# Redes Neurais Artificiais Multicamadas e o Algoritmo Backpropagation

*Sérgio Luiz Tonsig* [1]

**2º semestre de 2000**

[1] Analista de Sistemas, Especialista em Sistemas de Informação pela UFSCar e Mestrando em Gerência de Sistemas de Informação pela PUC Camp

# Sumário

**1. INTRODUÇÃO** ..... **3**

**2. UMA BREVE REVISÃO** ..... **4**

2.1. O MODELO DO NEURÔNIO ARTIFICIAL ..... 4
2.2. O *PERCEPTRON* - CONCEITO ..... 5
2.3. O PERCEPTRON – NA PRÁTICA ..... 6
*2.3.1. Regra de Aprendizado Perceptron* ..... *7*
*2.3.2. Treinamento* ..... *7*

**3. MODELO MATEMÁTICO – PERCEPTRON** ..... **8**

**4. *PERCEPTRON* MULTICAMADA** ..... **12**

**5. ALGORITMO *BACKPROPAGATION*** ..... **14**

5.1. O FUNCIONAMENTO DO *BACKPROPAGATION* ..... 14
5.2. LIMITAÇÕES DO *BACKPROPAGATION* ..... 15
6.1. SISTEMA DE IDENTIFICAÇÃO AUTOMÁTICA DE VEÍCULOS ..... 17
*6.1.1. Descrição do Sistema* ..... *18*
*6.1.2. Algoritmo de localização da placa* ..... *18*
*6.1.3. Segmentação e redimensionamento dos caracteres* ..... *19*
*6.1.4. Reconhecimento dos caracteres através de uma rede neural* ..... *19*
*6.1.5. Resultados Preliminares* ..... *19*

**6. CONCLUSÃO** ..... **21**

**REFERÊNCIAS BIBLIOGRÁFICAS** ..... **22**

# 1. Introdução

Durante as últimas décadas o homem pesquisou formas que possibilitasse a uma máquina aprender. Aprender como as pessoas aprendem. Com o advento das redes neurais, consegue-se uma vertente na Inteligência Artificial sobre a qual este objetivo poderá estar sendo alcançado.

Como se ensina uma criança a reconhecer o que é uma cadeira ? Você mostra a ela exemplos e diz: Isto é uma cadeira; aquele outro não é uma cadeira. Até que a criança aprende o conceito do que é uma cadeira. Nesta fase, é importante que a criança possa olhar aos exemplos mostrados para que seja possível a ela responder corretamente à pergunta: “Isto aqui é uma cadeira ?”.

Além disso, se é mostrado a criança novos objetos anteriormente não vistos, pode-se esperar que estes sejam reconhecidos corretamente quanto a tratar-se ou não de um cadeira, considerando que se tenha submetido esta criança a muitos exemplos de casos positivos e negativos de reconhecimento de cadeiras.

Este mesmo procedimento natural, pode ser espelhado para o ensino de uma rede neural. É sobre este aspecto a abordagem do presente trabalho.

Este trabalho considera que o leitor já possua um prévio conhecimento sobre redes neurais (básico que seja), pois o mesmo é um aprofundamento de um aspecto dentro do tema, decorrente de nosso trabalho anterior mais genérico (Simulando o Cérebro: Redes Neurais).

Na sua primeira parte o presente trabalho se preocupa em resgatar alguns conceitos básicos de redes neurais, fazendo-se acompanhar pela conceituação matemática envolvida. Mostra o significado do neurônio artificial, sua funcionalidade, e faz um detalhamento do *perceptron.*

Em seguida, coloca todas as questões que envolvem o aspecto de multicamada das redes neurais artificiais, bem como demonstra o algoritmo de aprendizagem *backpropagation.*

O trabalho encerra apresentando alguns casos de aplicação prática de redes neurais artificiais, multicamadas e que utilizam o algoritmo de aprendizagem *backpropagation*.

# 2. Uma Breve Revisão

## 2.1. O Modelo do Neurônio Artificial

Fazendo uma analogia entre células nervosas vivas e o processo eletrônico num trabalho publicado em 1943, sobre "neurônios formais", onde é simulado o comportamento do neurônio natural, o neurofisiologista Warren McCulloch, do MIT, e o matemático Walter Pitts da Universidade de Illinois, fizeram a primeira proposta para um modelo de neurônio artificial[2] (figura 01). Este modelo apresentava apenas uma saída, que era uma função da soma (*threshold*) do valor de suas diversas entradas. **[Vellasco, 2000]**

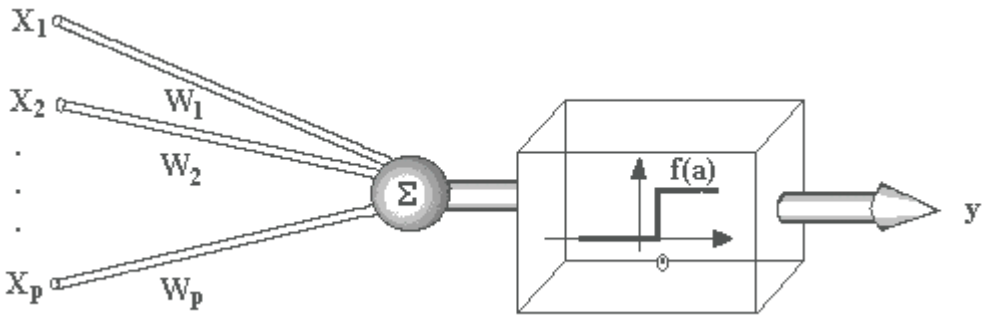


**Fig. 01 - Neurônio artificial projetado por McCulloch e Pitts**

O modelo de McCulloch-Pitts inspirado no funcionamento de um neurônio biológico consiste de muitas entradas, correspondente aos dentritos conectados através das junções sinápticas. O modelo, matematicamente, é descrito por:

$$\sum_{k} W_{ik}.X_{ik} + BIAS_i$$

$x_i$ = entradas, *i = 0, 1, 2, ..., N,*

*bias = para controlar o neurônio (Threshold),*

$w_i$ = pesos variáveis.

