Algoritmos Genéticos e Redes Neurais

Prof. Pasteur Ottoni de Miranda Junior – DCC PUC MG
Disponível em www.pasteurjr.blogspot.com

1-Algoritmos genéticos

Os algoritmos genéticos (AG) são uma família de algoritmos computacionais inspirados na teoria da evolução, que incorporam conceitos semelhantes aos de cromossomo, seleção, reprodução e mutação para resolver, principalmente, problemas de otimização.
O conjunto de estados possíveis de um problema é denominado *população.* Cada estado constitui um indivíduo e é codificado como um *cromossomo.* Usualmente, os cromossomos são representados por vetores binários cujos elementos indicam a presença (1) ou ausência (0) de uma característica. Entretanto, representações por meio de vetores inteiros, reais ou de caracteres, também são possíveis. A combinação dessas características determina o aspecto (fenótipo) final do indivíduo.
Os indivíduos se reproduzem, gerando novas populações. Nessas populações os indivíduos são selecionados de forma que os mais aptos têm mais chance de se reproduzir, transmitindo sua herança genética às populações seguintes.. Para quantificar essa aptidão, uma *função de avaliação* é utilizada, de forma que retorne valores maiores para os melhores indivíduos. Essa função depende do problema abordado. No processo de reprodução, tal como no mundo real, aplicam-se aos cromossomos operações de *crossing-over* e mutação, que vão gerar os novos indivíduos.

Seleção

A seleção é feita de forma a privilegiarem-se indivíduos mais aptos (maiores valores da função de avaliação). Um dos métodos mais populares é o denominado *método da roleta,* em que cada cromossomo, em uma roleta, possui uma área proporcional a sua aptidão. O "ponteiro" da roleta é impulsionado e a área apontada pelo mesmo ao parar, corresponde a um indivíduo selecionado. Assim, a área a_i de um indivíduo i que possui uma determinada $nota_i$ proporcional a sua aptidão é dada por:

$$a_i = \frac{nota_i}{\sum_{j=1}^{N} nota_p}$$

Onde N é o número de indivíduos a selecionar (número de vezes em que o "ponteiro" é impulsionado).

Contudo, se estas áreas forem muito próximas, pode não haver favorecimento dos mais aptos. (Pádua, 2000), sugere que a fatia seja determinada pela posição que o cromossomo ocupa no *ranking* de todas os notas. A fatia correspondente a cada cromossomo é definida pela escolha de um valor entre 0,0 e 1,0. Cada indivíduo, em ordem crescente do *ranking,* ocupa, da área total da roleta ainda não ocupada, uma fatia proporcional ao valor escolhido. O indivíduo de menor *ranking* ocupará a área que restou na roleta.

De forma a preservarem-se os melhores indivíduos, pode-se passá-los, sem cruzamento e alteração, para a geração seguinte. Este procedimento, denominado *elitismo*, melhora o desempenho do algoritmo, pois evita que tais indivíduos sejam destruídos no cruzamento ou na ocorrência de mutação. A % dos indivíduos restantes que se cruzam é denominada *taxa de cruzamento.* Em geral essa elite é constituída por aproximadamente 2% da população. O restante da população efetua o cruzamento. Entretanto, em determinadas situações, o elitismo pode gerar populações muito homogêneas, com bons indivíduos, mas com chances menores de produzir indivíduos ainda melhores.

Reprodução

O processo de reprodução combina cromossomos entre si, de forma a produzir indivíduos diferentes na próxima geração, para consequentemente aumentar a possibilidade de aparecimento de melhores variações. Esse processo ocorre por meio de dois operadores sucessivos:

- Cruzamento. É o denominado *crossing-over,* em que dois cromossomos fornecem aos filhos partes extraídas a partir de pontos de corte escolhidos aleatoriamente. Pode haver um único, ou múltiplos cortes. Uma outra forma de *crossing-over* utiliza uma máscara (vetor) de bits, com o mesmo tamanho dos cromossomos, para determinar que genes serão herdados do pai e da mãe.

Nessa máscara, se o í-ésimo elemento for 1, o í-ésimo gene do filho 1 será igual ao do pai, se for 0, será igual ao da mãe. O inverso ocorre no filho 2. A Figura 34 exibe esquematicamente esses processos.

