UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA

Otimização Global em Redes Neurais Artificiais

Trabalho de Graduação

Diogo da Silva Severo

Orientadora: Dra. Teresa Bernarda Ludermir
Co-Orientador: Dr. Cleber Zanchettin

Recife, 2010

UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA

Otimização Global em Redes Neurais Artificiais

Trabalho de Graduação

Diogo da Silva Severo

Projeto de Graduação apresentado no Centro de Informática da Universidade Federal de Pernambuco por Diogo da Silva Severo, orientado pela PhD. Teresa Bernarda Ludermir, como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação.

Orientadora: Dra. Teresa Bernarda Ludermir
Co-Orientador: Dr. Cleber Zanchettin

Recife, 2010

"O futuro pertence àqueles que acreditam na beleza de seus sonhos."
Elleanor Roosevelt

DEDICATÓRIA

À minha família

AGRADECIMENTOS

Primeiramente agradeço a Deus por me dar saúde e forças para atingir mais essa conquista.

À minha mãe, pelo amor e dedicação. Por me ajudar ao longo de todo o curso. Sem ela, nada disso seria possível. A ela sou muito grato.

Às minhas avós Célia (Dona Lu) e Raquel (Tina) *in memoriam*. Uma pena não tê-las ao meu lado para compartilhar comigo esse momento.

À minha irmã por seus conselhos e apoio. Às vezes se mostra mais centrada do que eu. Ao meu avô pelo apoio e suporte. A minha família como um todo.

A Sandro, amigo de longa data. Amigo desde o ensino fundamental e que por obra do destino nos aproximamos novamente depois de quatro anos.

À Leyla, Salgueiro, Gleicy e Lessa, amigos que formei durante a graduação logo no início do curso. Aos momentos de brincadeiras e descontração. Leyla, nos perdoe (a Lessa em grau maior, a mim e a Salgueiro em grau menor) por pegarmos sempre no seu pé.

Aos meus amigos Everson, Yane e Nivan. Três grandes amigos com os quais passei e espero passar vários bons momentos.

À minha orientadora, professora Teresa, por ter aceitado me orientar, a ela sou muito grato.

Ao meu co-orientador, professor Cleber, por tirar minhas dúvidas, pela paciência e atenção.

A todos, meu muito obrigado!

RESUMO

Este trabalho apresenta uma técnica de otimização global e local, que integra as potencialidades das heurísticas de três algoritmos de busca global (*Tabu Search, Simulated Annealing* e Algoritmos Genéticos) e de um algoritmo de busca local (*backpropagation*).

Tal técnica é empregada na tarefa de realizar a otimização simultânea da topologia, dos pesos das conexões e da função de ativação de redes neurais artificiais do tipo *Multilayer Perceptron* com o objetivo de gerar topologias mínimas e com alto desempenho para cada problema de forma automática.

Para verificar a eficiência do método, foram realizados experimentos com sete bases de classificação e com duas bases de dados de previsão.

Palavras-chave: otimização global; otimização local; redes neurais artificiais

ABSTRACT

This work presents a global and local optimization technique, which integrates the potential of three global search algorithms (Tabu Search, Simulated Annealing and Genetic Algorithms) and of a local search algorithm (backpropagation).

This technique is used to optimize simultaneously topology, connections weights and MultiLayer Perceptron neural networks activation function, generating minimal topologies with high performance for each problem automatically.

To check the method efficiency, experiments were made with seven classification and three prediction datasets.

Key-words: global optimization; local optimization; neural networks

SUMÁRIO

Lista de Figuras

Lista de Tabelas

1.INTRODUÇÃO

Redes Neurais Artificiais (RNAs) têm sido utilizadas com sucesso na resolução dos mais diversos problemas em uma variedade de áreas, desempenhando tarefas tais como: classificação de padrões, processamento de sinais, mineração de dados, reconhecimento de odores, análise de crédito, dentre outras [1].

Por sua vez, o uso de RNAs para resolução de um determinado problema é um trabalho empírico em que muitos dos parâmetros que constituem a rede são modificados com o objetivo de gerar uma topologia "ótima", mínima e que consiga resolver, satisfatoriamente, variações do mesmo problema.

Para atingir essa configuração "ótima" que seja adequada para solucionar o problema em questão, o projetista da rede neural tem como tarefa estimar, modificar e testar vários parâmetros livres da rede. No caso de uma rede do tipo *Multilayer Perceptron* (MLP), arquitetura adotada no presente trabalho, alguns desses parâmetros seriam: número de camadas escondidas, número de unidades de processamento nas camadas escondidas, tipo da função de ativação das unidades de processamento, taxa de aprendizado, dentre outros.

Além da necessidade de conhecimento teórico acerca de RNAs exigido por parte do projetista para que o ajuste dos parâmetros livres acarrete um bom desempenho do sistema, também lhe é exigido um bom conhecimento sobre o domínio do problema de forma que os ajustes possam ser conduzidos da melhor forma possível. Na ausência de todo esse conhecimento exigido do projetista, a presença de um especialista se faz necessária para condução dos experimentos.

Como as redes neurais envolvem um grande número de parâmetros ajustáveis e diversas topologias possíveis, constata-se a grande dificuldade que é a tarefa de projetar uma rede neural artificial que consiga resolver adequadamente o problema para o qual foi proposta. Logo, o caminho para se chegar a uma configuração "ótima" mostra-se como uma tarefa árdua, difícil e maçante. Segundo Jeffrey et al. [2], a busca por essa arquitetura "ótima" é um problema NP-difícil.

Para retirar do projetista essa grande responsabilidade, trabalhos envolvendo a automatização do processo de geração de topologias de RNAs vêm ganhando cada vez mais espaço no meio acadêmico.

Segundo Abraham [3], a grande utilização de RNAs na resolução de problemas se deve ao fato do sucesso conseguido através de trabalhos que envolvem tarefas de otimização.

Por sua vez, otimização é o processo de encontrar a melhor solução em um espaço de busca formado por diversas soluções possíveis. Um problema de otimização é formado por uma função objetivo, que se deseja maximizar ou minimizar dependendo do problema, e um conjunto de restrições que precisam ser satisfeitas [4].

Do exposto, concluímos que o uso de algoritmos de otimização no projeto de redes neurais artificiais é uma abordagem interessante e bastante útil, uma vez que serão geradas, automaticamente, topologias menores e mais eficientes, diminuindo tempo e custos empregados no processo de busca por tais arquiteturas.

1.1 Objetivo

O objetivo dessa monografia é realizar uma expansão no trabalho de Zanchettin [5], de forma que o algoritmo, denominado GaTSa (combinação das heurísticas de busca dos algoritmos *backpropagation*, Algoritmos Genéticos, *Tabu Search* e *Simulated Annealing*) possa realizar não somente a otimização simultânea da topologia e pesos das conexões de redes *Multilayer Perceptron*, mas que também incorpore a otimização da função de ativação dos nodos que compõem a rede.

A fim de verificar a eficácia da modificação no algoritmo GaTSa, serão realizados testes com os mesmos conjuntos de dados adotados por Zanchettin com o intuito de verificar a taxa de acerto do algoritmo com e sem a modificação proposta pelo presente trabalho. Além disso, serão realizados testes com conjuntos de dados que não foram explorados pelo autor.

1.2 Estrutura Geral do Trabalho

A monografia está dividida em seis capítulos. Abaixo, segue uma breve descrição acerca do conteúdo presente em cada um deles.

Capítulo 2: Sistemas Conexionistas

O capítulo 2 trata dos principais conceitos referentes aos sistemas conexionistas (Redes Neurais Artificiais). Nele, são abordados pontos cruciais no projeto de RNAs bem como alguns problemas a que a técnica está sujeita.

Capítulo 3: Otimização Global

No capítulo 3 serão abordadas as técnicas de otimização global (*Tabu Search*, *Simulated Annealing* e Algoritmos Genéticos) que compõem o algoritmo proposto.

Também será mostrado um estudo de trabalhos que falam sobre a tarefa de otimização de redes neurais artificiais. Principalmente no que diz respeito ao foco do presente trabalho (otimização de função de ativação).

Capítulo 4: Algoritmo GaTSa

No capítulo 4, serão descritas as principais características do algoritmo proposto bem como a modificação proposta para a realização da otimização da função de ativação.

Capítulo 5: Experimentos e Resultados

O capítulo 5 trará uma descrição dos conjuntos de dados utilizados, da metodologia de experimentos empregada e dos resultados obtidos.

Capítulo 6: Conclusão e Trabalhos Futuros

O capítulo 6 trará um breve resumo do que foi exposto ao longo da monografia e sugestões de pesquisa para trabalhos futuros que envolvam otimização global em redes neurais artificiais.

2. SISTEMAS CONEXIONISTAS

Redes Neurais Artificiais (RNAs), sistemas paralelos e distribuídos ou ainda sistemas conexionistas são três maneiras distintas de referenciar uma técnica computacional inspirada no cérebro humano. Como principais características desta técnica estão a capacidade de aprender através de exemplos, a capacidade de generalização e sua tolerância a falhas [6].

