redes em camada (MLP). Consiste em calcular o erro na saída da rede e retropropagá-lo pela rede, modificando os pesos para minimizar o erro da próxima saída. O algoritmo baseia-se no método do gradiente descendente cuja idéia central é fazer modificações proporcionais ao gradiente do erro. A direção do gradiente é onde o erro é minimizado. O método utiliza a 1º derivada de todas as inclinações.

Basicamente o algoritmo de retropropagação consiste nos seguintes passos:

1. calcular o erro na da rede
2. retropropagar o erro e modificar os parâmetros para minimizar o erro da próxima saída.

A aprendizagem por retropropagação do erro pode ser descrita como:

a) um passo para a frente: propagação (o vetor de entrada é aplicado aos neurônios da rede e seu efeito se propaga através da rede, camada por camada, até finalmente produzir o conjunto de saída (resposta real da rede). Durante este passo, os pesos da rede ficam fixos.
b) Um passo para trás: retropropagação (os pesos são ajustados de acordo com uma regra de correção de erro. A resposta real da rede é subtraída da resposta desejada (alvo) para produzir um sinal de erro. Este sinal é propagado para trás através da rede, daí o nome retropropagação de erro. Os pesos são ajustados para aproximar a resposta da rede a resposta desejada (num sentido estatístico).

Na saída da rede como existe uma resposta desejável, existe um erro. Mas na camada oculta o erro não tem um sentido físico. Portanto, os neurônios de saída são as únicas unidades visíveis para as quais o sinal de erro pode ser diretamente calculado. Dessa forma, o algoritmo oferece um tratamento diferenciado aos neurônios da camada oculta e da camada de saída. O objetivo é minimizar o erro médio. Para isso são feitas modificações nos pesos padrão a padrão.

O raciocínio para implementar o algoritmo inicia-se da seguinte forma: se as

modificações nos pesos são proporcionais ao gradiente do erro e este gradiente é calculado utilizando a regra da cadeia dada pela expressão da figura 11.

$$\frac{\partial \xi(n)}{\partial w_{ji}(n)} = \frac{\partial \xi(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial \upsilon_j(n)} \frac{\partial \upsilon_j(n)}{\partial w_{ji}(n)}$$

Figura 25: gradiente do erro

Então, é preciso obter as derivadas dessa expressão:

$$e_j(n) = d_j(n) - y_i(n) \qquad (1)$$

Onde:

- j :refere-se ao neurônio de saída.
- n: interação
- ej: erro obtido propagando-se a entrada para frente
- dj: resposta desejada
- yj:é a saída da rede

A função custo é dada por:

$$\xi(n) = \frac{1}{2} \sum_{j \in C} e_j^2(n) \qquad (2)$$

Sabendo que ‘c’ é o conjunto de todas as unidades da camada de saída.

A função custo é uma medida de desempenho. O somatório refere-se a soma dos erros de todos os neurônios da camada de saída.

Em seguida, pega-se a função custo e tira-se a média para todos os padrões, obtendo-se o erro médio com relação aos “n” padrões:

$$\xi_{av} = \frac{1}{N}\sum_{n=1}^{N}\xi(n)$$

(3)

O próximo passo é calcular a entrada do j-ésimo neurônio:

$$\upsilon_j(n) = \sum_{i=0}^{p} w_{ji}(n)y_i(n)$$

(4)

Lembrando que w_0 = polarizador e ‘ p’ é o número total de ligações que chegam ao neurônio j.

A saída do neurônio ‘j’ na interação ‘n’ é dada por:

$$y_j(n) = \varphi_j(\upsilon_j(n))$$

(5)

Como pode ser visto, a saída é dependente do nível de ativação aplicado ao neurônio, que está relacionado a função de ativação.

Quanto a resolução das derivadas da figura 25:

$$\frac{\partial\xi(n)}{\partial e_j(n)} = e_j(n)$$

(6)

$$\frac{\partial e_j(n)}{\partial y_j(n)} = -1.$$

(7)

$$\frac{\partial y_j(n)}{\partial \upsilon_j(n)} = \varphi'_j(\upsilon_j(n))$$

(8)

$$\frac{\partial \upsilon_j(n)}{\partial w_{ji}(n)} = y_i(n)$$

(9)

$$\frac{\partial \xi(n)}{\partial w_{ji}(n)} = \frac{\partial \xi(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial \upsilon_j(n)} \frac{\partial \upsilon_j(n)}{\partial w_{ji}(n)}$$

(10)

Colocando todas essas derivadas (6), (7), (8), (9) na expressão da regra da cadeia (figura 25) chega-se ao gradiente do erro sendo possível realizar a atualização dos pesos.

$$\Delta w_{ji}(n) = -\eta \frac{\partial \xi(n)}{\partial w_{ji}(n)}$$

(REGRA DELTA)

Ou

$$\Delta w_{ji}(n) \equiv w_{ji}(n-1) - w_{ji}(n)$$

O sinal negativo na frente de η (constante de aprendizagem) deve-se ao fato de se estar caminhando na direção da descida mais ímgreme. A constante de aprendizado será adiante mais detalhada.