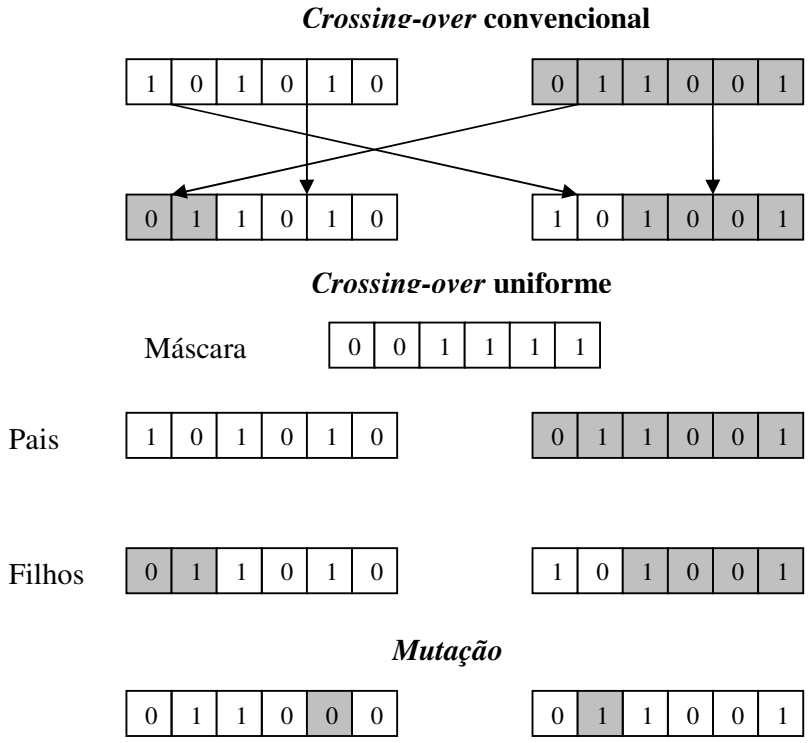

Figura 34-Operações de *crossing-over* e mutação (Inspirado em (Pádua, 2000))

- Mutação. Consiste em uma mudança aleatória na cadeia de genes dos filhos, que ocorre após a operação de *crossing-over.* É a responsável pela variabilidade genética na população e ocorre em uma freqüência determinada pela *taxa de mutação.* A Figura X10 exibe o funcionamento desse operador.

O algoritmo genético

Criar um conjunto Inicial de cromossomos;
Repetir
    Utilizar uma função de avaliação para atribuir uma nota a cada cromossomo;

Selecionar os cromossomos mais aptos;

Se for realizar elitismo

De acordo com a taxa de cruzamento, retirar os mais aptos para passar diretamente à geração seguinte;

Sobre os cromossomos restantes:

Aplicar *crossing-over;*

Aplicar mutação considerando-se a taxa de mutação;

Senão

Sobre os cromossomos mais aptos:

Aplicar *crossing-over;*

Aplicar mutação considerando-se a taxa de mutação;

Até que sejam obtidos cromossomos adequados, ou um certo número de mutações.

Desempenho do algoritmo

A escolha de parâmetros do algoritmo pode influenciar no desempenho e no comportamento do mesmo e devem ser determinados, geralmente por tentativa e erro, em função das características e peculiaridades do problema em questão. Segundo (Jong, 1980) e (Pádua, 2000) os parâmetros mais importantes a serem considerados são os seguintes:

- *Tamanho da população:* Pequenas populações comprometem o desempenho do algoritmo por representarem um espaço de busca pequeno. Grandes populações diminuem a chance de ocorrência de mínimos locais, mas devem ser usadas com parcimônia, por demandarem muito esforço computacional.
- *Taxa de cruzamento:* Um valor maior acarreta maior variabilidade genética. Entretanto, se for muito alta, aumenta-se a chance de eliminação de bons indivíduos. Se for muito baixa, a população tende a estagnar, diminuindo-se a chance de produção de bons indivíduos.

- *Taxa de mutação:* Uma baixa taxa de mutação previne estagnações de determinadas posições. Quando muito alta, introduz-se uma aleatoriedade excessiva no processo de busca.
- *Intervalo de geração:* Controla a porcentagem da população a ser substituída na próxima geração. Um valor alto aumenta a chance de perda de bons indivíduos. Um valor muito baixo torna o algoritmo lento, pois pode ser necessário um grande número de gerações até convergir.

2.Redes Neurais Artificiais

2.1-Definição

Segundo (Pádua, 2000), Redes Neurais Artificiais (RNAs) são sistemas paralelos distribuídos, constituídos por unidades (nós) de processamento simples que modelam um neurônio biológico, o *neurônio artificial*, capazes de aplicar funções matemáticas simples a dados recebidos em sua entrada. Este modelo é denominado MCP, por ter sido proposto por McCulloch e Pitts, em 1943.