[2] O neurônio artificial é uma estrutura lógico-matemática que procura simular a forma, o comportamento e as funções de um neurônio biológico. Assim sendo, os dendritos foram substituídos por *entradas*, cujas ligações com o corpo celular artificial são realizadas através de elementos chamados de *peso* (simulando as sinapses). Os estímulos captados pelas entradas são processados pela *função de soma*, e o limiar de disparo do neurônio biológico foi substituído pela *função de transferência*. **[DENIZ, 1998]**

Todo o conhecimento de uma rede neural está armazenado nas sinapses, ou seja, nos pesos atribuídos às conexões entre os neurônios. De 50 a 90% do total de dados deve ser separado para o treinamento da rede neural, dados estes escolhidos aleatoriamente, a fim de que a rede "aprenda" as regras e não "decore" exemplos. O restante dos dados só é apresentado à rede neural na fase de testes a fim de que ela possa "deduzir" corretamente o inter-relacionamento entre os dados. **[VELLASCO, 2000]**

McCulloch e Pitts não desenvolveram nenhum método através do qual o neurônio pudesse adaptar seus pesos em um processo de "aprendizagem".

Em 1949, Donald Hebb foi o primeiro a propor uma lei de aprendizagem especifica para as sinápses dos neurônios. Demonstrou que a capacidade da aprendizagem em redes neurais vem da alteração da eficiência sináptica, isto é, a conexão somente é reforçada se tanto as células pré-sinápticas quanto as pós-sinápticas estiverem excitadas. Hebb postulou uma fórmula matemática simples para mudar os pesos dos neurônios em proporção para as ativações do neurônio. **[RUSSELL & NORVIG, 1995]**

$$\Delta w_i = \mu Y(x) x_i$$

$i$ = $0, 1, 2, ..., N$

$x$ = vetor de ($N$+1) entradas

$\mu$ = parâmetro de aprendizado

## 2.2. O *Perceptron* - Conceito

Em 1958, Rosenblatt, demonstrou algumas aplicações práticas usando o *perceptron*. O *perceptron*, conceitualmente, é um neurônio de McCulloch-Pitts de um simples nível de conexão (Figura 02). O *perceptron* é capaz de separar linearmente vetores de entrada em classes de padrões através de hiperplanos. **[RUSSELL & NORVIG, 1995]**

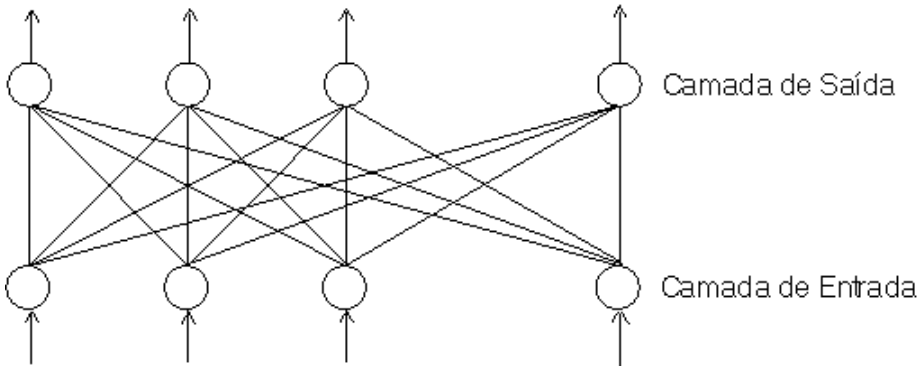


**Fig. 02 - Rede de perceptrons proposta por Rosenblatt**

Rosemblatt derivou a regra de aprendizado baseado no ajuste do peso na proporção do entre os neuronios de saida e as saidas desejadas. O ajustamento dos pesos são dados por:

$$\Delta w_i = \mu (y_i^d - y_i) x_i$$

$i$ = *1, 2, ..., M,*
$y^d$ = vetor saída desejada.
$\boldsymbol{x}$ = vetor de ($N$+1) entradas
$\mu$ = parâmetro de aprendizado

## 2.3. O Perceptron – Na prática

Na prática, o *perceptron* é um *software* que aprende conceitos. Ele pode aprender a responder com Verdadeiro (1) ou Falso (0) às entradas que forem fornecidas a ele, através do estudo de exemplos apresentados repetidamente. O *perceptron*, gera um grande interesse dado a sua capacidade de generalização. Geralmente são aplicados para problemas simples que envolvem classificação de padrões. A técnica de aprendizado utilizada é chamada de regra de aprendizado *perceptron*.

O *perceptron* é uma rede neural de uma camada só, na qual se pode treinar pesos vinculados a padrões de entradas que são fornecidos a rede, além do uso de um conceito de controle do neurônio (*bias*), para se obter uma saída correta. O conceito *bias* é uma forma de controlar o *threshold* do neurônio. *Threshold* (ou limiar) é o valor a partir do qual a somatória dos pesos convencionais determinará se o neurônio estará ativo ou inativo. **[RUSSELL & NORVIG, 1995]**

Considerando uma rede *perceptron* constituída de apenas um neurônio (figura 03), conectado a duas entradas e seus respectivos pesos, com um *bias*, o cálculo do valor de saída para este modelo, pode ser expresso na equação que segue: **[RUSSELL & NORVIG, 1995]**

$$Ei * Wi + b > 0$$

onde *E* é o vetor de entrada, *W* é o vetor dos pesos e *b* é o *bias*. Em nosso exemplo, o $i = 1, 2$.