Todas as derivadas apresentadas para resolver a regra da cadeia relativa a figura 25

mostram como modificar os pesos dos nós que estão ligando a camada oculta à saída. Contudo, também é necessário modificar os pesos dos nós da camada oculta.

Para isso, o procedimento utilizado é similar ao descrito anteriormente.

Inicialmente troca-se (j) por (k).

k: refere-se a camada de saída

j: refere-se a camada oculta (anteriormente (j) referia-se ao neurônio da camada de saída.

Redefine-se δ como:

$$\delta_j(n) = -\frac{\partial \xi(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial \upsilon_j(n)}$$
$$= -\frac{\partial \xi(n)}{\partial y_j(n)} \varphi'_j\left(\upsilon_j(n)\right)$$

(11)

$$\xi(n) = \frac{1}{2} \sum_{k \in C} e_k^2(n)$$

(12)

$$e_k(n) = d_k(n) - y_k(n)$$
$$= d_k(n) - \varphi_k(\upsilon_k(n))$$

(13)

$$\frac{\partial e_k(n)}{\partial \upsilon_k(n)} = -\varphi'_k(\upsilon_k(n))$$

(14)

$$\upsilon_k(n) = \sum_{j=0}^{q} w_{kj}(n) y_j(n)$$

(15)

Onde (q) é o nº de ligações que chegam ao neurônio k.

Diferenciando a expressão (11) chega-se em:

$$\frac{\partial \upsilon_k(n)}{\partial y_j(n)} = w_{kj}(n)$$

(16)

$$\frac{\partial \xi(n)}{\partial y_j(n)} = -\sum_k e_k(n)\varphi'_k(\upsilon_k(n))w_{kj}(n)$$
$$= -\sum_k \delta_k(n) w_{kj}(n)$$

(17)

$$\delta_j(n) = \varphi'_j(\upsilon_j(n))\sum_k \delta_k(n) w_{kj}(n)$$

↓ ↓

delta camada oculta (18)

delta camada de saída

O sentido delta está no fato de que é possível modificar os pesos se os deltas são conhecidos. Como pode ser visto pela fórmula (18), o delta da camada oculta está sendo calculado em função do delta da camada de saída. Este é o sentido da retropropagação do erro. Com o delta da camada de saída, calcula-se o delta da camada anterior e com este o da camada anterior, e assim por diante.

Como a modificação dos pesos depende da derivada da função de ativação, uma condição para que o algoritmo de retropopagação seja coerente é que a função seja diferenciável e não decrescente (pois a derivada não pode ficar trocando de sinal, caso contrário, nunca converge).

Enfim, o algoritmo de retropropagação pode ser descrito em 5 etapas:

a) apresentação do padrão;
b) propagar o sinal para a frente e calcular a saída resultante para todos os neurônios de saída;
c) calcular os δ's para todos os neurônios da camada de saída;
d) retropropagar o sinal de erro de saída, calculando os δ's das unidades ocultas;
e) atualizar os pesos, apresentar novo padrão e retornar ao passo (a).

Retornando a constante de aprendizagem (η), que aparece na regra delta de modificações dos pesos, o algoritmo de retropropagação fornece uma "aproximação" para a trajetória no espaço dos pesos calculada pelo método de descida mais íngreme. Assim:

- Quanto menor for "η" de uma interação para a outra, menor será a variação nos pesos. Então a taxa de aprendizado é mais lenta.

- Quando "η" for grande, para acelerar a taxa de aprendizado, as modificações nos pesos podem tornar a rede instável.

Uma forma de aumentar a taxa de aprendizado evitando o problema da instabilidade é modificar a regra delta incluindo o Termo Momento. A inclusão do termo momento no algoritmo de retropropagação tem um efeito estabilizador por penalizar os movimentos em sentidos opostos da derivada parcial:

$$\frac{\partial \xi(n)}{\partial w_{ji}(n)} \qquad (19)$$

Se as derivadas (19) apresentam o mesmo sinal em interações consecutivas, os pesos são ajustados em uma valor maior do que no caso em que as derivadas oscilam.