2.1 Introdução

Apesar dos primeiros trabalhos referentes a redes neurais artificiais terem sido registrados na década de 40 [7], foi somente em meados da década de 80 que a área teve um crescimento expressivo no número de pesquisadores envolvidos com assuntos relativos a RNAs.

O principal motivo para a retomada do interesse em RNAs por parte da comunidade científica foi, sem dúvida, a publicação do trabalho entitulado "*Learning Representations by Back-Propagating Errors*" por Rumelhart, Hinton e Williams em 1986 na prestigiada revista *Nature* [8]. Nele, foi proposto um algoritmo de aprendizagem para treinar redes de múltiplas camadas até então inexistente.

Como dito anteriormente, as redes neurais artificiais foram inspiradas nas redes neurais biológicas presentes nos cérebros de seres humanos. As principais características em comum com as redes neuronais biológicas são a capacidade de processamento de informação de forma paralela e distribuída, a presença de unidades de processamento de informação simples, a presença de detectores de características e a capacidade de aprendizado através de exemplos [7].

O principal componente de uma RNA é uma unidade de processamento, também conhecida como neurônio ou ainda nodo [9]. Tais unidades realizam operações matemáticas elementares e estão dispostas em camadas. Entre os neurônios de camadas adjacentes, há conexões sinápticas responsáveis por transmitir o sinal calculado pelo nodo da camada *i*-th para o nodo da camada *(i + 1)*-th.

Tais conexões podem ser do tipo *feedforward* ou *feedback*. As conexões do tipo *feedforward* são conexões unidirecionais, ou seja, sempre partem de um neurônio para outro neurônio que esteja a sua direita (figura 1). Já nas conexões *feedback,* há retroalimentação, isto é, a saída de um neurônio pode servir como entrada para um outro neurônio que esteja a sua esquerda (figura 2).

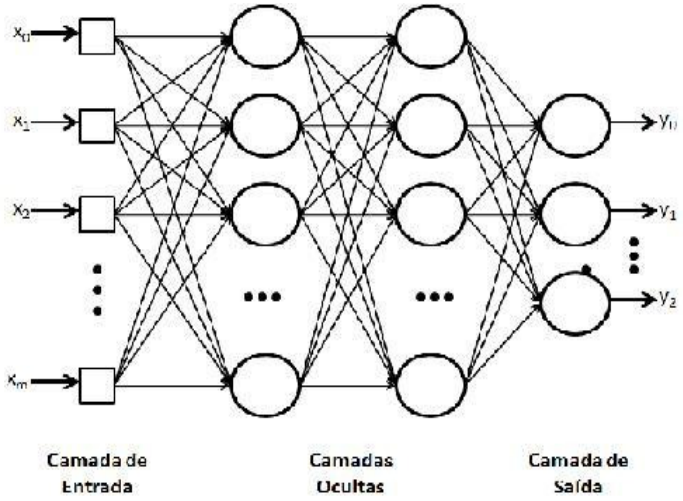

Figura 1 - Rede neural com conexões *feedforward*

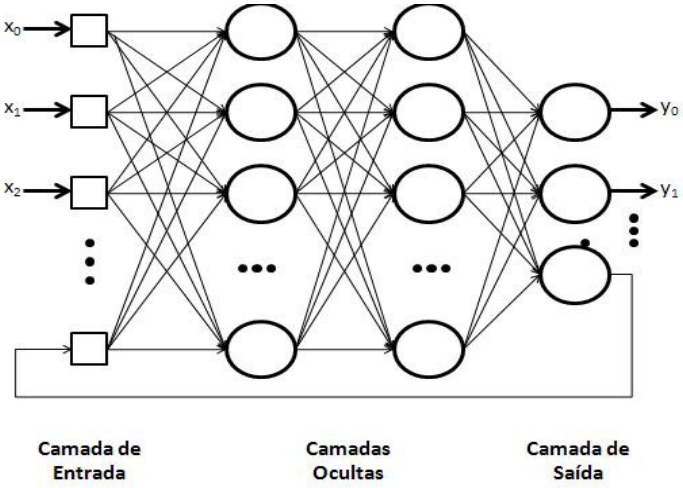

Figura 2 - Rede neural com conexões *feedforward* e *feedback*

Na maioria dos modelos de RNAs, à conexão está associado um valor real, chamado peso, responsável por ponderar o sinal propagado pelo nodo, podendo alterar ou não o sinal transmitido. São nos pesos que a RNA armazena o conhecimento adquirido sobre determinado problema que lhe é apresentado.

Cada nodo pode estar conectado a vários outros nodos de forma que a saída produzida por um nodo é dada pela soma de suas entradas ponderadas pelos respectivos pesos. Na figura 3, é mostrado um exemplo de um nodo de uma RNA.

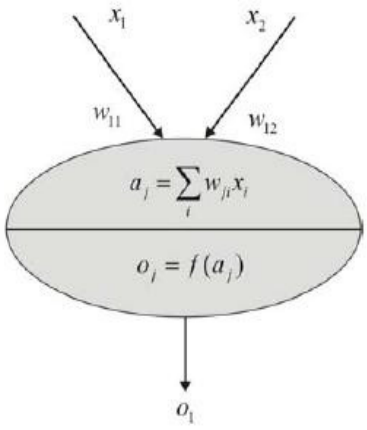

Figura 3 - Exemplo de uma unidade de processamento de uma RNA

A ativação de um nodo *j* é dada pela equação 2.1:

$$\alpha_j = \sum_j W_{ji}\, O_i \qquad (2.1)$$

Onde W_{ji} é o peso associado à conexão que parte do neurônio *i* e chega ao neurônio *j* com *j* variando no intervalo [1,n] onde "n" é o número total de conexões que chegam ao neurônio *j* (*fan-in*) e O_ié o sinal propagado pelo neurônio *i* [9].

Antes de o sinal ser propagado para o neurônio da camada seguinte, aplica-se uma função de ativação, também chamada função de transferência, geralmente não-linear que transforma o valor recebido como parâmetro para um dado intervalo [a,b] onde "a" e "b" são valores dependentes da função de ativação em questão.

Após a aplicação de uma função de ativação f ao sinal de ativação, a saída do neurônio pode ser reescrita como: $O_j =$ f $(\sum_j W_{ji}\, O_i)$. Há várias funções de ativação na literatura, sendo a linear, a sigmóide logística e a tangente hiperbólica as mais comuns [9]. As funções de ativação serão abordadas com mais detalhes na seção 2.3.

Além das entradas que cada neurônio recebe, cada nodo possui um parâmetro adicional chamado *bias* ou limiar (b), responsável por aumentar ou diminuir o valor da ativação do nodo passado para a função de transferência [9]. Com esse valor adicional, a fórmula que calcula a saída de um neurônio pode ser reescrita através da equação 2.2:

$$O_{j=} \mathrm{f}\ (\sum_j W_{ji}\ O_i + \mathrm{b}) \qquad (2.2)$$

Apesar de realizarem operações matemáticas simples, os vários nodos em conjunto possibilitam à rede neural a capacidade de solucionar problemas complexos. Daí a grande utilização de RNAs como método para solução dos mais diversos problemas: reconhecimento de odores [10], reconhecimento de caracteres manuscritos [11], predição de estruturas secundárias de proteínas [12], entre outros.

2.2 Redes *Multi-layer Perceptron*

A primeira rede neural proposta e implementada surgiu no final da década de 50 [13]. Tal rede foi denominada *perceptron*. Ela era formada por uma camada de entrada, responsável por captar sinais externos, uma camada intermediária com pesos fixos e uma camada de saída responsável por fornecer a resposta produzida pela rede neural [7].

Como bem apontaram Minsky e Papert [14], a rede *perceptron* possui um grande problema que é a capacidade de resolver apenas problemas que sejam linearmente separáveis (classe pequena de problemas), nos quais os dados de entrada podem ser divididos em duas regiões separadas por uma reta.

Devido à grande repercussão do trabalho de Minsky e Papert, a área de redes neurais foi deixada em segundo plano por aproximadamente dezessete anos. Foi nessa época, especificamente em 1986, que um grupo de cientistas propôs o *perceptron* multicamadas, uma rede neural formada por múltiplas camadas intermediárias com pesos variáveis, e um algoritmo de aprendizado específico, denominado *backpropagation*, para treinar esses novos tipos de redes. Com isso, era superada a limitação das redes *perceptrons* de só conseguir resolver problemas que fossem linearmente separáveis.

A partir de então, vários modelos de redes neurais surgiram. Porém, as redes *perceptron* multicamadas ou também chamadas MLP (do inglês *Multi-layer Perceptron*) se tornaram o modelo mais difundido e utilizado devido, em grande parte, a sua facilidade de implementação e simplicidade [9].