A Figura 26 exibe um esquema do neurônio MCP. Ele é constituído por um vetor x de entradas $x_1, x_2, ..., x_n$ (que representam os dendritos de um neurônio biológico) e de uma saída y (que representa o axônio de um neurônio biológico). As sinapses de um neurônio biológico são representadas por um vetor w de pesos $w_1, w_2, ..., w_n$, acopladas às entradas do neurônio. A saída y de um neurônio MCP é ativada através da aplicação de uma *função de ativação f*, que recebe usualmente como parâmetro uma combinação linear *in* dos sinais de entrada x_i, da seguinte maneira:

$$in = \sum_{i=1}^{n} w_i x_i \quad (1)$$

A saída y é obtida então por

$$y=f(in) \quad (2)$$

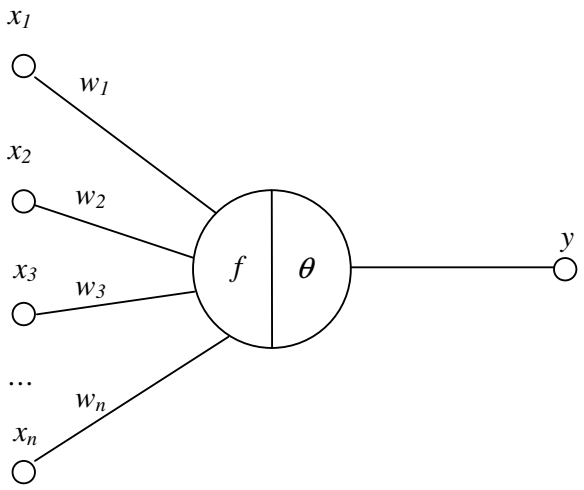

Figura 26-Esquema de um neurônio artificial

Podem ser utilizadas diferentes funções de ativação, mas as mais comuns são as seguintes:

- A função degrau, $\begin{cases} 1 & \sum w_i x_i \geq \theta \\ 0 & \sum w_i x_i < \theta \end{cases}$ onde θ é o limiar a partir do qual o neurônio dispara.
- A função sigmóide $f(x) = \frac{1}{1+e^{-\alpha x}}$

Ao utilizar-se a função degrau, a condição de disparo de um nodo MCP é

$$\sum_{i=1}^{n} w_i x_i = \theta$$

Para efeito de simplificação pode-se fazer

$$\sum_{i=1}^{n} w_i x_i - \theta = 0$$

Isto equivale a adicionar às entradas da rede uma entrada com valor fixo $x_i = 1$ com um peso de valor $-\theta$. Assim tem-se

$w=\{ -\theta, w_1, w_2, ..., w_n \}$

$x=\{ 1, x_1, x_2, ..., x_n \}$

$$\sum_{i=1}^{n} w_i x_i = 0$$

Segundo (Pádua, 2000), a função degrau restringe o uso do neurônio MCP à solução de problemas ditos *linearmente separáveis,* ou seja, aqueles cuja solução é obtida pela separação do espaço em duas regiões por meio de uma reta (se houver apenas duas entradas) ou um hiperplano (para n entradas). Se considerarmos apenas duas entradas x_1 e x_2 com pesos w_1 e w_2 e limiar θ, a condição de disparo do neurônio é $w_1 x_1 + w_2 x_2 = \theta$, que pode ser descrita no formato de uma equação de reta $x_2 = -(w_1 / w_2) x_1 + (\theta / w_2)$. A Figura 27 abaixo exibe a solução para os problemas das portas lógicas E e OU, com as respectivas retas-solução. Conforme pode ser visto, tais retas dividem o plano em duas regiões, evidenciando o caráter linearmente separável dos problemas. Entretanto, a Figura 28 mostra que o problema do OU-exclusivo (XOU) não é linearmente separável, não havendo portanto, solução para a equação acima nessas condições.

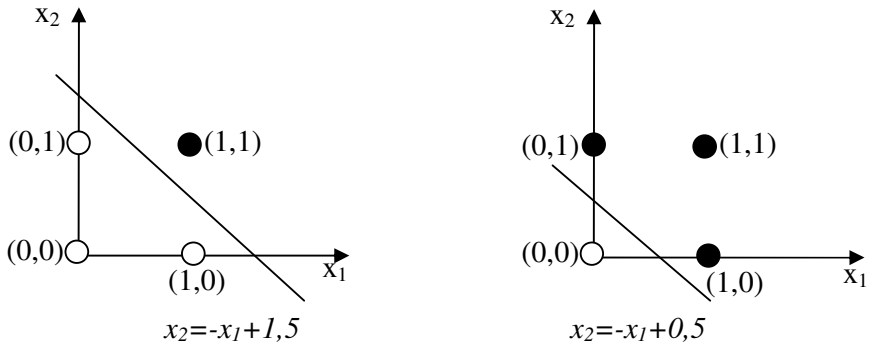

Figura 27 – E e OU lógicos são linearmente separáveis-Fonte: (Russel, 2004)