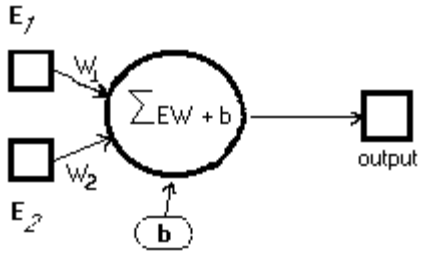


**Fig. 03 - Rede de perceptrons 2 entradas, um *bias* e uma saída**

### 2.3.1. Regra de Aprendizado *Perceptron*

O *perceptron* é treinado para responder a cada vetor de entrada com uma correspondente saída que pode ser 0 ou 1. Foi provado que a regra de aprendizagem converge em uma solução em tempo finito se uma solução existe. **[RUSSELL & NORVIG, 1995]**

A regra de aprendizagem pode ser resumida na seguinte duas equações:

Para todo *i:*

$$W(i) = W(i) + [\ T - A\ ] * E(i)$$

$$b = b + [\ T - A\ ]$$

onde, *W* é o vetor dos pesos, *E* são as entradas apresentadas para a rede, *T* é o resultado correto que a rede deve mostrar, *A* é a atual saída mostrada pela rede e *b* é o bias.

### 2.3.2. Treinamento

Conjunto de vetores de entrada para treinamento são apresentados a rede. Se a saída apresentada pela rede estiver correta, nenhuma alteração é feita. Caso, contrário os valores dos pesos e do *bias* são alterados, utilizando-se a regra de aprendizagem *perceptron*. No treinamento, uma passagem inteira por todo circuito da rede, é chamado de ciclo.

Quando Redes Neurais Artificiais de uma só camada são utilizadas, os padrões de treinamento apresentados à entrada são mapeados diretamente em um conjunto de padrões de saída da rede, ou seja, não é possível a formação de uma representação interna. Neste caso, a codificação proveniente do mundo exterior deve ser suficiente para implementar esse mapeamento.

Tal restrição implica que padrões de entrada similares resultem em padrões de saída similares, o que leva o sistema à incapacidade de aprender importantes mapeamentos.

Como resultado, padrões de entrada com estruturas similares, fornecidos do mundo externo, que levem a saídas diferentes não são possíveis de serem mapeados por redes sem representações internas, isto é, sem camadas intermediárias. Um exemplo clássico deste caso é a função ou-exclusivo (XOR).

Um único *perceptron* ou uma combinação das saídas de alguns perceptrons poderia realizar uma operação *XOR,* porém, seria incapaz de aprendê-la. Para isto são necessárias mais conexões, os quais só existem em uma rede de perceptrons dispostos em camadas. **[MENDES FILHO, 2000]**

# 3. Modelo Matemático – Perceptron

Para apresentação completa do modelo matemático[3] que envolve o *perceptron,* algumas convenções foram estabelecidas para maior clareza do conteúdo:

➔ As multiplicações são indicadas pelo '.', ou seja, A vezes B é A.B

➔ Variáveis são escritas em maiúsculas, índices em minúsculas

➔ EXP(X) = Função que representa o logaritmo Neperiano - "e" elevado a potência "x", ou simplesmente como costuma-se dizer "e na x" (logaritmo natural -> base 'e')

Lembrando que o *perceptron* é uma rede neural de uma camada só, na qual se pode-se treinar pesos vinculados a padrões de entradas que são fornecidos a rede, pode-se seguir os passos abaixo, para ter-se um modelo adequado para tal fim.

a) Ativação da rede . Neste passo, ativa-se os neurônios, para calcular o valor obtido na saída da rede.

$OUT_i$ = Valor obtido na saída no neurônio 'i'

$$OUT_i = F ( SomaPond_i )$$

$$\text{onde } SomaPond_i = \sum_k W_{ik}.X_{ik} + BIAS_i$$

logo podemos re-escrever assim,

$$OUT_i = F \left( \sum_k W_{ik}.X_{ik} + BIAS_i \right]$$

$$\text{onde } F ( X ) = \frac{1}{1 + EXP^{(-x)}}$$

então, teremos que

$$OUT_i = \frac{1}{1 + EXP( - \sum W_{ik}.X_{ik} + BIAS_i )}$$

[3] Toda conceituação matemática exposta neste capítulo, pode ser encontrada nas referências bibliográficas que seguem: **[RUSSELL & NORVIG, 1995]** ; **[VELLASCO, 1995]**

Sendo que no somatório $(\Sigma)$, 'k' varia de 0 até N, onde N é o número de conexões de entrada que possui o neurônio em questão (neurônio 'i').

Xik representa o sinal da entrada 'k' do neurônio em questão (neurônio 'i').

Wik representa o peso do neurônio 'i' associado a entrada 'k' (este mesmo neurônio possui N entradas, e obviamente N pesos).

Exemplo: O neurônio 3 tem quatro entradas e portanto sua ativação seria igual a:

$$OUT_3 = F ( W_{31}.X_{31} + W_{32}.X_{32} + W_{33}.X_{33} + W_{34}.X_{34} + BIAS_3)$$

| |
| -- Nro. da entrada ('k')
|
+--- Nro. do neurônio ('i')

b) Aprendizado - Cálculo do erro e correção dos pesos

No caso de uma rede de uma só camada, o erro é aplicado diretamente sobre os neurônios da última e única camada sem precisar retro-propagar para as camadas superiores.

$$ERRO_i = F'( SomaPond_i ) . ERRO\_ESTIMADO_i$$

onde, $ERRO\_ESTIMADO_i = SAIDA\_DESEJADA_i - SAIDA\_OBTIDA_i$

podemos também escrever:

$ERRO\_ESTIMADO_i = SAIDA\_DESEJADA_i - OUT_i$ pois...

$SAIDA\_OBTIDA_i$ é na realidade o valor $OUT_i$

logo podemos re-escrever assim,

$$ERRO_i = F'( SomaPond_i ) . ( SAIDA\_DESEJADA_i - SAIDA\_OBTIDA_i )$$

nde F'(X) = Derivada da função F(X), ou seja, derivada da função sigmoïde:

$$F(X) = \frac{1}{1 + EXP^{(-X)}}$$ (função sigmoïde)

para a função acima (sigmoïde), ao derivar-se ela, obtem-se:

$$F'(X) = F(X) . ( 1 – F(X) )$$ (derivada da sigmoïde)

que pode ser re-escrita da seguinte maneira,

$$F'(X) = \frac{1}{1 + EXP^{(-X)}} \cdot \left( 1 - \frac{1}{1 + EXP^{(-X)}} \right)$$

X é o valor que é fornecido para a função. Neste caso, X será a soma ponderada:

$$SomaPond_i = \sum W_{ik}.X_{ik} + BIAS_i$$

Nas equações da página anterior, temos portanto os seguintes elementos:

$ERRO_i$ = O erro obtido para a saída do neurônio 'i'

$ERRO\_ESTIMADO_i$ = O erro estimado, ou seja, a diferença da resposta que queríamos obter na saída e a resposta que realmente foi obtida na saída do neurônio 'i' (uma rede pode ter vários neurônios na camada de saída).