A arquitetura de uma rede neural do tipo MLP é determinada pela sua topologia e pela função de ativação de cada nodo da rede [15]. Por sua vez, a topologia de uma rede MLP é formada pela camada de entrada, pelas camadas intermediárias, pela camada de saída e pela quantidade de neurônios que forma cada camada.

Basicamente, uma rede MLP é formada por uma camada de entrada, uma ou mais camadas intermediárias (também chamadas de camadas ocultas ou escondidas)

e uma camada de saída sendo totalmente conectada, ou seja, de um neurônio da *i*-ésima camada partem conexões para todos os neurônios da *(i + 1)*-ésima camada. A figura 4 ilustra uma arquitetura típica de uma rede MLP.

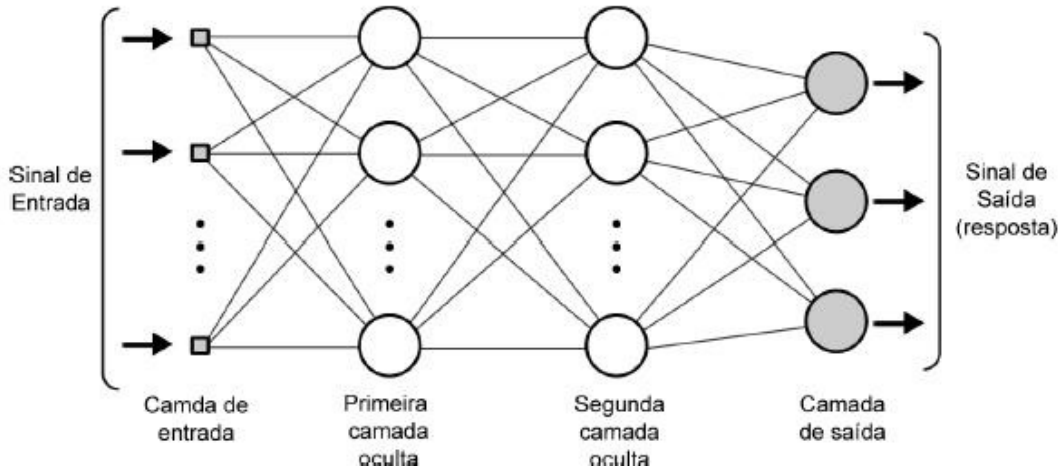

Figura 4 - Exemplo de uma arquitetura com duas camadas escondidas de uma MLP

Cada camada da rede possui uma função bem definida. A camada de entrada é formada por nodos que não realizam processamento de informação. Tal camada comporta-se como um *buffer* que armazena os padrões de entrada para só então passá-los para a primeira camada escondida. A quantidade de neurônios presente nessa camada é dependente da dimensão do vetor de características que representa os padrões de dados que compõem um determinado problema.

As camadas intermediárias, por sua vez, são formadas por nodos que possuem capacidade computacional e são essas camadas que conferem a não-linearidade às redes MLP, uma vez que as funções de ativação desses nodos são funções não-lineares.

Essas camadas funcionam como extratores de características visto que elas extraem as características mais relevantes presentes nos dados para que a rede neural crie a sua própria representação do problema que está sendo aprendido.

A quantidade de camadas intermediárias bem como a quantidade de unidades de processamento que compõe cada camada escondida é definida empiricamente e depende da natureza do problema que está sendo tratado. Porém, segundo Cybenko, [16] uma camada escondida é suficiente para aproximar qualquer função contínua, e duas são suficientes para aproximar qualquer função matemática [17].

A camada de saída, que também é formada por neurônios com capacidade computacional, é responsável por fornecer a resposta gerada pela rede para cada padrão que lhe é apresentado. Da mesma forma que a camada de entrada, a quantidade de unidades de processamento presente na camada de saída depende do problema que está sendo tratado.

2.3 Funções de Ativação

Na seção 2.1, foi visto que uma RNA é formada por unidades de processamento que realizam determinadas operações matemáticas sobre as entradas que recebe. Tipicamente, em neurônios de redes MLP, essas operações matemáticas se resumem a uma soma das entradas ponderadas pelos pesos.

Porém, antes desse sinal ser propagado para os neurônios da camada seguinte ou ser definido como a saída produzida pela rede, ele é submetido a uma função de ativação, que converte o valor dessa soma para uma faixa de valores que dependerá da função que estiver sendo utilizada.

O tipo de função de ativação a ser utilizado pelos neurônios de uma rede é um dos parâmetros livres a ser definido pelo projetista e tem grande influência no desempenho de RNAs.

A escolha de qual tipo de função será utilizada pelos nodos está intimamente relacionada à natureza do problema, podendo acarretar um aumento ou diminuição na velocidade de convergência [18].

Existem muitos tipos de função de ativação, sendo uns mais simples e outros mais complexos. Geralmente, utiliza-se um mesmo tipo de função de ativação para todos os nodos de uma rede neural. Porém, existem outras abordagens, muito pouco exploradas, que definem diferentes funções de ativação para cada neurônio ou para camada da rede [19].

As funções de ativação mais conhecidas são a função linear, a função sigmóide logística e a função sigmóide tangente hiperbólica, ou simplesmente, função tangente hiperbólica. As duas últimas são as mais utilizadas como funções de ativação dos nodos, principalmente em redes treinadas com algoritmos baseados em gradiente descendente, que exigem que as funções sejam contínuas para o cálculo de derivadas utilizadas pelo algoritmo [9].

A faixa de valores possível para a função sigmóide logística é o intervalo fechado [0,1], para a função tangente hiperbólica é o intervalo fechado [-1,+1]. Como a função linear não é contínua, ela só admite os valores 0 e 1como resposta.

As funções linear, sigmóide logística e tangente hiperbólica podem ser formalizadas, respectivamente, através das equações 2.3, 2.4 e 2.5.

$$f\ (\alpha) = \theta\alpha \tag{2.3}$$

$$f(\alpha) = \frac{1}{1+\exp\ (-b\alpha)} \tag{2.4}$$

$$f(\alpha) = \frac{1-\exp\ (-b\alpha)}{1+\exp\ (-b\alpha)} \tag{2.5}$$

Nas três equações acima, α é soma ponderada das entradas definida pela equação 2.1, θ é um número real e b é o parâmetro de inclinação da curva sigmóide.

Abaixo, seguem, respectivamente, os gráficos da função linear (figura 5), sigmóide logística (figura 6) e tangente hiperbólica (figura 7).

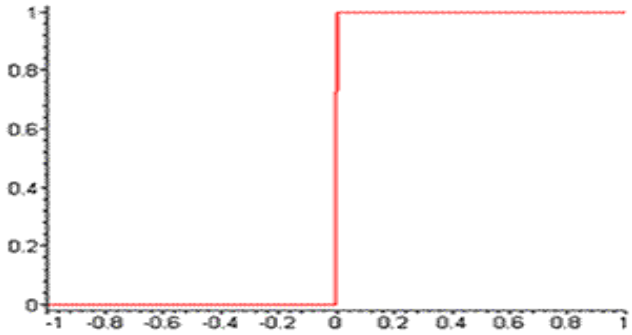

Figura 5 - Gráfico da função linear

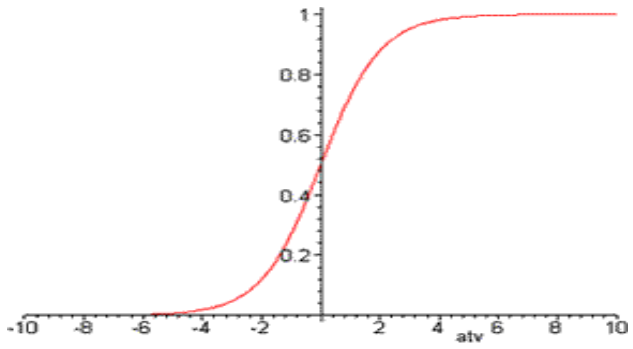

Figura 6 - Gráfico da função sigmóide logística

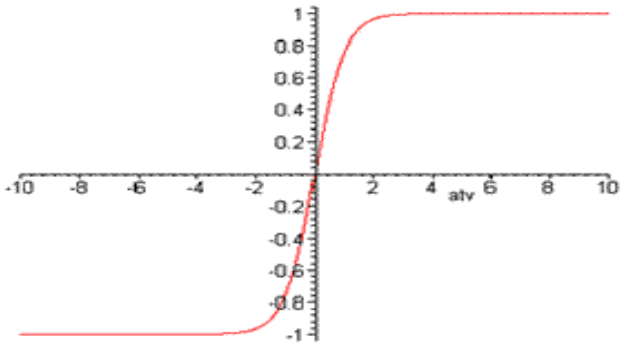

Figura 7 - Gráfico da função tangente hiperbólica

2.4 Treinamento

Como visto em seções anteriores, um dos principais atrativos para se utilizar redes neurais artificiais na resolução de problemas é a sua capacidade de aprender através de exemplos. Portanto, antes de uma RNA ser utilizada para resolver um determinado problema, ela passa por um processo de aprendizado.