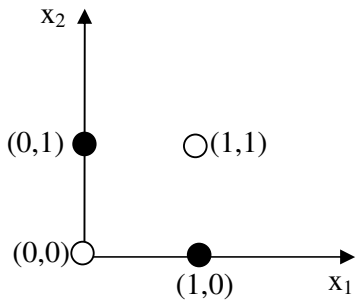

Figura 28-OU-exclusivo não permite a colocação de uma reta separando regiões de 1 e 0- Fonte: (Russel, 2004)

RNAs são muito utilizadas para classificação de um conjunto de dados fornecidos em sua entrada. O chamado *treinamento,* realizado por meio de *algoritmos de aprendizado,* permite ajustar os pesos das conexões, de forma que os dados de entrada sejam classificados discretamente na saída . Os algoritmos para aprendizado de redes neurais são classificados em dois tipos:

-Algoritmos para treinamento supervisionado: o treinamento é feito por meio de conjuntos de treinamento, ou seja, exemplos de entradas/saídas fornecidos à rede, que ajusta seus pesos com base neles;

-Algoritmos de treinamento não-supervisionado: tais algoritmos não demandam conjuntos de treinamento para aprendizado.

2.2- O Perceptron

Segundo (Pádua,2000) o trabalho de McCulloch e Pitts focou-se no modelamento matemático do neurônio biológico. Somente com o trabalho de Frank Rosemblatt, em 1958, o conceito de aprendizado de redes neurais foi introduzido. Rosemblatt propôs o denominado modelo *perceptron,* uma estrutura de rede contendo nós MCP e com um algoritmo de aprendizado.

O perceptron é composto por unidades de entrada (sensores, receptores de dados), unidades de associação (nós MCP com pesos fixos pré-determinados) e unidades de resposta (uma única camada de nós MCP a serem devidamente treinados).

(Pádua, 2000) demonstra que algoritmos de treinamento de perceptrons sempre convergem em tempo finito, para problemas linearmente separáveis. O algoritmo de treinamento para um nodo de um perceptron simples é descrito a seguir (Pádua, 2000).

O algoritmo de treinamento do perceptron

Algoritmos de treinamento supervisionado de RNAs normalmente se focam no ajuste de valores de w que permitam a classificação proposta no conjunto de treinamento constituído por pares de vetores entrada-saída (x^i, y^i). Em cada iteração t do algoritmo obtém-se um incremento Δw, adicionado ao vetor w obtido na interação anterior: $w(t+1)=w(t)+\Delta w$. Este Δw pode ser obtido de forma a se reduzir o chamado *erro quadrático médio* $E = \frac{1}{2}(Err)^2$, onde $Err = y_d - y_a$, y_d é a saída desejada e $y_a = f(\sum_{i=1}^{n} w_i x_i)$, a saída atual do perceptron.

Para reduzir o erro quadrático médio, utiliza-se o declínio de gradiante de E, que consiste em se calcular a derivada parcial de E em relação a cada peso. Assim tem-se (Russel, 2004):

$$\frac{\partial E}{\partial w_i} = \frac{\partial E}{\partial Err}\frac{\partial Err}{\partial w_i}$$

$$\frac{\partial E}{\partial w_i} = Err\frac{\partial Err}{\partial w_i}$$

$$\frac{\partial E}{\partial w_i} = Err\frac{\partial}{\partial w_i}(y_d - f(\sum_{i=1}^{n} w_i x_i))$$

Finalmente:

$$\frac{\partial E}{\partial w_i} = -Err \quad f'(\sum_{i=1}^{n} w_i x_i) \quad x_i$$