$SAIDA\_DESEJADA_i$ = A saída que queríamos obter no neurônio 'i' (esta informação está presente no arquivo de aprendizado, onde damos o exemplo e a resposta que deve ser fornecida quando aquele exemplo for apresentado na entrada da rede)

$SAIDA\_OBTIDA_i$ = A saída que o neurônio 'i' deu... ou seja o valor da ativação deste que nós calculamos no passo 1 de propagação *forward* (é o valor de $OUT_i$).

c) De posse do erro $ERRO_i$ do neurônio 'i' podemos ajustar seus pesos usando a seguinte fórmula:

$$W_{ik}(t+1) = W_{ik}(t) + ERRO_i \cdot X_{ik}$$

sendo que:

$W_{ik}(t)$ é o valor do peso W do neurônio 'i' conectado a entrada 'k' em um dado instante de tempo (t)

$W_{ik}(t+1)$ é o valor do peso W do neurônio 'i' conectado a entrada 'k' em um dado instante de tempo (t+1), ou seja, logo após a correção deste peso que é calculada pela expressão acima.

A fórmula de ajuste de pesos acima é bem simplificada, pois sempre usamos um fator chamado de velocidade de aprendizado(também denominado de *alfa* ou *epsilon*), que permite avançar mais lentamente e evitar de mudar os pesos muito rapidamente "passando" de seu valor ideal.

A nova fórmula ficará assim:

$$W_{ik}(t+1) = W_{ik}(t) + ERRO_i \cdot X_{ik} \cdot VELOC\_APRENDIZADO$$

Sendo que VELOC_APRENDIZADO (alfa) é um valor real maior que 0 e menor que 1. Um valor igual a 0.5 reduz a "velocidade" da correção dos pesos pela metade.

Outra pequena alteração da fórmula acima é a introdução da inércia, também conhecida como *momentum* ou momento.

Para garantir o sucesso do aprendizado devemos usar uma velocidade de aprendizado muito baixa, algo do tipo 0.1 ou 0.01 o que faz com que o algoritmo se torne lento demais... para compensar este problema tentamos acelerar a velocidade, usando a inércia, que no entanto não irá causar grandes estragos como seria o caso do uso de uma velocidade grande. Porque pode-se usar a inércia e não se pode usar sempre apenas a velocidade ? É simples, a inércia se baseia na alteração anterior dos pesos, se a alteração foi grande, então na próxima alteração não precisamos ir tão devagar, mas se estamos indo devagar não devemos de um instante para o outro sair correndo! O *momentum* garante uma aceleração gradual e uma desaceleração também gradual da velocidade de alteração dos pesos.

Velocidade de alteração significa para nós o tamanho da mudança feita no valor dos pesos. E aqui está a fórmula com a inércia incluída:

$$W_{ik}(t+1) = W_{ik}(t) + ERRO_i \cdot X_{ik} \cdot VA + MOMENTUM \cdot \Delta W_{ik}(t-1)$$

Onde:

VA aparece aqui como uma abreviatura para Veloc_aprendizado.
$\Delta W_{ik}(t-1)$ e' igual ao último ajuste feito nos pesos.

Ou seja, podemos re-escrever a fórmula acima em duas etapas:

$$\Delta W_{ik}(t) = ERRO_i \cdot X_{ik} \cdot VA + MOMENTUM \cdot \Delta W_{ik}(t-1)$$

$$W_{ik}(t+1) = W_{ik}(t) + \Delta W_{ik}(t)$$

Assim nós guardamos o valor da última alteração dos pesos para usar junto a próxima alteração destes. O valor do *momentum* também é um valor real entre 0 e 1, mas normalmente usamos uma inércia alta, de digamos 0.9, pois a inércia ajuda a chegar mais rápido num valor dos pesos que nos satisfaça (reduzindo o erro na saída do neurônio mais rapidamente) sem no entanto provocar perturbações exageradas no processo de convergência em direção ao ponto de mínimo na curva de erro.

# 4. *Perceptron* Multicamada

As capacidades do *perceptron* de uma simples camada é limitada a fronteiras de decisão linear e funções lógicas simples.

Minsky e Papert (*Perceptrons*, 1969) analisaram matematicamente o *Perceptron* e demonstraram que redes de uma só camada, não são capazes de solucionar problemas que não sejam linearmente separáveis. Como não acreditavam na possibilidade de se construir um método de treinamento para redes com mais de uma camada, eles concluíram que as redes neurais seriam sempre suscetíveis a essa limitação. **[RUSSELL & NORVIG, 1995]**

Identificado as limitações relativas ao *perceptron* de camada simples, foi desenvolvido o *multilayer perceptron* (MLP), que em geral, consiste de uma camada de entradas, uma ou mais camadas intermediárias (escondidas) e uma camada de saída . Trata-se de um caso em particular de topologia de rede, conforme mostra a taxionomia da rede na figura 03. **[VELLASCO, 2000]**