A preocupação com o aprendizado de redes neurais artificiais teve início no final da década de 40 através de Donald Hebb [20]. Por meio de seu trabalho, Hebb mostrou como o processo de aprendizagem em RNAs era conseguido através da modificação dos pesos associados às conexões sinápticas.

De lá para cá, muitos foram os algoritmos de treinamento propostos para realizar a tarefa de alterar os pesos das conexões. Sendo a maneira como ocorre a modificação dos pesos a principal diferença entre os vários algoritmos sugeridos.

Geralmente em redes MLP, os nodos de uma camada estão totalmente conectados aos nodos da camada adjacente através de conexões sinápticas. A cada conexão associa-se um peso que serve tanto para armazenar o conhecimento adquirido pela rede como para ponderar os sinais de entrada. Ao ajuste da matriz de pesos que compõe uma RNA, dá-se o nome de treinamento.

Quanto aos parâmetros que são modificados durante o processo de treinamento, os algoritmos são classificados em estáticos e dinâmicos.

Durante o processo de treinamento, utilizando algoritmos estáticos, a estrutura da rede (número de camadas, número de neurônios em cada camada e a quantidade de conexões) não sofre alteração. Já nos algoritmos dinâmicos, tal estrutura sofre alteração, diminuindo ou aumentando [6].

Dentre os vários algoritmos de treinamento criados para treinar redes do tipo *MultiLayer Perceptron*, o algoritmo *backpropagation* (também chamado de regra delta generalizada) se destaca como o algoritmo mais utilizado e difundido [6].

Apesar de sua larga aceitação, o *backpropagation* apresenta alguns problemas como a grande quantidade de iterações necessárias para convergir para uma região de baixo erro, a diminuição da capacidade de generalização (*overfitting*) e a possibilidade de ficar preso em mínimos locais.

Na seção seguinte, o algoritmo de treinamento *backpropagation* será descrito em detalhes. Nas seções 2.5 e 2.6, alguns dos problemas que afetam o *backpropagation* serão discutidos.

2.5 Algoritmo *Backpropagation*

Desde o surgimento do primeiro algoritmo de treinamento proposto por Hebb [20], muitos foram os algoritmos propostos para treinar redes neurais artificiais. Dentre eles, pode-se citar: algoritmo *backpropagation* [8], algoritmo de gradiente conjugado escalonado [21], algoritmo Fletcher-Reeves [22], algoritmo Levenberg-Marquardt [23], dentre outros.

Proposto em 1986 [8], o *backpropagation* ou regra delta generalizada é o algoritmo de treinamento mais utilizado para ajuste de pesos de redes neurais artificiais.

O surgimento desse algoritmo caracterizou-se como um grande avanço para a área de redes neurais artificiais uma vez que, através dele, conseguiu-se realizar a atualização dos pesos das conexões associadas a neurônios das camadas intermediárias.

Também chamado de método do gradiente descendente, por se basear no cálculo do gradiente do erro total cometido pela rede para ajuste dos pesos, o algoritmo *backpropagation* requer que as funções de ativação dos nodos sejam contínuas e, geralmente, não decrescentes, de forma que seja possível o cálculo da derivada da função [6].

O erro total cometido pela rede é a função de custo a ser minimizada pelo algoritmo. Tal função é definida como a média da soma dos erros quadráticos sendo definida pela equação 2.6:

$$E = \frac{1}{2N} \sum_{i=1}^{N} (d_i - y_i)^2 \qquad (2.6)$$

Na equação acima, E é o erro médio total cometido pela rede, N é a quantidade de padrões que forma o conjunto de treinamento, d_i é a saída desejada para o i-ésimo padrão fornecido à rede como entrada e y_i é o valor produzido pela rede como saída para o i-ésimo padrão.

O algoritmo *backpropagation* é um algoritmo supervisionado já que trabalha com pares de valores (entrada, saída-desejada). Quando não há o conhecimento *a priori* da saída desejada para o padrão de entrada, o algoritmo de treinamento é dito não-supervisionado. Supervisionado e não-supervisionado são as duas principais classificações de algoritmos quanto à forma de aprendizagem.

Formado por duas etapas bem definidas chamadas de fase *forward* (fase responsável por calcular as saídas de cada nodo para cada padrão de entrada apresentado à rede) e fase *backward* (fase responsável por atualizar os pesos das conexões da rede através da retro-propagação do erro cometido pela rede), o algoritmo *backpropagation* é descrito como:

Algoritmo 1 – Pseudo-código *Backpropagation*

1. Inicializar pesos e bias
2. Apresentar o padrão de entrada juntamente com sua respectiva saída desejada
3. Propagar esse padrão de camada em camada de forma que seja calculada a saída para cada nodo da rede
4. Comparar a saída gerada pela rede com a saída desejada e calcular o erro cometido pela rede para os nodos da camada de saída
5. Atualizar os pesos dos nodos da camada de saída com base no erro competido por tais nodos
6. Até chegar à camada de entrada:
    a. Calcular o erro dos nodos da camada intermediária baseado no erro cometido pelos nodos imediatamente seguintes ponderado pelos pesos entre os nodos da camada atual e os nodos imediatamente seguintes
7. Repetir os passos 2, 3, 4, 5 e 6 até obter um erro mínimo ou até atingir um dado número de iterações

O erro cometido por um nodo *j* da camada de saída é formalizado pela equação 2.7:

$$\delta_j = (d_j - y_j).\, f'(net_j) \qquad (2.7)$$

O valor d_j é a saída desejada para o nodo, y_j é a saída que foi produzida pela rede, f' é a derivada da função de ativação para o valor net_j, que por sua vez representa a ativação do nodo *j* dada pela equação 2.8:

$$net_j = \sum_{i=1}^{n} x_i\, w_{ji} \qquad (2.8)$$

O valor n representa o número de nodos da camada imediatamente anterior conectados ao nodo *j*, x_i é a saída de cada nodo da camada anterior que serve como entrada do nodo *j* e W_{ji} é o peso associado à conexão entre cada nodo da camada imediatamente anterior e o nodo *j*.

Para um nodo *j* da camada intermediária, seu erro é dado pela equação 2.9:

$$\delta_j = f'(net_j) \sum_n \delta_n\, W_{nj} \qquad (2.9)$$

O valor δ_n representa os erros cometidos pelos nodos imediatamente à direita de *j* e W_{nj} representa os pesos das conexões existentes entre *j* e os nodos imediatamente a sua direita.

Por fim, a fórmula de ajuste dos pesos pode ser definida pelas equações 2.10 e 2.11:

$$W_{ji}(k+1) = W_{ji}(k) + \Delta W_{ji} \qquad (2.10)$$

$$\Delta W_{ji} = \eta \delta_j x_i \qquad (2.11)$$

Em 2.7, η é um valor no intervalo (0,1) que representa a taxa de aprendizagem, lambda (δ_j) é o erro produzido por um nodo da camada de saída ou camada intermediária e x_i é o valor de entrada do nodo j.

2.6 *Overfitting*

RNAs possuem a capacidade de aprender através de exemplos. Antes de passar pelo algoritmo de treinamento, a massa de dados que compõe o problema é dividida em dois conjuntos disjuntos chamados treinamento e teste. Tais conjuntos possuem, respectivamente, as funções de ajuste de pesos e de medir a capacidade de generalização da rede.

Durante o processo de treinamento, a rede está sujeita a sofrer um problema chamado *overfitting* ou problema do sobre-ajuste dos pesos. Tal problema é caracterizado como a perda da capacidade de generalização, ou seja, a diminuição da capacidade de fornecer respostas coerentes para padrões que não foram utilizados para treiná-la. O *overfitting* ocorre quando a rede passa a "decorar" os padrões de treinamento incorporando até eventuais ruídos presentes nos dados. Essa memorização dos dados de treinamento está associada ao grande número de conexões presentes na rede neural. Número esse mais do que o necessário para aprender o problema em questão [24].

O número de conexões está atrelado à definição, por parte do projetista da rede neural, do número de camadas escondidas e do número de nodos em cada camada escondida. Esses parâmetros devem ser definidos de forma que a estrutura da rede possa resolver, satisfatoriamente, o problema em questão.

Uma rede com muitos nodos e, consequentemente, com muitas conexões, além de levar a um aumento no tempo necessário de treinamento, acarreta a memorização dos dados de entrada (*overfitting*). Em contrapartida, um número insuficiente de nodos na(s) camada(s) escondida(s) implica outro problema denominado *underfitting*, que é a incapacidade da rede neural de resolver o problema abordado, isto é, a rede neural possui uma complexidade inferior ao problema. Logo, a estimação desses parâmetros torna-se um ponto crítico no projeto de redes neurais que deve ser definido com muito cuidado por parte do projetista.

Com o intuito de evitar a incidência de *overfitting*, uma parte do conjunto original de treinamento é reservada para formar o conjunto de validação e o restante dos padrões do conjunto original de treinamento forma o novo conjunto de treinamento. Os padrões do conjunto de validação não são usados para treinar a rede, eles possuem uma função específica.

