

Capítulo

1

Redes Neurais Artificiais

Júlio Cesar Nievola

Abstract

This chapter presents the notion of artificial neural networks, the computational technique built on the idea of its biological counterpart. Its various models allow the solution of many problems in pattern recognition and simulation fields. Here the main kinds of artificial neural networks and its behavior are presented. It is also described how to connect artificial neural networks and symbolic systems.

Resumo

Este capítulo apresenta a noção de redes neurais artificiais, as quais são técnicas computacionais desenvolvidas a partir da noção biológica equivalente. Os vários modelos existentes permitem a resolução de muitos problemas nas áreas de reconhecimento de padrões e simulação. Neste texto, são analisados os principais tipos de redes neurais artificiais e seu comportamento. Também é descrita a interligação entre redes neurais artificiais e os sistemas simbólicos.

1. Introdução

1.1. Histórico

As Redes Neurais Artificiais (RNA) constituem-se em um paradigma computacional baseado no modelo biológico do cérebro humano. As suas primeiras utilizações surgiram já na década de 1950, quando da reunião no Dartmouth College. Nesta reunião surgiu o termo Inteligência Artificial para designar um conjunto de técnicas que permitiriam o desenvolvimento de sistemas suficientemente genéricos para resolver qualquer problema, desde que devidamente representado. Desta forma, as RNAs, também chamadas de abordagem conexionista, surgiram como uma alternativa à abordagem simbólica, baseada na lógica [Kratzer, 1991]. Os trabalhos pioneiros utilizavam uma rede chamada de Perceptron, desenvolvida por Rosenblatt [Rosenblatt, 1958].

Uma das motivações para o desenvolvimento da abordagem conexionista foi o fato de que as abordagens clássicas, embora tenham capacidade de resolver muitos problemas complexos, não conseguiram ter um desempenho adequado para tarefas que são rotineiras, e em certos casos até intuitivas, para o ser humano [Brunak, 1993]. Foi então questionado que a forma humana de trabalho nestes casos estava em um nível anterior ao raciocínio lógico, a nível sub-simbólico. O estudo de tais elementos revelou um conjunto de

características que em certos pontos opõe a abordagem simbólica (baseada na lógica) da abordagem sub-simbólica (baseada nos elementos biológicos). A tabela 01 apresenta uma comparação entre estas duas abordagens.

Tabela 1 – Comparação de Características.

Característica	Redes Neurais Artificiais	Sistemas Convencionais
Velocidade de Processamento	ms (disparo de um neurônio)	4,2ns (clock do Cray3)
Processamento	Paralelo	Serial
Quantidade	10^{11} - 10^{14} neurônios	Um ou poucos processadores
Interligações	10^3 - 10^4 conexões/neurônio	Poucas
Armazenamento do conhecimento	Adaptativo	Estritamente relocável
Tolerância a falhas	Boa	Mínima ou inexistente
Controle do processamento	Anárquico, distribuído	Autocrático, centralizado

Em 1969, Minsky e Papert [Minsky e Papert, 1969] publicaram um livro no qual apresentavam o modelo formal do perceptron e demonstraram que o mesmo apresentava uma grande limitação, a saber, ele só era capaz de resolver corretamente problemas com dados linearmente separáveis. Como tal característica não está presente na maioria dos sistemas reais, a abordagem conexionista sofreu um grande impacto e durante a década de 1970 a área de IA voltou-se bastante para a abordagem lógica. Entretanto, alguns grupos continuaram buscando alternativas que suplantassem a limitação apresentada e para obter verbas para suas pesquisas passaram a utilizaram uma denominação que não redes neurais. Devido a este fato, encontram-se muitas vezes outros termos que são sinônimos de redes neurais artificiais, tais como: Sistemas Neurais Artificiais, Sistemas Adaptativos, Neuro-computadores, Sistemas Maciçamente Paralelos, entre outros [Hassoun, 1995].

Esta situação foi alterada de maneira drástica com a apresentação de um artigo que demonstrava uma maneira de expandir o algoritmo do perceptron para uma rede com várias camadas [Hoffmann, 1993]. O modelo matemático desenvolvido provava que tal rede, denominada de Perceptron Multicamadas (ou MLP, “Multi-Layer Perceptron”) poderia ser treinada para atingir qualquer valor desejado de precisão, para qualquer tipo de problema [Rumelhart et al., 1986]. Isto despertou novamente um grande interesse na comunidade científica fez da década de 1980 uma década de pesquisas relacionadas ao tema.

1.2. Etapas de Projeto

As Redes Neurais fazem parte de uma área denominada Aprendizagem de Máquina, a qual pode ser definida como: *“A computer program is said to **learn** from experience E with respect to some class of tasks T and performance measure P, if its performance at*

tasks in T, as measured by P, improves with experience E." [Mitchell, 1997]. Existem modelos específicos de RNA capazes de realizar qualquer um dos três tipos de aprendizagem, a saber:

a)Aprendizagem Supervisionada: Consiste no mapeamento de um conjunto de variáveis, ditas variáveis de entrada ou independentes, em um segundo conjunto de variáveis, chamadas de variáveis de saída ou dependentes. A rede atua comparando a saída com o valor desejado e realizando correções em seu modelo até atingir um erro aceitável. No caso da variável de saída ser contínua tem-se um problema de regressão; caso a variável seja discreta tem-se um problema de classificação;

b)Aprendizagem Não-Supervisionada: Nesta situação, tem-se um conjunto de dados de entrada e deseja-se agrupar estes dados em subconjuntos de forma tal que dados colocados em um mesmo conjunto sejam semelhantes entre si (ou tenham um conjunto de propriedades em comum) e dados colocados em grupos diferentes tenham características distintas;

c)Aprendizagem com Reforço: Tal como no caso da aprendizagem supervisionada, existe um conjunto de variáveis de entrada e de saída. Entretanto, não se sabe qual o valor de saída correto para cada exemplo de entrada; somente pode-se identificar se uma saída está correta ou não.

A utilização das RNAs segue uma metodologia que pode ser decomposta em sucessivas etapas [Kinnebrock, 1992]. Em alguns momentos é necessário retornar a uma etapa anterior, caso a finalidade daquela etapa não tenha sido alcançada ou então para melhorar os resultados. De uma maneira geral as atividades que compõe o desenvolvimento de uma RNA são:

a)Definição do Problema: Nesta etapa deve-se verificar se as características do problema sendo considerado são condizentes com o uso de uma RNA. Algumas características de uma boa aplicação para RNAs são: regras de resolução do problema desconhecidas ou difíceis de determinar e disponibilidade de um grande conjunto de exemplos. Alguns exemplos de área de aplicação típicos são: reconhecimento de formas, tratamento de sinal, visão, fala, previsão e modelagem, auxílio à decisão, robótica;

b)Escolha e adequação dos dados: Para que se possa utilizar uma RNA é fundamental que se disponha de dados em quantidade e qualidade suficientes. Caso a quantidade de dados seja pequena, a rede não conseguirá criar um modelo suficientemente representativo para se ter um bom desempenho quando aplicado em situações reais após o seu desenvolvimento. Além disto, os dados devem englobar todos os aspectos do problema em questão, a fim de que o modelo criado seja genérico. Em geral, tais dados precisam ser convertidos para um formato padrão para utilização pelas RNAs;

c)Treinamento da Rede: Implica em aplicar o algoritmo de aprendizagem escolhido a cada um dos exemplos do conjunto de treinamento iterativamente, até que a rede atinja o comportamento desejado. Caso a rede não consiga atingir o desempenho mínimo, deve-se avaliar se os dados são adequados e realizar uma readequação dos mesmos, expandindo-os ou reduzindo-os;

d)Teste da Rede: Após a rede ter sido treinada, deve-se testar a mesma para verificar se o modelo criado pela mesma tem um índice de acerto aceitável em dados "novos", ou seja, que ainda não foram aplicados à rede. Em caso positivo, passa-se ao uso em campo da rede neural. Se o desempenho não for adequado, deve-se retornar a uma etapa anterior e refazer

aquela parte do processo, como por exemplo, alterar o algoritmo de aprendizagem, a topologia da rede, ou eventualmente o próprio conjunto de dados;

e)Uso em Campo: Se as etapas anteriores apresentaram sucesso, o modelo criado pode ser implementado (independente do algoritmo de aprendizagem) para uso contínuo. Ainda assim, durante o uso podem surgir problemas, o que implica em retornar a uma etapa anterior e reiniciar o processo.

2. Terminologia e Formalização

2.1. Neurônio Biológico

As redes neurais biológicas são compostas por uma grande quantidade de elementos básicos, denominados neurônios (fig. 01). A estrutura básica dos neurônios é composta por:

a)*Dendritos*: Correspondem a filamentos que recebem as informações de outros elementos, sejam estes outros neurônios ou sensores. Não existe uma conexão física entre os dendritos e os elementos antecedentes. A ligação entre eles é eletro-química, denominada *sinapse*;

b)*Corpo Celular*: Corresponde à maior área do neurônio e, a partir dos sinais recebidos pelos dendritos, sofre uma série de transformações internas, resultando em um sinal que é propagado ao elemento seguinte;

c)*Axônio*: Recebe o resultado dos processos químicos internos ao corpo celular e propaga a outros elementos, através das sinapses que se formam entre ele e os elementos subseqüentes.

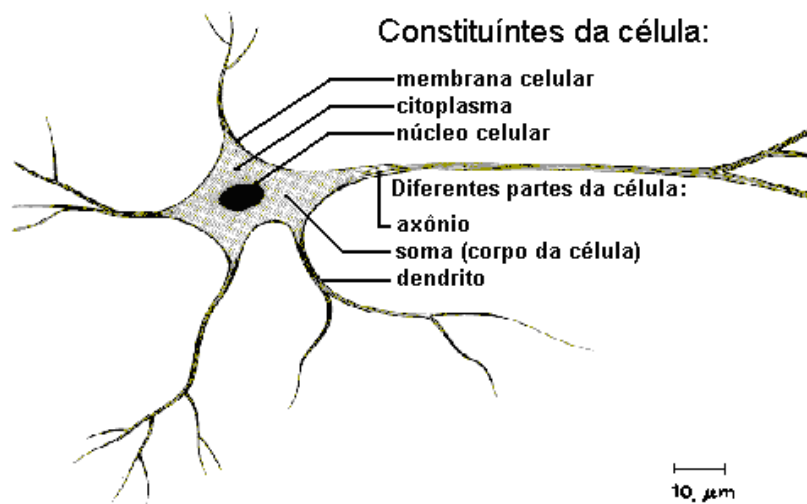


Figura 1 – Neurônio Biológico e seus componentes.

2.2. Modelo Matemático do Neurônio Biológico

McCulloch e Pitts estudaram o neurônio anteriormente apresentado e propuseram um modelo matemático de seu funcionamento [McCulloch e Pitts, 1943]. Neste modelo, considera-se que o neurônio recebe sinais provenientes de várias entradas. A combinação destes sinais é propagada ao corpo celular. Ao longo do tempo cada uma das entradas do neurônio passa a ter uma maior ou menor influência sobre o processamento que ocorre no corpo celular. Mais do que isso, elas podem excitar ou inibir o processamento. Este comportamento é representado através de uma soma ponderada dos sinais de entrada, ou

seja, cada sinal de entrada tem o seu valor multiplicado por um peso (geralmente entre +1 e -1), indicando a influência daquela entrada no processo que ocorre internamente ao corpo celular. Desta forma, se chamarmos de x_1, x_2, \dots, x_n o valor do sinal em cada entrada do neurônio, e indicarmos por w_1, w_2, \dots, w_n a influência que cada uma tem no processamento que ocorre no corpo celular, temos que a entrada total no corpo celular ent_{total} é dada por:

$$ent_{total} = \sum_{i=1}^n x_i \cdot w_i \quad (1)$$

Em muitos casos, considera-se que o valor da entrada total sobrepõe-se a um valor fixo pré-existente, denominado de *bias*, o qual pode assumir qualquer valor também entre +1 e -1. Neste caso considera-se que o bias representa uma entrada extra, sempre com valor unitário, sofrendo a ponderação através de um peso correspondente, w_0 . Neste caso a entrada total será dada por:

$$ent_{total} = \sum_{i=1}^n x_i \cdot w_i + w_0 \quad (2)$$

ou seja,

$$ent_{total} = \sum_{i=0}^n x_i \cdot w_i \quad (3)$$

O valor total da entrada sofre uma transformação interna no corpo celular, produzindo o valor que é propagado para os outros elementos através do axônio. Esta transformação do valor de entrada total no corpo celular, ent_{total} , na saída do neurônio, s , é representada pela aplicação de uma função $f(\bullet)$, conforme a equação:

$$s = f(ent_{total}) \quad (4)$$

A função $f(\bullet)$ em princípio pode ser qualquer função matemática, embora na prática exista um conjunto de funções que são as mais utilizadas. A figura 2 apresenta uma lista de algumas delas.



Figura 2 – Exemplos de função de ativação de neurônios.

2.3. Formalização das Redes Neurais Artificiais

A partir do neurônio formalizado, pode-se realizar conexões entre diversos neurônios, constituindo-se as Redes Neurais Artificiais (RNAs). Em função do tipo de conexão realizada, passa-se a ter uma topologia específica, a qual, juntamente com o algoritmo de aprendizagem que realiza a atualização dos pesos entre os neurônios, determina o tipo de rede.

A partir da especificação dos elementos anteriores tem-se cada uma das RNAs, as quais podem ser agrupadas segundo suas semelhanças. Uma possibilidade de agrupamento está indicada na figura 3.

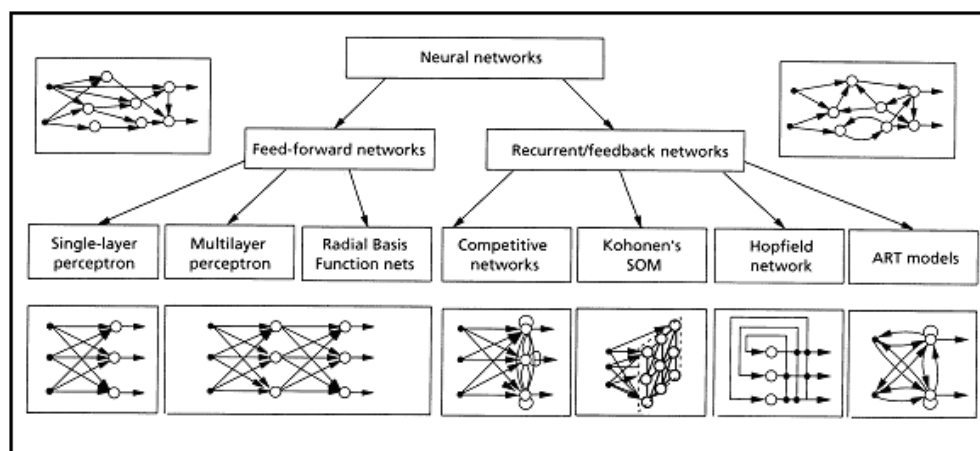


Figura 3 – Agrupamento das redes neurais em função de suas características [Jain et al., 1996].

Uma descrição hierárquica de redes neurais foi proposta por Fiesler [Fiesler, 94]. Segundo a mesma, a especificação da rede neural pode ser dividida em uma topologia, um conjunto de limitações, um estado inicial e um conjunto de funções de transição. A topologia da rede neural é definida em função dos elementos indicados. As limitações definem a faixa de valores para os pesos, as transições locais (ou deslocamentos) e os valores de ativação (ou atividades), os quais podem ser vistos como os valores “de saída” dos neurônios. Exemplos de limitações são: o conjunto dos valores reais que um peso pode assumir e um conjunto limitado de inteiros que pesos discretos podem assumir.

A *dinâmica* de uma rede neural pode ser completamente descrita pelo estado inicial da rede, consistindo de um conjunto de valores iniciais para os pesos e transições locais, mais os padrões de entradas (e objetivo), os quais tipicamente determinam as ativações, mais as funções de transição que podem ser usadas para determinar os sucessivos estados da rede neural, respeitando-se as limitações dadas [Mehrotra et al., 1997].

Para que uma RNA possa ser específica de maneira única, deve-se indicar os seguintes elementos:

- a) Topologia da rede: Conjunto de interligações dos neurônios;
- b) Algoritmo de aprendizagem: Define como os pesos da rede neural, e eventualmente suas conexões, são alterados durante o treinamento;
- c) Codificação dos dados de entrada: Em muitos casos os dados de entrada devem ser convertidos para uma representação diferente daquela original;
- d) Decodificação dos valores de saída: As saídas correspondem a valores numéricos contínuos. Se esta não for a representação desejada, ela deve ser convertida para o formato adequado.

Outra forma de se agrupar as redes neurais segundo suas características em termos do paradigma (aprendizagem supervisionada, não supervisionada ou híbrida), da regra de aprendizagem empregada, da topologia, do algoritmo de aprendizagem utilizado e da tarefa apresentada está indicada na tabela 2.

Tabela 2 – Características de RNAs em função do tipo de aprendizagem [Jain et al., 1996].

Paradigma	Regra de Aprendizagem	Arquitetura	Algoritmo de Aprendizagem	Tarefa
Supervisionada	Correção do erro	Perceptron com uma camada	Algoritmos de aprendizagem do perceptron	Classificação de padrões
		Perceptron com várias camadas	Retro-propagação; Adaline e Madaline	Aproximação de funções, predição e controle
	Boltzmann	Recorrente	Algoritmo de aprendizagem de Boltzmann	Classificação de padrões
	Hebb	Multicamadas em avanço	Análise discriminante linear	Análise de dados, classificação de padrões
	Competitiva	Competitiva	Quantização do vetor de aprendizagem	Categorização em classes internas, compressão de dados
		Rede ART	ARTMAP	Classificação de padrões, categorização em classes internas
Não supervisionado	Correção do erro	Multicamadas em avanço	Projeção de Sammon	Análise de dados
	Hebb	Em avanço ou competitiva	Análise da componente principal	Análise de dados, compressão de dados
		Rede Hopfield	Aprendizagem de memória associativa	Memória associativa
	Competitiva	Competitiva	Quantização de vetores	Categorização, compressão de dados
		SOM (Kohonen)	SOM (Kohonen)	Categorização, análise de dados
		Rede ART	ART1, ART2	Categorização
Híbrido	Correção de erros e competitiva	Rede RBF	Algoritmo de aprendizagem RBF	Classificação de padrões, aproximação de funções, predição, controle

3. Perceptron, MLP e Redes Construtivas

3.1. Perceptron

O Perceptron foi a primeira RNA utilizada. Ele é composto basicamente por duas camadas. A camada de entrada executa a função de simplesmente repassar à camada seguinte as variáveis de interesse, sem executar nenhuma operação sobre as mesmas. Em função disto, a maioria dos autores não a considera uma camada de neurônios no sentido formal, ou seja, um perceptron é composto por somente uma camada de neurônios formais, conforme a fig 4. A camada de saída possui um neurônio, correspondente à saída desejada. Na representação mais clássica do perceptron, as entradas e saídas são binárias e a função de ativação do neurônio de saída é do tipo limiar (“threshold”).

