2.
Redes Neurais Artificiais

A motivação pelo uso das Redes Neurais Artificiais (RNA), comumente chamadas de "redes neurais", é pelo reconhecimento de que o cérebro processa informações de uma forma diferente dos computadores convencionais. O cérebro é um computador (sistema de processamento) altamente complexo, não-linear e paralelo, com capacidade de organizar estruturalmente os neurônios de forma a realizar funções como o reconhecimento de padrões e percepção, muito mais rápido que qualquer computador existente. O computador tem um processamento rápido e preciso na execução de seqüência de instruções porém é muito mais lento que o cérebro, por exemplo, no reconhecimento de padrões [2].

A rede neural é uma máquina que modela a maneira como um cérebro realiza determinadas tarefas ou funções de interesse. Uma rede neural pode ser implementada com componentes eletrônicos ou por programação em um computador. Podemos definir uma rede neural, conforme abaixo:

Inspiradas nos neurônios biológicos e na estrutura maciçamente paralela do cérebro, as Redes Neurais Artificiais (RNA) são modelos matemáticos que adquirem, armazenam e utilizam conhecimento através da experiência [1].

A maneira pela qual as redes neurais adquirem conhecimento é chamado de algoritmo de aprendizagem, que tem como função modificar, através de um processo iterativo, os pesos sinápticos da rede (treinamento), de modo a alcançar um objetivo desejado [2].

As características básicas das redes neurais são:

- Procura paralela e endereçamento pelo Conteúdo – O cérebro não possui endereço de memória e não procura a informação seqüencialmente;
- Aprendizado – A rede aprende por experiência, não precisando explicitar os algoritmos para executar uma determinada tarefa;
- Associação – Capacidade da rede em fazer associações entre padrões diferentes;

- Generalização – A rede neural produz saídas adequadas para entradas que não estavam presentes durante o treinamento (aprendizagem);
- Robustez e Degradação – Devido à natureza distribuída da informação armazenada na rede, o seu desempenho se degrada suavemente sob condições de operações adversas, tal como a perda de elementos processadores ou conexões sinápticas. Para que a resposta da rede seja degradada seriamente, o dano deve ser extenso.

2.1. Modelo de Neurônio Artificial

O neurônio artificial ou Elemento Processador (EP) é uma unidade fundamental para a operação de uma rede neural. Seu modelo foi inspirado na estrutura neural biológica. A Figura 1 ilustra uma forma de um tipo comum de neurônio biológico. Uma rede neural pode ter centenas ou milhares de EPs, já o cérebro de um mamífero poder ter bilhões de neurônios.

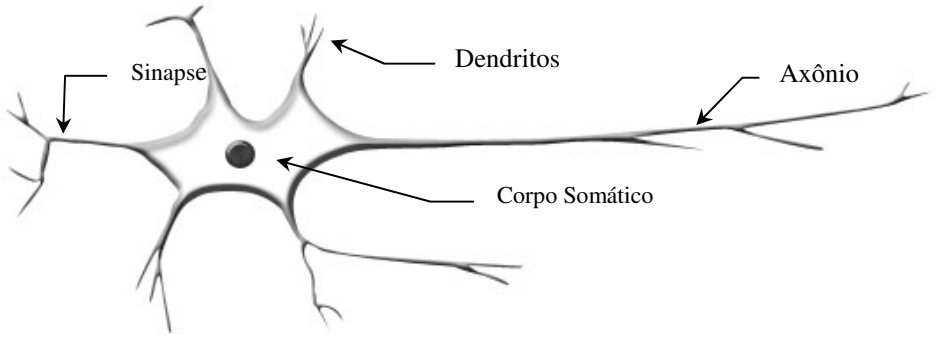


Figura 1 - Neurônio Biológico

A Figura 2 mostra o modelo de um neurônio artificial, que forma a base para um projeto de rede neurais. Este pode ser dividido em três elementos básicos:

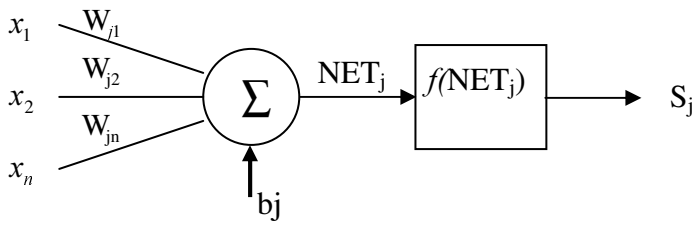


Figura 2 – Modelo Neurônio Artificial

Elemento 1 - Representado pelo conjunto de sinapses ou elos de conexões, que para cada uma existe um peso sináptico determinando o efeito da entrada sobre o processador. A Figura 2 mostra as entradas neurais X_1 a X_n, os pesos sinápticos seriam W_{j1} a W_{jn}, onde j refere-se ao neurônio em questão e **n** à **n**-ésima sinapse.