A função do conjunto de validação é mensurar a capacidade de generalização da rede durante a fase de treinamento, interrompendo, precocemente, o processo de treinamento. O treinamento deverá ser interrompido quando o erro do conjunto de

validação começar a subir em relação ao erro do conjunto de treinamento por um número *n* consecutivo de vezes (*n* é um parâmetro livre da rede). Esse crescimento do erro indicará a redução da capacidade de generalização da rede [6].

2.7 O problema do Mínimo Local

Outro problema que acomete os algoritmos baseados em gradiente descendente e que tem influência direta no desempenho de redes neurais artificiais treinadas com tais algoritmos é a ocorrência dos mínimos locais.

Visto que o algoritmo *backpropagation* é basicamente uma técnica de "subida de encosta", ele corre o risco de ficar preso em mínimos locais onde qualquer pequena mudança nos pesos sinápticos acarreta um aumento da função de custo [9].

Entretanto, em algum lugar do espaço de busca há um conjunto de pesos sinápticos para o qual a função de custo é menor do que o mínimo local [9].

Do exposto, vemos que a finalização precoce do processo de treinamento em um mínimo local deve ser fortemente combatido. Na figura 8, é mostrada uma superfície de erro na qual há dois pontos de mínimos locais e um ponto de mínimo global (resposta ótima para o problema).

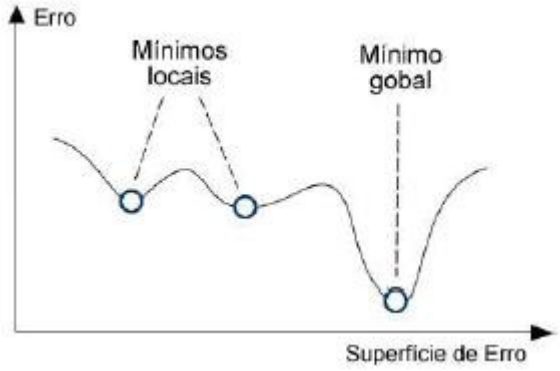

Figura 8 - Mínimos locais e mínimo global em uma superfície de erro

Algumas alternativas foram propostas com a intenção de evitar a ocorrência de mínimos locais durante a fase de treinamento. São elas:

- Utilização de taxa de aprendizado decrescente;
- Adição de neurônios nas camadas intermediárias;
- Utilização do termo *momentum*;
- Adição de ruídos aos dados.

Dentre as técnicas acima, a adição do termo *momentum* é uma das mais adotadas [25]. O termo *momentum* passa a ser mais um parâmetro ajustável no projeto de redes neurais artificiais sendo definido pela equação 2.12.

$$\varphi = \alpha\ (W_{ji}\,(\mathrm{t})\ \text{-}\ W_{ji}(\mathrm{t}-1)) \qquad (2.12)$$

Com a adição desse termo, a velocidade no processo de treinamento sofre um aumento, reduzindo as oscilações em regiões próximas a mínimos locais. A equação de ajuste dos pesos com o termo *momentum* passa a ser definida pela equação 2.13.

$$W_{ji}(\mathrm{t}+1) = W_{ji}(\mathrm{t}) + \eta\delta_i(\mathrm{t})\ X_i(\mathrm{t}) + \alpha\ (W_{ji}\,(\mathrm{t})\ \text{-}\ W_{ji}(\mathrm{t}-1)) \qquad (2.13)$$

2.8 Considerações Finais

Neste capítulo, vimos como as redes neurais são largamente utilizadas na resolução de problemas do mundo real, apresentando resultados satisfatórios, em sua grande maioria. Além da sua inspiração biológica, foram cobertos os principais tópicos referentes ao projeto de RNAs, com foco nas redes MLP, como por exemplo, os tipos de função de ativação, o algoritmo de treinamento e a arquitetura. Ademais, foram apresentados alguns dos problemas mais comuns que afetam as redes MLP treinadas com algoritmo *backpropagation*.

3 OTIMIZAÇÃO GLOBAL

O uso de RNAs na solução de determinados problemas não se mostra uma tarefa fácil. Como visto anteriormente, o projeto de uma RNA é um trabalho empírico em que vários dos parâmetros ajustáveis que a constituem devem ser estimados pelo projetista da rede neural.

Dependendo da experiência do projetista e de seu conhecimento teórico acerca de RNAs e da natureza do problema a ser solucionado, o resultado obtido pode ser satisfatório ou estar longe do esperado.

Diante desse cenário, técnicas das mais variadas têm sido usadas para otimizar RNAs com o intuito de tirar do projetista essa responsabilidade, eliminando assim os erros decorrentes da má escolha feita pelo projetista [26].

A aplicação dessas técnicas ao projeto de RNAs origina os Sistemas Inteligentes Híbridos (SIHs), uma das áreas de pesquisa mais promissora e estudada na atualidade [27].

Sistemas Inteligentes Híbridos são sistemas que combinam as potencialidades de duas ou mais técnicas, formando um novo sistema que se mostra mais eficaz na resolução de um dado problema quando comparado com as técnicas originais separadas. Quando uma das técnicas envolvidas é uma rede neural artificial, o SIH recebe o nome de SNH (Sistema Neural Híbrido) [6].

No campo dos SIHs, diversos trabalhos têm sido desenvolvidos utilizando técnicas de otimização global e redes neurais artificiais, sendo os algoritmos genéticos [28], *simulated annealing* [29] e *tabu search* [30] as técnicas de otimização global mais adotadas, como será visto na subseção 3.4.

3.1 Algoritmos Genéticos

Pesquisando na literatura que trata de otimização de redes neurais artificiais, constata-se a grande utilização de algoritmos genéticos (AGs) para tal tarefa. Uma característica que contribui para essa larga utilização é o fato dos algoritmos genéticos terem, assim como as RNAs, uma inspiração biológica.

Os AGs foram inspirados na teoria da evolução das espécies, elaborada pelo cientista Inglês Charles Darwin em sua obra de título “A origem das espécies” [31]. Por sua vez, as RNAs foram inspiradas no cérebro humano [7].

Segundo Murray [32], os AGs são a técnica de otimização mais adequada para solucionar problemas de otimização de topologias de RNAs uma vez que o cérebro humano é resultado do processo de evolução biológico.

Segundo Pham et al. [33], Algoritmo Genético é uma técnica de pesquisa randômica direcionada, criada por Holland [28] em 1975 capaz de achar a solução ótima global de um problema de otimização em complexos espaços de busca multidimensionais.

Diferentemente de outras técnicas de otimização, os AGs realizam uma busca paralela no espaço de soluções visto que um AG trabalha com um conjunto de soluções simultaneamente.

A esse conjunto de soluções, é dado o nome de população. Cada solução da população recebe o nome de indivíduo, cromossomo ou genótipo. A cada indivíduo da

população está associado um valor denominado aptidão que mede a qualidade da solução para resolver o problema em questão [28].

Um algoritmo genético é um algoritmo iterativo, logo, ele funciona através de uma sequência de iterações também chamada de gerações. A cada geração, a população passa pelos processos de seleção (escolha dos indivíduos mais aptos para reprodução) e reprodução (combinação e/ou modificação das informações contidas nos indivíduos originais para gerar novos indivíduos) [34].

Abaixo, segue o pseudo-código de um AG.

Algoritmo 2 – Pseudo-código Algoritmo Genético

1. t ← 0 índice de gerações inicial
2. P(t) ← inicializa P(t): população é inicializada
3. Avalia P(t): Aptidão dos Indivíduos da População é avaliada
4. Para t = 0 até critérios de parada:
    a. t ← t + 1: índice da geração é atualizado
    b. Pais(t) ← seleciona Pais de P(t): Pais são selecionados
    c. Filhos(t) ← recombina Pais(t): Filhos são gerados por recombinação
    d. Filhos(t) ← muta Pais(t): Filhos são gerados por mutação
    e. Avalia P(t): Aptidão dos Indivíduos da População é avaliada
    f. P(t) ← seleciona sobreviventes P(t): Indivíduos sobreviventes são selecionados para a próxima geração
5. Retorne a melhor solução de P(t)

Na figura 9 é mostrado um fluxograma de um algoritmo genético com as suas principais etapas.

Figura 9 - Fluxograma de um Algoritmo Genético

AGs não usam muito conhecimento a priori sobre o problema a ser otimizado e não lidam diretamente com os parâmetros do problema. Eles trabalham com as codificações dos parâmetros [33]. Assim, para que seja obtido sucesso ao utilizar AGs, que nem sempre é conseguido, para solucionar problemas de otimização, alguns requisitos devem ser obedecidos. São eles:

1. Como representar os parâmetros do problema;
2. Como criar a população inicial de possíveis soluções;
3. Como selecionar ou criar um conjunto de operadores genéticos;

4. Como projetar uma interface entre o problema e o AG propriamente, de forma que o AG tenha conhecimento sobre a aptidão de cada solução.