Em síntese:

- Termo Momento ($\alpha \geq 0$**)**: é uma variação pequena, mas com efeito benéficos:

$$\Delta wji(n) = \alpha\ wji(n-1) + \eta\ \delta j\ yi\ (n)$$

Para inicializar o treinamento, os pesos devem ser distribuídos uniformemente com média zero e limites pequenos. Como a modificação nos pesos é proporcional a derivada da função de ativação, quanto mais próximo do centro da função se estiver, maior a modificação introduzida. Nas duas regiões de saturação, as derivadas são próximas de zero e, por conseqüência, haverá um ajuste muito pequeno nos pesos, mesmo se o erro for significativo. Em outras palavras, sempre deve-se iniciar os pesos em algum ponto entre valores extremos.

Quando encerrar o ajuste de pesos (critérios de parada), pode-se dizer que não existem critérios bem definidos para demonstrar que o algoritmo convergiu ou que indiquem que sua operação pode ser encerrada. Entretanto, em geral, encerra-se o ajuste de pesos quando ocorre alguma das situações abaixo mencionadas:

- erro atingiu uma tolerância pré-determinada.
- erro caiu a uma taxa suficientemente pequena de acordo com uma parâmetro pré-determinado.
- a cada época testa-se a generalização, parando se o desempenho for considerado adequado.

Apesar do algoritmo de retropropagação ser uma técnica muito difundida, existem problemas na sua implementação, a saber alguns deles:

1. Mínimos locais: como a superfície do erro geralmente é cheia de vales e desníveis e o algoritmo emprega um tipo de gradiente descendente, há possibilidade do algoritmo ficar preso em um mínimo local.
2. Paralesia da rede: durante o treinamento se os pesos são ajustados para valores muito grandes, podem levar a derivada da função de ativação para zero. Isto impede a rede de aprender o conjunto de treinamento.

A constante de aprendizagem e o termo momento são algumas heurísticas para tentar melhorar o desempenho da rede, entre outras tais como: normalização dos dados de entrada, utilização da função de ativação hiperbólica (o algoritmo aprende mais rápido), método padrão a padrão é mais rápido que por lote (batelada), apresentação aleatória dos exemplos.

Destacando a questão do mínimo local, sabe-se que a taxa de aprendizado não deve ser nem muito grande (pois gera oscilações durante o treinamento prejudicando a convergência) e nem muito pequena (para não tornar o treinamento lento). Logo, se η é pequeno e o algoritmo está preso em um mínimo local, qualquer mudança nos pesos faz o erro aumentar. Se não for possível calcular um Δw que faça a rede neural sair do mínimo local, a rede não conseguirá aprender com a precisãoespecificada (considerando um erro mínimo).

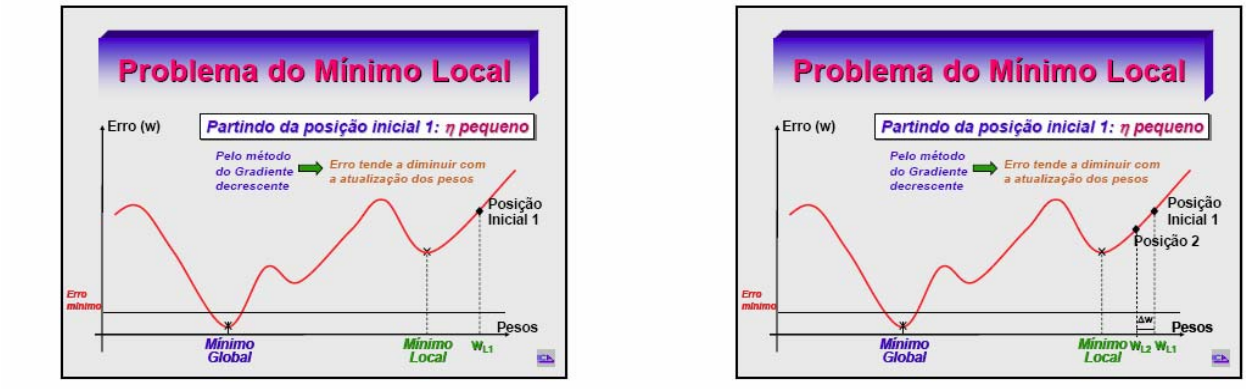

Figura 26: problema do mínimo local (para η pequeno)