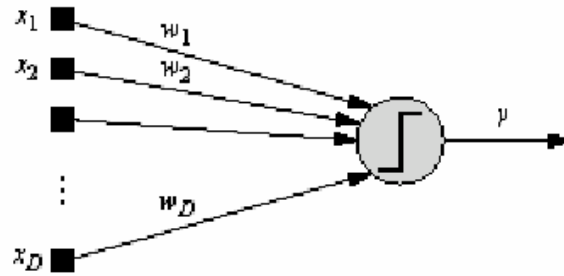


Figura 4 – Diagrama de um Perceptron com d entradas e uma saída.

A entrada total do perceptron é dada por:

$$e_T = \sum_{i=1}^D x_i \cdot w_i \quad (5)$$

e a saída y é dada em função da entrada total, ou seja:

$$y = \begin{cases} 1, & \text{se } e_T \geq 0, \\ 0, & \text{em caso contrário} \end{cases} \quad (6)$$

Na maioria dos casos utiliza-se o termo independente (“bias”), fazendo com que a equação 5 seja generalizada para a equação 7:

$$e_T = \sum_{i=0}^D x_i \cdot w_i \quad (7)$$

Pode-se provar que o perceptron somente encontra solução (isto é, consegue classificar todos os exemplos corretamente) para conjuntos de treinamento linearmente separáveis. No algoritmo de aprendizagem do Perceptron busca-se um vetor W que tenha projeção positiva (produto interno) com todos os exemplos positivos e projeção negativa com todos os exemplos negativos. A aprendizagem do perceptron sempre tem sucesso em tempo finito para um conjunto de treinamento finito e separável de exemplos de treinamento.

Na figura 5 encontra-se o Algoritmo do Perceptron [Gallant, 1993]. Para um conjunto finito de exemplos de treinamento E , com componentes inteiros (ou racionais), o algoritmo de aprendizagem do perceptron, em tempo finito ou produzirá um vetor peso que

satisfaz todos os exemplos de treinamento (se e somente se E é separável) ou então abandonará e reutilizará um vetor peso (se e somente se E é não-separável).

1. Fazer W ser o vetor nulo;
2. Selecionar um exemplo de treinamento E_k (com a correspondente classificação C_k). Isto pode ser feito de maneira cíclica (em ordem) através dos exemplos de treinamento ou pegando um exemplo aleatoriamente;
3. Se W classifica E_k corretamente, isto é, se
 $\{W \cdot E_k \geq 0 \text{ e } C_k = +1\}$ ou se $\{W \cdot E_k < 0 \text{ e } C_k = -1\}$
 Então: Não fazer nada.
 Senão: *Passo de alteração*: Modificar W somando ou subtraindo E_k de acordo com a saída correta ser +1 ou -1: $W' = W + C_k E_k$.
4. Ir ao passo 1.

Figura 5 – Algoritmo do Perceptron Simples [Gallant, 1993].

Se um conjunto de exemplos de treinamento E é não-separável, então por definição não existe um vetor de pesos W que classifique corretamente todos os exemplos de treinamento em E utilizando o algoritmo de aprendizagem do perceptron. A alternativa mais natural é encontrar um vetor de pesos W^* que classifique tantos exemplos de treinamento quanto possível de E . Tal conjunto de pesos é chamado de ótimo.

O algoritmo de aprendizagem do perceptron tem comportamento pobre: se o algoritmo é terminado, mesmo após um grande número de iterações, não se tem garantia da qualidade dos pesos que são produzidos, já que o mesmo usa somente reforço negativo, ignorando totalmente os exemplos que são corretamente classificados.

Para contornar este problema utiliza-se o chamado Algoritmo do Bolso. O algoritmo do bolso leva em conta as classificações corretas mantendo um conjunto de pesos em separado W_{bolso} mantidas “no bolso”, juntamente com o número de iterações consecutivas para a qual W_{bolso} classificou corretamente os exemplos de treinamento escolhidos. Na figura 6 encontra-se uma descrição do Algoritmo do Bolso.

1. Fazer W ser o vetor nulo;
2. Selecionar um exemplo de treinamento E_k (com a correspondente classificação C_k);
3. Se W classifica E_k corretamente, isto é, se
 $\{W \cdot E_k \geq 0 \text{ e } C_k = +1\}$ ou se $\{W \cdot E_k < 0 \text{ e } C_k = -1\}$
 Então:
 3a. Se a sequência atual de classificações corretas com W é maior que a sequência de classificações corretas para o vetor peso W_{bolso} :
 3aa. Substitua os pesos no bolso W_{bolso} por W e registre o número de vezes que ela é correta.
 Senão: *Passo de alteração*: Modificar W somando ou subtraindo E_k de acordo com a saída correta ser +1 ou -1: $W' = W + C_k E_k$.
4. Ir ao passo 1.

Figura 6 – Algoritmo do Perceptron com Bolso [Gallant, 1993].

Para problemas de tamanho médio e grande o algoritmo do bolso normalmente não produz pesos ótimos após um razoável número de iterações, mas produz bons pesos. Na execução do algoritmo do bolso não há nada que previna um mau conjunto de pesos de ter uma “seqüência feliz” de respostas corretas e acabe substituindo um bom conjunto de pesos no bolso. Pior ainda, se a “seqüência feliz” for longa então o conjunto de maus pesos pode permanecer no bolso por um grande número de iterações.

Para evitar isto, utiliza-se o Algoritmo do Bolso com Catraca, no qual, para conjuntos finitos de exemplos de treinamento tem-se melhores resultados, conforme a figura 7.

1. Fazer W ser o vetor nulo;
2. Selecionar um exemplo de treinamento E_k aleatoriamente (com a correspondente classificação C_k);
3. Se W classifica E_k corretamente, isto é, se

$$\{W \cdot E_k \geq 0 \text{ e } C_k = +1\} \text{ ou se } \{W \cdot E_k < 0 \text{ e } C_k = -1\}$$
 Então:
 - 3a. Se a seqüência atual de classificações corretas com W é maior que a seqüência de classificações corretas para o vetor peso W_{bolso} :
 - 3aa. Se W classifica corretamente mais exemplos de treinamento que W_{bolso} :
 - 3aaa. Substitua os pesos no bolso W_{bolso} por W e registre o número de vezes que ela é correta.
 Senão: *Passo de alteração:* Modificar W somando ou subtraindo E_k de acordo com a saída correta ser +1 ou -1: $W' = W + C_k E_k$.
4. Ir ao passo 1.

Figura 7 – Algoritmo Perceptron com Bolso e Catraca [Gallant, 1993].

O algoritmo do Perceptron, em qualquer de suas variantes, pode eventualmente não encontrar a solução mesmo em caso dela existir. Isto pode acontecer caso o número de iterações seja insuficiente. Desta forma, quando ele não encontra a solução, não se sabe se o número de iterações foi pequeno ou se o conjunto de treinamento é não linearmente separável. Um algoritmo que determina um limite superior para o número de iterações, indicando se os dados são ou não linearmente separáveis é de bastante utilidade.

Os algoritmos de programação linear têm a característica desejável de crescerem polinomialmente com a precisão das entradas, podendo ser usados para determinar os pesos para um conjunto de exemplos de treinamento linearmente separáveis, garantindo um limite polinomial no tempo de aprendizagem.

O algoritmo de Khachiyan tem melhor tempo de aprendizagem médio e melhor tempo de aprendizagem no pior caso para problemas separáveis que o algoritmo do perceptron [Aspvall, 1980]. Para o caso em que as ativações estão em $\{+1, -1\}$ verificou-se também uma melhora no limite superior do número máximo de iterações. Uma descrição algorítmica do algoritmo de Khachiyan é apresentada na figura 8.

1. Igualar W ao vetor nulo;
 2. Fazer H ser a matriz identidade $(p + 1) \times (p + 1)$;
 3. Realizar no máximo um número de iterações igual a

$$(p + 1)3\ln(p + 1) + (p + 1)2\ln(\pi(p + 1))$$
 - 3a. Se todos os exemplos de treinamento estão corretamente classificados, isto é, todo exemplo de treinamento E_k cuja saída correta é C_k satisfaz $\{W \cdot E_k \geq 0 \text{ e } C_k = +1\}$ ou $\{W \cdot E_k < 0 \text{ e } C_k = -1\}$, então os exemplos de treinamento são separáveis; neste caso fornecer W e sair. Caso contrário, fazer E_k ser um exemplo de treinamento que não é corretamente classificado;
 - 3b. Calcular o vetor $Q = H \cdot E_k$;
 - 3c. Calcular:

$$W = W + \frac{C_k}{(p + 2)\sqrt{Q \cdot E_k}} Q$$

$$H = \frac{(p + 1)^2}{p(p + 2)} \left(H \frac{2}{(p + 2)Q \cdot E_k} Q \otimes Q \right)$$
 4. Se não for encontrada solução, então os exemplos de treinamento não são linearmente separáveis.
- Obs.: $Q \otimes Q$ denota o produto externo de Q , dado por $(Q \otimes Q)_{i,j} = Q_i Q_j$.

Figura 8 – Algoritmo de Khachiyan [Gallant, 1993].

Uma das principais características do perceptron com M saídas é que ele pode dividir o espaço de padrões em M regiões distintas, ou hiperplanos. Os hiperplanos que contribuem para a definição da superfície de decisão devem ser contíguos. As regiões de decisão do perceptron sempre são regiões convexas, pois durante o treinamento exige-se que uma e somente uma das saídas seja positiva. Entretanto, o Perceptron resolve somente problemas lineares, e a aprendizagem on-line sofre com dados ruidosos, embora tenha maior possibilidade de evitar mínimos locais.

3.2. MLP

Uma extensão natural do Perceptron é a rede MLP (“Multi-Layer Perceptron” – Perceptron Multi-Camadas), na qual coloca-se uma camada dita escondida, tendo em vista que os neurônios desta camada não tem ligações nem com a entrada nem com a saída. Em particular, tais neurônios tem como função de ativação uma função do tipo não linear. Conceitualmente, as redes MLP podem ser vistas como uma cascata de perceptrons, conforme figura 9.

Introduzindo-se uma camada extra no perceptron, altera-se qualitativamente o formato das funções discriminantes, sendo que o número máximo de regiões distintas no espaço de entrada é controlado pelo número de neurônios escondidos. Os neurônios de saída criam regiões de decisão que não são mais convexas. Isto faz com que exista mais que um conjunto de pesos para uma combinação particular de regiões de decisão.

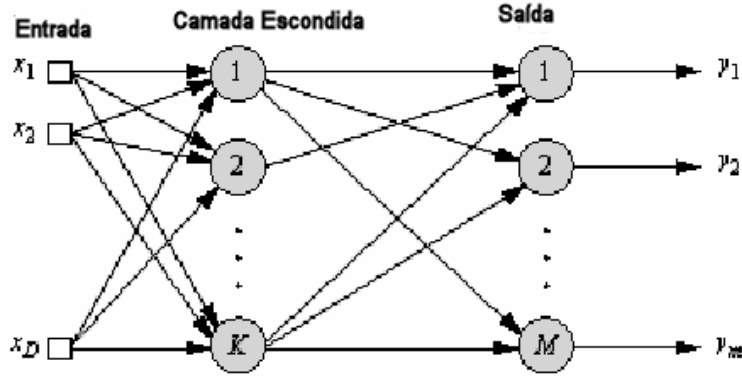


Figura 9 – MLP com uma camada escondida.

A rede MLP é capaz de construir uma saliência no espaço de entrada. Uma rede MLP com uma camada escondida e neurônios com função de ativação sigmoideal (com um número adequado de neurônios na camada escondida) é um mapeador universal, isto é, pode aproximar com uma precisão arbitrária qualquer região de decisão contínua.

O treinamento de uma rede MLP é uma aprendizagem com correção do erro, ou seja, aprendizagem supervisionada [Michalewicz, 2000]. Para adaptar os pesos, calcula-se o erro no i -ésimo neurônio usando-se um erro derivado da camada mais próxima da saída. Este erro é o erro de saída retro-propagado e devidamente escalado. A sensibilidade é automaticamente calculada pela regra da cadeia [Bender, 1996].

3.2.1. Regra Delta Generalizada

A ativação é uma função diferenciável da entrada total:

$$a_i^p = \mathfrak{Z}(i_i^p) \quad (8)$$

na qual tem-se:

$$i_i^p = \sum_j w_{ij} a_j^p + \theta_i \quad (9)$$

Para obter a generalização correta da regra delta deve-se ter:

$$\Delta_p w_{ij} = -\gamma \frac{\partial E^p}{\partial w_{ij}} \quad (10)$$

A medida do erro E_p é definida como o erro total quadrático para o padrão p nas unidades de saída:

$$E^p = \frac{1}{2} \sum_{i=1}^{N_o} (d_i^p - a_i^p)^2 \quad (11)$$

Como uma medida total do erro tem-se:

$$E = \sum_p E^p \quad (12)$$

Usando a regra da cadeia:

$$\frac{\partial E^p}{\partial w_{ij}} = \frac{\partial E^p}{\partial i_i^p} \frac{\partial i_i^p}{\partial w_{ij}} \quad (13)$$

Observa-se que:

$$\frac{\partial i_i^p}{\partial w_{ij}} = a_j^p \quad (14)$$

Para obter uma atualização como a regra delta faz-se:

$$\delta_i^p = -\frac{\partial E^p}{\partial i_i^p} \quad (15)$$

Tem-se uma descida do gradiente se as alterações forem:

$$\Delta_p w_{ij} = \gamma \delta_i^p a_j^p \quad (16)$$

Usando a regra da cadeia:

$$\delta_i^p = -\frac{\partial E^p}{\partial i_i^p} = -\frac{\partial E^p}{\partial a_i^p} \frac{\partial a_i^p}{\partial i_i^p} \quad (17)$$

O segundo elemento é, portanto:

$$\frac{\partial a_i^p}{\partial i_i^p} = \mathfrak{I}'(i_i^p) \quad (18)$$

Existem dois casos para o primeiro fator da regra da cadeia. No primeiro caso considera-se que i é uma unidade de saída e portanto:

$$\frac{\partial E^p}{\partial a_i^p} = -(d_i^p - a_i^p) \quad (19)$$

Neste caso obtém-se:

$$\delta_i^p = (d_i^p - a_i^p) \mathfrak{I}'(i_i^p) \quad (20)$$

Se i não for uma unidade de saída pode-se obter uma medida do erro como uma função da soma ponderada da camada escondida para a camada de saída:

$$E^p = E^p(i_i^1, i_i^2, \dots, i_i^p, \dots) \quad (21)$$

Usando-se a regra da cadeia:

$$\begin{aligned} \frac{\partial E^p}{\partial a_i^p} &= \sum_{h=1}^{N_o} \frac{\partial E^p}{\partial i_h^p} \frac{\partial i_h^p}{\partial a_i^p} = \sum_{h=1}^{N_o} \frac{\partial E^p}{\partial i_h^p} \frac{\partial}{\partial a_i^p} \sum_{k=1}^{N_h} w_{hk} a_k^p \\ &= \sum_{h=1}^{N_o} \frac{\partial E^p}{\partial i_h^p} w_{hi} = -\sum_{h=1}^{N_o} \delta_h^p w_{hi} \end{aligned} \quad (22)$$

Substituindo na primeira regra da cadeia:

$$\delta_i^p = \mathfrak{I}'(i^p) \sum_{h=1}^{N_o} \delta_h^p w_{hi} \quad (23)$$

3.2.2. Funcionamento do Algoritmo de Retro-propagação (“Backpropagation”)

A entrada é apresentada e propagada para frente (etapa avante do algoritmo) através da rede, calculando as ativações para cada unidade de saída. Cada unidade de saída é comparada com o valor desejado, resultando em um valor de erro. Na sequência, calculam-se os erros em cada unidade e são realizadas alterações nos pesos (etapa de retorno do algoritmo).

Descrição do Algoritmo de Retro-propagação:

a) Escolher um pequeno valor positivo para o tamanho do passo ρ , e assinalar pesos iniciais pequenos aleatoriamente selecionados $\{w_{ij}\}$ para todas as células.

b) Repetir até que o algoritmo convirja, isto é, até que alterações nos pesos e no erro médio quadrático ϵ tornem-se suficientemente pequenas:

b1) Escolher o próximo exemplo de treinamento E e sua saída correta C (a qual pode ser um vetor).

b2) *Passo de propagação avante*: Fazer uma passagem da entrada para a saída através da rede para calcular as somas ponderadas, S_i , e as ativações, $u_i = f(S_i)$, para todas as células.

b3) *Passo de retropropagação*: Iniciando com as saídas, fazer uma passagem de cima para baixo através das células de saída e intermediárias calculando:

$$f'(S_i) = u_i(1 - u_i) \quad (24)$$

$$\delta_i = \begin{cases} (C_i - u_i)f'(S_i), & \text{se } u_i \text{ é uma unidade de saída} \\ \left(\sum_{m>i} w_{mi} \delta_m \right) f'(S_i), & \text{para outras unidades} \end{cases} \quad (25)$$

b4) Atualizar os pesos:

$$w_{ij}^* = w_{ij} + \rho \delta_i u_j \quad (26)$$

3.2.3. Características Gerais de uma rede MLP

O uso de uma rede MLP com o algoritmo de retro-propagação com muitos neurônios escondidos faz com que a solução seja atingida rapidamente durante o treinamento, mas o poder de generalização é sacrificado, ou seja, o desempenho em casos não vistos tende a piorar. Durante o treinamento, o sistema posiciona as funções discriminantes que classificam corretamente a maioria dos exemplos, para então lentamente classificar áreas com poucos exemplos. Por outro lado, o erro estabilizará em um alto valor se os graus de liberdade não forem suficientes.

Um MLP com duas ou mais camadas escondidas é um aproximador universal, ou seja, realiza qualquer mapeamento entrada-saída. Isto acontece porque cada neurônio na primeira camada cria uma saliência e a segunda camada combina estas saliências em regiões disjuntas do espaço, conforme figura 10. Entretanto, um problema ainda sem solução é a determinação do número ótimo de camadas escondidas e do número de neurônios em cada camada para um dado problema.