A representação dos indivíduos tem um grande impacto no desempenho de um AG. Dependendo do tipo de representação escolhido, pode haver um acréscimo ou decréscimo no tempo computacional exigido pelo AG para manipular as soluções [32].

Os dois métodos mais comumente usados são a codificação binária e a codificação real. Na codificação binária, um indivíduo é representado como uma cadeia de bits. A principal desvantagem desse método está na quantidade de bits necessária para representar um indivíduo. Se muitos bits forem necessários para representar uma solução candidata, o desempenho do AG é comprometido pela dificuldade de manipulação. Em contrapartida, se poucos bits forem usados, informações úteis podem ser descartadas devido à escassez de bits para representar com precisão uma solução [35].

Já a codificação real tem a vantagem de não precisar codificar e decodificar constantemente as soluções candidatas. Porém, sofre pela falta de operadores genéticos para manipular soluções codificadas dessa maneira.

Os AGs trabalham em cima de uma população inicial. A formação dessa população inicial se dá através de duas maneiras: uma forma consiste em usar soluções produzidas randomicamente. Este método é aconselhado para problemas em que não há um conhecimento a priori sobre a natureza do problema. A outra maneira emprega um conhecimento a priori sobre o problema de otimização em questão. Neste caso, esse conhecimento é usado para obter um conjunto de critérios de forma que as soluções que satisfazem esses critérios são reunidas para formar a população inicial.

A diferença principal entre os dois métodos, em termos de eficiência, está no tempo necessário para convergir para uma solução ótima. A segunda maneira de formar uma população inicial requer menos tempo do que a primeira para convergir [33].

Os operadores genéticos são responsáveis pela seleção dos indivíduos mais aptos da população para reproduzirem e pela garantia da diversidade gênica. Muitos desses operadores foram inspirados na natureza e o seu uso está atrelado à maneira como o indivíduo foi codificado [33].

O operador genético de seleção desempenha um importante papel no processo evolucionário que é o de direcionar a busca para regiões em que se encontram as melhores soluções para o problema abordado [33].

Os dois principais operadores de seleção são a seleção proporcional (método da roleta) e a seleção baseada no ranking. No método da roleta, indivíduos da população ocupam espaços (*slots)* na roleta proporcionais aos seus valores de aptidão de forma que indivíduos com maiores valores de aptidão terão maiores chances de serem selecionados para a fase de reprodução [34].

No método de seleção baseado no ranking, cada indivíduo gera um número de descendentes que é baseado no *rank* que lhe é atribuído e não no valor de sua aptidão [36].

Depois de selecionados os indivíduos, tem início o processo de reprodução. Os operadores genéticos responsáveis pela etapa de reprodução são os operadores de mutação, recombinação e inversão.

A mutação e a inversão são operadores unários, isto é, trabalham em cima de um único indivíduo, que modificam os genes (parâmetros do problema codificados) de um indivíduo, forçando o algoritmo a buscar novas áreas do espaço de busca para serem exploradas. Por sua vez, a recombinação é um operador binário no qual um novo indivíduo é gerado a partir das informações genéticas dos indivíduos pais [33].

Por fim, o projeto da interface entre o AG e o problema está fortemente ligado à maneira como as soluções estão codificadas (codificação binária ou real) e ao tipo de problema que está sendo tratado.

3.2 Simulated Annealing

A técnica *simulated annealing* é inspirada no processo de esfriamento de sólidos no qual os sólidos têm reduzidas as suas temperaturas lentamente até que sejam formadas estruturas cristalinas perfeitas [29].

Na analogia entre um problema de otimização combinatória e o processo de aquecimento (*annealing*), os estados dos sólidos representam as possíveis soluções para o problema de otimização, as energias dos estados correspondem aos valores da função objetivo computados para estas soluções e o estado de energia mínima corresponde à solução ótima do problema [33].

O algoritmo é formado por um sequência de iterações onde cada iteração consiste em mudar randomicamente a solução corrente a fim de criar uma nova solução na vizinhança que seja melhor do que a solução atual [29].

Uma vez criada a solução, o custo dessa nova solução é computado e a variação no custo é usada para decidir pela aceitação ou não da nova solução como solução atual. Se a variação for negativa, a solução produzida é tirada do posto de solução atual. Caso contrário, ela é aceita de acordo com o critério *Metropolis* baseado na probabilidade de Boltzman [37].

Segundo esse critério, se a diferença (∆E) entre os valores da função de custo das soluções atual e produzida for igual ou maior que zero, um valor randômico (δ) no intervalo fechado [0,1] é gerado a partir de uma distribuição uniforme.

Se $\delta \leq e^{(-\frac{\Delta E}{T})}$, então a nova solução produzida é aceita como solução atual. Caso contrário, a solução atual não sofre alteração.

Abaixo, segue o pseudo-código do algoritmo *simulated annealing*:

Algoritmo 3 – Pseudo-código *Simulated Annealing*

1. $s_0 \leftarrow$ Solução inicial em S
2. $T_0 \leftarrow$ Temperatura inicial
3. Para t = 0 até I - 1:
    a. Gera nova solução S'
    b. Se $f\ (S') < f\ (S_i)$
    c. $S_{i+1} \leftarrow S'$
    d. senão
    e. $S_{i+1} \leftarrow S'$ com probabilidade $e^{\frac{f(S')-f(S_i)}{T_{i+1}}}$
    f. Atualiza temperatura T_i
4. Retorne S_i

No algoritmo acima, S representa o espaço de possíveis soluções e f representa uma função de custo real. O objetivo do algoritmo é encontrar o mínimo global s, tal que f(s) $\leq f(s')$, $\forall\ s' \in$ S. O algoritmo tem fim quando é atingida a quantidade máxima de iterações (I), retornando a melhor solução encontrada (solução atual) [38].

3.3 Tabu Search

O algoritmo de busca *tabu search* foi desenvolvido independentemente por Glover [39] e Hansen [40] para solucionar problemas de otimização combinatória.

Tabu search é um método de busca iterativa e é caracterizado pelo uso de uma memória flexível. A cada iteração do algoritmo, um número limitado de novas soluções é gerado e avaliado. Dessa quantidade de soluções avaliadas, a melhor solução, em termos de função de custo, é aceita como a nova solução atual mesmo que o seu custo seja maior do que o custo da solução corrente. Assim, explorando a vizinhança e escolhendo a melhor solução dessa vizinhança para substituir a solução atual, o algoritmo *tabu search* consegue escapar de mínimos locais [30].

Durante as iterações, o algoritmo faz uso de uma lista (memória) onde são armazenadas as N soluções visitadas recentemente. A finalidade da lista é evitar que haja ciclos durante a exploração da vizinhança sinalizando quais movimentos devem ser evitados (*tabu*) nas próximas iterações [33].

O tamanho da lista *tabu* é um fator crucial que influencia diretamente o desempenho do algoritmo. Ao definir o tamanho da lista, deve-se tomar cuidado para que não seja escolhido um valor muito pequeno ocasionando, assim, ciclos na busca e limitando o espaço de busca a pequenas regiões, nem que seja escolhido um valor muito grande, acarretando um aumento no custo computacional [33].

Definido o tamanho da lista como T, serão armazenadas na memória as T soluções recentemente visitadas. Quando o tamanho da lista é atingido, a solução mais antiga armazenada na lista é removida para que seja armazenada a nova solução que não é *tabu*, obedecendo à política FIFO (*first-in-first-out*).

No algoritmo, S representa o espaço de possíveis soluções, S_{BSF} representa a melhor solução encontrada até o momento e f representa uma função de custo real. O objetivo do algoritmo é encontrar o mínimo global s, tal que f (s) $\leq f(s')$, $\forall\ s' \in$ S. O algoritmo tem fim quando é atingida a quantidade máxima de iterações (I) retornando a melhor solução encontrada (S_{BSF}). Abaixo, segue o pseudo-código:

Algoritmo 4 – Pseudo-código *Tabu Search*

1. $s_0 \leftarrow$ Solução inicial em S
2. Atualize S_{BSF} com s_0 (melhor solução encontrada até o momento

3. Insira s_0 na lista *tabu*
4. Para t = 0 até I - 1:
    a. Gere um conjunto de novas soluções a partir da solução atual
    b. Escolha a melhor solução S' desse conjunto que não seja *tabu*
    c. $S_{i+1} \leftarrow S'$
    d. Atualize a lista *tabu* inserindo S_{i+1}
    e. Atualize S_{BSF} se $f\ (S_{i+1}) < f\ (S_{BSF})$
5. Retorne S_{BSF}

3.4 Trabalhos Relacionados

Em [5], são apresentados vários artigos que tratam da integração das três técnicas de otimização abordadas nesse trabalho (*tabu search*, *simulated annealing* e algoritmos genéticos) para diferentes aplicações como, por exemplo, roteamento de veículos [40], planejamento de engenharia [42] e segmentação de imagens [43].

Zanchettin propôs em sua tese a integração das três técnicas para a otimização dos pesos e da arquitetura de redes neurais do tipo MLP.