Entretanto, a regra de declínio de gradiente determina que o ajuste Δw deva ser feito na direção contrária à de $\frac{\partial E}{\partial w_i}$ ou seja , $\Delta w \quad \alpha - \frac{\partial E}{\partial w_i}$,podendo ser escrito como $\Delta w = -\eta \frac{\partial E}{\partial w_i}$, onde a constante de proporcionalidade η é denominada *taxa de aprendizado,* determinante da rapidez com que os pesos são atualizados. Assim, obtemos finalmente a expressão de atualização de pesos:

$$w(t+1)=w(t)+ \eta \quad Err \quad f'(\sum_{i=1}^{n} w_i x_i) \quad x_i$$

Se *f* for a função degrau, o termo $f'(\sum_{i=1}^{n} w_i x_i)$ pode ser omitido, pois é o mesmo para todos os pesos e sua omissão somente altera a magnitude, e não a orientação da atualização global de pesos para cada exemplo (Russel, 2004).
O algoritmo de treinamento pode ser assim formalizado (inspirado em (Pádua, 2000)):

1-Iniciar o vetor de pesos w com valores entre -1 e +1 e a taxa de aprendizado η;
2-Repetir

2.1-Para cada par entrada-saída desejada (x^i, y^i) de um conjunto de treinamento com *p* elementos $T=\{ (x^1, y^1), (x^2, y^2)... (x^p, y^p) \}$ faça:

2.1.1-Atualizar o vetor de pesos para cada um dos nodos da rede da seguinte forma:

$$y_a = f(\sum_{i=1}^{n} w_i x_i);$$

$$Err = y_d - y_a;$$

$$w(t+1)=w(t)+ \eta \quad Err \quad f'(\sum_{i=1}^{n} w_i x_i) \quad x_i;$$

Até que Err=0 (ou algum outro critério de parada), para todos os *p* elementos do conjunto de treinamento e todos os nodos da rede.

2.3- Redes Multicamadas (MLP- *Multi Layer Perceptron)*

Conforme visto anteriormente, perceptrons de uma única camada somente resolvem problemas linearmente separáveis. (Pádua, 2000) afirma que a solução de problemas

não linearmente separáveis somente é obtida com o uso de redes com uma ou mais camadas intermediárias ou escondidas. (Cybenko, 1988, 1989) demonstra que uma rede com uma única camada escondida pode implementar qualquer função contínua e que duas camadas escondidas permitem a aproximação de qualquer função. As redes do tipo *perceptron multicamadas* ou MLP (*multilayer perceptron)* caracterizam-se por apresentar a seguinte disposição de camadas (Figura 29):

- Camada de Entrada: onde os padrões de entrada são apresentados à rede;
- Camadas Intermediárias ou Escondidas: segundo (Russel, 2004), responsável pelo aumento do espaço de hipóteses que a rede é capaz de representar. Essencialmente são detectores de características (Pádua, 2000);
- Camada de Saída: onde o resultado do processamento é apresentado.

Em uma rede MLP, o processamento realizado por cada nodo é definido pela combinação linear dos processamentos realizados pelos nodos da camada anterior que estão conectados a ele (Pádua, 2000).