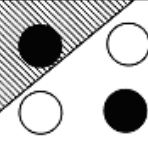


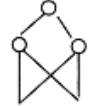
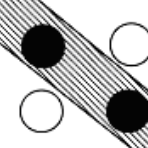


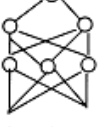
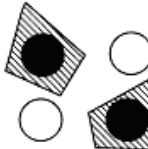


Structure	Description of decision regions	Exclusive-OR problem	Classes with meshed regions	General region shapes
 Single layer	Half plane bounded by hyperplane			
 Two layer	Arbitrary (complexity limited by number of hidden units)			
 Three layer	Arbitrary (complexity limited by number of hidden units)			

Figura 10 – Regiões criadas no espaço por uma rede MLP [Jain et al., 1996].

O uso do algoritmo de retro-propagação (“backpropagation”) para o treinamento de redes MLP é sistemático. Um ponto interessante de ser observado é que os erros de classificação e o erro médio quadrático (“Mean Squared Error” – MSE) podem variar em sentidos opostos. O erro médio quadrático é sensível à diferença entre a resposta desejada e a atual, enquanto que o número de classificações erradas é uma quantidade digital que depende somente da maior saída [Bishop, 1995].

O algoritmo de retro-propagação pode ser aplicado a qualquer topologia do tipo direta em redes MLP, sendo necessário somente seguir a sequência dos três passos indicadas na figura 11. Ele pode ser implementado de maneira eficiente usando-se a idéia da rede dual, conforme indicado na figura 12.

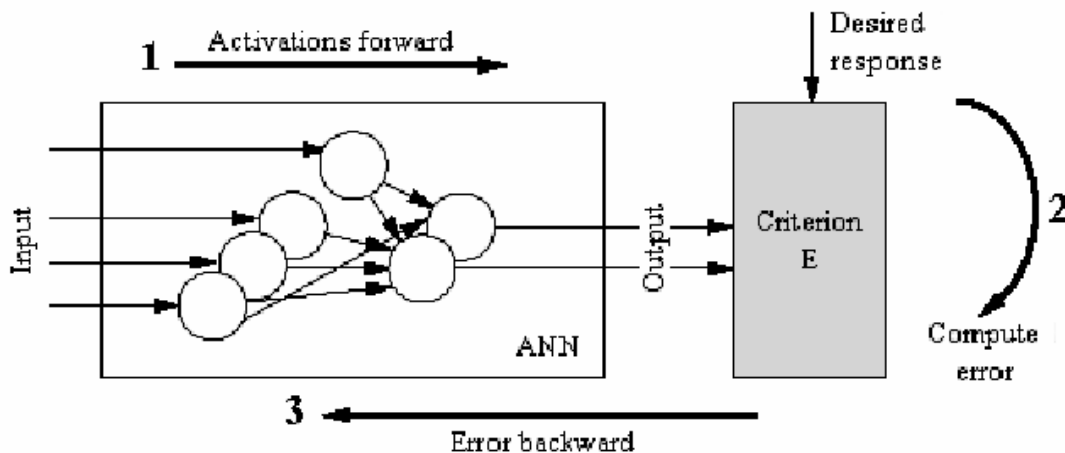


Figura 11 – Sequência de passos para o uso do algoritmo de retro-propagação [Príncipe et al., 2000].

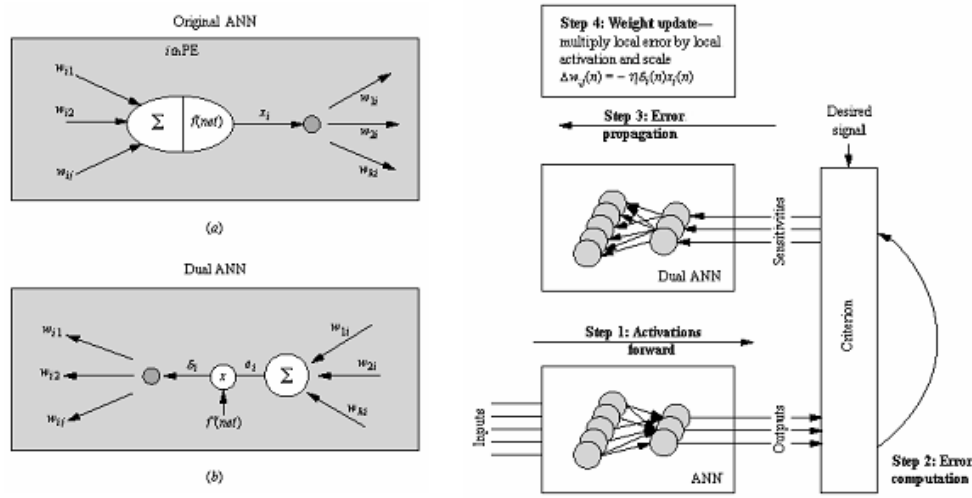


Figura 12 – Implementação Dual do Algoritmo de Retro-Propagação [Príncipe et al., 2000].

Um classificador ótimo deve criar funções de discriminação arbitrárias que separem os agrupamentos de dados de acordo com a probabilidade a posteriori. A rede MLP pode fazer isto desde que: a) hajam neurônios suficientes para fazer o mapeamento; b) hajam dados em quantidade e qualidade; c) a aprendizagem convirja para o mínimo global; e d) as saídas estejam entre 0 e 1 com a soma das mesmas com valor unitário (neurônio softmax).

3.2.4. Controle da Aprendizagem em uma Rede MLP

Há procedimentos sistemáticos na busca pelo conjunto de pesos que produz um resultado desejado. Entretanto, esta busca deve ser controlada heurísticamente. O usuário atua na busca através da definição de quatro elementos: a) da seleção dos pesos iniciais; b) das taxas de aprendizagem; c) do algoritmo de busca; e d) do critério de parada.

Conjuntos finais de pesos diferentes surgem com mesma topologia e mesmo treinamento [Stanley, 1991]. Isto se deve a vários fatores, entre eles o fato de existirem muitas simetrias no mapeamento entrada-saída, a não existência de garantias que o problema tenha somente uma solução, e às condições iniciais aleatórias do conjunto de pesos. Deve-se sempre lembrar que a aprendizagem é um processo estocástico, e, portanto, deve-se treinar cada rede várias vezes com condições iniciais diferentes e usar a melhor [Reed, 1999].

É comum iniciar os pesos aleatoriamente. Entretanto, um neurônio na região linear aprende mais rapidamente que na região saturada. Em função disto, verificou-se uma regra prática para não-linearidade *tanh*: ajustam-se os pesos iniciais com variância:

$$\sigma^2 = \left(\frac{-2,4}{I}, \frac{2,4}{I} \right) \quad (27)$$

onde I é o número de entradas do neurônio.

Em redes não-lineares, como praticamente sempre é o caso de uma rede MLP, a seleção do tamanho do passo é muito importante. Em princípio deseja-se que o passo seja o maior possível no início do treinamento para acelerar a convergência, mas ao longo do processo este valor deve diminuir para se obter uma maior precisão do resultado. Uma alternativa é o uso do escalonamento do tamanho do passo, o qual pode ser dado por:

$$\eta(n) = \frac{\eta_0}{1 + \frac{n}{n_0}}, \quad \text{onde} \begin{cases} \eta_0 \text{ é o tamanho inicial do passo} \\ n_0 \text{ é um contador iterativo} \end{cases} \quad (28)$$

Experimentalmente obtém-se η_0 e n_0 . Se η_0 é muito grande então há divergência. A aprendizagem torna-se muito lenta se n_0 é muito pequeno. Se n_0 é muito grande a aprendizagem torna-se demorada. O escalonamento ajuda a escapar de mínimos locais em superfícies não-convexas.

A aprendizagem, seguindo-se o algoritmo de retro-propagação, requer alterações no peso proporcionais a $\partial E_p / \partial w$, ou seja, a descida do gradiente requer passos infinitesimalmente pequenos. Na prática deseja-se uma constante de aprendizagem tão grande quanto possível, o que leva a oscilações. Para evitar isto, acrescenta-se um termo α , denominado momento à equação de correção dos pesos [Caudill, 1992] [Fausett, 1994]:

$$\Delta w_{ij}(t+1) = \gamma \delta_i^p a_j^p + \alpha \Delta w_{ij}(t) \quad (29)$$

Existem várias formas propostas de realizar a atualização dos pesos durante o treinamento de uma rede MLP com o algoritmo de retro-propagação. Além do uso do escalonamento simples do tamanho do passo e do uso do momento, uma variante bastante utilizada é o algoritmo Delta-Bar-Delta, que atualiza o tamanho do passo em função da variação do sinal do erro [Jacobs, 1988]. Uma outra variante baseada na mesma idéia consiste no algoritmo com tamanho de passo adaptativo de Almeida [Almeida et al., 1997].

Uma decisão muito importante quando se está na etapa de treinamento de uma rede neural é quando parar. Um dos critérios mais utilizados é o número de iterações. Entretanto, este é um critério que não garante que o erro tenha atingido um patamar aceitável e pode levar a duas situações diferentes. Se o número de iterações for pequeno, o erro estará muito elevado ainda e será necessário treinar a rede novamente, com um número maior de iterações. Em caso contrário, ou seja, se o número de iterações for muito alto, pode-se chegar a um valor de erro aceitável, porém não há nenhuma indicação de qual será o desempenho da rede para outros exemplos além daqueles utilizados no treinamento.

Uma segunda alternativa para determinar o momento de parada do treinamento consiste em se observar o erro médio quadrático ("MSE"). Entretanto, este é apenas uma variável indireta em problemas de classificação. Mais do que isso, não se tem garantias de que o sistema possa atingir um valor de erro médio quadrático especificado. Outro procedimento utilizado consiste em se observar a taxa de decréscimo do erro médio quadrático, pois ela indica quando não há mais extração de informação. Entretanto, pode ocorrer uma parada prematura no caso de regiões planas da superfície de erro.

Uma das técnicas mais utilizadas para determinar o momento adequado de se interromper o treinamento consiste no uso de um conjunto de validação. Neste caso, o conjunto de dados é dividido em dois subconjuntos: um conjunto de treinamento (com algo em torno de dois terços dos dados), utilizado para treinar a rede, e um conjunto de validação (com o restante dos dados), utilizado para determinar o melhor momento de parada do treinamento. O conjunto de validação é apresentado à entrada da rede neural a cada n épocas, calculando-se o erro para este conjunto (sendo que os pesos da rede não são corrigidos para o conjunto de validação). Quando o erro no conjunto de validação aumentar, tem-se a indicação de que a rede está deixando de generalizar e começando a se

tornar específica para os dados de treinamento. Este procedimento implica em não se utilizar uma parcela dos dados para a determinação da rede, o que pode ser um inconveniente quando não se dispõe de uma grande quantidade de dados [Cohen, 1995].

Embora não exista um consenso em relação a quantos dados são suficientes para que uma rede modele adequadamente os dados, tem-se uma regra prática que um número aproximado de padrões N requerido para classificar exemplos de teste com um erro δ , em uma rede com W pesos é dado por:

$$N > \frac{W}{\delta} \quad (30)$$

considerando-se dados representativos. Quando isto não acontece, tem-se o problema inverso, ou seja, dado um conjunto limitado de exemplos, como criar uma rede com tamanho mínimo que não permita a ocorrência do sobre-ajuste [Pham, 2000]. Uma alternativa é o uso de conexões esparsas da camada de entrada para a camada escondida, ou então o uso de pré-processamento para reduzir a dimensionalidade dos dados.

Algumas heurísticas para melhorar o tempo de treinamento e o desempenho de redes MLP são as seguintes:

- a) Normalizar os dados em relação à faixa de ativação da rede;
- b) Usar não-linearidade do tipo tangente hiperbólica;
- c) Usar neurônios do tipo softmax na camada de saída, os quais fazem com que a soma das saídas tenha um valor unitário;
- d) Normalizar o sinal desejado ligeiramente acima ou abaixo do limite (p.ex. $\pm 0,9$ e não ± 1);
- e) Adicionar um valor constante de 0,05 na derivada da não-linearidade;
- f) Ajustar um tamanho do passo maior nas camadas mais próximas da entrada;
- g) Em aprendizagem online, alterar a seqüência do conjunto de treinamento periodicamente;
- h) Inicializar os pesos da rede na região linear da não-linearidade;
- i) Usar métodos de aprendizagem mais sofisticados (delta-bar-delta, gradiente conjugado etc.);
- j) Sempre ter mais padrões de treinamento que o número de pesos na rede e treinar a rede até que o MSE se torne menor que $\varepsilon/2$;
- k) Usar validação cruzada para parar o treinamento;
- l) Executar o treinamento da rede várias vezes para medir o desempenho.

3.3. Redes Construtivas

Um dos problemas quando da definição de uma topologia de uma RNA em camadas consiste em determinar o número das mesmas e a quantidade de neurônios em cada uma. Cada uma das possíveis configurações conduz a comportamentos diferentes. Se o número de camadas e neurônios por camada for muito maior que o necessário, haverá uma tendência de sobre-ajuste (“*overfitting*”), isto é, a rede treinará bem, atingindo um valor baixo para o erro, mas terá um desempenho ruim no conjunto de testes (e no subsequente uso em campo). Para evitar isto, busca-se a rede de menor tamanho que cumpra a sua função. Entretanto, se o tamanho da rede for muito pequeno, a mesma poderá ser incapaz de atingir o desempenho desejado mesmo durante o treinamento [Vonk, 1997].

Para tentar diminuir este problema existem duas estratégias para a escolha de uma topologia para uma rede neural:

- a) *Redes Construtivas*: Parte-se de uma rede básica, do tipo perceptron, e aumenta-se gradativamente a quantidade de ligações, através do aumento do número de camadas e de neurônios por camada, até que o desempenho seja satisfatório;
- b) *Redes com Poda*: Parte-se de uma rede “grande”, que resolve o problema e retiram-se conexões (que implicam em redução do número de neurônios e/ou número de camadas) até atingir a rede de tamanho mínimo que satisfaça os requisitos.

A seguir serão apresentadas algumas estratégias para redes construtivas.

3.3.1. Algoritmo da Torre (Fig 13):

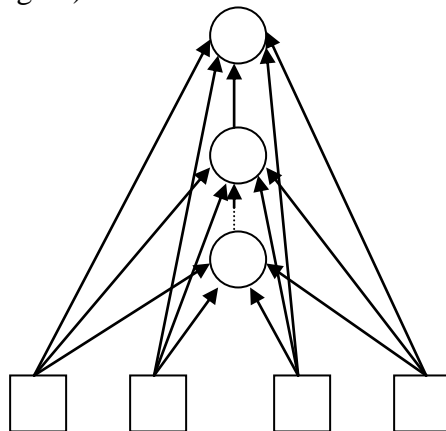


Figura 13 – Esquema de uma RNA usando o Algoritmo da Torre.

As características principais do algoritmo da Torre são: a) Todas as células intermediárias e de saída vêm as de entrada e a imediatamente anterior; b) Usa-se o algoritmo do bolso com catraca para treinar cada célula intermediária e de saída; c) Quando uma célula está treinada seus pesos são “congelados” e treina-se a próxima célula no caminho para a saída.

O algoritmo da Torre pode ser executado da forma descrita a seguir. Inicialmente usa-se o algoritmo do bolso para gerar um modelo de célula única e congelam-se seus pesos após o treinamento. Em seguida, cria-se uma nova célula que vê as p entradas mais a ativação da célula mais recentemente treinada. Feito isto, executa-se o algoritmo do bolso com catraca para treinar os $p+2$ pesos (incluindo bias) para esta célula. Se a rede tem desempenho melhorado com a nova célula, os seus pesos são congelados e retorna-se ao passo anterior criando uma nova célula; caso contrário remove-se a última célula inserida e o processo é terminado.

3.3.2. Algoritmo da Pirâmide (Fig 14):

As características do algoritmo da Pirâmide são idênticas às do algoritmo da torre, exceto que cada nova célula recebe conexões de todas as outras células. Embora as conexões e pesos adicionais pareçam melhorar a sua aprendizagem, as simulações não foram conclusivas.

3.3.3. Algoritmo de Correlação em Cascata

As características do algoritmo de correlação em cascata são ter um topologia similar à pirâmide, usar ativação contínua (sigmóide) e em cada estágio, adiciona-se e treina-se uma nova célula intermediária e treina-se novamente a célula de saída (que tem mais uma entrada).

O algoritmo de correlação em cascata trabalha basicamente em duas etapas. Inicialmente treina-se a nova célula para a correlação com o erro residual da célula de

saída. Usando u_{ik} para indicar o valor de u_i para o exemplo de treinamento k e \bar{x} indicar a média da variável x sobre todos os N exemplos de treinamento, busca-se maximizar

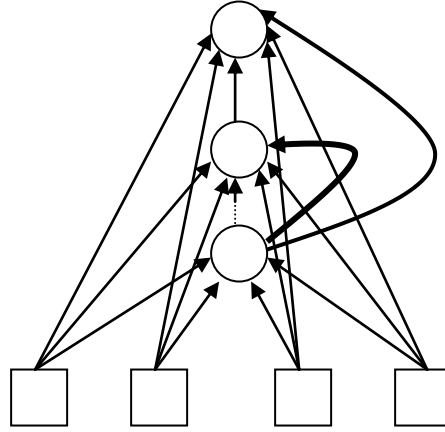


Figura 14 – Rede construída usando o Algoritmo da Pirâmide.

$$\chi = \left| \sum_{k=1}^N (u_i^k - \bar{u}_i) \left\{ (C^k - u_n^k) - (\bar{C} - \bar{u}_n) \right\} \right| \quad (31)$$

Na etapa seguinte, usa-se o método da descida do gradiente para encontrar pesos $\{w_{i,j}\}$ que maximizem χ . A seguir congelam-se os pesos para a nova célula e treina-se novamente a saída usando retro-propagação para encontrar $w_{n,i}$.

3.3.4. Algoritmo do Telhado (figura 15)

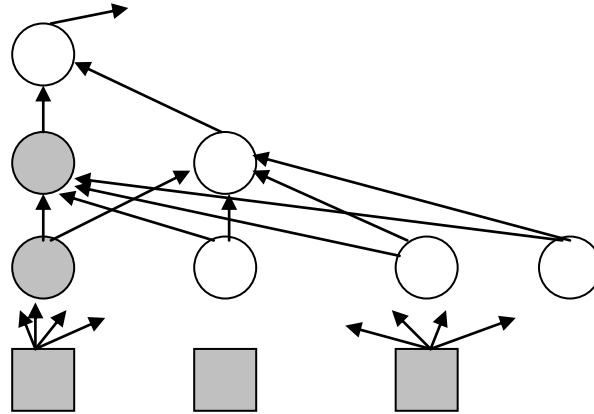


Figura 15 – Rede Neural construída segundo o Algoritmo do Telhado.