São muitos os trabalhos que tratam da otimização da arquitetura, dos pesos e dos parâmetros livres dos algoritmos de treinamento, mas pouco se produz no campo da otimização da função de ativação dos nodos que constituem a rede neural, embora, tenha sido demonstrado o importante papel da função de ativação no projeto das redes neurais artificiais e o impacto significante no desempenho das mesmas [15].

Como o foco do presente trabalho é a otimização da função de ativação dos nodos, serão discutidos adiante artigos que trabalham em cima desse tópico.

Em Yung et al [44], é apresentada uma aplicação original para otimização dos pesos e da arquitetura de RNAs onde os parâmetros da arquitetura (valores dos pesos, número de camadas escondidas, número de nodos em cada camada e tipo de função de ativação dos nodos) foram codificados através de um esquema de codificação numérico misto. Os pesos foram codificados através de números reais e os outros parâmetros foram codificados através de números inteiros. Como funções de ativação, foram utilizadas as funções sigmóide logística, linear, tangente hiperbólica e função de base radial. Os resultados foram obtidos validando o sistema com uma única base de predição. Todo o processo evolutivo foi conduzido por um algoritmo evolucionário (AE).

Abraham [3] propôs um AE para otimização do algoritmo de aprendizagem, da função de ativação e da arquitetura de RNAs. O esquema de codificação adotado foi o binário e as únicas funções de ativação usadas foram a sigmóide logística e a tangente hiperbólica. O algoritmo foi validado apenas com bases de predição. Abraham relata a obtenção de bons resultados e destaca a influência das funções de ativação na velocidade de treinamento e no desempenho da generalização.

Yao et al [45] propuseram um método automático para projetar RNAs com diferentes funções de ativação para os nodos de diferentes camadas. Foram utilizados um AE e novos operadores de mutação para conduzir a otimização simultânea da arquitetura e dos pesos. As funções de ativação adotadas foram a função gaussiana e a sigmóide logística.

Ferentinos [46] tratou da otimização dos parâmetros do algoritmo de aprendizado, da arquitetura e dos tipos de função de ativação. A codificação binária foi utilizada e o processo evolucionário foi conduzido por um único AG. Como funções de ativação, foram usadas a sigmóide logística e a tangente hiperbólica. Ferentinos aponta como vantagens do seu método a busca automática por configurações de RNAs ótimas, a diminuição da interferência humana, a robustez quando aplicado a redes *feedforward* e a capacidade de explorar o processamento paralelo. Como desvantagens, ele aponta a incidência em mínimos locais quando a população inicial do AG é muito pequena e automatização incompleta já que os parâmetros do AG devem ser escolhidos pelo usuário.

3.5 Considerações Finais

Neste capítulo, vimos o funcionamento e as principais características de três algoritmos iterativos gerais com destaque para os algoritmos genéticos devido a sua grande utilização em problemas que envolvem a otimização de redes neurais artificiais. Além disso, foram abordados alguns trabalhos que envolvem otimização de função de ativação, assunto pouco tratado na literatura.

4 ALGORITMO GaTSa

Como visto anteriormente, os algoritmos genéticos [28], *Tabu Search* [30] e *Simulated Annealing* [29] são os algoritmos mais utilizados para solucionar problemas de otimização. Como principais características de cada método, pode-se destacar a

capacidade do *tabu search* de escapar de mínimos locais explorando a vizinhança e escolhendo a melhor solução da vizinhança como solução atual. Além disso, o método *tabu search* faz uso de uma memória com o intuito de evitar ciclos (mais de uma visita a mesma solução) durante a busca. Já o método *simulated annealing* consegue evitar os mínimos locais escolhendo entre aceitar ou não uma solução que deteriore o custo da solução atual. Os algoritmos genéticos, por conseguinte, fazem uso de operadores genéticos (seleção, recombinação e mutação) que são aplicados às soluções candidatas de modo a obter a melhor solução a partir da população inicial de soluções.

Em [5], os pontos fortes dessas três técnicas são combinados originando um novo algoritmo de otimização batizado como GaTSa (acrônimo formado pelas inicias em inglês das três técnicas envolvidas).

Essa nova técnica de otimização é formada por duas fases: a fase de otimização global e a fase de otimização local. A otimização global é responsável pela busca da melhor solução presente no espaço de soluções que forma o problema, ficando a cargo da integração das heurísticas de *simulated annealing*, *tabu search* e algoritmos genéticos. Já a fase de otimização local tem como objetivo realizar o ajuste fino da solução retornada pela fase global. Esse ajuste fino é feito por um algoritmo de otimização local. No referido trabalho, o algoritmo de otimização local utilizado foi o algoritmo *backpropagation* [8].

A busca pela melhor solução realizada pelo algoritmo é feita seguindo uma abordagem construtiva, de forma que as soluções iniciais possuem a topologia mínima de uma RNA válida. Durante o processo de otimização, conexões e unidades de processamento são adicionadas ou eliminadas à medida que se faz necessário.

4.1 Descrição

Além de serem algoritmos de otimização, funcionarem através de uma sequência de iterações para retornar a melhor solução encontrada durante a busca, as três técnicas que formam o algoritmo GaTSa (algoritmos genéticos, *tabu search* e *simulated annealing*) incorporam conhecimento específico sobre o domínio do problema para realizar a otimização.

A principal diferença entre os três algoritmos está na maneira como o conhecimento é utilizado e onde ele é utilizado. Na técnica *simulated annealing*, o conhecimento está na função de custo que é utilizada para decidir entre aceitar ou não a melhor solução da vizinhança como solução atual.

Já nos algoritmos genéticos, o conhecimento está presente no valor de aptidão das soluções, na seleção dos indivíduos, nos operadores genéticos e na maneira como são geradas as novas populações.

No método *tabu search*, o conhecimento acerca do domínio do problema está presente na forma como é gerada a vizinhança, no tamanho da lista e nas soluções *tabu* armazenadas na lista.

Do exposto, nota-se um ponto forte da técnica proposta que é a maior quantidade de informação acerca do problema utilizada durante a busca, advinda das três técnicas envolvidas.

O objetivo do algoritmo é retornar a melhor solução *s* do espaço de busca, sendo formalizado da seguinte forma: seja *S* um conjunto de soluções e f uma função de custo real, o algoritmo procura o mínimo global *s*, tal que $f(s) \leq f(s'), \forall\ s' \in S$.

Abaixo, segue o pseudo-código do algoritmo proposto [5]:

Algoritmo 5 – Pseudo-código GaTSa

1. $P_0 \leftarrow$ população inicial com K soluções s_k e tamanho s_z
2. *Lista* ← ∅ (lista tabu)
3. $T_0 \leftarrow$ temperatura inicial
4. Atualize S_{BSF} com a melhor s_k de P_0 (melhor solução até o momento)
5. Atualize *Lista* ← S_{BSF}
6. Para i = 0 até $I_{max} - 1$
7. Se i + 1 não for múltiplo de I_T
8. $T_{i+1} \leftarrow T_i$
9. Senão
10. $T_{i+1} \leftarrow$ nova temperatura
11. Se critério de parada baseado em validação for satisfeito
12. Interrompa execução da busca global
13. Aumente o tamanho dos cromossomos de P_i inserindo *z* nodos
14. $P_i \leftarrow P_z$
15. Para j = 0 até g_n
16. Gere uma nova população P' a partir de P_i
17. $P_i \leftarrow P'$
18. Escolha a melhor solução s_k de P_i que não seja solução tabu
19. Se $f(s_k) < f(s_i)$

20. $s_{k+1} \leftarrow s_k$
21. *Lista* ← *Lista* – (solução mais antiga) + (solução s_k)
22. Senão
23. $s_{k+1} \leftarrow s_k$ com probabilidade $e^{\frac{f(s')-f(s_k)}{T_{i+1}}}$
24. Lista ← Lista – (solução mais antiga) + (solução s_k)
25. Se $f(s_{k+1}) < f(S_{BSF})$
26. Atualize S_{BSF}
27. Fixe a topologia contida em S_{BSF} e use o valor das conexões como valores iniciais para o algoritmo de busca local

---

No algoritmo acima, a temperatura é atualizada a cada I_T iterações. Utiliza-se um esquema de esfriamento geométrico baseado no cálculo de uma probabilidade (ver seção 4.6). A temperatura desempenha um papel importante no algoritmo da seguinte maneira: se a temperatura for muito baixa, a exploração no espaço de busca fica comprometido visto que em temperaturas muito baixas a chance de aceitar uma solução que deteriore o custo da solução atual é reduzida. Por outro lado, se for muito alta, o algoritmo precisará de muitas iterações para convergir.

Para cada iteração, uma nova população é gerada a partir da população atual. Cada solução da nova população tem seu custo avaliado e a solução com menor custo é escolhida para substituir a solução atual.