Elemento 2 - Um somador, que soma as entradas ponderadas pelos respectivos pesos sinápticos, definido como uma regra de propagação NETj (Equação 1). No modelo da Figura 2 foi incluído também o bias, que tem a função de aumentar ou diminuir a entrada da função de ativação, pois pode ser negativo ou positivo [1].

$$NET_j = \sum_i W_{ji}.x_i + b_j$$

Equação 1 – Regra de Propagação NET

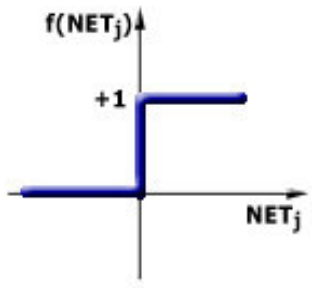
Elemento 3 - A função de ativação, também conhecida como função restritiva, restringe a amplitude de saída de um neurônio (Equação 2) em um intervalo de valor finito. Tipicamente usa-se o intervalo unitário fechado [0,1] ou [-1,1]. A função de ativação que gera a saída do EP.

$$S_j = f(NET_j)$$

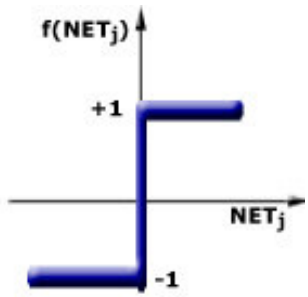
Equação 2 – Cálculo de Saída

Os três tipos básicos de função de ativação são:

- *Função Limiar.* Representada pela Função Degrau (Figura 3a) de intervalo [0,+1] ou Degrau Simétrico (Figura 3b) de intervalo [-1,1].



(a)



(b)

Figura 3 - Função Limiar. (a) Degrau (b) Degrau Simétrico

- *Função Pseudo Linear* A Figura 4 mostra um função pseudo-linear, também conhecida como função linear por partes.

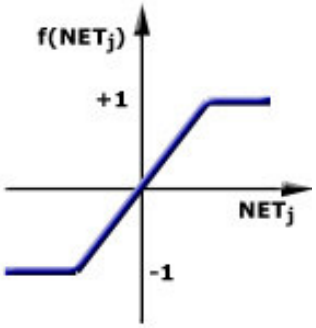


Figura 4 – Função Pseudo –Linear

- *Função Sigmóide.* Representada na Figura 5, é a função de ativação mais utilizada no projeto de redes neurais artificiais

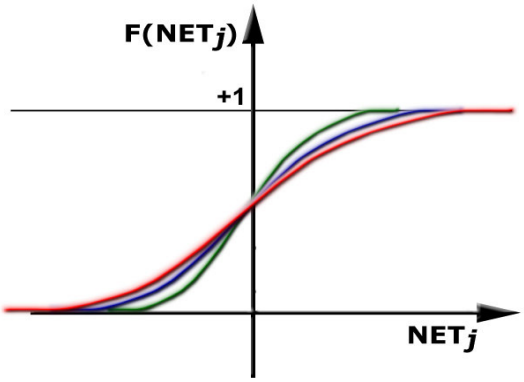


Figura 5 – Função Sigmóide

2.2. Características Gerais

As principais características para especificar uma rede neural são: Topologia de rede, características dos nós computacionais (saídas dos neurônios), e algoritmo (regra) aprendizado.

Os algoritmos de aprendizagens utilizados para treinar as redes neurais estão diretamente ligados à topologia de rede, estes algoritmos são estruturados [1].

2.2.1. Topologia de Rede

Quanto à topologia de rede podemos identificar, basicamente 2 tipos, as redes alimentadas adiante mais conhecidas como "Feed-Forward" e as redes recorrentes ou "Feed-Backward".

As redes "Feed-Forward" são as arquiteturas mais utilizadas nas RNAs, que podem ter seus conjuntos de neurônios dispostos em uma ou mais camadas de neurônios artificiais, tendo seu fluxo de dados sempre em direção a camada de saída. A Figura 6 ilustra a forma de rede de camada única. A designação camada única se refere à camada de saída, pois não contamos a camada de entrada, onde não há qualquer computação nesta camada.

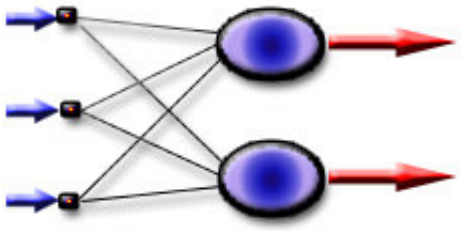


Figura 6 - Rede "Feed-Forward" de Camada Única

Rede alimentada diretamente com múltiplas camadas ou rede "Multi-Layer", é a configuração mais completa e apresenta uma ou mais camadas intermediárias, também chamadas de camadas ocultas. As camadas ocultas possibilitam as redes neurais extrair estatísticas de ordem mais elevadas, esta característica permite redes "Multi-Layer" apresentar solução para problemas mais complexos ou impossíveis para rede de camada única. A Figura 7 ilustra

uma rede neural “Feed-Forward” de múltiplas camadas com 1 camada oculta, ela é referida como uma rede 5-3-2, sendo 5 entradas, 3 EPs na camada oculta e 2 EPs na camada de saída.

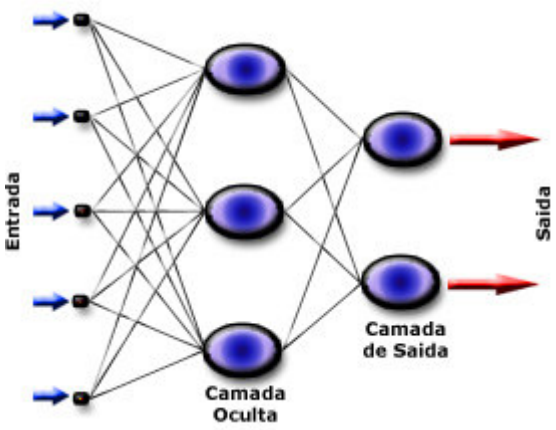


Figura 7- Rede “Feed-Forward” de Múltiplas Camadas

Redes Recorrentes ou “Feedbackward” representada na Figura 8 se distingue da “Feed-Forward” por possuírem laços de realimentação. Estes laços podem estar ligados à mesma camada ou com camadas anteriores, podendo apresentar camadas ocultas ou não.