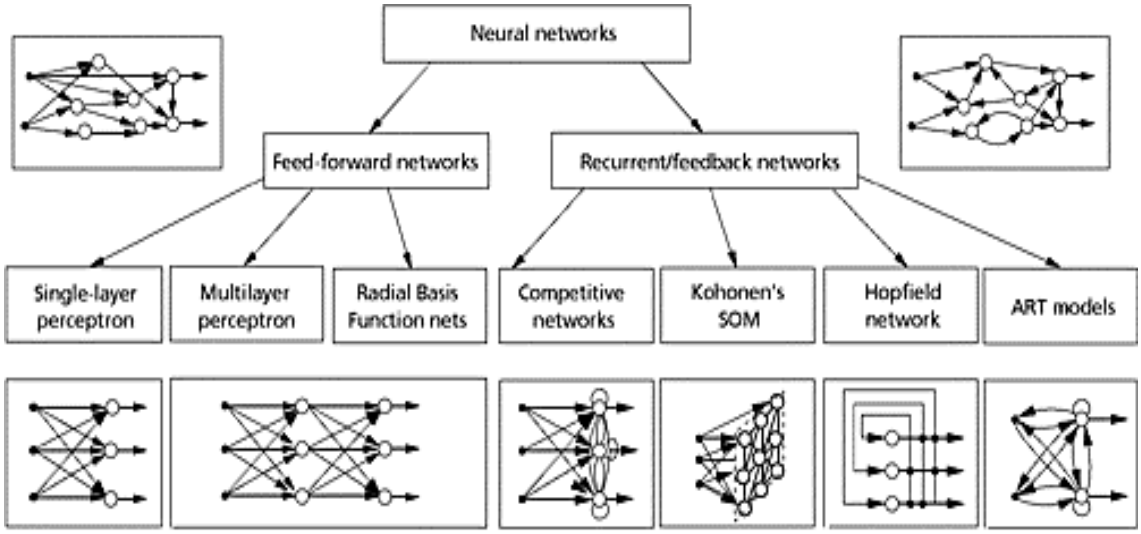


**Fig. 03 – Taxionomia das Arquiteturas de Redes Neurais Artificiais**

Nas redes *multilayer perceptron*, cada camada tem uma função específica. A camada de saída recebe os estímulos da camada intermediária e constrói o padrão que será a resposta. As camadas intermediárias funcionam como extratoras de características, seus pesos são uma codificação de características apresentadas nos padrões de entrada e permitem que a rede crie sua própria representação, mais rica e complexa, do problema.

Arquiteturas neurais são tipicamente organizadas em camadas (figura 04), com unidades que podem estar conectadas às unidades da camada posterior. Por convenção a camada que inicialmente recebe os dados é chamada de *input* (entrada), a camada intermediária é chamada de *hidden* (oculta) e por último, a camada de saída. Cada camada pode ter de 1 a n neurônios artificiais com as características apresentadas na figura 09. **[DHAR & STEIN, 1997]**

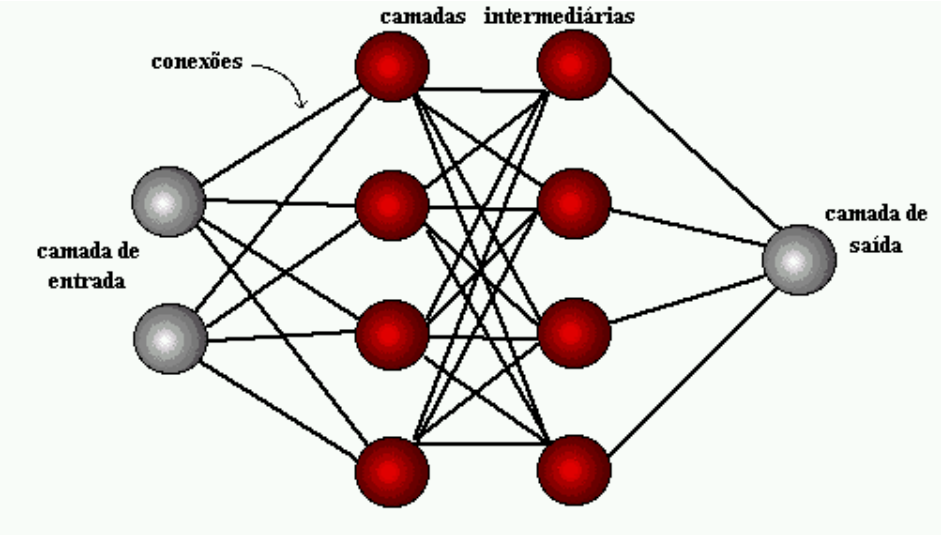


**Fig. 04 – Exemplo de Rede Neural Artificial Multicamada**

O comportamento inteligente de uma Rede Neural Artificial vem das interações entre as unidades de processamento da rede. A maioria dos modelos de redes neurais possui alguma regra de treinamento, onde os pesos de suas conexões são ajustados de acordo com os padrões apresentados; ou seja, elas aprendem através de exemplos. **[YAMAMOTO & NIKIFORUK, 2000]**

# 5. Algoritmo *Backpropagation*

O mais popular método para aprendizado em redes de multicamadas é chamado de *backpropagation* (retro-propagação). Ele foi inicialmente criado em 1969 por Bryson e Ho, mas foi ignorado até  o inicio dos anos 80.

## 5.1. O Funcionamento do *Backpropagation*

Durante o treinamento[4] com o algoritmo *backpropagation*, a rede opera em uma seqüência de dois passos (figura 05). Primeiro, um padrão é apresentado à camada de entrada da rede. A atividade resultante flui através da rede, camada por camada, até que a resposta seja produzida pela camada de saída. No segundo passo, a saída obtida é comparada à saída desejada para esse padrão particular. Se esta não estiver correta, o erro é calculado. O erro é propagado a partir da camada de saída até a camada de entrada, e os pesos das conexões das unidades das camadas internas vão sendo modificados conforme o erro é retropropagado. **[CARVALHO, 2000]**