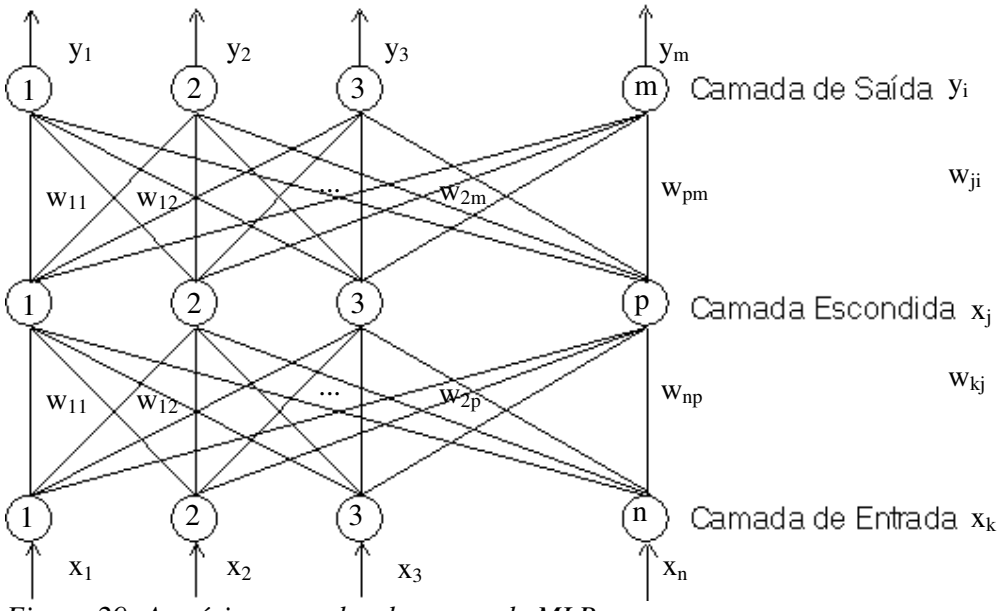

Figura 29- As várias camadas de uma rede MLP

Determinação da topologia de redes MLP

Na camada de entrada recomenda-se que o número de nodos MCP seja igual ao número de variáveis que produzem as saídas da rede MLP. Para determinar o número de neurônios na camada de saída, deve-se considerar o uso pretendido da rede. Se este for, por exemplo, classificar itens em grupos, é recomendável que haja um nodo MCP para cada grupo. Para modelagem de funções matemáticas, um neurônio é suficiente. Já para filtragem de ruídos, o ideal é que o número de neurônios na camada de saída seja igual ao da camada de entrada.

Sabemos que uma camada intermediária é suficiente para modelar qualquer função contínua e que duas camadas intermediárias modelam qualquer função. Entretanto, observa-se que problemas que requeiram duas camadas intermediárias são raros e que simplesmente não existe qualquer razão teórica para se utilizar mais que duas camadas.

A determinação do número de neurônios MCP nas camadas intermediárias é uma importante questão no projeto da arquitetura de uma RNA. Subestimar esse número pode resultar no chamado *underfitting:* a estrutura é insuficiente para manipular as entradas, dificultando sobremaneira a convergência dos algoritmos de treinamento. Por outro lado, um número excessivo de neurônios na camada intermediária usualmente gera *overfitting,* ou seja, a diminuição da capacidade de generalização. A determinação desse número depende de vários fatores, tais como:

- Número de neurônios de entrada e saída
- Número de exemplos de treinamento
- Quantidade de ruído presente nos exemplos
- Função de avaliação utilizada
- Complexidade da classificação ou da função a ser aprendida

Algumas sugestões para um ponto de partida na determinação do número de neurônios na camada intermediária são as seguintes:

- Um número intermediário entre o número de neurônios na camada de entrada e o número de neurônios na camada de saída
- 2/3 do número de neurônios na entrada mais o número de neurônios na saída

- Um número menor que o dobro de neurônios na camada de entrada (por exemplo, 2n - 1, onde n é o número de neurônios na entrada);
- Dez por cento do número de exemplos;

Entretanto, a experiência tem mostrado que a determinação do número de neurônios na camada intermediária é em essência um processo de tentativa e erro. Felizmente, existem refinamentos que podem ser aplicados à RNA, de forma a melhorar sua topologia para que possa modelar mais precisamente a classificação ou função embutida nos exemplos, mas que a torne também apta a generalizar, quando necessário, considerando-se também, inclusive, o ruído possivelmente contido nos dados, que se deseja, não seja modelado.

Segundo (Reed, 1993) o *overfitting* pode ser diminuído utilizando-se um conjunto de treinamento para modificar os pesos mediante um algoritmo de aprendizado e um *conjunto de validação* para verificar a capacidade de generalização apreendida durante o aprendizado: interrompe-se o treinamento quando o erro produzido pelo conjunto de validação começar a subir.

As denominadas técnicas de *poda (pruning)* envolvem a eliminação de nodos e pesos da estrutura da rede. Uma primeira abordagem propõe a exclusão sistemática de nodos da rede, avaliando-se em seguida o impacto no erro de saída. Se este for pequeno, mantém-se a exclusão, pois sua ausência não afeta a capacidade da mesma em modelar os dados (Reed, 1993). Uma segunda abordagem, apresentada em (Pádua, 2000), consiste em relacionar o erro quadrático médio com a norma do vetor *w,* de forma a se obter soluções com pesos de norma mínima. Durante o treinamento, começa-se com uma rede superdimensionada e removem-se os pesos irrelevantes. (Hinton, 1987) propõe uma alteração na expressão do erro quadrático médio (exibida anteriormente), que incorpora o termo correspondente à norma do vetor *w:*

$$E = \frac{1}{2}(Err)^2 + \frac{1}{2}\lambda\|w\|^2$$

Essa expressão é então minimizada, obtendo-se uma solução com pesos de norma mínima, na qual aqueles muito pequenos são eliminados. O parâmetro de regularização λ deve ser ajustado: se for muito grande, a solução para o vetor de pesos tende para w=0, se for nulo, somente a soma dos erros quadráticos é minimizada. Portanto, o desafio consiste em obter um valor intermediário de λ que seja satisfatório, entre esses extremos.