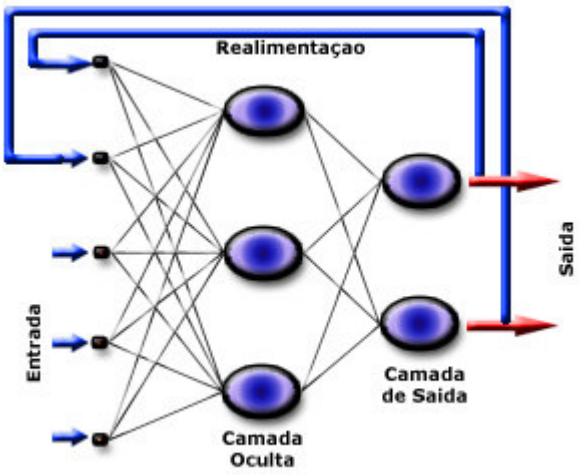


Figura 8- Redes Recorrentes com Neurônios Ocultos

2.2.2. Processamento Neural

O processamento neural pode ser dividido em duas fases: aprendizado (Learning) e execução (Recall). A fase de treinamento ou aprendizado é o processo de atualização dos pesos sinápticos. Depois da rede estar treinada,

quando o erro estiver no nível satisfatório, seus pesos sinápticos são armazenados em uma matriz peso (W). Na fase de execução a rede é utilizada apenas no modo progressivo (Feed-Forward), ou seja, novas entradas são apresentadas, são processadas nas camadas intermediárias e os resultados apresentados na camada de saída.

2.2.3. Tipos Básicos de Aprendizagem

Aprendizagem supervisionada

Neste tipo de treinamento, representado pelo esquema da Figura 9, também denominado aprendizado com um professor, as atualizações dos pesos sinápticos são feitos com apresentação dos dados de entrada e as respectivas respostas desejadas (pares de treinamento ou vetor de treinamento), sendo influenciados pelo sinal de erro. O ajuste é feito passo a passo, até que o erro seja mínimo.

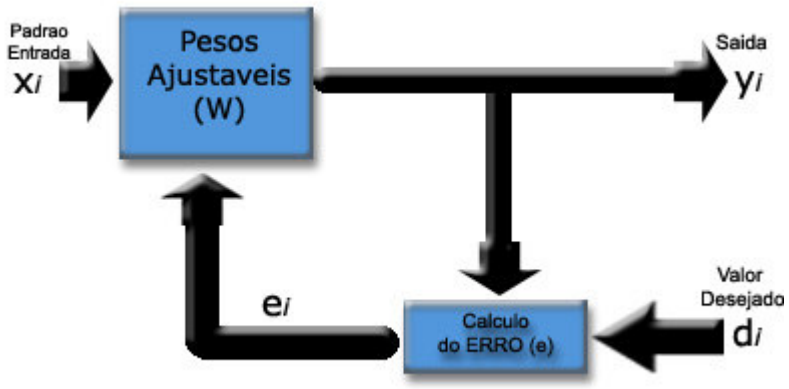


Figura 9 – Aprendizagem Supervisionada

Aprendizagem Não-Supervisionada

Conhecida também como aprendizagem auto-organizada. Neste treinamento, representado pelo esquema da Figura 10, não há as respostas desejadas, ou seja, não há exemplo rotulado de função a ser aprendida. A rede cria novas classes, decodificando as características da entrada.

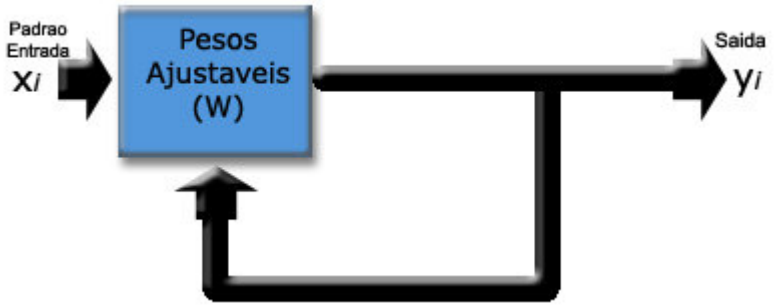


Figura 10 – Aprendizagem Não Supervisionada

Aprendizagem por Reforço *"Reinforcemment Learnig"*

Semelhante a aprendizagem supervisionada, ou seja, existe um objetivo mas não existe uma resposta desejada para cada entrada. Há uma realimentação, que avalia a saída como "boa" ou "ruim". Seu objetivo é maximizar a quantidade de "reforço positivo".

As correções dos pesos sinápticos das redes ocorrem num ciclo, onde ciclo seria a apresentação de todos os pares de treinamento e podem ser feitas de dois modos:

Modo Padrão – A correção dos pesos é feita a cada apresentação de pares de treinamento, baseada somente no erro de cada par apresentado. Então teríamos tantas correções quanto o número de pares apresentados a cada ciclo (época – "epochs").