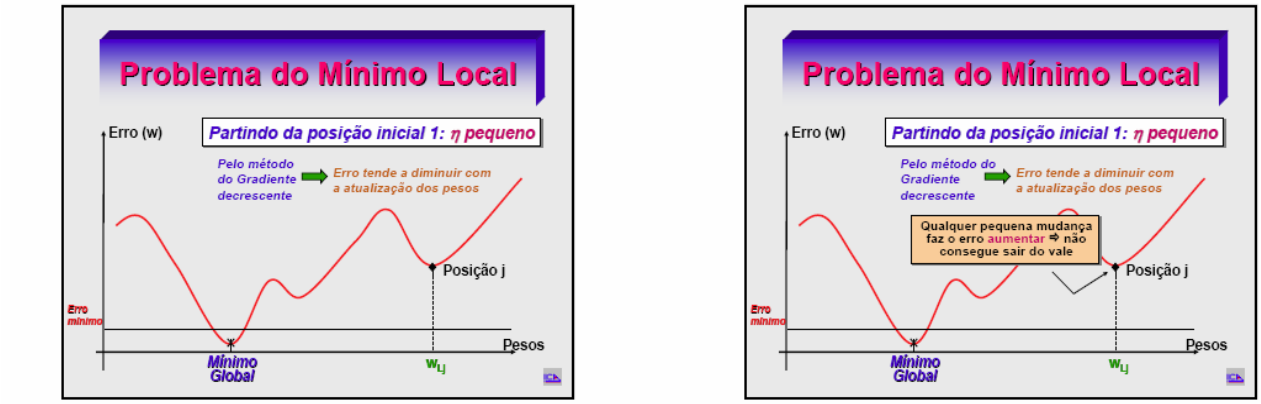

Figura 26(continuação): problema do mínimo local (para η pequeno)

Entretanto, se η é grande, as mudanças não são suficientemente pequenas para levar a uma configuração de erro mínimo global. Os valores de Δw são grandes causando oscilações. Nesta situação, a rede neural também não consegue aprender com a precisão especificada.