As principais características do algoritmo do Telhado são: a construção ocorre em camadas onde células da camada L recebem ativações somente da camada $L-1$, a célula-mestre da camada L classifica corretamente mais exemplos que aquela da camada $L-1$, e as células subordinadas garantem que não existem dois exemplos de treinamento com classificações diferentes que tenham o mesmo conjunto de ativações na camada L .

O algoritmo do telhado atua executando as ações descritas a seguir. Inicialmente faz-se a camada $L = 2$ (camada 1 é a camada de entrada). Em seguida, usa-se o algoritmo do bolso com catraca para criar a célula-mestre para a camada L usando todas as ativações da camada $L-1$. Se a célula-mestre da camada L classifica corretamente todos os exemplos de treinamento, então terminar. Caso contrário, continua-se a adicionar células

subordinadas até que a camada L se torne fiel. Deve-se encontrar um subconjunto de exemplos de treinamento de tamanho máximo com mais de uma classificação, que produz as mesmas ativações para todas as células na camada L. Usa-se então o algoritmo do bolso com catraca para treinar uma nova célula subordinada para a camada L usando somente o subconjunto de exemplos de treinamento do passo anterior. Neste momento a camada L tornou-se fiel. Fazer $L = L + 1$ e retornar ao segundo passo.

3.3.5. *Algoritmo Grandioso* (figura 16):

As características do algoritmo Grandioso são: entradas e saídas discretas em $\{+1, -1\}$, o treinamento de uma única célula u_n usando o algoritmo do bolso com catraca, no caso de dados não separáveis linearmente treinam-se unidades u_{n+} e u_{n-} para corrigir os erros de u_n , a unidade u_{n+} fornece reforço positivo quando u_n classifica incorretamente uma saída -1, o mesmo acontecendo com a unidade u_{n-} com saída +1, após o treinamento u_{n+} e u_{n-} são unidas a u_n por grandes pesos positivos e negativos respectivamente, e se u_{n+} e u_{n-} não classificam seus respectivos conjuntos de treinamento repete-se a construção recursivamente.

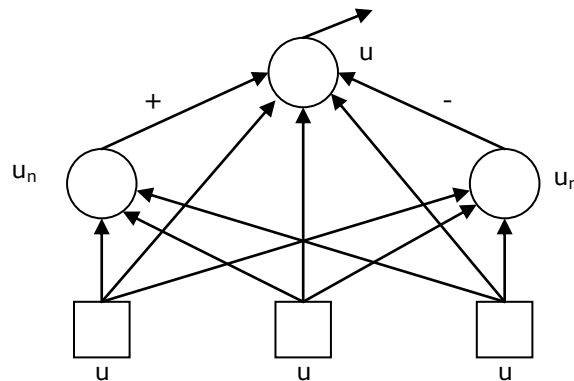


Figura 16 – Rede Construída segundo o princípio do Algoritmo Grandioso.

4. Aproximação de Funções

4.1. Introdução

A tarefa de regressão consiste na busca da representação do relacionamento entre os dados de entrada e saída, onde a saída é representada por uma ou mais variáveis contínuas. Já na tarefa de classificação considera-se que a variável de saída pertence a uma de várias classes e o objetivo é separá-la nas classes tão corretamente quanto possível. A tarefa mais genérica é chamada de aproximação de funções e ela transforma-se na regressão linear com o uso de topologias lineares ou em classificação com o uso de funções especiais chamadas funções indicadoras.

O objetivo da aprendizagem consiste então em descobrir a função de mapeamento $f(\bullet)$, dado um número finito (na prática desejável pequeno) de pares entrada-saída (x, d) . As redes neurais são úteis para aproximação de funções porque elas são aproximadores universais e eficientes, e porque elas podem ser implementadas como uma máquina de aprendizagem.

Podemos definir o objetivo da aproximação de funções como sendo: Em uma área compacta S do espaço de entrada descrever uma função $f(x)$, pela combinação de funções $\varphi_i(x)$ mais simples

$$\hat{f}(\mathbf{x}, \mathbf{w}) = \sum_{i=1}^N w_i \varphi_i(\mathbf{x}) \quad (32)$$

onde \mathbf{w}_i são elementos reais do vetor $\mathbf{w}=[w_1, \dots, w_N]$ tais que

$$\left| f(\mathbf{x}) - \hat{f}(\mathbf{x}, \mathbf{w}) \right| < \varepsilon \quad (33)$$

e ε pode ser arbitrariamente pequeno. A função $\hat{f}(\mathbf{x}, \mathbf{w})$ é chamada de função aproximante e as funções $\{\varphi_i(\mathbf{x})\}$ são chamadas de *funções elementares*.

As decisões básicas na aproximação de funções consistem na escolha das funções elementares $\{\varphi_i(\mathbf{x})\}$, em como calcular os pesos \mathbf{w}_i , e na seleção do número de funções elementares. Se o número de vetores de entrada \mathbf{x}_i é igual ao número de funções elementares $\{\varphi_i(\mathbf{x})\}$ a solução torna-se

$$\mathbf{w} = \Phi^{-1} f \quad (34)$$

Uma condição importante neste caso é que a inversa de Φ deve existir, e, portanto, para a escolha para as funções elementares tem-se como requisito que $\Phi^{-1}(\mathbf{x})$ deve existir. Isto é obtido se as funções elementares constituírem uma base, isto é, se elas forem linearmente independentes, ou seja,

$$w_1 \varphi_1(\mathbf{x}) + \dots + w_N \varphi_N(\mathbf{x}) = 0 \quad \text{sse} \quad (w_1, \dots, w_N) = 0 \quad (35)$$

Pode-se admitir a hipótese simplificadora seguinte: impor que as funções elementares usem bases ortonormais, ou seja,

$$\int_S \varphi_i(\mathbf{x}) \varphi_j(\mathbf{x}) d\mathbf{x} = \delta_{ij}(\mathbf{x}) \quad (36)$$

onde $\delta(x)$ é a função delta de Dirac

Alguns exemplos de funções elementares são a função *sinc* (figura 17), a série de Fourier (figura 18) e a wavelet (figura 19).

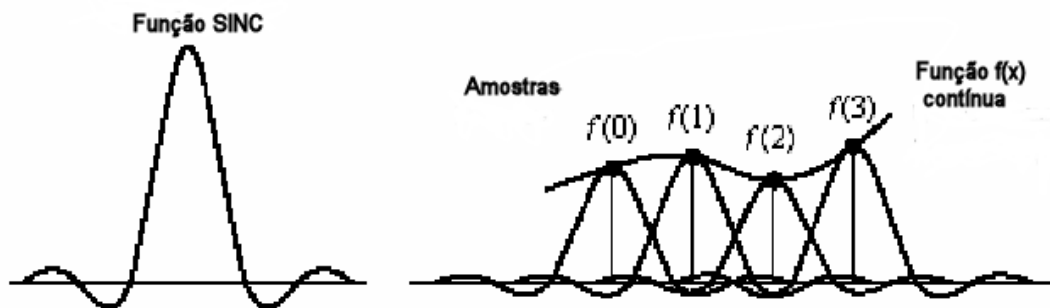


Figura 17 – Função elementar sinc.

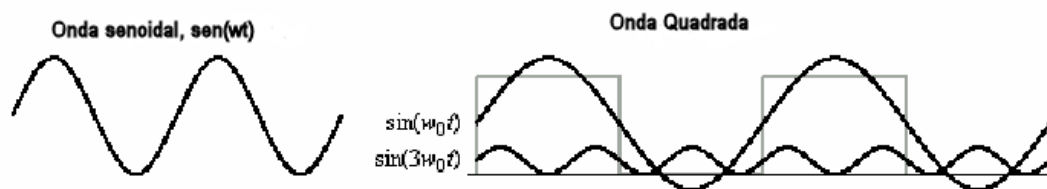


Figura 18 – Função elementar do tipo série de Fourier.

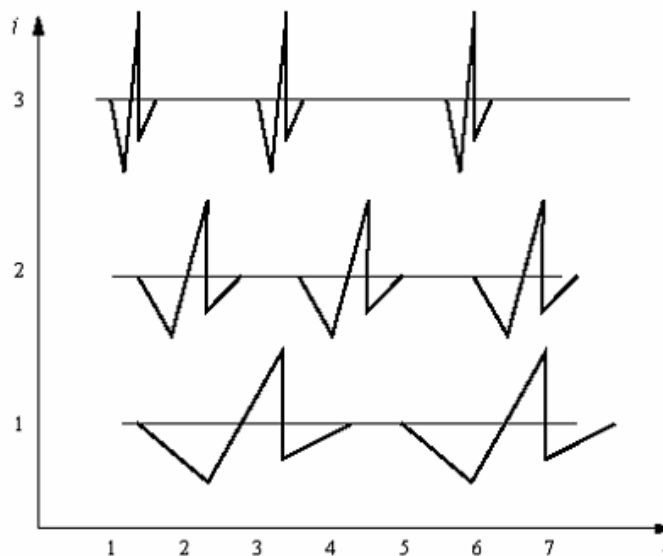


Figura 19 – Wavelet.

Existem diversas bases possíveis para a aproximação de funções na rede MLP. A divisão mais geral em relação às funções elementares utilizadas é a seguinte:

- a) funções globais, quando as funções utilizadas compreendem todo o espaço de entrada; e
- b) funções locais, quando as funções utilizadas respondem de maneira especial a uma área limitada do espaço de entrada.

Uma rede MLP com uma camada escondida e com um neurônio de saída linear pode ser considerada como uma implementação de um sistema para aproximação de funções, onde as bases são os neurônios escondidos. O neurônio sigmoidal responde a todo o espaço de entrada, ou seja, a MLP implementa uma aproximação com funções elementares globais.

A tarefa de aproximação de funções usando rede MLP tem as seguintes características:

- a) A rede MLP realiza aproximação de funções com um conjunto adaptativo de bases, determinados a partir dos dados entrada-saída;
- b) As bases são alteradas em função dos dados, ou seja, o espaço de projeção é dependente dos dados;
- c) O treinamento é mais difícil pois não somente a projeção como também a base está sendo alterada;
- d) As representações são mais compactas;
- e) Devido à alta conectividade e natureza global das funções elementares, um bom ajuste é obtido com poucas bases, isto é, com poucos neurônios escondidos;

f)O treinamento é mais difícil, pois as bases não são ortogonais entre si.

As redes MLP são mais eficientes que polinômios para aproximação de funções em espaços de alta dimensão. Um exemplo de resultado obtido está apresentado na figura 20.

A rede MLP pode ser utilizada tanto para classificação quanto para aproximação de funções. Entretanto, em termos de projeto as duas situações diferenciam-se pelos seguintes fatores:

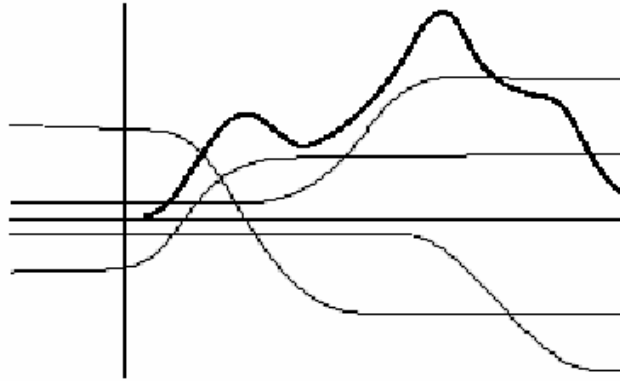


Figura 20 – Aproximação de funções com uma rede MLP com 4 neurônios escondidos.

a) Quanto ao elemento de saída: Para aproximação de funções usa-se um neurônio com função de ativação linear, enquanto que para classificação o neurônio deve utilizar uma função de ativação não-linear;

b) Quanto ao ponto de operação dos neurônios escondidos: Para aproximação de funções eles devem estar longe da saturação para que mapeamento seja suave, enquanto que para classificação os neurônios escondidos devem operar na região de saturação, já que as saídas devem tender para 1 ou 0.

4.2. Rede RBF

Utilizando-se uma base alternativa em redes MLP para sistemas não-lineares, tem-se a chamada rede RBF (“Radial Basis Function”). Para as funções de base radial (RBF) tem-se a condição

$$\phi_i(\mathbf{x}) = \gamma(|\mathbf{x} - \mathbf{x}_i|) \quad (37)$$

onde $\gamma(\bullet)$ é normalmente uma função gaussiana, ou seja,

$$\begin{aligned} G(\mathbf{x}) &= \exp\left(-\frac{x^2}{2\sigma^2}\right), & \text{unidimensional} \\ G(\mathbf{x}) &= \exp\left(-\frac{\mathbf{x}^T \Sigma^{-1} \mathbf{x}}{2}\right), & \text{multidimensional} \end{aligned} \quad (38)$$

com variância σ^2 ou covariância $\Sigma = \sigma^2 \mathbf{I}$. Uma gaussiana centrada em \mathbf{x}_i com variância σ^2 , é uma função elementar local. Um exemplo em uma dimensão pode ser visto na figura 21.

A aproximação de funções em uma área limitada do espaço de entrada requer o posicionamento das gaussianas localizadas para cobrir o espaço, o controle da largura de cada gaussiana e o ajuste da amplitude de cada gaussiana.

Como as bases RBF são locais, a alteração em uma delas não perturba a aproximação em outras áreas do espaço. Isto implica em que se necessita exponencialmente mais RBFs para cobrir espaços de alta dimensão. Entretanto, com os centros já determinados, as RBFs treinam eficientemente, já que o erro é linear nos pesos. Mais do que isso, se os centros forem otimamente ajustados, garante-se a convergência para o mínimo global [Sundararajan, 1999].

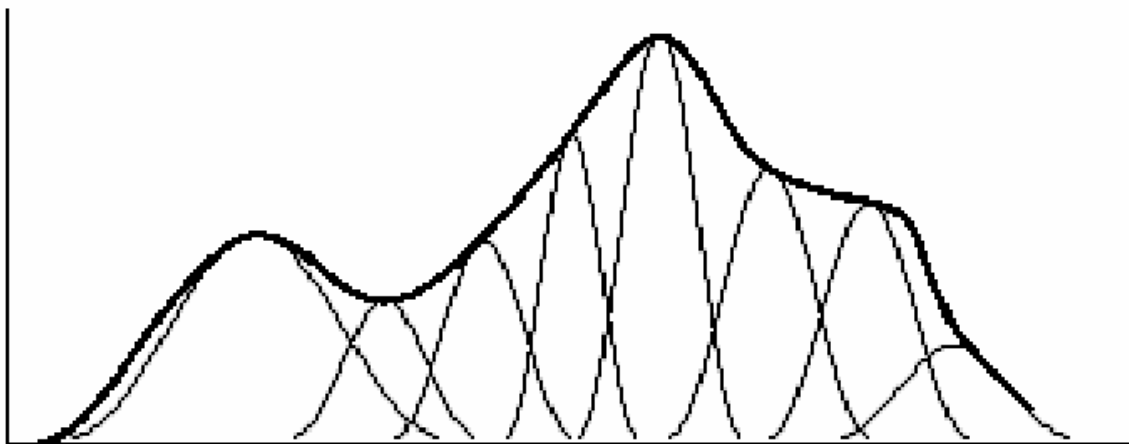


Figura 21 – Aproximação de funções usando uma rede RBF com função elementar gaussiana.

Se for usado o critério do erro médio quadrático durante o treinamento, pode-se ter uma interpretação probabilística do mapeamento, pois a rede MLP com RBF realiza regressão não-linear. Ela é sendo capaz de descobrir qualquer relação entrada-saída determinista com ruído aditivo de média zero. Os requisitos para isto são de que haja a convergência para mínimo global, que o número graus de liberdade seja suficiente e de que existam dados suficientes para o treinamento.

Existem vários meios de realizar a adaptação do centro e da variância das gaussianas. O método mais simples consiste em distribuir uniformemente os centros das gaussianas. Para funções que cobrem todo o espaço isto pode ser realizado sem grandes prejuízos. Entretanto, se existirem agrupamentos de dados este procedimento não é indicado. Duas alternativas são o método supervisionado e o método auto-organizado.

O método supervisionado faz uso do algoritmo de retro-propagação para treinar RBFs. O uso do mesmo implica em se ter um treinamento lento e também faz com que as variâncias se tornem muito grandes e a RBF perca a sua natureza de processo local.

Já no método auto-organizado de treinamento de RBFs, existem duas etapas:

- a) Na primeira etapa ocorre a adaptação independente dos pesos da primeira camada. Os agrupamentos de dados atuam como atratores para os centros das gaussianas e as variâncias são estimadas para cobrir a distribuição dos dados de entrada;
- b) Na segunda etapa acontece a adaptação dos pesos de saída, usando o algoritmo LMS, tendo em vista que o problema de adaptação é linear nos pesos. Nesta etapa mantém-se a primeira camada “congelada”.

Uma das decisões mais importantes é a seleção do número de bases, pois ela define as características principais do sistema. Com poucas bases elementares a aproximação é fraca. Além disto o número de bases é dependente do tamanho da rede e dos valores dos coeficientes. O uso de um polinômio de grau muito baixo causa bias do modelo, enquanto que um polinômio de grau muito alto resulta em grande oscilação, ou seja, variância do modelo. Por isto, busca-se um compromisso entre baixo bias e baixa variância ao longo do domínio.

5. Redes Temporais

5.1. Introdução

Problemas que não envolvem o tempo (tais como classificação de padrões) são chamados estáticos. Já os sistemas dinâmicos que envolvem uma ou mais variáveis são funções do tempo, o qual estabelece uma ordem nos dados de entrada. O tempo é um continuum e as variáveis físicas também são contínuas, compondo os sinais analógicos.

O trabalho com sistemas dinâmicos é mais complexo do que com sistemas estáticos, pois embora o tempo ajude a remover a ambigüidade dos dados, torna-se necessário o uso de uma memória de curto prazo.