Modo Batch – As correções dos pesos sinópticos é feita por ciclo. Todos os pares de treinamento são apresentados à rede, seu erro médio é calculado e a partir deste erro é feita a correção dos pesos.

2.2.4. Recuperação de Dados

Quando uma rede é treinada e seus pesos sinópticos forem armazenados, a rede neural pode executar várias tarefas. Estas tarefas são influentes na escolha do algoritmo de aprendizagem. Abaixo relacionamos algumas tarefas de aprendizagem que podemos aplicar ao uso das redes neurais.

Associações de Padrões – A associação pode assumir duas formas: auto-associação e heteroassociação. Na auto-associação, representada pela Figura 11, a

rede recupera um padrão original a partir de uma descrição parcial ou ruidosa deste padrão original, isto é, assumindo que um conjunto de padrões tenha sido armazenado. Na heteroassociação, representada pela Figura 12, a rede neural armazena a assossiação entre um par de padrões, recuperando um padrão diferente do padrão de entrada.[1].

Figura 11 – Auto-associação

Figura 12 - Heteroassociação

Reconhecimento de Padrões – Nós somos capazes de reconhecer um rosto familiar mesmo que esta pessoa esteja um pouco diferente, ou uma voz de uma pessoa conhecida no telefone, mesmo que a ligação não esteja perfeita, ou mesmo identificar pela visão se uma fruta está madura ou não. Esta capacidade se deve ao fato de realizar o reconhecimento de padrões através de um processo de aprendizagem, isto também ocorre com as redes neurais. A Figura 13 ilustra o reconhecimento de Padrões ou classificação que seria um caso especial de heteroassociação, onde a rede neural fornece dentro de um número pré-determinado, a classe ou categoria a qual o padrão de entrada pertence.

Figura 13 – Reconhecimento de Padrões

Aproximação de Funções – Nesta tarefa de aprendizagem, a rede neural deverá fornecer uma função F(x) que se aproxima de uma função desconhecida f(x). Considere um mapeamento não-linear de entrada-saída representado por $d = f(x)$, onde "x" é o vetor de entrada e "d" o vetor de saída. Após o treinamento a rede neural deverá aproximar a função f(x) através de uma função F(x), de modo que:

$$|F(x) - f(x)| \prec \varepsilon \text{ para todo "x".}$$

Sendo ε (erro) um número positivo e pequeno.

A Figura 14 mostra um esquema de aprendizagem supervisionada para a aproximação de funções, sendo o vetor de treinamento o par [xi, di].

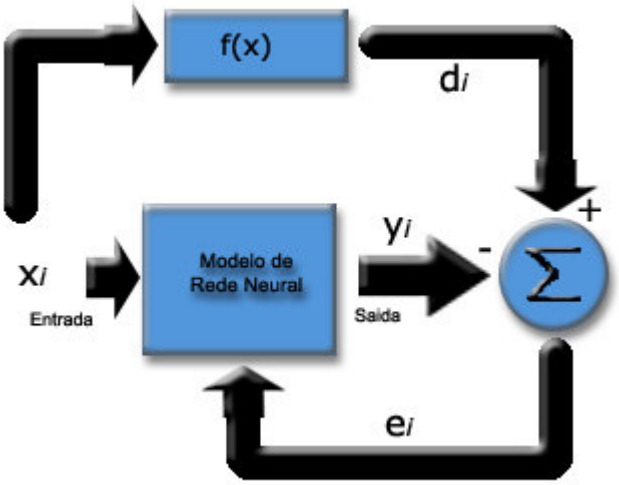


Figura 14 - Identificador de Sistema

2.3. Perceptron

O aprendizado supervisionado do modelo de rede Perceptron, proposto por Frank Rosemblat (1958), foi à estrutura mais simples de uma RNA, usada com sucesso em classificação de padrões.

O Perceptron de Rosemblat é construído em torno de um neurônio não-linear (modelo de McCullosh & Pitts), tendo as seguintes características básicas: topologia constituída de apenas uma camada com um ou mais neurônios; valores de entrada e saída em binários; função de ativação degrau; algoritmo de aprendizado supervisionado e regra de propagação definida como $NET_j = \sum x_x . w_{ij} + b_j$.[2]

Na Figura 2 vimos um neurônio artificial, que para representar um perceptron de Rosemblat, à função de ativação deve ser a função degrau bipolar que resulta em uma saída igual a +1 se o valor de NET_j for positivo e saída igual a -1, se o resultado de NETj for negativo. A supervisão da rede é feita por d_j (resposta desejada) e o sinal de erro correspondente, geralmente ($d_j - s_j$). Os pesos sinápticos são atualizados em cada iteração até que o erro seja satisfatoriamente pequeno, onde geralmente utiliza-se a regra de aprendizado da Equação 3.

$$\Delta w_{ij} = \eta.(d_j - s_j).x_i$$

Equação 3 – Delta Rule

Na equação anterior η representa a taxa de aprendizado que mede o passo adotado pela rede em busca do ponto de convergência (mínimo global). A taxa de aprendizado (η) é uma constante positiva, varia entre 0 e 1, que corresponde à velocidade do aprendizado, porém quando η excede um valor crítico, o algoritmo se torna instável (diverge).