O algoritmo de treinamento backpropagation

É o algoritmo mais conhecido para treinamento de redes MLP. Trata-se de um algoritmo supervisionado que utiliza pares (entrada, saída desejada) para, por meio de um mecanismo de correção de erros, ajustar os pesos da rede (Pádua, 2000). O treinamento ocorre em duas fases, cada uma delas percorre a rede em um sentido: para frente (*forward*) em que a saída da rede é definida para determinado padrão de entrada, e para trás (*backward*), em que utiliza-se uma propagação do erro (diferença entre as saídas desejada e produzida pela rede) da camada de saída para as camadas ocultas de forma atualizar os pesos das conexões.

Em redes MLP tem-se um vetor de saídas y_i, assim, o erro quadrático médio deve ser expresso como um somatório dos erros quadráticos médios de cada saída y_i:

$E = \frac{1}{2}\sum_{i=1}^{m}(Err)^2$ onde $Err = yd_i - y_i$, yd_i é a saída desejada e $y_i = f(\sum_{j=1}^{p} w_{ji} x_j)$.

Vamos considerar inicialmente a correção dos pesos w_{ji} que conectam a camada escondida à camada de saída, ou seja, vamos obter o gradiente de E em relação aos pesos w_{ji}. Nesse caso específico, expandimos somente yi, pois todos os outros termos do somatório não são afetados por w_{ji} (Russel, 2004). Assim, nesse caso tem-se $E = \frac{1}{2}(Err)^2$.

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial Err}\frac{\partial Err}{\partial w_{ji}}$$

$$\frac{\partial E}{\partial w_{ji}} = Err\frac{\partial Err}{\partial w_{ji}}$$

$$\frac{\partial E}{\partial w_{ji}} = Err\frac{\partial}{\partial w_{ji}}(yd_i - f(\sum_{j=1}^{p} w_{ji}x_j))$$

$$\frac{\partial E}{\partial w_{ji}} = -(yd_i - f(\sum_{j=1}^{p} w_{ji}x_j)) \quad f'(\sum_{j=1}^{p} w_{ji}x_j) \quad x_j$$

Escrevemos finalmente:

$$\frac{\partial E}{\partial w_{ji}} = -x_j \Delta_i \text{, onde } \Delta_i = (yd_i - f(\sum_{j=1}^{p} w_{ji} x_j)) \quad f'(\sum_{j=1}^{p} w_{ji} x_j)$$

Analogamente ao que se considerou para o perceptron, podemos escrever então:

$$\Delta w_{ji} = -\eta \frac{\partial E}{\partial w_{ji}} = \eta x_j \Delta_i \text{, onde } \eta \text{ é a taxa de aprendizado.}$$

Consideremos agora a correção de pesos w_{kj} que conectam a camada escondida à camada de entrada. Nesse caso não podemos ignorar o somatório, pois cada saída y_i pode ser afetada por w_{kj}.

$$\frac{\partial E}{\partial w_{kj}} = \sum_{i=1}^{m} \frac{\partial E}{\partial Err} \frac{\partial Err}{\partial w_{kj}}$$

$$\frac{\partial E}{\partial w_{kj}} = \sum_{i=1}^{m} Err \frac{\partial Err}{\partial w_{kj}}$$

$$\frac{\partial E}{\partial w_{kj}} = \sum_{i=1}^{m} ((yd_{\mathrm{i}} - f(\sum_{j=1}^{p} w_{ji} x_j)) \frac{\partial}{\partial w_{kj}} ((yd_i - f(\sum_{j=1}^{p} w_{ji} x_j))))$$

$$\frac{\partial E}{\partial w_{kj}} = -\sum_{i=1}^{m} ((yd_{\mathrm{i}} - f(\sum_{j=1}^{p} w_{ji} x_j)) f'(\sum_{j=1}^{p} w_{ji} x_j) \frac{\partial}{\partial w_{kj}} (f(\sum_{j=1}^{p} w_{ji} x_j)))$$

Mas, $\Delta_i = (yd_i - f(\sum_{j=1}^{p} w_{ji} x_j)) \quad f'(\sum_{j=1}^{p} w_{ji} x_j)$

$$\frac{\partial E}{\partial w_{kj}} = -\sum_{i=1}^{m} (\Delta_i \frac{\partial}{\partial w_{kj}} (f(\sum_{j=1}^{p} w_{ji} x_j)))$$

$$\frac{\partial E}{\partial w_{kj}} = -\sum_{i=1}^{m} (\Delta_i w_{ji} \frac{\partial}{\partial w_{kj}} x_j)$$