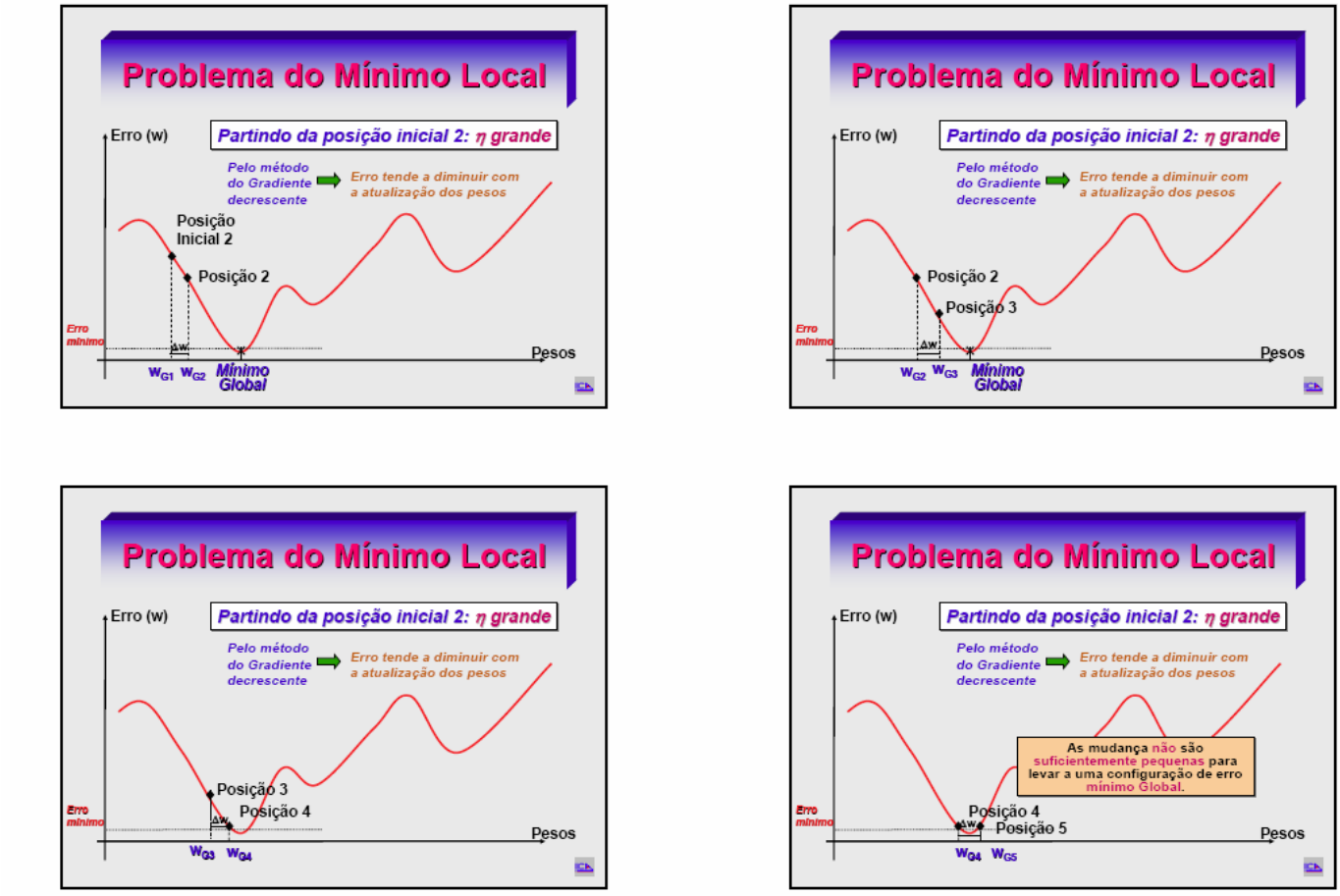

Figura 27: problema do mínimo local (para η grande)

Para um η adequado, a atualização dos pesos consegue chegar a um valor de erro que seja mínimo global e a rede neural aprende com a precisão desejada.