A primeira etapa de trabalho nos sistemas dinâmicos envolve a amostragem e quantização. A amostragem consiste na obtenção periódica da variável de interesse, e a quantização refere-se ao valor discretizado desta variável. Em um conversor analógico-digital, a cada T segundos, chamado período de amostragem, o sinal analógico $x(t)$ é medido, produzindo um sinal $\{x(nT)\}$, chamado seqüência ou série temporal (conforme a figura 22):

$$\{x(nT)\} = x(T), x(2T), x(3T), \dots, x(NT) \quad (39)$$

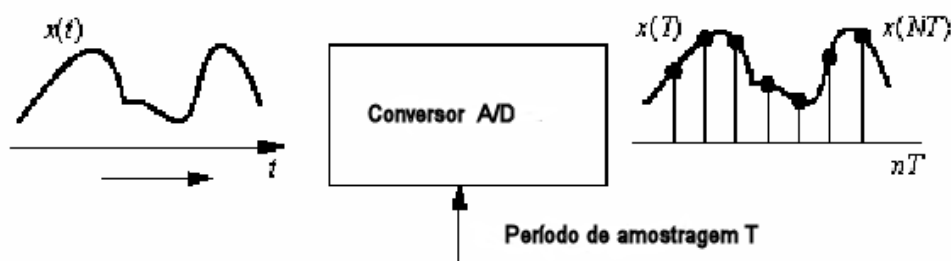


Figura 22 – Conversor analógico-digital.

Denomina-se elemento de atraso ideal, denotado por z^{-1} , um componente que atrasa o sinal em uma amostra. Uma linha de atraso é um sistema com uma entrada e várias saídas compostas pela ligação em cascata de vários operadores de atraso (figura 23).

A maioria dos sistemas dinâmicos trabalha com a noção de janela de tempo. O tamanho do espaço de reconstrução determina o comprimento N de uma janela de tempo que desliza sobre a série temporal completa (figura 24). Este comprimento corresponde ao tamanho da linha de atraso e estabelece a dimensionalidade do espaço de reconstrução.

A filtragem pode ser vista como uma projeção, onde o objetivo de um sistema linear é escolher uma direção de projeção tal que a informação desejada contida no sinal de

entrada seja preservada. Um combinador linear (FIR) com $N+1$ pesos tem uma memória de somente N amostras. Portanto, filtragem é uma distorção seletiva do sinal de entrada, definida pelo usuário.

Considerando-se que a resposta desejada está disponível, pode-se usar a idéia de adaptação automática para criar filtros. Um filtro adaptativo é um tipo especial de rede neural linear (sem bias), ou seja, pode ser estudada no espaço de vetores lineares. Se a ordem de um filtro FIR é grande o suficiente, qualquer função de transferência pode ser criada com o combinador linear.

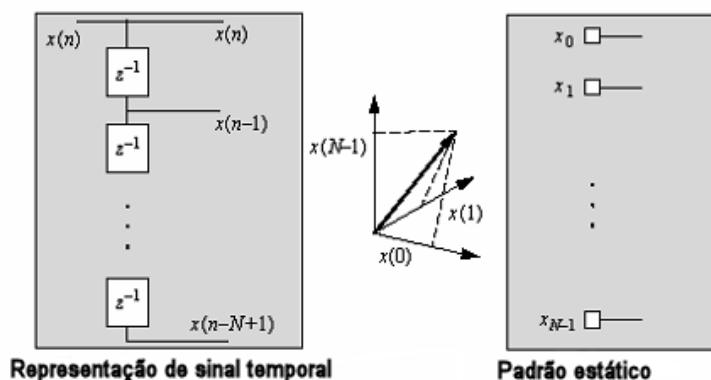


Figura 23 – Linha de atraso.

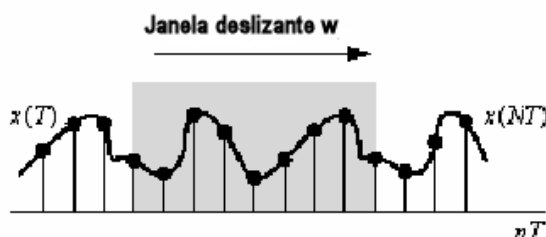


Figura 24 – Janela de tempo deslizando.

O combinador linear adaptativo tem o mesmo diagrama de blocos da regressão. Portanto, a função do combinador linear adaptativo pode ser interpretada como uma regressão linear (sem bias) da série temporal de entrada para a série temporal desejada.

É importante lembrar que existem diferenças entre os sistemas neurais e os sistemas adaptativos. Embora as RNAs tenham pesos treináveis, estes são fixados durante o teste, ou seja, a RNA não é adaptável após o treinamento. Já os filtros adaptativos estão sendo continuamente adaptados, não havendo divisão em conjunto de treinamento e conjunto de teste, ou seja, eles rastreiam as alterações na estrutura temporal. Portanto, as RNAs modelam os dados como memória de longo prazo em seus pesos, enquanto que o combinador linear é basicamente um rastreador, ou seja, possui somente memória de curto prazo.

Em séries temporais usam-se muitas vezes as redes hebbianas (que serão vistas no tópico 6). O neurônio estima os autovalores e auto-vetores da auto-correlação de entrada, o qual é um sinal temporal. Esta versão discreta é chamada PCA temporal, também chamada de Transformada de Karhunen-Loeve (figura 25).

Os mapeadores estáticos implementam transformações estáticas entre a entrada e a saída. Portanto, a resposta é calculada instantaneamente e não se altera se a entrada é mantida constante, e o tempo de transiente é nulo. Já o combinador linear preserva os valores anteriores da entrada na linha de atraso e a resposta é obtida após determinado tempo, resultando em um tempo transiente finito.

Comparando a memória de curto prazo com a memória de longo prazo verifica-se que os mapeadores estáticos funcionam como repositórios de informações do passado, não diferenciando relações temporais, pois a informação temporal está colapsada nos pesos. Os sistemas dinâmicos são sensíveis à sequência de apresentação dos dados e os pesos codificam diferenças dentro da janela temporal de observação.

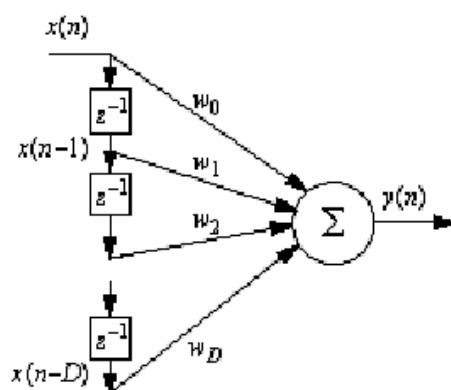


Figura 25 – Rede com linha de atraso.

As redes neurais dinâmicas são topologias projetadas para incluir relações temporais de forma explícita no mapeamento entrada-saída. Em particular, as TLFN (“Time-Lagged Feedforward Network”) integram a estrutura de um filtro linear dentro de uma rede neural para estender a capacidade de mapeamento não-linear de uma rede com a representação do tempo. Neste caso, a linha de atraso do filtro linear é chamada de memória de curto prazo.

Quando se realiza a extração de informações temporais busca-se o reconhecimento de padrões temporais a partir de um sinal $x(n)$ localmente estacionário, M -dimensional. Para tanto, a ordem de amostragem deve ser preservada quando se realiza processamento temporal, surgindo a necessidade de se definir como selecionar o tamanho da janela temporal, a qual determina o tamanho do espaço de reconstrução (figura 26).

Uma melhoria na eficiência da representação temporal é obtida levando a memória para dentro da máquina de aprendizagem. Isto implica na escolha do tamanho da janela temporal e da escolha da importância das amostras na janela. Nestes modelos ocorre apenas o recebimento da entrada atual, como no modelo biológico e o sistema de aprendizagem pode utilizar a filtragem (informação no domínio da frequência), resultando em um novo paradigma para processamento de padrões temporais (figura 27).

Este tipo de rede, chamada de TDNN (“Focused Time-Delay NN”), é dita concentrada (“focused”), pois indica que existe memória somente na camada de entrada. Na TDNN substituem-se os neurônios de entrada de uma MLP por uma linha de atraso. O ponto positivo é que ela pode ser treinada com o algoritmo de retro-propagação estático, desde que o sinal desejado esteja disponível a cada instante. Por outro lado surge como

dificuldade de projeto, além daquelas da MLP, também a escolha do tamanho da linha de atraso (figura 28).

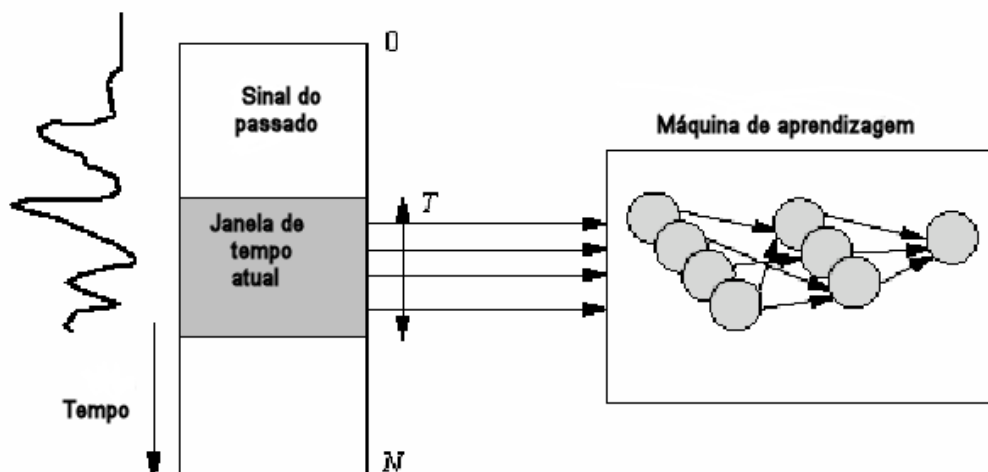


Figura 26 – Janela temporal utilizada por uma máquina de aprendizagem.

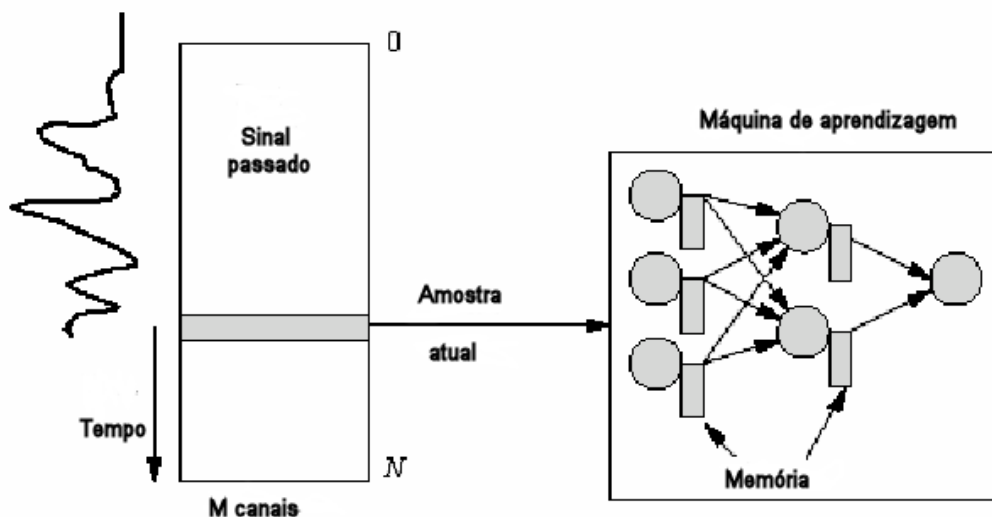


Figura 27- Sistema neural janela de tempo interna usa apenas uma amostra por vez.

As principais tarefas nas quais se usa uma TDNN são: a) Classificação, ou seja, para encontrar diferentes padrões temporais; b) Identificação de sistemas, onde se busca tornar a saída atual da rede o mais próxima possível da saída do sistema a ser modelado; c) Previsão, onde o objetivo é aproximar a próxima amostra como sendo uma combinação não-linear das amostras anteriores do sinal de entrada. A rede TDNN representa um compromisso entre simplicidade da estrutura da rede e poder de processamento.

A figura 29 mostra um novo neurônio com memória, que é um mecanismo de memória de curto prazo. Este neurônio com memória recebe várias entradas $x_i(n)$ e produz várias saídas $y=[y_0(n), \dots, y_D(n)]_T$, que são versões atrasadas da entrada combinada $y_0(n)$:

$$y_k(n) = g(y_{k-1}(n)) \quad y_o(n) = \sum_{j=1}^p x_j(n) \quad (40)$$

Uma rede TLFN (“Time-Lagged Feedforward NN”) é um arranjo em avanço de neurônios com memória e neurônios não-lineares (por exemplo uma TDNN). A memória de curto prazo em TLFNs pode ser de qualquer tipo e distribuídas em qualquer camada. Elas possuem as propriedades interessantes das redes em avanço e podem capturar a informação presente no sinal de entrada.

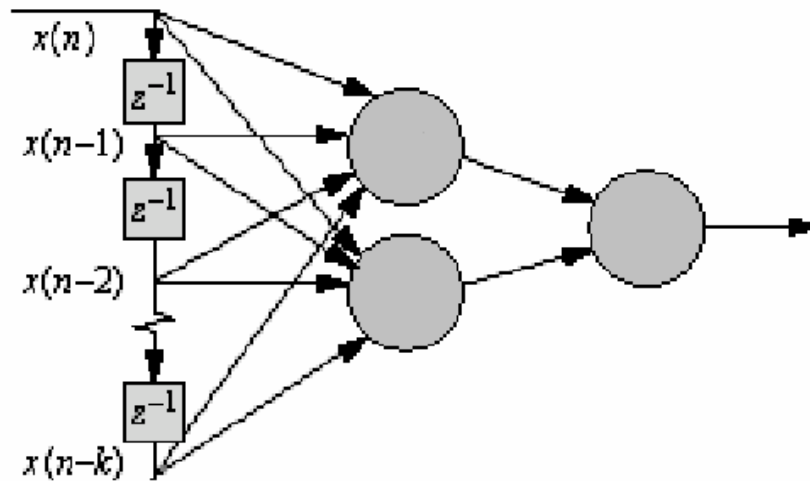


Figura 28 – Rede TDNN com k atrasos de tempo.

Nas TLFN focadas, os neurônios com memória estão na camada de entrada. Nelas há dois estágios: primeiramente há a representação temporal linear (com a memória) e em seguida há o estágio de mapeamento (usando uma rede MLP ou RBF) conforme a figura 30. Usando uma TLFN com uma MLP como mapeador, ficam normalmente acoplados a dimensão do espaço de projeção (número de filtros de memória) e o número de neurônios de entrada da MLP, embora não haja razão para acoplar o tamanho do espaço ao número de neurônios escondidos. Pode-se desacoplar o número de eixos do número de neurônios escondidos através de uma camada extra de neurônios lineares entre ambos.

Quando se utiliza uma rede neural para realizar processamento temporal deve-se sempre atentar para o problema da estabilidade. No caso de uma rede TLFN deve-se criar um mecanismo de memória de curto prazo estável, o que não é muito complexo. No caso de redes recorrentes é bem mais difícil este cálculo. Em função disto, o treinamento de uma rede TLFN é mais fácil que o treinamento de uma rede recorrente genérica. Entretanto, apesar da rede TLFN funcionar como um mapeador universal, certas funções exigem um tamanho muito grande da rede para atingir as características especificadas.

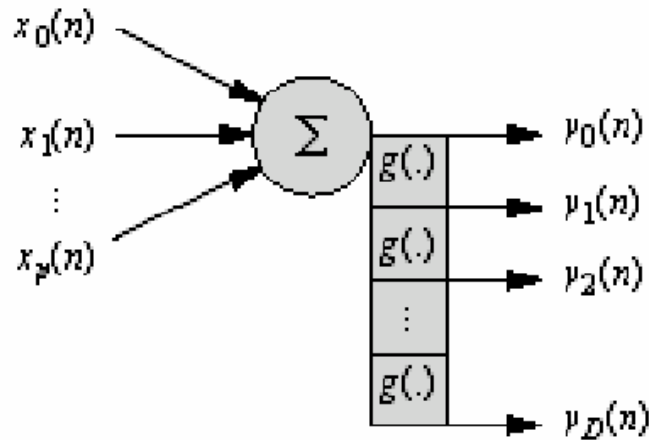


Figura 29 – O neurônio com memória.

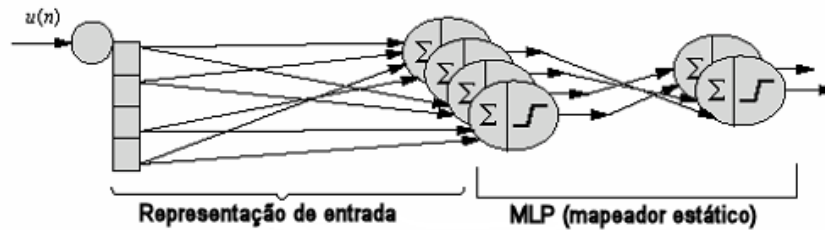


Figura 30 – Uma rede TFLN focada com uma entrada.

A rede de Elman e a rede de Jordan são redes mais simples que as anteriores, sendo baseadas em neurônios de contexto [Golden, 1995]. Elas são fáceis de treinar (pois os parâmetros de realimentação são fixos), e realizam o mapeamento com topologias pequenas, onde não há recorrência no caminho entrada-saída (figura 31).

As redes de Elman e Jordan podem ser treinadas de forma aproximada com o algoritmo de retro-propagação. A memória engloba várias camadas, realizando um mapeamento não-linear dinâmico. Elas foram usadas inicialmente para reconhecimento de seqüências (redes seqüenciais). A rede de Jordan pode associar a mesma entrada fixa a várias seqüências de saída em função do contexto.

6. Aprendizagem Hebbiana

O princípio da aprendizagem estudado por Hebb diz que a comunicação entre dois neurônios é facilitada pela excitação repetida dos mesmos [Hebb, 1949]. A Regra de Hebb diz que se a saída do i -ésimo neurônio é y_i e a ativação do j -ésimo neurônio é x_j , então

$$\Delta w_{ij} = \eta x_j y_i \quad (41)$$

onde η é o tamanho do passo. Para aplicar a regra de Hebb, somente os sinais de entrada precisam fluir através da rede, ou seja, a regra de Hebb é local ao peso. A regra de Hebb é uma regra biologicamente plausível e os modelos biológico e artificial coincidem (figura 32).