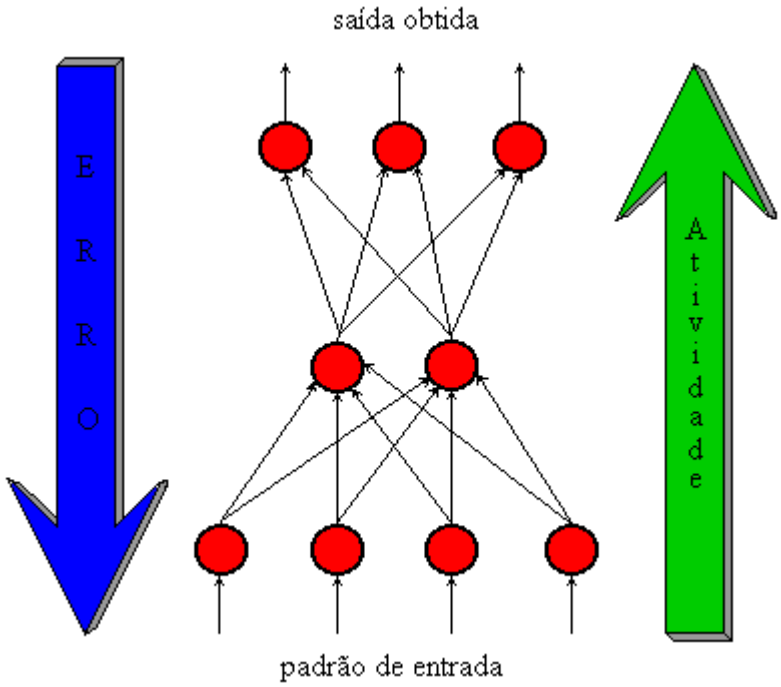


**Fig. 05 – Operação do Algoritmo Backpropagation**

As redes que utilizam *backpropagation* trabalham com uma variação da regra delta, apropriada para redes multicamadas: a regra delta generalizada.

[4] O treinamento é a atividade que leva a rede ao aprendizado. Denomina-se algoritmo de aprendizado a um conjunto de regras bem definidas para a solução de um problema de aprendizado. Existem muitos tipos de algoritmos de aprendizado específicos para determinados modelos de redes neurais, estes algoritmos diferem entre si principalmente pelo modo como os pesos são modificados. Aprendizagem, para uma rede neural, envolve o ajuste destes pesos. **[DHAR & STEIN, 1997]**

A regra delta padrão essencialmente implementa um gradiente descendente no quadrado da soma do erro para funções de ativação lineares. Redes sem camadas intermediárias, podem resolver problemas onde a superfície de erro tem a forma de um parabolóide com apenas um mínimo. Entretanto, a superfície do erro pode não ser tão simples, e suas derivadas mais difíceis de serem calculadas. Nestes casos devem ser utilizadas redes com camadas intermediárias. Ainda assim, as redes ficam sujeitas aos problemas de procedimentos "*hill-climbing*", ou seja, ao problema de mínimos locais. **[CARVALHO, 2000]**

Depois que a rede estiver treinada e o erro estiver em um nível satisfatório, ela poderá ser utilizada como uma ferramenta para classificação de novos dados (figura 06). Para isto, a rede deverá ser utilizada apenas no modo progressivo (*feed-forward*). Ou seja, novas entradas são apresentadas à camada de entrada, são processadas nas camadas intermediárias e os resultados são apresentados na camada de saída, como no treinamento, mas sem a retropropagação do erro. A saída apresentada é o modelo dos dados, na interpretação da rede. **[CARVALHO, 2000]**

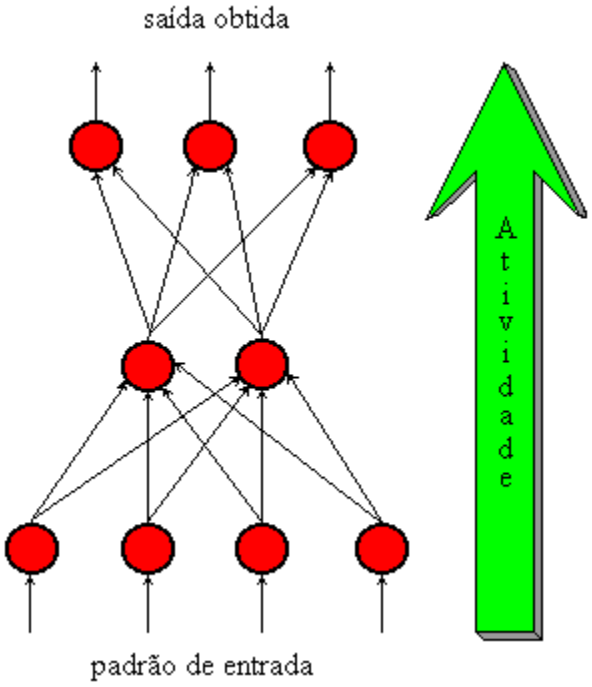


**Fig. 06 – *Backpropagation* em modo progressivo (*feed-forward)***

## 5.2. Limitações do *Backpropagation*

As redes neurais que utilizam *backpropagation*, assim como muitos outros tipos de redes neurais artificiais, podem ser vistas como "caixas pretas", na qual quase não se sabe porque a rede chega a um determinado resultado, uma vez que os modelos não apresentam justificativas para suas respostas. Neste sentido, muitas pesquisas vêm sendo realizadas visando a extração de conhecimento de redes neurais artificiais, e na

criação de procedimentos explicativos, onde se tenta justificar o comportamento da rede em determinadas situações.

Uma outra limitação refere-se ao tempo de treinamento de redes neurais utilizando *backpropagation*, que tende a ser muito lento. Algumas vezes são necessários milhares de ciclos para se chegar à níveis de erros aceitáveis, principalmente se estiver sendo simulado em computadores seriais, pois a CPU deve calcular as funções para cada unidade e suas conexões separadamente, o que pode ser problemático em redes muito grandes, ou com grande quantidade de dados. Muitos estudos estão sendo realizados para implementação de redes neurais em computadores paralelos, além de construção de *chips* neurais como *Intel* 80170NX *Eletronically Trainable ANN* ou placas aceleradoras como *BrainMaker Accelerator Board* CNAPS. **[CARVALHO, 2000]**

É muito difícil definir a arquitetura ideal da rede de forma que ela seja tão grande quanto o necessário para conseguir obter as representações necessárias, ao mesmo tempo pequena o suficiente para se ter um treinamento mais rápido. Não existem regras claras para se definir quantas unidades devem existir nas camadas intermediárias, quantas camadas, ou como devem ser as conexões entre essas unidades. Para resolver este tipo de problema, Algoritmos Genéticos poderiam ser utilizados para encontrar automaticamente boas arquiteturas de redes neurais, eliminando muitas armadilhas associadas às abordagens de engenharia humana. **[CARVALHO, 2000]**

# 6. Exemplo de Aplicação

Para mostrar na prática uma aplicação que utiliza rede neural artificial multicamadas com algoritmo *backpropagation*, será apresentado a síntese de um trabalho desenvolvido na Universidade Federal do Rio Grande do Sul **[SUSIN, 2000]**, através dela poderá ter-se uma idéia da importância deste algoritmo de aprendizagem atuando em uma rede multicamada.