A equação 3, é o método mais utilizado, conhecida como “Delta Rule” e sua equação é obtida utilizando o método do Gradiente Descendente (Equação 4) e o Erro Médio Quadrático (Equação 5). Esta regra é conhecida como algoritmo de ”Least Mean Square” (LMS), regra de Windrow-Hoff ou regra ADALINE, trata-se de uma generalização do Perceptron, aplicadas as entradas e saídas contínuas.

$$\Delta w_{ij} = \frac{\delta E_j}{\delta w_{ij}}$$

Equação 4 – Gradiente Descendente

$$E_j = \frac{1}{2}\left(d_j - s_j\right)^2$$

Equação 5 – Erro Médio Quadrático

A figura ilustra um Perceptron classificador de duas variáveis de entrada X1 e X2 para qual a fronteira de decisão toma a forma de uma linha reta. Quando o ponto estiver acima da linha de fronteira é atribuído à classe C1, caso o ponto esteja abaixo é atribuído à classe C2.

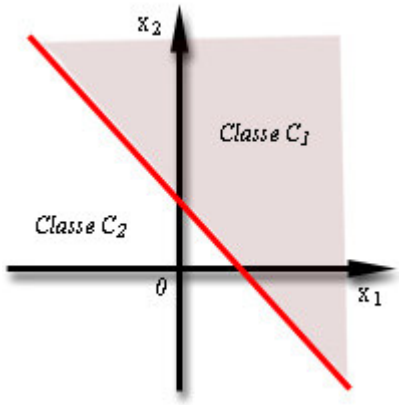


Figura 15 – Perceptron Classificador Bidimensional

2.4. Back Propagation

Responsável pelo renascimento da RNA, este algoritmo permitiu que as redes neurais de múltiplas camadas apresentassem capacidade de aprendizado. O Back Propagation, baseado no modelo de aprendizado supervisionado, retropropaga o erro da camada de saída até a camada de entrada, permitindo a atualização dos pesos sinápticos entre as camadas intermediárias (ocultas). Os modelos de RNAs mais utilizados ultimamente são os baseados no "*Perceptron de Rosemblat*", mas com múltiplas camadas, chamadas de Perceptron Multicamadas ou "Multi Layer Perceptron.", treinadas com o algoritmo Back Propagation.

Nas redes Perceptron Multicamadas, cada camada tem uma função específica. A camada de entrada apresenta os padrões a redes e não efetua nenhum

cálculo. As camadas intermediárias que funcionam como extratoras de características, codificam estas características (pesos sinápticos), transferindo estímulos para camada de saída que constrói o padrão de resposta da rede. O número de camadas intermediárias e o número de neurônios artificiais em cada camada dependem da característica do problema a ser representado.

Cybenko [5] provou, a partir do "Teorema de Kolmogoroff", que as redes Perceptron Multicamadas é um aproximador universal, pois uma camada intermediária é suficiente para aproximar qualquer mapeamento contínuo do hipercubo [-1,+1] no intervalo (-1,+1). E que duas camadas, no máximo, com um número suficiente de neurônios artificiais, podem reproduzir quaisquer mapeamentos [2].

Não há uma regra que determine os parâmetros da(s) camada(s) intermediária(s), confiavelmente, os pesos são ajustados de acordo com o desempenho da rede e depende do profissional que está implementando uma RNA.

Durante o treinamento com o algoritmo do Back Propagation, a rede opera em uma seqüência de dois passos. O primeiro consiste na apresentação dos padrões a rede, os estímulos fluem da camada de entrada, passando pela(s) camada(s) intermediária(s), até que uma resposta seja produzida pela camada de saída (Feed-Forward). No segundo passo a saída é comparada com os valores desejados, então é calculado o erro (ε), caso este erro seja maior que um valor de tolerância, ele será retropropagado a partir da camada de saída até a camada de entrada, modificando os valores dos pesos sinápticos das camadas intermediárias (Feed backward).

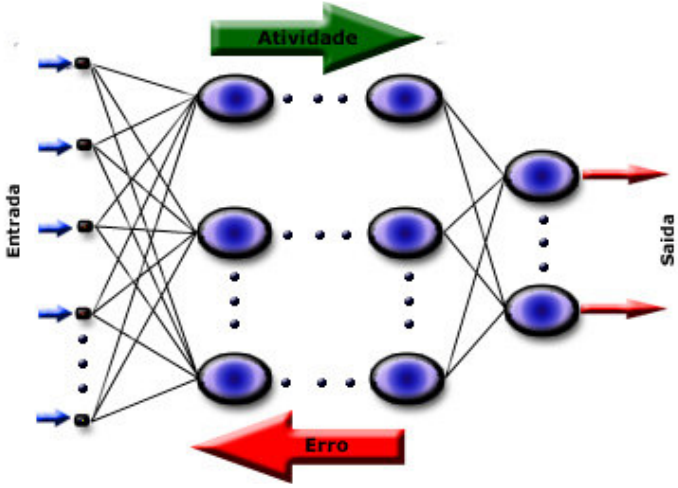


Figura 16 – Treinamento Back Propagation

O algoritmo de "Back Propagation" trabalha com uma variação da Regra Delta (Delta Rule), chamada de "Regra Delta Generalizada", que é apropriada para redes multicamadas. Podemos ver a seguir todos os passos deste algoritmo.

1 – Inicializar todos os pesos aleatoriamente com valores pequenos ($W_{ij} \leq 0.1$).