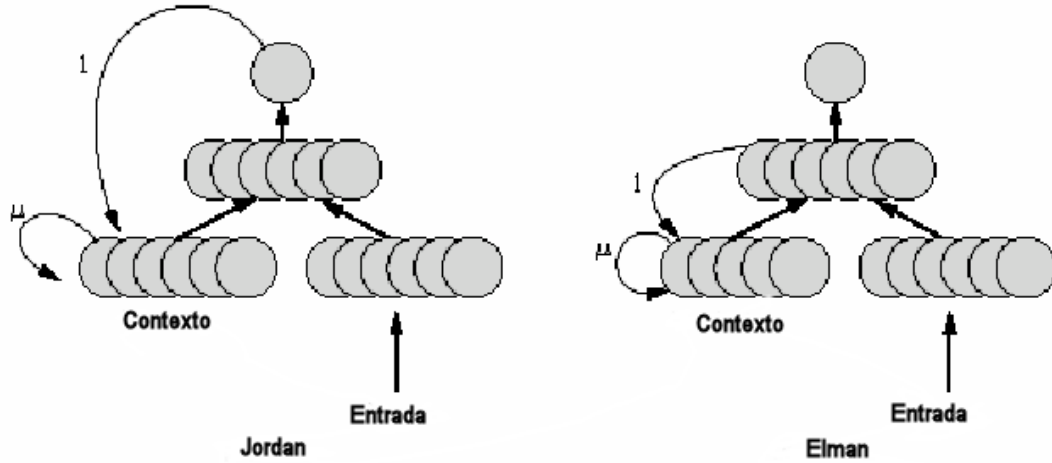


Figura 31 – Topologia da rede de Jordan e da rede de Elman.

Podemos analisar o efeito da atualização hebbiana considerando que a mesma atualiza os pesos de acordo com

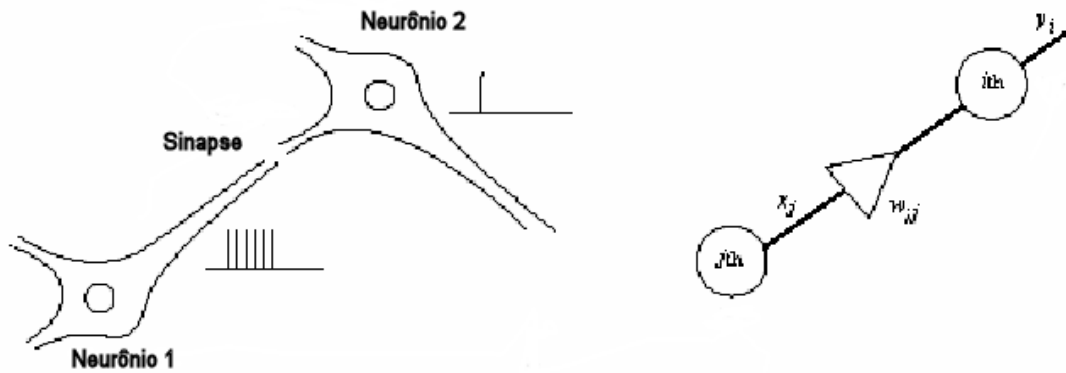


Figura 32 – Sistema biológico e artificial.

$$w(n+1) = w(n) + \eta x(n)y(n) \quad (42)$$

na qual n é o número da iteração e η o tamanho do passo. Para um neurônio com função de ativação linear, $y = wx$ e, portanto,

$$w(n+1) = w(n)[1 + \eta x^2(n)] \quad (43)$$

Pela equação anterior, observa-se que a aprendizagem hebbiana é intrinsecamente instável, produzindo pesos muito grandes, o que torna o seu uso não prático.

Pode-se realizar outra análise do neurônio com várias entradas (figura 33). Em notação vetorial a saída do neurônio é

$$y = \mathbf{w}^T \mathbf{x} = \mathbf{x}^T \mathbf{w} \quad \text{ou seja,} \quad y = |\mathbf{w}| |\mathbf{x}| \cos \theta \quad (44)$$

Assumindo que as entradas e os pesos estão normalizados, um valor de y maior significa que a entrada está mais “próxima” da direção do vetor peso (figura 34). Durante a

aprendizagem os dados expostos aos pesos condensam toda informação nos valores dos pesos. Portanto, os pesos representam a memória de longo termo.

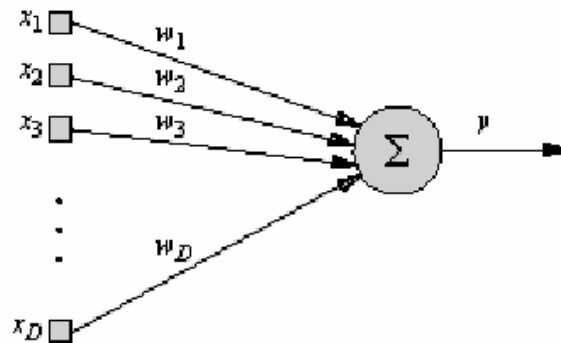


Figura 33 – Neurônio linear com D entradas.

O neurônio hebbiano é simples e cria uma medida de similaridade (produto interno) no espaço de entrada de acordo com a informação contida nos pesos. A saída do neurônio responde em nível alto ou baixo, de acordo com a similaridade entre a entrada atual e o que ele “recorda” do treinamento. Desta forma, o neurônio hebbiano implementa um tipo de memória chamada memória associativa.

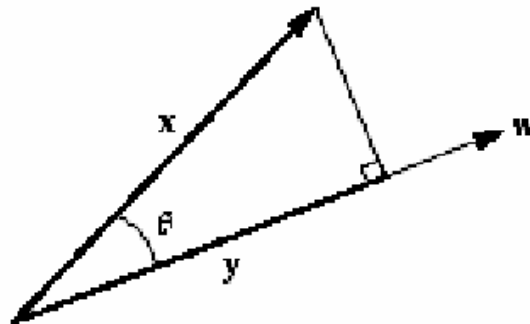


Figura 34 – Saída de um neurônio linear no espaço vetorial.

Na aprendizagem batch a regra de Hebb atualiza os pesos com uma estimativa amostral da função de auto-correlação

$$\Delta \mathbf{w} = \eta \hat{\mathbf{R}}_x \mathbf{w}(0) \quad \text{onde} \quad \mathbf{R}_x = E[\mathbf{x}\mathbf{x}^T] \quad (45)$$

Com a regra de Hebb, o algoritmo faz a subida do gradiente (busca do máximo) dos dados de entrada. A regra de adaptação on-line é simplesmente uma versão estocástica e tem o mesmo comportamento. No caso da representação de dados em espaços multidimensionais, tem-se que os pesos da rede treinada com a regra de aprendizagem de Hebb indicam a direção do gradiente do campo de entrada. A saída da rede indica a projeção de maior variância, ou seja, os eixos do sistema principal de coordenadas onde a saída projetada tem a maior variância. Esta é, portanto, uma forma de criar pesos de rede ajustados às estatísticas de segunda ordem dos dados de entrada (figura 35).

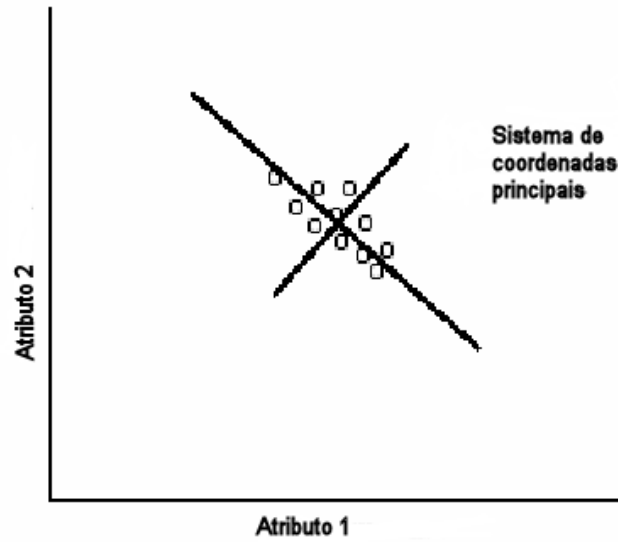


Figura 35 – Sistema de coordenadas principais.

Para criar uma forma útil da aprendizagem de Hebb é necessário normalizar os pesos. A forma mais simples de se fazer isto foi proposta por Oja [Oja, 1982]:

$$w_i(n+1) = \frac{w_i(n) + \eta y(n)x_i(n)}{\sqrt{\sum_i (w_i(n) + \eta y(n)x_i(n))^2}} \quad (46)$$

Assumindo um tamanho de passo pequeno, Oja aproximou a equação anterior por

$$\begin{aligned} w_i(n+1) &= w_i(n) + \eta y(n)[x_i(n) - y(n)w_i(n)] \\ &= w_i(n)[1 - \eta y^2(n)] + \eta x_i(n)y(n) \end{aligned} \quad (47)$$

A regra de Oja pode ser considerada a atualização de Hebb com uma atividade normalizada. Ela evita o crescimento ilimitado dos pesos aplicando um “termo de esquecimento” proporcional ao quadrado da saída. Entretanto, se o padrão não estiver presente freqüentemente, ele será esquecido, já que a rede esquece associações antigas.

A Regra de Oja implementa a idéia do máximo autofiltro, pois o treinamento de um neurônio linear com o algoritmo de Oja produz um vetor de pesos que é o autovetor da matriz de autocorrelação de entrada e na saída o maior autovalor [Chen, 1984]. O vetor de pesos obtido usando o algoritmo de Oja indica a direção onde os dados produzem a maior projeção. Uma rede com um único neurônio, treinada com a regra de Oja, extrai o máximo possível de informação da entrada. Uma projeção de um agrupamento de dados nos componentes principais pode ser vista na figura 36 [Hinton, 1999].

O associador linear, também chamado de memória associativa linear (LAM) fornece um paradigma alternativo à memória computacional. A topologia do associador linear, treinado com a regra hebbiana forçada pode ser usado como uma memória associativa. Tal dispositivo pode ser treinado para associar uma entrada \mathbf{x} a uma resposta \mathbf{d} . Então, na ausência de \mathbf{d} , \mathbf{x} pode produzir uma saída \mathbf{y} que é similar a \mathbf{d} .

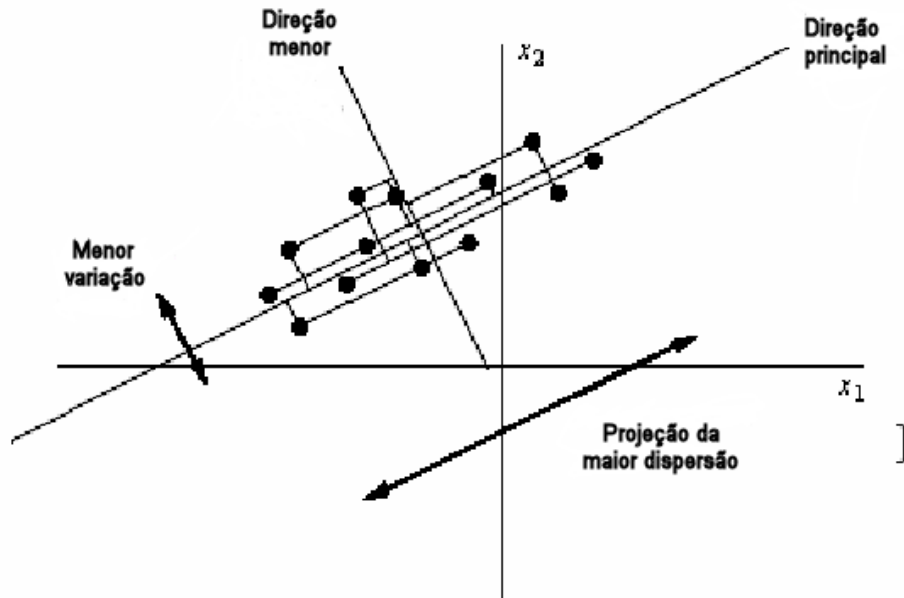


Figura 36 – Projeção de um agrupamento nas componentes principais.

As topologias da LAM e do problema do regressor linear são semelhantes. Isto tem algumas implicações. Se observarmos a relação entre a quantidade de padrões na camada de entrada veremos que na LAM têm-se mais equações que dados, enquanto que no regressor tem-se mais dados que equações.

Portanto, a rede linear pode memorizar (trabalhando como uma LAM) ou generalizar as propriedades estatísticas dos pares entrada-saída (trabalhando como um regressor). Uma função de aproximação com um pequeno número de exemplos (para regressão não-linear ou classificação) corresponde na realidade a uma memória associativa e, portanto, não generaliza bem.

7. Rede de Kohonen e Rede ART

Na natureza, a competição por recursos representa uma maneira de diversificar e otimizar a função dos elementos de um sistema distribuído, conduzindo à otimização a nível local sem controle global para assinalar recursos do sistema [Haykin, 1999]. Da mesma forma, os neurônios de redes competitivas recebem informação idêntica das entradas mas competem pelos recursos através de conexões laterais na topologia, ou através da formulação da regra de aprendizagem, especializando-se em áreas diferentes da entrada. A competição é intrinsecamente uma operação não-linear e, portanto, o tratamento matemático não está tão desenvolvido como em outras áreas de sistemas adaptativos.

Existem dois tipos básicos de competição: a competição forte (“hard”), na qual somente um neurônio ganha os recursos, e a competição fraca (“soft”), na qual há um vencedor, mas seus vizinhos também recebem uma parte dos recursos.

A rede competitiva mais simples é chamada de rede o-vencedor-leva-tudo (“winner-takes-all”), a qual consiste de uma camada de neurônios, todos com mesma entrada. O neurônio com melhor saída é o vencedor. Na rede o-vencedor-leva-tudo não há controle global. Se houverem N entradas, a saída será dada por:

$$y_k = \begin{cases} 1, & \text{para o maior } x_k \\ 0, & \text{para os demais} \end{cases} \quad (48)$$

Na rede o-vencedor-leva-tudo há realimentação lateral entre os neurônios competitivos e a saída leva algum tempo até estabilizar, conforme a figura 37. Embora a amplitude da diferença na entrada possa ser pequena, a saída é bem definida.

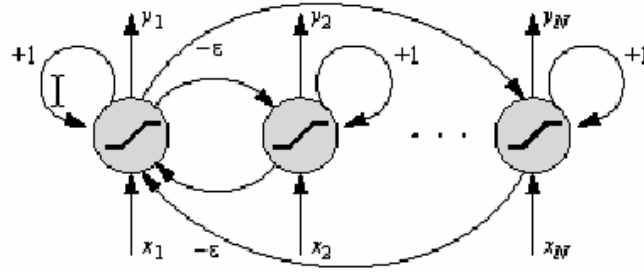


Figura 37 – Rede o-vencedor-leva-tudo.

O objetivo da aprendizagem competitiva é criar uma regra de aprendizagem que possa ser aplicada a uma topologia de camada única e que atribua os neurônios a áreas diferentes do espaço de entrada. A aprendizagem competitiva é um paradigma de aprendizagem não-supervisionada, a qual extrai informação somente dos padrões de entrada sem a necessidade de uma resposta desejada.

Grossberg propôs uma rede como um único neurônio de McCulloch-Pitts treinado com a regra estrela (“instar rule”) [Grossberg, 1982]:

$$w_{ij}(n+1) = w_{ij}(n) + \eta y_i(n)(x_j(n) - w_{ij}(n)) \quad (49)$$

na qual $y(n)$ é a saída, limitada aos valores zero e um, ou seja, o peso é atualizado somente se o neurônio está ativo (figura 38).

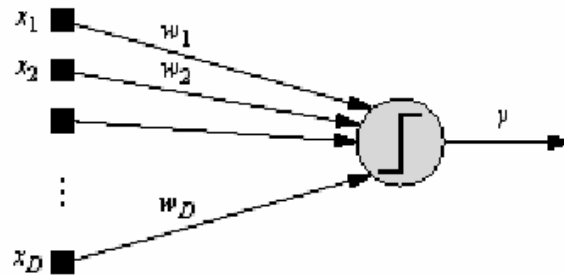


Figura 38 – Rede Instar de Grossberg.

Na rede Instar, quando o neurônio está ativo, a rede move os pesos na direção da entrada em uma linha direta proporcional ao tamanho de η . Um neurônio treinado com a regra instar fornece saída 1 para exemplos próximos do padrão de treinamento (com o bias controlando a vizinhança), ou seja, é capaz de reconhecer se um vetor padrão é “similar” à uma classe armazenada nos pesos (figura 39).

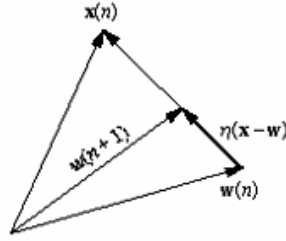


Figura 39 – Movimento dos pesos na regra instar.

Verifica-se aqui que regra *instar* é similar à regra de Hebb e à de Oja:

$$\text{Regra de Hebb : } \Delta w = \eta y x$$

$$\text{Regra de Oja : } \Delta w = \eta (yx - y^2 w) \quad (50)$$

$$\text{Regra instar : } \Delta w = \eta (yx - yw)$$

Como acontece na regra de Oja, o segundo termo da regra instar evita a instabilidade presente na regra de Hebb. Usando a regra instar com um neurônio não-linear, padrões não-freqüentes são preservados, enquanto o neurônio estiver inativo. A regra competitiva torna-se então

$$\mathbf{w}_{i^*}(n+1) = \mathbf{w}_{i^*}(n) + \eta [\mathbf{x}(n) - \mathbf{w}_{i^*}(n)] \quad (51)$$

na qual i^* é o neurônio que venceu a competição. Todos os outros neurônios mantêm seus pesos anteriores. O tamanho do passo η ($0 < \eta < 1$) controla o tamanho da atualização em cada iteração.

O neurônio mais próximo à entrada atual deve vencer a competição, portanto precisa-se de uma medida de proximidade. Usando-se para tanto o produto interno, deve-se levar em conta que o mesmo é sensível não somente às direções mas também ao comprimento dos vetores, portanto a entrada e os pesos devem ser normalizados. Uma alternativa ao produto interno é o uso da distância Euclidiana como métrica para definir o vencedor:

$$vencedor = \max_i (\mathbf{w}_i^T \mathbf{x}) \quad \text{ou} \quad vencedor = \min_i (\|\mathbf{x} - \mathbf{w}_i\|^2) \quad (52)$$

Como a raiz quadrada é uma função computacionalmente cara, a distância métrica é menos eficiente que o produto interno:

$$\|\mathbf{x} - \mathbf{w}\| = \sqrt{\sum_k (x_k - w_k)^2} \quad (53)$$

As vezes a métrica de Manhattan é usada, pois só envolve subtrações e valores absolutos.

Usando a regra competitiva, uma rede linear de camada única agrupa e representa dados que residem em uma vizinhança do espaço de entrada. Cada vizinhança é representada por um único neurônio. Os pesos de cada neurônio representam pontos no espaço de entrada chamados vetores protótipos. Se os vetores forem unidos por uma linha e forem traçadas perpendiculares na metade de cada uma, as mesmas se encontrarão e

formarão uma estrutura semelhante a uma colméia de abelhas. Esta estrutura é chamada de tesselação de Voronoi (figura 40).