## 6.1. Sistema de Identificação Automática de Veículos

Quando ao objetivo, este sistema visa dotar uma máquina da capacidade de localizar e interpretar o conteúdo da placa de um veículo qualquer através da utilização de técnicas de processamento de imagens e inteligência artificial.

Algumas aplicações deste tipo de sistema são:

- Medição e planejamento do fluxo de tráfego
- Identificação de veículos para recuperação em caso de furto
- Controle automático de pedágios e estacionamentos pagos
- Aplicação das leis de trânsito (identificação automática de veículos infratores, estacionamento em áreas proibidas, etc. )

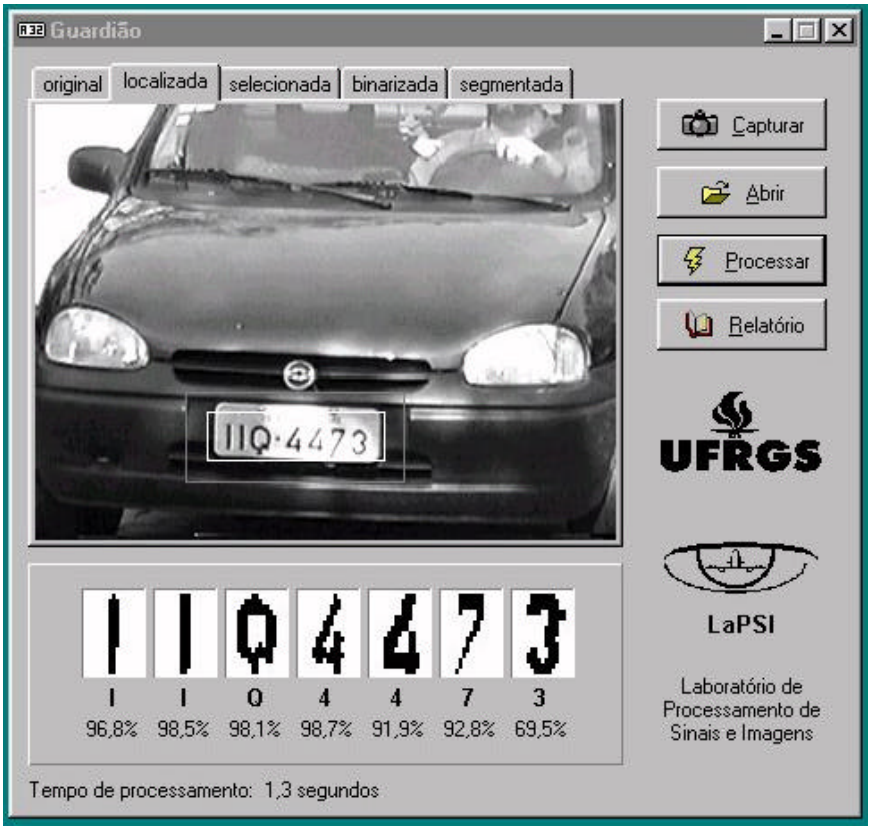


**Fig. 07 – Sistema de Identificação Automática de Veículos**

### 6.1.1. Descrição do Sistema

A plataforma de desenvolvimento do projeto foi um microcomputador PC e o software foi desenvolvido em C++ para Windows 95. Este computador, equipado com uma placa digitalizadora de vídeo, permite a aquisição de imagens com resolução de 320x 240 pixels. A figura 07 mostra o aplicativo desenvolvido.

### 6.1.2. Algoritmo de localização da placa

O algoritmo de localização da placa pode ser dividido em dois procedimentos:

a) No primeiro procedimento temos a localização da placa através dos algoritmos abaixo:

- Procura por variação tonal padronizada através da análise do gradiente positivo na área da placa;
- Binarização local adaptativa sobre a região encontrada ou, se necessário, sobre a imagem inteira;

Nesta etapa, temos como resultado as coordenadas do provável local da placa.

b) No segundo procedimento há a confirmação do local através da análise do número de dígitos encontrados:

- Procura por dígitos de tamanhos pré-definidos;
- Análise dos dígitos selecionados verificando a formação de algum grupo válido;

Como resultado final temos a confirmação do local correto, conforme ilustra a figura 08.

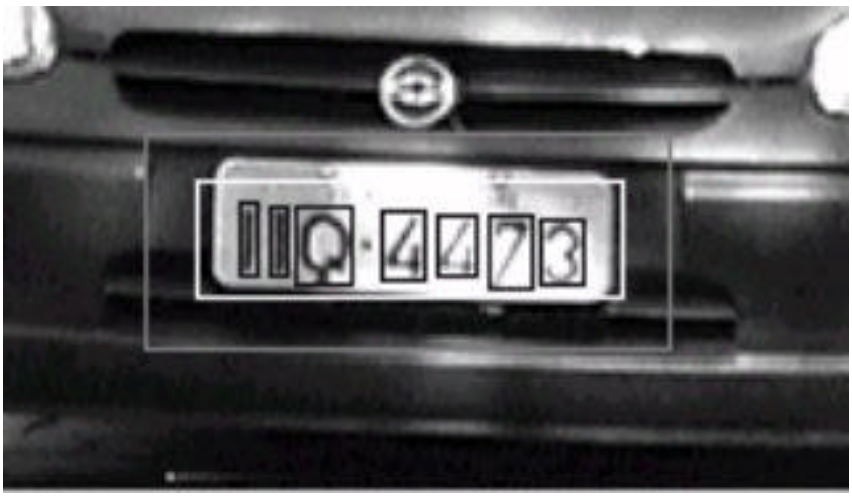


**Fig. 08 – Confirmação Local Correto da Placa do Veículo**

### 6.1.3. Segmentação e redimensionamento dos caracteres

Uma vez encontrada a placa é necessário segmentar os caracteres (figura 09) a fim de separá-los do resto da imagem e redimensioná-los para a rede neural. Foi utilizada uma técnica de crescimento controlado, dentro de cada entidade encontrada na área da placa, e um conjunto de heurísticas para descartar ruídos indesejáveis e selecionar corretamente os caracteres.