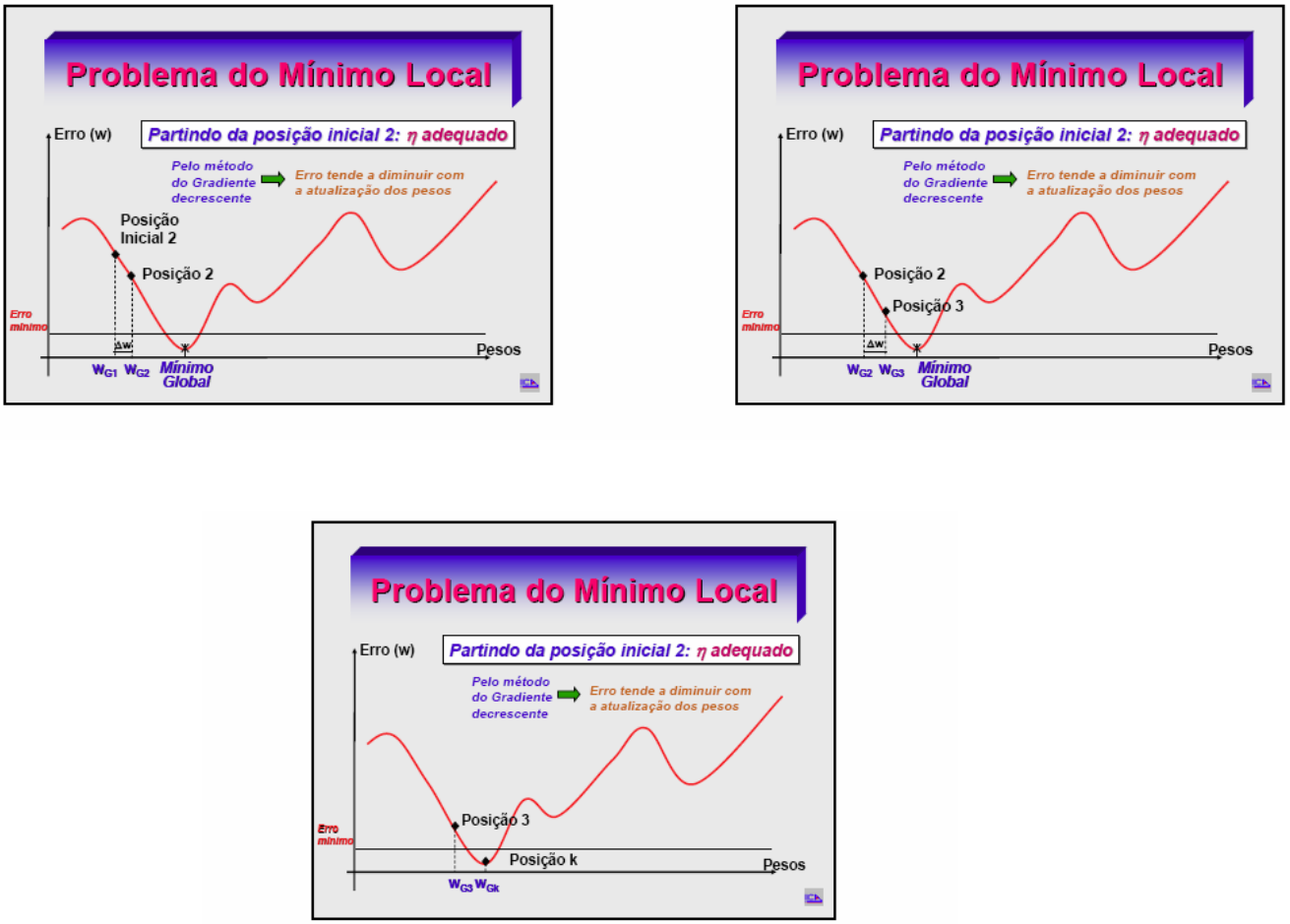

Figura 28: : problema do mínimo local (para η adequado)

A inclusão do termo momento permite a rede convergir mais rapidamente, pois penaliza a troca de sinais da derivada da função de ativação. Assim, a atualização dos pesos acontece mais rapidamente.

Como pode ser visto na figura 29, que reporta como se dá a convergência para diferentes valores atribuídos ao termo momento, percebe-se que a rede converge mais rápido (menos época) quando se aumenta o valor de α. Contudo, nota-se também que se α for um valor muito alto, a rede começa a oscilar e o erro começa a aumentar.

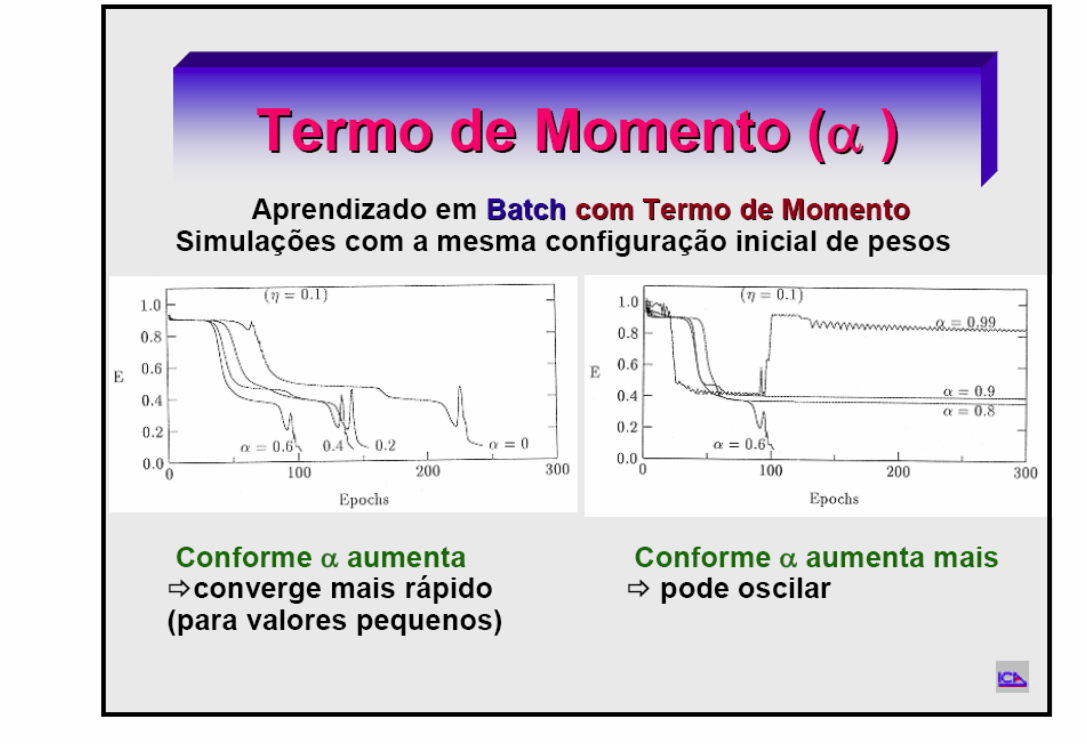

Figura 29: efeitos da inclusão do termo momento