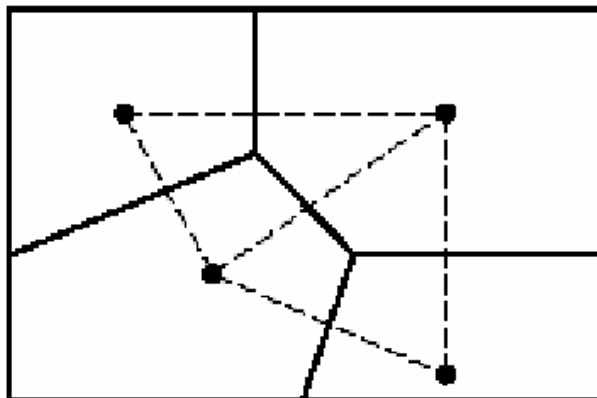


Figura 40 – Tesselação de Voronoi.

Na tesselação de Voronoi, as amostras de dados que estão nas regiões são assinaladas aos correspondentes vetores-protótipo. Disto observa-se que agrupamento é uma transformação de contínuo para discreto. Do ponto de vista teórico, agrupamento é uma forma de estimação não-paramétrica de densidade.

O algoritmo não-neural típico de agrupamento é o k-médio, o qual encontra a melhor divisão de N amostras em k grupos, tal que a distância total entre as amostras agrupadas e seus respectivos centros, isto é, a variância total, seja minimizada. As redes competitivas implementam uma versão on-line do agrupamento k-médio em vez das operações de adaptação batch.

Agrupamento é o processo de agrupar amostras de entradas que são vizinhos espaciais, sendo um processo não-supervisionado. Já a classificação consiste na rotulação de amostras de entrada através de algum critério externo, sendo um processo supervisionado.

Como agrupamento é não-supervisionado, ele não pode ser usado diretamente para classificação. Em várias aplicações práticas, os dados de cada classe tendem a ser densos e, portanto, há um vale natural entre as classes. Nestes casos o agrupamento pode ser utilizado como um pré-processador para a classificação. Com isto obtém-se redes de classificação mais simples.

A competição fraca (“soft competition”) cria uma “bolha” de atividade no espaço de saída, onde o neurônio mais próximo é o mais ativo e seus vizinhos são menos ativos. Uma rede softmax pode ser criada usando realimentação lateral, onde os pesos laterais variam com a distância dos neurônios conectados. A competição fraca cria uma relação de vizinhança entre neurônios, isto é, eles ligam-se por uma métrica de similaridade. Isto implica que mapeamentos topológicos do espaço de entrada para o espaço dos neurônios são possíveis.

A rede SOM (“Self-Organizing Map”) de Kohonen realiza um mapeamento de um espaço contínuo de entrada para um espaço discreto de saída, onde as propriedades topológicas da entrada são preservadas. A rede SOM de Kohonen é uma rede linear de

camada única totalmente conectada, cuja saída é organizada geralmente em uma ou duas dimensões, conforme a figura 41. Quando a SOM se adapta a entradas de altas dimensões, ela deve se estender e enrolar para cobrir o espaço de entrada (figura 42).

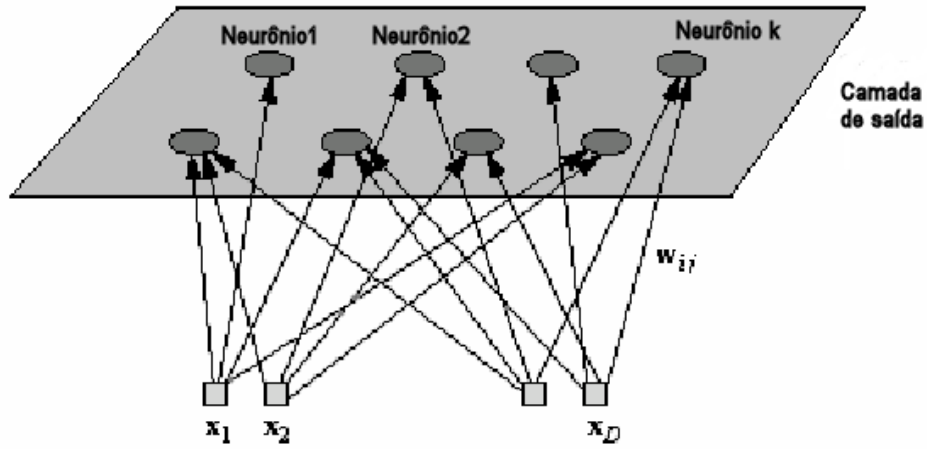


Figura 41 – Rede SOM de Kohonen com saída em duas dimensões.

O algoritmo SOM de aprendizagem assume que a rede de inibição lateral produz uma distribuição gaussiana centrada no neurônio vencedor. Como aplica-se a regra de aprendizagem do tipo instar, que escala a regra competitiva pela atividade de saída de cada neurônio, a regra competitiva SOM de Kohonen torna-se

$$\mathbf{w}_i(n+1) = \mathbf{w}_i(n) + \Lambda_{i,i^*}(n)[\mathbf{x}(n) - \mathbf{w}_i(n)] \quad (54)$$

na qual a função Λ_{i,i^*} é uma função de vizinhança centrada no neurônio vencedor. Normalmente, tanto o tamanho do passo quanto a vizinhança diminuem com o tempo, e a função de vizinhança Λ é normalmente uma gaussiana:

$$\Lambda_{i,i^*}(n) = \exp\left(\frac{-d_{i,i^*}^2}{2\sigma^2(n)}\right) \quad (55)$$

com uma variância que decresce com a iteração. Inicialmente ela cobre todo o mapa, mas reduz-se progressivamente a uma vizinhança de zero, isto é, somente o neurônio vencedor é atualizado. À medida que a vizinhança é reduzida, a rede move-se de uma competição “muito fraca” (na qual quase todo neurônio é atualizado) para uma competição “forte” (na qual somente o neurônio vencedor é atualizado).

Há evidências que a rede SOM cria um espaço de saída discreto onde relações topológicas dentro das vizinhanças do espaço de entrada são preservadas. A rede SOM é criada de uma maneira não-supervisionada e a seleção de parâmetros é crucial para a preservação de topologia.

A experiência mostrou que há duas fases na aprendizagem SOM: a) Fase de ordenação topológica dos pesos, ou seja, definição das vizinhanças; e b) Fase de convergência com o ajuste fino da distribuição de entrada.

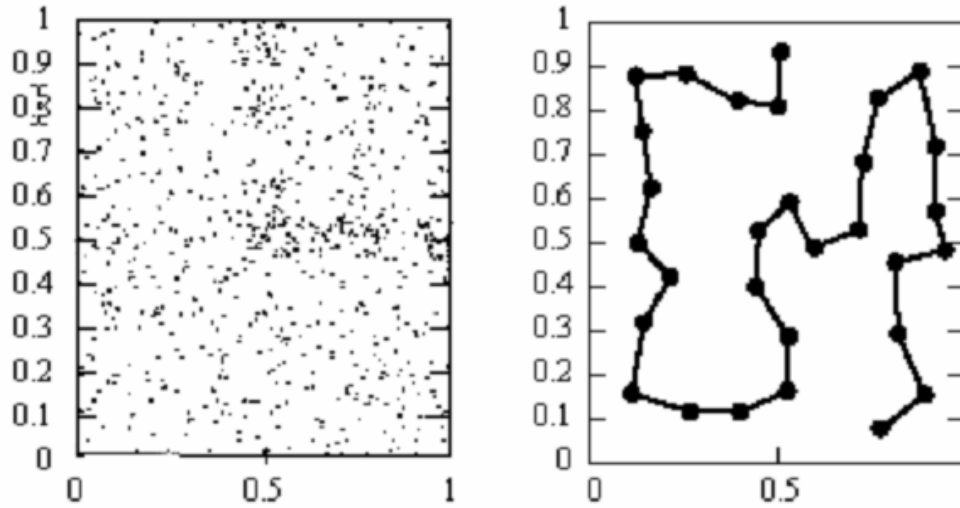


Figura 42- Saída unidimensional e bidimensional para uma rede SOM de Kohonen.

Na fase de ordenação topológica dos pesos, suposta tendo N_o iterações, a função de vizinhança decresce, em geral linearmente, com um raio definido por:

$$\sigma(n) = \sigma_0 \left(1 - \frac{n}{N_o} \right) \quad (56)$$

Normalmente a taxa de aprendizagem é alta (acima de 0.1) para permitir à rede se auto-organizar. Ela também é linearmente ajustada através da fórmula

$$\Delta\eta(n) = \eta_0 \left(1 - \frac{n}{N + K} \right) \quad (57)$$

na qual η_0 é a taxa de aprendizagem inicial e K é a taxa final de aprendizagem.

A fase de convergência é a fase mais demorada, onde se mantém uma taxa de aprendizagem pequena (0.01) e usa-se a menor vizinhança (somente o neurônio vencedor ou seus vizinhos mais próximos são atualizados). A escolha do número de neurônios é feita experimentalmente. O número de saídas afeta a precisão do mapeamento e o tempo de treinamento. O aumento do número de neurônios atualizados aumenta a resolução mas aumenta em muito o tempo de treinamento.

As principais propriedades da rede SOM são as seguintes:

- a) Aproximação do Espaço de Entrada: A rede SOM é capaz de preservar a estrutura do espaço de entrada relativamente bem;
- b) Ordenamento Topológico: Os neurônios na saída da rede SOM estão topologicamente ordenados no sentido de que neurônios vizinhos correspondem a regiões similares no espaço de entrada;
- c) Manutenção da densidade: Regiões no espaço de entrada com maior densidade de pontos são mapeadas para regiões maiores no espaço de saída.

É possível criar classificadores a partir de redes competitivas. Uma das redes mais conhecidas neste sentido são as redes do tipo LVQ (“Learning Vector Quantization”). Este tipo de rede faz uso de informações de classes para ajustar os limites das regiões de

tesselação. Se a regra competitiva produz a saída certa, ela não causará alteração; entretanto, se a saída estiver errada, os pesos dos neurônios devem ser “repelidos” do agrupamento atual, pela regra

$$\Delta w_{i^*j} = \begin{cases} \eta(x_j - w_{i^*j}), & \text{para classe correta} \\ -\eta(x_j - w_{i^*j}), & \text{para classe incorreta} \end{cases} \quad (58)$$

Esta regra apresenta problemas se os dados de entrada forem multimodais. Kohonen propôs uma série de redes (LVQ1, LVQ2, LVQ3 e outras) que são capazes de trabalhar melhor com dados multimodais [Kohonen, 1995].

Grossberg também propôs a rede outstar, que pode associar uma entrada escalar a um vetor de saída para recuperação de padrões (figura 43). A regra de aprendizagem é obtida alterando os papéis das entradas com a saída, isto é, fazendo-se

$$\mathbf{w}_{ij}(n+1) = \mathbf{w}_{ij}(n) + \eta x_j(n)(y_i(n) - w_{ij}(n)) \quad (59)$$

A rede instar-outstar mapeia dados de entrada em padrões de saída. Esta arquitetura pode ser usada para descoberta de grupos se a resposta desejada torna-se a entrada, $d(n)=x(n)$ e a saída do sistema torna-se a camada competitiva. O maior problema é a instabilidade devido à aprendizagem instar e recursos limitados. A partir desta dificuldade surgiu o que ficou conhecido como Dilema Estabilidade-Plasticidade (D-E-P): “Como fazer um sistema se adaptar, evitando-se simultaneamente que as entradas atuais destruam o treinamento passado”. Para resolver o D-E-P pode-se usar a porção outstar da rede e exigir que a entrada atual esteja a uma distância pré-especificada do centro do grupo.

Grossberg criou um método para adicionar novos neurônios à camada competitiva quando não há “ressonância” entre o dado atual e os grupos existentes. Neste método, a especificação da vizinhança é controlada por um parâmetro de vigilância. Esta rede é chamada de ART (“Adaptive Resonance Theory”). Ela é capaz de se dimensionar para realizar o agrupamento de dados e pode ser usada para detectar padrões não vistos anteriormente, quando cria novos grupos após o período de aprendizagem [Braga, 2000]. Entretanto, o ajuste do parâmetro de vigilância é delicado.

8. Extração do Conhecimento de Redes Neurais Treinadas

8.1. Integração entre Conhecimento Neural e Simbólico

Neurocomputação é um dos termos continuamente ligados às redes neurais artificiais. Sua ênfase está no uso e representação de conhecimento específico do problema. Aqui aplica-se a máxima de que “Conhecimento é poder”. A modelagem explícita do conhecimento representado por um sistema neurcomputacional continua sendo um dos principais temas de pesquisa hoje. A neurocomputação baseada em conhecimento diz respeito a métodos e técnicas que trabalham com a representação e processamento explícito de conhecimento onde um sistema de neurocomputação está envolvido. Historicamente ela está inspirada na IA simbólica e nas redes neurais artificiais [Fu, 1993].

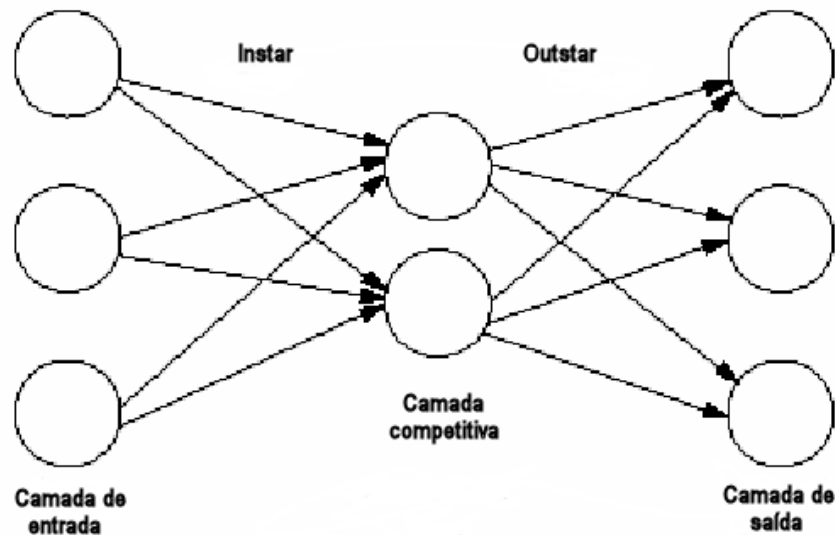


Figura 43 – Rede outstar de Grossberg.

Existem basicamente três tipos de arquitetura na área:

- a) Abordagem Híbrida: Neste caso, os módulos neural e simbólico são componentes distintos, partilhando ou transferindo conhecimento entre si;
- b) Abordagem Unificada: Aqui o conhecimento é modelado usando conexões locais e/ou distribuídas entre os neurônios;
- c) Abordagem Translacional: Constitui-se um meio termo entre as abordagens anteriores.

A abordagem tradicional que envolve uma representação explícita do conhecimento é a de Sistemas Especialistas. Os sistemas especialistas têm como objetivo a representação e uso de grandes quantidades de conhecimento, assegurando a sua integridade, consistência e exploração efetiva. Eles realizam uma tarefa de tomada de decisão complexa dentro de um domínio de problema bem específico.

A arquitetura de sistemas especialistas é bastante modular sendo composta, geralmente, pelos seguintes módulos: Base de Conhecimentos, Base de Fatos, Máquina (ou Motor) de Inferência, Mecanismo de Explicação e pela Interface com o Usuário. Nos sistemas especialistas o usuário descreve o problema de maneira interativa, e a partir daí o sistema deve inferir uma solução, mesmo com informação incompleta ou imprecisa. Ele também deve fornecer ao usuário alguma explicação de suas conclusões para justificar suas inferências (explicações normalmente designadas COMO? e POR QUÊ?).

Uma comparação entre as abordagens dos sistemas especialistas (também chamados de sistemas baseados em regras) com os sistemas neurais envolve os seguintes elementos: tamanho da tarefa, aquisição e edição do conhecimento, considerações sobre matching parcial, considerações sobre informação incompleta e sobre a capacidade de explicação.

Existem várias estratégias possíveis para a integração de conhecimento neural e simbólico. Dentre os sistemas híbridos, podemos ter várias abordagens, tais como:

- a) Dividir-e-conquistar: Consiste no particionamento do problema em unidades menores, utilizando-se em cada subunidade o paradigma mais apropriado. Um exemplo de tal

sistema foi desenvolvido para escalonamento de caminhões de transporte de cargas [Caudill, 1991];

b)RNA embutida: Neste caso a rede neural faz parte de um sistema baseado em regras. A rede neural pode ser utilizada, por exemplo, para a avaliação das condições de algumas regras, ou ainda para realizar algumas atividades da máquina de inferência tradicional. Alguns exemplos de sistemas desenvolvidos baseados nesta forma de integração são o sistema COLE (“COnectionist Logic programming Environment”) [Kasabov and Petkov, 1992], o sistema COPE (“COnectionist Production systems Environment”) [Kasabov, 1993] e o sistema FEL-EXPERT [Mařík et al., 1992];

c)Implementação neural de conhecimento explícito: A idéia principal nesta estratégia é a de criar a rede neural utilizando-se de conhecimento explícito acerca do problema, ou seja, ao se definir a rede neural que será treinada, os neurônios representarão conceitos, proposições etc. e as conexões entre os mesmos representarão relações causais e dependências lógicas. Um exemplo de sistemas baseados neste princípio é o sistema RUBICON [Samad, 1988];

d)Incorporação de regras em redes neurais treinadas: Parte-se do princípio que há uma rede neural treinada para a solução do problema e incorpora-se conhecimento explícito à mesma [Mazný, 1995];

e)Extração de regras a partir de redes neurais treinadas: Uma rede neural pode ser treinada, a partir do conjunto de dados disponíveis, para resolver o problema e então analisada para a obtenção de um conjunto de regras. Esta é uma abordagem que pode reduzir significativamente o tempo de desenvolvimento de um sistema especialista. Alguns exemplos desta abordagem são o algoritmo VIA (“Validity Interval Analysis”) [Thrun, 1995] e o sistema TREPAN [Craven and Shavlik, 1996]. É possível também utilizar uma abordagem evolucionária para a geração do melhor conjunto de regras, em termos de precisão e compreensibilidade, a partir de redes neurais artificiais [Santos et al., 2000].