Mas $x_j = f(\sum_{k=1}^{n} w_{kj} x_k)$

$$\frac{\partial E}{\partial w_{kj}} = -\sum_{i=1}^{m} (\Delta_i w_{ji} \frac{\partial}{\partial w_{kj}} f(\sum_{k=1}^{n} w_{kj} x_k))$$

$$\frac{\partial E}{\partial w_{kj}} = -\sum_{i=1}^{m} (\Delta_i w_{ji} f'(\sum_{k=1}^{n} w_{kj} x_k) x_k)$$

Finalmente

$$\frac{\partial E}{\partial w_{kj}} = -x_k \Delta_j \text{, onde } \Delta_j = f'(\sum_{k=1}^{n} w_{kj} x_k) \sum_{i=1}^{m} (\Delta_i w_{ji})$$

E neste caso, também podemos fazer:

$$\Delta w_{kj} = -\eta \frac{\partial E}{\partial w_{kj}} = \eta x_k \Delta_j \text{, onde } \eta \text{ é a taxa de aprendizado.}$$

Uma vez derivadas as correções dos pesos entre as camadas, vamos utilizá-las para descrever detalhadamente o algoritmo *backpropagation:*

1-Iniciar pesos e parâmetros;

2-Repetir até o erro ser mínimo ou até a realização de um dado número de ciclos:

2.1-Para cada padrão de treinamento X

2.1.1-Definir a saída da rede através da fase *forward* (descrita abaixo);

2.1.2-Atualizar pesos dos nodos através da fase *backward* (descrita abaixo).

Considerando-se:

M= número de camadas da RNA

x_j^l = entrada dos nós na camada *l*,

y_j^l = saída dos nós na camada *l*,

w_{ij}^l = pesos na camada *l*,

Δ_j^l é o erro nas saídas da camada *l*

Fase *forward:*

Para *l=2* até *M* faça

p = número de nós da camada l;

q= número de nós da camada (l-1);

Para *j = 1 até p* faça

$$y_j^l = f(\sum_{i=1}^{q} w_{ij}^l \; x^l{}_i) ;$$

Se (l=M) faça

$$\Delta_j^M = (yd_j - y_j^M) \quad f'(\sum_{i=1}^{q} w_{ij}^l \; x_i^M) ;$$// Calcula o erro na camada de saída

Fase *backward*

Para l=M até 2 faça

p = número de nós da camada l;

q= número de nós da camada (l-1);

r= número de nós da camada (l-2);

Para *i = 1 até q* faça

Para *j = 1 até p* faça

$$w_{ij}^l = w_{ij}^l + \eta x_i^l \Delta^l{}_j ;$$

Se $l \neq 2$ então

Para *i = 1 até q* faça

$$\Delta_i^{l-1} = f'(\sum_{j=1}^{r} w_{ji}^{l-1} x_j^{l-1}) \sum_{j=1}^{p} (\Delta^l{}_j w_{ij}^l) ;$$

Melhorias no treinamento

O principal problema no treinamento de RNAs pelo algoritmo *backpropagation* é a possibilidade de ocorrência de mínimos locais na superfície de erro, que provê uma solução incorreta. Para reduzir a incidência deste problema, bem como acelerar a convergência do algoritmo, pode-se fazer o seguinte (Pádua, 2000):

- Diminuir a taxa de aprendizado;
- Adicionar mais nós nas camadas intermediárias;
- Adicionar ruído aos dados;
- Acrescentar o chamado fator *momentum* à atualização dos pesos:

$w_{ij}(t) = w_{ij}(t-1) + \eta x_i \Delta_j + \alpha(w_{ij}(t-1) - w_{ij}(t-2))$, onde t é a iteração atual do algoritmo, e α ajusta a intensidade do fator.

Quando houver a possibilidade de todas as entradas em uma RNA serem nulas, é fácil perceber o risco de não haver a atualização dos pesos. Esse problema pode ser evitado colocando-se uma entrada de valor unitário, denominada *bias,* nas camadas onde houver o referido risco. Esta entrada produz pesos adicionais à camada onde esta conectada e é tratada como se fosse uma entrada comum.

Referências

CYBENKO, G. *Continuous Valued Neural Networks with Two Hidden Layers are Sufficient.* Technical Report, Department of Computer Science, Boston: Tufts University, 1988.

CYBENKO, G. Aproximation by Superpositions of a Sigmoid Function. In *Mathematics of Control, Signals and Systems.* New York: *1989*

PADUA, A. et al. *Redes Neurais Artificiais-Teoria e Aplicações*. Rio de Janeiro, LTC, 2000.

RUSSEL, S. & NORVIG, P. *Inteligência Artificial*. Rio de Janeiro: Elsevier, 2ª edição, 2004.