2 – Apresentar à rede os pares de treinamento(x,d).

3 – Calcular as respostas de cada neurônio artificial ou elemento processador.

4 – Calcular o erro (E_p) entre a resposta desejada (d) e o padrão de saída da rede(s_j). Se o valor do erro (Equação 6) for igual ou menor que a tolerância para todos os elementos processadores, voltar ao passo (2), caso contrário, calcular o valor do sinal de erro para atualização de cada camada.

$$E_j = \frac{1}{2}\sum_{i=1}^{n}\left(d_j - s_j\right)^2$$

Equação 6 - Erro Médio Quadrático Total

$$s_j = f\left(NET_j\right) = f\left(\sum_{i=1}^{N} s_i.w_{ij} + b_j\right)$$

$$\frac{\partial E_p}{\partial w_{ij}} = \frac{\partial E_p}{\partial f(NET)} \times \frac{\partial f(NET)}{\partial w_{ij}}$$

$$\frac{\partial (NET)}{\partial w_{ij}} = \frac{\partial f\left(\sum_{i=1}^{N} s_i.w_{ij} + b_j\right)}{\partial w_{ij}} = s_i$$

$$\frac{\partial E_p}{\partial w_{ij}} = \frac{\partial E}{\partial (NET)}.s_i$$

Então o sinal do erro, será definido por:

$$e_j \cong -\frac{\partial E_p}{\partial (NET_j)}$$

Equação 7 - Sinal do Erro

e a atualização dos pesos será por:

$$\Delta w_{ij} = -\eta.e_j.s_s$$

Equação 8 - Função de custo dos Pesos

$$W_{novo} = W_{anterior} + \Delta w_{ij}$$

5 – Definição do cálculo do erro para cada camada.

6 – Calcular o valor do sinal do erro (E_j) e atualização dos pesos sinápticos na camada de saída (j).

$$e_{js} \cong -\frac{\partial E_p}{\partial (NET_j)} = -\frac{\partial E_p}{\partial (s_j)} \times \frac{\partial (s_j)}{\partial (NET_j)}$$

$$s_j = f(NET_j)$$

$$\frac{\partial (s_j)}{\partial (NET_j)} = f'(NET_j)$$

$$e_{js} \cong -\frac{\partial E_p}{\partial (NET_j)} = -f'(NET_j) \times \frac{\partial E_p}{\partial (s_j)}$$

sendo,

$$E_p = \frac{1}{2}\sum_{j=1}^{n}(d_j - s_j)^2$$

temos,

$$\frac{\partial E_p}{\partial (s_j)} = -(d_j - s_j)$$

$$e_{js} = f'(NET_j) \times (d_j - s_j)$$

então,

$$\Delta w_{ij} = \eta . e_{js} . s_i = \eta . f'(NET_j) . (d_j - s_j) . s_i$$

Equação 9 - Função de custo da camada de saída

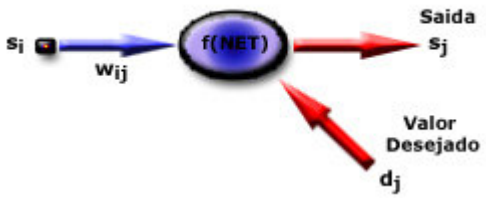


Figura 17 - Elemento processador j da camada de saída

7 - Calcular o valor do sinal do erro (E_{je}) e atualização dos pesos sinápticos da(s) camada(s) intermediária(s).

$$e_{je} \cong -\frac{\partial E_p}{\partial (NET_j)} = -\frac{\partial E_p}{\partial (s_j)} \times \frac{\partial (s_j)}{\partial (NET_j)}$$

$$s_j = f(NET_j)$$

$$\frac{\partial (s_j)}{\partial (NET_j)} = f'(NET_j)$$

$$E_p = f(s_j) = ?$$

Supor que k pertença a camada de saída

$$E_p = \frac{1}{2}\sum_{k=1}^{n}(d_k - s_k)^2$$

$$\frac{\partial E_p}{\partial(s_j)} = \frac{\partial(\frac{1}{2}\sum_{k=1}^{n}(d_k - s_k)^2)}{\partial(s_j)}$$

$$= 2\times\frac{1}{2}\times\left[\frac{1}{2}\sum_{k=1}^{n}(d_k - s_k)\right]\times(-1)\times\frac{\partial(s_k)}{\partial(s_j)}$$

$$= -\left[\sum_{k=1}^{n}(d_k - s_k)\right]\times\frac{\partial(s_k)}{\partial(NET_k)}\times\frac{\partial(NET_k)}{\partial(s_j)}$$

$$\frac{\partial(s_k)}{\partial(NET_k)} = f'(NET_k)$$

$$\frac{\partial(NET_k)}{\partial(s_j)} = w_{ij}$$

$$e_k = (d_k - s_k).f'(NET_k)$$

$$e_{je} = -\left[-\sum_{k=1}^{n} e_k \times w_{jk}\right]\times f'(NET_j)$$

$$e_{je} = f'(NET_j)\times\left[\sum_{k=1}^{n} e_k \times w_{jk}\right]$$

$$\Delta w_{ij} = \eta.e_{je}.s_i$$

$$\Delta w_{ij} = \eta\times\left[f'(NET_j)\times\sum e_k \times w_{jk}\right]\times s_i$$