Muitas vezes utiliza-se a denominação de sistemas neurais especialistas para alguns dos sistemas neurocomputacionais que buscam diminuir as desvantagens da representação implícita do conhecimento, introduzindo heurísticas. Nestes casos, a rede neural é “enriquecida” com outras funcionalidades para ter as características de sistemas especialistas. Alguns exemplos de sistemas neurais especialistas são o sistema MACIE (“MAtrix Controlled Inference Engine”) [Gallant, 1993] e o sistema EXPSYS [Šima, 1995].

Uma das possibilidades de desenvolvimento de sistemas neurocomputacionais baseados na abordagem unificada consiste no desenvolvimento de arquiteturas que permitam a criação de redes neurais artificiais a partir de um conjunto de regras simbólicas, tendo em vista que o conhecimento simbólico sobre determinados domínios freqüentemente está disponível na forma de regras de produção [Towell, 1994]. Desta forma, conhecimento a priori sobre uma determinada aplicação pode ser embutido na etapa de definição inicial de uma rede neural, antes do treinamento.

Um algoritmo que realiza esta tarefa é o chamado VL₁ANN [Cloete and Zurada, 2000]. Este algoritmo aceita regras que obedecem às convenções de VL₁ [Theron and Cloete, 1996], sendo compostos por dados numéricos contínuos e dados nominais como atributos de entrada. A rede neural gerada é do tipo avante (sem conexões recorrentes), a qual pode ser treinada usando algoritmos tradicionais como o de retro-propagação na etapa

posterior à geração da mesma, permitindo o refinamento da representação do conhecimento inicial. O algoritmo VL₁ANN está descrito na figura 44.

1. Codificar variáveis de entrada como valores reais (numéricos)
2. Para cada regra fazer
 1. Para cada átomo da regra fazer
 1. Codificar o átomo como nova unidade relacional na camada 1 conectada à entrada na camada 0
 2. Conectar todas as unidades relacionais da regra a uma nova unidade AND na camada 2
3. Conectar todas as unidades AND representando regras com a mesma conclusão a uma nova unidade OR na camada 3
4. Para cada variável de saída *Attr* que não esteja ortogonalmente codificada fazer
 1. Mapear todas as unidades OR de *Attr* na camada 3 à uma única nova unidade de saída na camada 4

Figura 44 – Algoritmo VL₁ANN para representação de regras em redes neurais avante.

A sequência de atividades para o desenvolvimento de arquiteturas translacionais consiste em obter conhecimento simbólico, ou seja, uma forma estruturada de descrição do problema, a seguir traduzir o conhecimento em uma rede neural artificial, treiná-la para revisar e refinar o conhecimento embutido na mesma, extrair o conhecimento simbólico da rede neural treinada e finalmente refinar o conhecimento simbólico para posterior uso.

As principais características dos métodos translacionais são as seguintes:

- a) Tipo de representação do conhecimento prévio e final, o qual pode ser sob a forma de regras para o caso de redes neurais em avanço, sob a forma de autômatos para o caso de redes neurais recorrentes, sob a forma de grafos direcionados para a representação de neurônios recursivos, ou ainda sob a forma de árvores de decisão;
- b) Restrições baseadas na arquitetura: Como a arquitetura deve mapear a representação estruturada desejada, deve-se levar em conta o número de camadas, que define o nível requerido para mapear regras à topologia da rede neural, os tipos de função de ativação, que em geral são a sigmóide ou então baseada em fatores de certeza, os pesos, que podem estar na faixa $\{-1,0,1\}$ ou $[-1,1]$, e o tipo de entradas: $\{0,1\}$ ou $\{-1,1\}$;
- c) Método de treinamento: Em função das restrições nos parâmetros durante treinamento tem-se pesos fixos ou em uma faixa de valores, podem ocorrer modificações na topologia para adicionar ou retirar unidades. Além disto, pode haver modificação da função de aprendizagem através do uso de termos de regularização que criam penalidade para obter pesos na faixa desejada;
- d) Os métodos de extração do conhecimento podem ser baseados em requisitos de treinamento especializado e uma arquitetura de rede neural restrita, ou dirigidos a uma rede neural genérica, sem restrições com relação ao tipo de treinamento a ser efetuado sobre o conhecimento simbólico embutido.

A técnica de extração de regras consiste na tarefa de converter modelos de redes neurais treinadas em representações mais facilmente compreensíveis pelos seres humanos. Estas técnicas podem ser agrupadas em três tipos: a)decomposicional, na qual a extração ocorre ao nível de associações escondidas e de saída; b)pedagógica, na qual a rede neural é vista como “caixa preta”, e a extração de regras ocorre sobre arquiteturas sem restrições; e c)eclética, que é uma composição das duas anteriores.

Os métodos de extração de regras são categorizados pela sua:

- a)Compreensibilidade: Indica o quanto são compreensíveis por seres humanos;
- b)Fidelidade: A que nível as regras modelam a rede neural da qual foram extraídas;
- c)Precisão: Indica o desempenho sobre exemplos não vistos;
- d)Escalabilidade: Demonstra como evolui o tempo de extração em relação a grandes espaços de entrada, unidades e conexões;
- e)Generalidade: Necessidade de treinamento especial ou restrições.

Uma etapa fundamental da extração de regras é baseada nos métodos de busca [Luger, 1993]. Para tanto, um dos primeiros algoritmos utilizados foi o SUBSET (figura 45). Uma das características do algoritmo SUBSET é a de que ele extrai regras dos neurônios das camadas intermediárias e de saída, buscando subconjuntos de pesos para cada neurônio cuja soma supera o limiar. Isto gera um grande número de possibilidades, o que implica em um processamento excessivo e o resultado implica em regras com grande número de antecedentes.

Um exemplo de uma rede neural e das regras extraídas da mesma usando o algoritmo SUBSET estão na figura 46.

Um algoritmo bastante poderoso para a extração de conhecimento é o TREPAN (“TREes Parroting Networks”) [Craven and Shavlik, 1996]. Este algoritmo independe da topologia da rede, fornecendo uma árvore de decisão como representação do conhecimento. Para atingir isto ele usa o método do melhor primeiro para realizar a busca [Russel et Norvig, 1995]. A classe de cada exemplo é definida por um oráculo (sendo utilizada para tanto a própria rede neural) e ele faz uso de exemplos complementares para garantir um número mínimo de exemplos.

Para cada neurônio da camada escondida e intermediária fazer

Formar S_p subconjuntos, combinando somente pesos positivos cujo somatório supera o limiar

Para cada elemento P dos subconjuntos S_p fazer

Formar S_n subconjuntos de N elementos, considerando as combinações mínimas de pesos negativos, tal que a soma absoluta destes pesos seja maior que a soma de P menos o valor do limiar

Formar a regra: Se P AND não N então neurônio

Figura 45 – Descrição do algoritmo SUBSET.

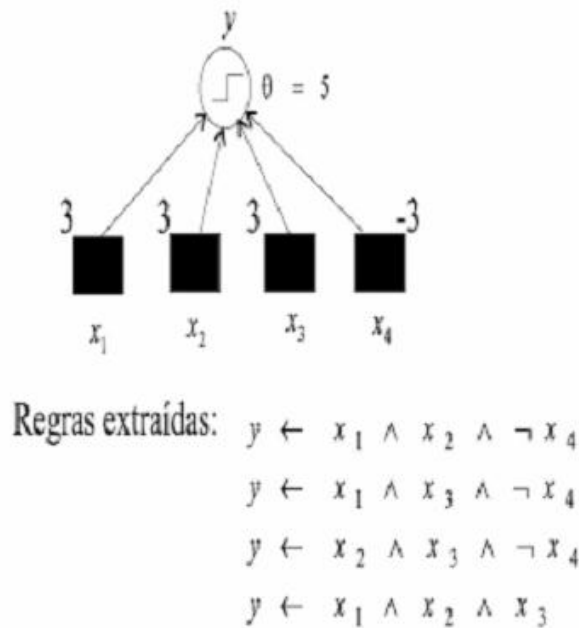


Figura 46 – Rede neural treinada e regras extraídas usando o algoritmo SUBSET.

9. Links

A seguir são fornecidos alguns links para endereços relativos às redes neurais artificiais.

1) Adaptive Resonance Theory clearinghouse, by Daniel Tauritz:

<http://web.umr.edu/~tauritzd/art/index.html>

b) Artificial Neural Networks Technology:

http://www.dacs.dtic.mil/techs/neural/neural_ToC.html

c) Applets for Neural Networks and Artificial Life:

<http://neuron.eng.wayne.edu/>

d) Arquivo contendo vários artigos e teses sobre Redes Neurais Artificiais:

<ftp://archive.cis.ohio-state.edu/pub/neuroprose/>

e) Homepage do Prof. Rudy Setiono:

<http://www.comp.nus.edu.sg/~rudys/>

f) EasyNNplus – Um simulador grátis de Redes Neurais Artificiais:

<http://www.easynn.com/easynnplus.html>

g) International Neural Network Society – Special Interest Group Brazil:

http://www.cpdee.ufmg.br/~apbraga/siginns_brazil.html

h) Redes SOM e LVQ:

<http://www.cis.hut.fi/research/som-research/nncr-programs.shtml>

i) Implementation of a Fast Artificial Neural Network Library in C:

<http://fann.sourceforge.net/report/report.html>

j) Java Applets for Neural Network and Artificial Life:

<http://staff.aist.go.jp/utsugi-a/Lab/Links.html>

k) JOONE - Java Object Oriented Neural Engine, um simulador gratis em Java:

<http://www.jooneworld.com/>

l) LENS – The light, efficient network Simulator, um simulador de redes neurais artificiais:

<http://tedlab.mit.edu/~dr/Lens/>

m) LIP6 - Connexionnist Home Page:

<http://www-connex.lip6.fr/index.php>

n) Neural Computing Research Group:

<http://www.ncrg.aston.ac.uk/GTM/>

o) Neural Nets by Kevin Gurney:

<http://www.shef.ac.uk/psychology/gurney/notes/index.html>

p) Evolutionary Design of Neural Networks:

<http://www.cs.iastate.edu/~gannadm/homepage.html>

q) Neural Networks at your Fingertips:

<http://www.geocities.com/CapeCanaveral/1624/>

r) Neural Networks Tool – Nenet:

<http://koti.mbnet.fi/~phodju/nenet/Nenet/General.html>

s) NeuroDimension, Inc.:

<http://www.nd.com/>

t) Neuronale Netze (página com muitos links sobre redes neurais artificiais):

http://www.informatik.uni-freiburg.de/~heinz/nn_page.html

u) SNNS – um excelente simulador de redes neurais artificiais grátis:

<http://www-ra.informatik.uni-tuebingen.de/SNNS/>

10. Referências Bibliográficas

Jain, A. K., Mao, J., and Mohiuddin, K. M., “Artificial Neural Networks: A Tutorial”, *IEEE Computer*, pp. 31-44, March 1996.

Almeida, L. B., Langlois, T., and Amaral, J. D., On-Line Step Size Adaptation, *Technical Report INESC RT07/97*, 1997.

Aspvall, B., and Stone, R.E., “Khachiyan’s Linear Programming Algorithm”, *Journal of Algorithms*, 1, 1980, pp. 1-13.

Bender, E. A., *Mathematical Methods in Artificial Intelligence*, IEEE Computer Society Press, 1996.

Bishop, C. M., *Neural Networks for Pattern Recognition*, Clarendon Press, Oxford, 1995.

Braga, A. P., Ludermir, T. B., and Carvalho, A. C. P. L. F., *Redes Neurais Artificiais: Teoria e Aplicações*, LTC – Livros Técnicos e Científicos Editora S.A., 2000.

Brunak, S., and Lautrup, B., *Neuronale Netze: Die Nächste Computer-Revolution*, Carl Hanser Verlag, 1993.

Caudill, M., Expert Networks, *Byte*, 16(10):108-116, 1991.

Caudill, M., and Butler, C., *Understanding Neural Networks: Computer Explorations – Volume 1: Basic Networks & Volume 2: Advanced Networks*, The MIT Press, 1992.

Chen, C.-T., *Linear System Theory and Design*, CBS College Publishing, 1984.

Cloete, I., and Zurada, J. M. (Eds.), *Knowledge-Based Neurocomputing*, The MIT Press, 2000.

Cohen, P. R., *Empirical Methods for Artificial Intelligence*, The MIT Press, 1995.

Craven, M. W., and Shavlik, J. W., Extracting Tree-Structured Representations of Trained Networks, *Advances in Neural Information Processing Systems*, 8, 1996.

- Fausett, L., *Fundamentals of Neural Networks – Architectures, Algorithms, and Applications*, Prentice Hall, Inc., 1994.
- Fiesler, E., “Classification and Formalization of Artificial Neural Networks”, in *Computer Standards and Interfaces*, Vol. 16, John Fulcher (Ed.), Elsevier Sciences Publishers B.V., Amsterdam, The Netherlands, 1994.
- Fu, L. M., Knowledge-Based connectionism for revising domain theories, *IEEE Transactions on Systems, Man and Cybernetics*, 23(1):173-182, 1993.
- Gallant, S. I., *Neural Network Learning and Expert Systems*, The MIT Press, 1993.
- Golden, R. M., *Mathematical Methods for Neural Network Analysis and Design*, The MIT Press, 1995.
- Grossberg, S., *Studies of Mind and Brain*, Reidel Publishing Company, Boston, 1982.
- Hassoun, M. H., *Fundamentals of Artificial Neural Networks*, The MIT Press, 1995.
- Haykin, S., *Neural Networks – A Comprehensive Foundation*, 2nd. Edition, Prentice Hall, Inc., 1999.
- Hebb, D., *The Organization of Behavior: A Neurophysiological Theory*, John Wiley & Sons, 1949.
- Hinton, G., and Sejnowski, T. J. (Eds.), *Unsupervised Learning: Foundations of Neural Computation*, The MIT Press, 1999.
- Hoffmann, N., *Kleines Handbuch Neuronale Netze – Anwendungsorientiertes Wissen zum Lernen und Nachschlagen*, Friedr. Vieweg & Sohn Verlagsgesellschaft mbH, 1993.
- Jacobs, R. A., Increased Rates of Convergence Through Learning Rate Adaptation, *Neural Networks*, 1, pp. 295-307, 1988.
- Kasabov, N. K., and Petkov, S. H., Approximate Reasoning with Hybrid Connectionist Logic Programming Systems, *Proceedings of the International Conference on Artificial Neural Networks ICANN'92*, eds. I. Aleksander and J. Taylor, pp. 749-752, Elsevier Science Publisher B.V., 1992.
- Kasabov, N. K., Hybrid Connectionist Production Systems: An Approach to Realising Fuzzy Expert Systems, *Journal of Systems Engineering*, 1:15-21, 1993.
- Kinnebrock, W., *Neuronale Netze – Grundlagen, Anwendungen, Beispiele*, R. Oldenbourg Verlag, GmbH, 1992.
- Kohonen, T., *Self-Organizing Maps*, Springer-Verlag, New York, 1995.
- Kratzer, K. P., *Neuronale Netze – Grundlagen und Anwendungen*, Carl Hanser Verlag, 1991.
- Luger, G. F., and Stubblefield, W. A., *Artificial Intelligence – Structures and Strategies for Complex Problem Solving*, 2nd. Edition, The Benjamin/Cummings Publishing Company, Inc., 1993.
- Mazný, M. *Integrating Rule-Based and Neural Approaches to Expert System Design*, Master Thesis, Department of Software Engineering, Faculty of Mathematics and Physics, Charles University, Prague, 1995.

- McCulloch, W.S., Pitts, W., "A Logical Calculus of the Ideas Immanent in Nervous Activity", *Bulletin of Mathematical Biophysics*, Vol. 5, pp. 115-133, 1943.
- Mehrotra, K., Mohan, C. K., and Ranka, S., *Elements of Artificial Neural Networks*, The MIT Press, 1997.
- Michalewicz, Z., and Fogel, D. B., *How to Solve It: Modern Heuristics*, Springer-Verlag, 2000.
- Minsky, M.L., Papert, S.A., *Perceptrons*, Cambridge, MA : MIT Press, 1969.
- Mitchell, T. M., *Machine Learning*, WCB McGraw-Hill, 1997.
- Oja, E., A Simplified Neuron Model as a Principal Component Analyzer, *Journal of Mathematical Biology*, 15:239-245, 1982.
- Pham, D.T., and Karaboga, D., *Intelligent Optimisation Techniques – Genetic Algorithms, Tabu Search, Simulated Annealing and Neural Networks*, Springer-Verlag, 2000.
- Príncipe, J. C., Euliano, N. R., and Lefebvre, W. C., *Neural and Adaptive Systems – Fundamentals Through Simulations*, John Wiley & Sons, Inc., 2000.
- Reed, R. D., and Marks III, R. J., *Neural Smithing – Supervised Learning in Feedforward Artificial Neural Networks*, The MIT Press, 1999.
- Rosenblatt, F., "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain", *Psychological Review*, Vol. 65, pp. 386-408, 1956.
- Rumelhart, D.E., Hinton, G.E., Williams, R.J., "Learning Representations of Back-Propagation Errors", *Nature (London)*, Vol. 323, pp. 533-536, 1986.
- Russel, S., and Norvig, P., *Artificial Intelligence – A Modern Approach*, Prentice-Hall, Inc., 1995.
- Samad, T., Towards Connectionist Rule-Based Systems, *Proceedings of the IEEE International Conference on Neural Networks*, San Diego, Vol. II, pp. 525-532, 1988.
- Santos, R. T., Nievola, J. C., and Freitas, A. A., Extracting Comprehensible Rules from Neural Networks via Genetic Algorithms, *2000 IEEE Symposium on Combinations of Evolutionary Computation and Neural Networks*, pp. 130-139, 2000.
- Šima, J., Neural Expert Systems, *Neural Networks*, 8(2):261-271, 1995.
- Stanley, J., and Bate, E., *Neuronale Netze – Computersimulation biologischer Intelligenz*, Systhema Verlag GmbH, 1991.
- Sundararajan, N., Saratchandran, P., and Wei, L. Y., *Radial Basis Function Neural Networks with Sequential Learning – MRAN and Its Applications*, World Scientific Publishing Co. Pte. Ltd., 1999.
- Theron, H. and Cloete, I., BEXA: A covering algorithm for learning propositional concept descriptions, *Machine Learning*, 24:5-40, 1996.
- Thrun, S., Extracting Rules from Artificial Neural Networks with Distributed Representation, *Advances in Neural Information Processing Systems*, 7, 1995.

Towell, G. G., and Shavlik, J. W., Knowledge-based artificial neural networks, *Artificial Intelligence*, 70:119-165, 1994.

Vonk, E., Jain, L.C., and Johnson, R.P., *Automatic Generation of Neural Network Architecture Using Evolutionary Computation*, World Scientific Publishing Co. Pte. Ltd., 1997.