**Fig. 09 - Placa Segmentada - 7 Entidades Encontradas**

### 6.1.4. Reconhecimento dos caracteres através de uma rede neural

Uma topologia de rede neural *feedforward* utilizando o algoritmo *backpropagation* para treinamento foi escolhida. Foram desenvolvidas duas redes distintas, uma para os caracteres e outra para os algarismos, com 255x26 x26 e 255x10x10 neurônios nas camadas de entrada x escondida x saída respectivamente.

### 6.1.5. Resultados Preliminares

O sistema foi testado com um conjunto de 300 imagens e comparado à uma versão demonstração de um software comercial israelense - See/Car. As imagens utilizadas no teste possuem as seguintes características:

- 256 tons de cinza e 320x240 pixels
- Dimensões da placa entre 70x20 e 120x40 pixels
- Imagens da frente e costas dos veículos com distribuição de luz sobre a superfície da placa homogênea e heterogênea

No See/Car:

- 76,2 % de sucesso na localização das placas

- 64,6 % de sucesso na segmentação correta dos caracteres

No SIAV[5]:

- 98,7 % de sucesso na localização das placas
- 87,1 % de sucesso na segmentação correta dos caracteres
- 82,4 % de sucesso na localização das placas com a rotina preliminar (variação tonal)
- 87 % de sucesso no reconhecimento dos caracteres segmentados
- 28 % de sucesso no reconhecimento correto das placas (7 dígitos)

O sistema desenvolvido possui um bom desempenho em ambientes com diferentes tipos de iluminação. O tempo de processamento de cada imagem é dependente da complexidade da imagem analisada e pode variar entre 1.4 e 130s (processador K6-2 400 MHz com 64 MB de RAM).

A baixa taxa de acertos no reconhecimento dos caracteres, deve-se, entre outras coisas, ao deficiente banco de amostras utilizadas no seu treinamento. Estão sendo feitos esforços para a formulação de um banco de dados apropriados e uma descrição mais eficiente dos caracteres para a rede.

[5] Uma cópia de demonstração deste software pode ser obtida no endereço: http://www.iee.ufrgs.br/iee/siav.zip

# 6. Conclusão

A maior parte do tempo necessário para construir um modelo de redes neurais é gasto no processo identificação da melhor topologia e no de treinamento. Determinar uma boa estrutura de rede, ou topologia, é um dos caminhos para se diminuir esse tempo, mas só isto não basta para se ter uma rede que ofereça resultados aceitáveis.

O binômio topologia x treinamento é fator que hoje apresenta-se como barreira a ser transposta para o sucesso de uma solução utilizando-se rede neural artificial.

Há um peso muito grande no aspecto de  treinamento, para se atingir resultados satisfatórios. Assim, é necessário que se tenha uma idéia, ainda que genérica, sobre os algoritmos disponíveis para treinamento das diversas topologias de redes existentes. O algoritmo *backpropagation* para uma rede de multicamanadas, exposto neste trabalho, é apenas um deles.

# Referências Bibliográficas


**[CARVALHO, 2000]** CARVALHO, André Ponce de Leon F., Home Page "*Perceptron Multicamadas"*. Departamento de Ciência da Computação. USP - SP http://www.icmc.sc.usp.br/~andre/neural2.html, consulta em 10/05/2000.

**[DENIZ, 1998]** DENIZ, Juan C., Home Page *"Neural Network Theory"*. MIT – Massachusetts Institute of Technology, http://web.mit.edu/denis/www/FuncAppx/Theory.html, consulta em 04/05/2000.

**[DHAR & STEIN, 1997]** DHAR, Vasant E STEIN, Roger. "*Seven Methods for Transforming Corporate Data Into Business Intelligence*". Prentice-Hall, New Jersey, 1997, p.77-106.

**[MENDES FILHO, 2000]** MENDES FILHO, Elson Felix, Home Page *"Redes Neurais Artificiais"*. Universidade de São Paulo, SP, http://www.icmsc.sc.usp.br/~prico/neural1.html, consulta em 28/04/2000.

**[RUSSELL & NORVIG, 1995]** RUSSELL, Stuart E NORVIG, Peter. "*Artificial Intelligence - A Modern Approach*". Prentice-Hall, New Jersey, 1995, p.563-597.

**[SUSIN, 2000]** , SUSIN, Dr. Altamiro Amadeu at alii. Home Page do Laboratório e Processamento de Sinais e Imagens da Universidade Federal do Rio Grande do Sul "*Projeto SIAV – Sistema de Identificação Automática de Veiculos"*. UFRGS - RS, http://www.iee.ufrgs.br/iee/siav.html, consulta em 13/05/2000.

**[VELLASCO, 2000]** VELLASCO, Marley Maria B.R, Home Page *"Redes Neurais"*. ICA: Núcleo de Pesquisa em Inteligência Computacional Aplicada – PUC - Rio, http://www.ele.puc-rio.br/labs/ica/icahome.html, consulta em 05/05/2000.

**[YAMAMOTO& NIKIFORUK, 2000]** YAMAMOTO, Yoshihiro e Nikiforuk Peter. *"A New Supervised Learning Algorithm for Multilayered and Interconnected Neural Networks"*. IEEE Transactins on Neural Networks, vol. 11, no. 1, 2000.