Equação 10 - Função de custo da camada escondida

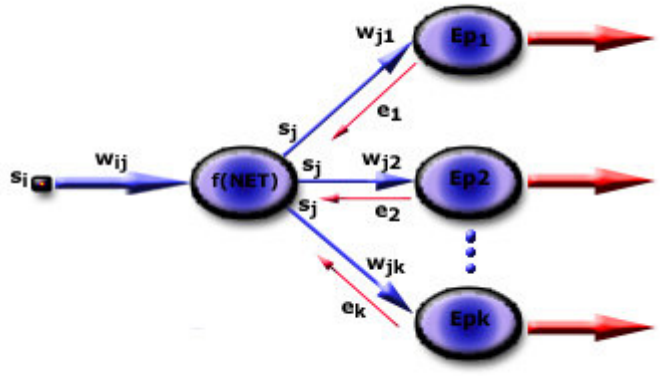


Figura 18 – Elemento processador j da camada intermediária

A Regra Delta Generalizada funciona quando são utilizados nas redes, neurônios artificiais com função de ativação diferencial e não decrescente. Os erros (e_{js}, e_{je}) são calculados em função da derivada de função de ativação, sendo a função sigmóide a mais utilizada.

Apesar do grande êxito, quando bem aplicado, o algoritmo de retropropagação apresenta algumas dificuldades, onde destacamos:

Tamanho da Rede . A definição do número de camada escondida e o número de processadores em cada camada são importantes, pois há um compromisso entre a "Convergência" e a "Generalização". Se a rede for pequena, não será capaz de armazenar todos os padrões do conjunto de treinamento (Convergência). Caso a rede seja muito grande, pode acontecer de responder incorretamente a padrões nunca vistos, ou seja, a rede ficaria específica aos padrões de treinamento. Uma forma de atacar este problema seria obter um tamanho fixo do conjunto de treinamento e obter a melhor arquitetura da rede, ou também utilizar algumas métricas, como por exemplo a Equação 11.

$$N_e \leq \frac{N \times \varepsilon_{train}}{N_{in} + N_{out}}$$

Equação 11 - Baum_Haussler

Onde,

N_e – Número de elementos processadores na camada escondida;

N – Número de padrões;

ε_{train} – Erro permitido no treinamento (tolerância);

N_{in} – Número de entradas;

N_{out} – Número de elementos processadores na camada de saída.

Baseados em algumas métricas para determinar o número de processadores na camada escondida, para alcançar uma boa generalização, nunca escolha o número de processadores nesta camada maior que 2 vezes o número de entrada da rede, assegure-se que tenha pelo menos $\frac{1}{\varepsilon}$ vezes mais padrões do que o número de pesos (W). [1]

Paralisia da Rede. A rede pode operar em uma região onde a derivada da função de ativação é nula ou infinitesalmente pequena (Figura 19), pois se a

derivada for nula o ajuste nos pesos sinápticos serão nulos, como podemos ver pelas equações 9 e 10. Para evitar este problema devemos escolher pesos e bias distribuídos dentro de um intervalo pequeno. Os neurônios devem operar na sua região linear e o número de neurônios artificiais nas camadas escondida deve ser pequeno[2].

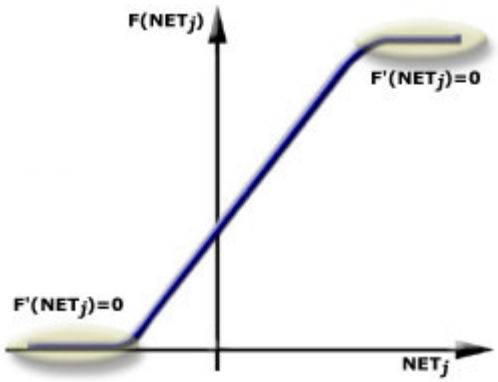


Figura 19 – Região de Paralisia

Mínimo Local. O algoritmo Back Propagation utiliza o método do gradiente descendente, ajustando seus pesos a um mínimo. Como o gradiente descendente requer que sejam tomados passos infinitesimais, o valor ideal da taxa de aprendizado (η), não deve ser muito pequeno pois resulta um tempo de treinamento extremamente longo e caso encontre um mínimo local o erro do conjunto de treinamento estaciona em um valor maior que a tolerância. E este valor também não deve ser muito grande, pois apesar de acelerar o treinamento, aumentando o valor de atualização dos pesos, pode causar oscilações , conseqüentemente nunca chegando a um mínimo global.

Uma maneira de aumentar a taxa de aprendizado (η), diminuindo a possibilidade de oscilações, é incluir o termo "Momentum" (α) nas atualizações dos pesos sinápticos (Equação 12)

$$\Delta w_{ij}(t+1) = \eta \times s_j \times e_j + \alpha \times \Delta w_{ji}(t)$$
Equação 12 - Função de custo com o termo Momentum

O termo "Momentum" (α) é um constante, geralmente menor que 0.9, que determina o efeito das mudanças passadas dos pesos na direção atual do movimento do peso, ou seja, leva em consideração o efeito de mudanças anteriores de pesos na direção do movimento atual.

2.5. Aceleração da Convergência do Back Propagation

A taxa de convergência do algoritmo Back Propagation tende ser, relativamente baixa., tornando este algoritmo lento no aspecto computacional. Por exemplo, na superfície de erro podemos ter uma região razoavelmente plana, conseqüentemente a derivada do erro seria pequena, necessitando de muitas iterações do algoritmo para produzir um ajuste significativo aplicado aos pesos sinápticos Apresentamos a seguir duas técnicas heurísticas que apresentam normas que podem ser usadas na aceleração do algoritmo Back Propagation.

Heurística 1 - Utilização de parâmetros de taxa de aprendizado diferente para cada peso sinápticos ajustável de rede. Um valor de taxa de aprendizagem utilizado para um ajuste de um determinado peso talvez não seja apropriado para o ajuste de outros pesos sinápticos.

Heurística 2 - O parâmetro de taxa de aprendizagem pode variar a cada iteração, pois a superfície de erro se comporta de forma diferente ao longo de diferentes regiões de uma única dimensão de peso.

2.6. Validação Cruzada

O objeto do algoritmo Back Propagation é codificar os pesos sinápticos e bias de um perceptron multicamadas, tornando a rede bem-treinada de modo que possa aprender sobre os padrões de treinamento o suficiente para generalizar sobre dados não apresentados ao modelo.

Uma ferramenta atrativa da estatística conhecida como “Validação Cruzada”, torna-se atraente como orientador. O conjunto de dados é dividido aleatoriamente em dois conjuntos, um para treinamento e o outro para teste do modelo, sendo o conjunto de treinamento subdividido em dois subconjuntos distinto: o subconjunto de estimação (seleciona o modelo) e o conjunto de validação que é utilizado para testar ou validar o modelo [2].

Caso os parâmetros da rede neural, com melhor desempenho, acabem ajustando excessivamente o subconjunto de validação o conjunto de teste é utilizado para medir o desempenho do modelo.

A determinação de como particionar o conjunto de treinamento em dois subconjuntos é uma questão importante. Um parâmetro "r", com intervalo de 0 a 1, determina a partição do conjunto de treinamento, considerando que o tamanho dos padrões disponíveis tenha tamanho "N", (1-r) x N padrões são destinados ao subconjunto de estimação e r x N destinado ao subconjunto de validação. Um estudo descrito por Keans em 1996, onde várias propriedades qualitativas de um r ótimo foram identificadas, determina que r = 0,2 é um valor apropriado para determinar o tamanho do subconjunto de validação [2].

No treinamento de um perceptron de múltiplas camadas com o algoritmo de Back Propagation é muito difícil perceber qual o melhor momento de encerrar o treinamento, para que o modelo apresente uma boa generalização, pois pode acontecer de a rede acabar sendo ajustada apenas para os dados do treinamento. Usando a técnica da validação cruzada o subconjunto de estimação é usado para treinar a rede, sendo que periodicamente o treinamento é interrompido e a rede é avaliada com o subconjunto de validação, este processo de treinamento seguido de validação é conhecido como " Método de Parada Antecipada".

A Figura 20 mostra as curvas de aprendizagem medidas em relação aos subconjuntos de estimação e validação. A curva de aprendizagem do subconjunto de validação decresce para um mínimo e então começa a crescer enquanto o treinamento continua decrescendo. O ponto de parada antecipada determina a partir de quando a rede irá se ajustar em relação somente aos dados de estimação, sugerindo que a rede esta se ajustando ao ruído contido nos dados de treinamento. Esta heurística sugere que este ponto mínimo seja utilizado como critério para encerrar o treinamento [2].

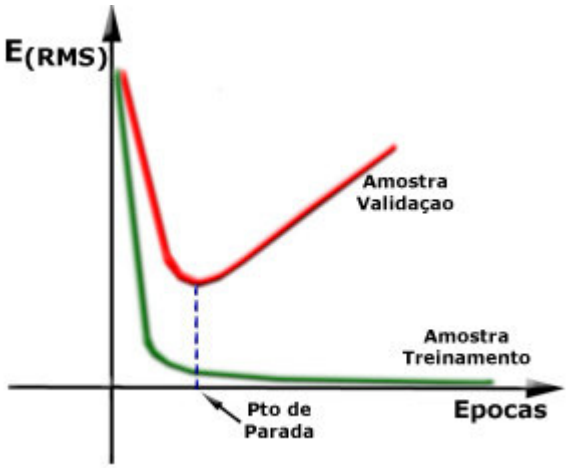


Figura 20 - Treinamento com parada antecipada

Amari et al. (1996) apresentou a teoria estatística do fenômeno de excesso de ajuste, sugerindo uma precaução na utilização do método de treinamento com parada antecipada, onde foram identificados dois modos de comportamentos da rede que depende do tamanho do conjunto de treinamento [2]:

Modo não-assintótico. Neste modo o conjunto de treinamento (N) é menor que o número de parâmetros a serem ajustados (W). O método de treinamento com parada antecipada apresenta um desempenho melhor de generalização do que o treinamento onde o conjunto completo de padrões é utilizado e não há interrupção no treinamento (treinamento exaustivo).

A Equação 13 mostra o valor ótimo de r que determina a partição dos dados de tamanho N<W.

$$r_{ótimo} = 1 - \frac{1}{\sqrt{2W}}$$

Equação 13 - Parâmetro $r_{ótimo}$

Modo assintótico. Onde N> 30W. Neste modo o treinamento com parada antecipada apresenta um desempenho de generalização pequeno em relação ao treinamento exaustivo, ou seja, o treinamento exaustivo é satisfatório para este caso.