A memória da técnica *tabu search* é utilizada para armazenar as soluções da vizinhança visitadas recentemente de forma a evitar ciclos no processo de busca. A diferença para a técnica *tabu search* original é que nem sempre a melhor solução da vizinhança, em termos da função de custo, é aceita. A solução é aceita ou não dependendo do cálculo de uma probabilidade como em *simulated annealing*.

O algoritmo tem fim após serem atingidas as I_{max} iterações ou se o critério de parada baseado no erro de validação for satisfeito. Ao final, a melhor solução encontrada até o momento é retornada S_{BSF} (b*est so far*).

4.2 Representação das Soluções

A arquitetura de uma rede neural artificial é formada pelo número de camadas escondidas, pela quantidade de nodos nessas camadas e pela função de ativação dos nodos das camadas escondidas e de saída [47]. Em [5], o número de camadas intermediárias foi fixado, ficando restrito a uma camada intermediária. As redes,

inicialmente, são *full-connected*, ou seja, entre nodos de camadas adjacentes são estabelecidas todas as possíveis conexões. Assim, a topologia máxima pertimitida por esse esquema fica definida pela fórmula 4.1.

$$N_{max} = N_1 \times N_2 + N_2 \times N_3 \tag{4.1}$$

Onde N_1 é o número de nodos (depende do problema) da camada de entrada, N_3 é o número de nodos (depende do problema) da camada de saída e N_2 é o número de nodos (definido pelo projetista da rede) na única camada intermediária.

O esquema de codificação original proposto por Zanchettin [5] foi uma mistura do esquema binário e real. Casa solução *s* é codificada por um vetor formado pelas C conexões e pelos P pesos. A informação contida nesse vetor fornece a existência ou não de uma conexão entre duas unidades de processamento e o valor do peso associado a essa conexão. A existência da conexão é representada por um bit de conectividade possuindo o valor 1 caso a conexão exista e o valor 0 caso não exista. Já os pesos são representados por números reais ($\Re$).

$$s = (C, P) \tag{4.2}$$

$$C = (c_1, c_2, \dots, c_{N_{max}}), \quad c_i \in \{0,1\}, \quad i = 1,2, \dots, N_{max} \tag{4.3}$$

$$P = (p_1, p_2, \dots, p_{N_{max}}), \quad p_i \in \Re, \quad i = 1,2, \dots, N_{max} \tag{4.4}$$

Segundo Zanchettin [5], o método foi desenvolvido como um algoritmo iterativo geral, podendo implementar uma diversidade de problemas, bastando para isso adaptar a representação da solução, função de custo e o mecanismo de gerar novas soluções sobre o espaço de busca.

A modificação proposta no presente trabalho, que é a de realizar não somente a otimização da arquitetura e dos pesos simultaneamente, mas também incluir a otimização da função de ativação, foi conseguida através da modificação da representação da solução.

Dessa forma, uma solução passa a ser formada pelos bits de conectividade seguidos dos respectivos valores de pesos de forma intercalada concatenados com os bits que representam o tipo da função de ativação dos nodos da camada de saída e da

única camada intermediária. Optou-se por adicionar novos bits à solução com o objetivo de aproveitar os operadores genéticos já utilizados no trabalho original [5], não sendo necessário criar um novo vetor e carregá-lo junto com o vetor que representa a solução ao longo de todo processo evolucionário.

A quantidade de bits usada para indicar qual o tipo de função utilizada pelos nodos da camada intermediária e da camada escondida depende da quantidade de funções que forma o conjunto de possibilidades. Em [44], são usados quatro diferentes tipos de funções. Nesse caso, dois bits seriam suficientes para representar as quatro funções utilizadas na representação adotada.

No presente trabalho, o conjunto de funções utilizado é formado por seis funções. Sendo três funções nativas do Matlab (função sigmóide logística, tangente hiperbólica e função de base radial) [48] e três funções provenientes do trabalho de Gomes [49], onde ela propõe, implementa e fundamenta, matematicamente, o uso dessas novas funções em redes do tipo *multi-layer perceptron*. Nesse caso, a quantidade de bits necessária para representar as seis funções utilizadas seriam três bits, gerando oito cadeias de três bits, sendo mais do que o necessário.

Funções e seus genótipos			
1 - 000	logsig	5 - 100	probit
2 - 001	tansig	6 - 101	clolog
3 - 010	radbas	7 - 110	**random**
4 - 011	sech	8 - 111	**random**

Tabela 1 - Conjunto de funções usadas com seus genótipos

Na tabela acima, as funções 1, 2 e 3 são provenientes do Matlab [48] e as funções 4 – 6 são provenientes do trabalho de Gomes [49]. Como são geradas duas codificações extras (mais do que o necessário), é gerado um valor randômico no intervalo [1,6] e a função associada ao valor retornado é atribuída aos genótipos 7 e 8. Na figura 10, é mostrada uma arquitetura de rede neural e a sua respectiva codificação pelo sistema adotado:

Figura 10 - Codificação de uma rede neural MLP

Na figura acima, os três penúltimos bits informam que a função de ativação usada pelos nodos da única camada intermediária é a função sech e os três últimos bits informam que a função de ativação usada pelos nodos da camada de saída é a função sigmóide logística (logsig).

4.3 Inserção de Neurônios na Camada Escondida

Como já mencionado, o processo de otimização é realizado seguindo uma abordagem construtiva (linha 13 do algoritmo 5), ou seja, nodos e suas respectivas conexões são adicionados à medida que forem necessários visando atingir uma topologia mínima que consiga resolver, satisfatoriamente, variações do mesmo problema.

Com o objetivo de evitar o crescimento demasiado das redes, as unidades de processamento são adicionadas seguindo uma regra probabilística. Segundo esta regra, a nova probabilidade de adicionar novas unidades é igual à probabilidade atual multiplicada por um fator de redução (ε), que deve ser menor do que *1*. A probabilidade inicial e o fator de redução (ε) devem ser definidos pelo usuário. Assim, a probabilidade λ_i da i-ésima iteração é definida pela equação 4.5.

$$\lambda_i = \begin{Bmatrix} \varepsilon\lambda_{i-1}, \ se\ i = kI_\lambda, \quad k = 1,2,\ldots,\frac{I_{max}}{I_\lambda} \\ \lambda_i, \ caso\ contrário \end{Bmatrix} \tag{4.5}$$

4.4 Mecanismo de Geração de Vizinhança

A geração de vizinhança é usada para obter novas soluções a partir da solução atual (linha 16 do algoritmo 5). A solução inicial é gerada de forma aleatória e representa uma rede neural de topologia válida, sendo N_1 (número de nodos de

entrada) e N_3 (número de nodos de saída) variáveis dependentes do problema e N_2 (número de nodos ocultos) menor igual a N_3.

Os cromossomos são classificados por *Rank Based Fitness Scalling* [50] para serem selecionados para reprodução posteriormente.

Para selecionar os pais da nova geração, utilizou-se *Universal Stochastic Sampling* [50].

Não foi utilizado elitismo no algoritmo. Para a combinação das informações gênicas dos pais, foi utilizado o operador *Uniform Crossover* [51]. Como os pais podem ter comprimentos diferentes, é utilizada uma máscara com bits aleatórios de tamanho igual ao menor dos pais, sinalizando em quais pontos as informações devem ser trocadas (ver figura 11). Se o bit da i-ésima posição da máscara for 0, os bits da i-ésima posição dos pais não devem ser trocados, caso contrário, são trocados. O operador de mutação utilizado foi o Gaussian Mutation [52] que adiciona um valor aleatório retirado de uma distribuição gaussiana a cada elemento indivíduo selecionado para criar a nova população.

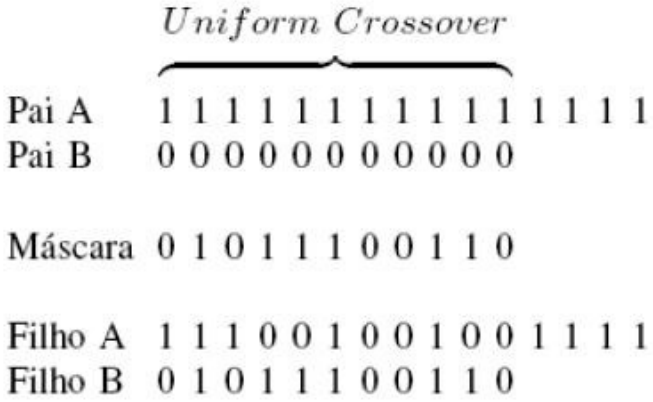

Figura 11 - Recombinação em cromossomos de tamanho diferentes

4.5 Avaliação de Custo

Como a cada iteração novas soluções são geradas, uma função de custo se faz necessária para avaliar qual das soluções é a melhor solução em termos de função objetivo (linha 18 do algoritmo 5).

Sendo N_C as possíveis classes de uma base de dados, a classe verdadeira de um padrão *x* do conjunto de treinamento P_t é dada pela equação 4.6:

$$\gamma(x) \in \{1,2,\ldots,N_C\},\ \forall\ x \in P_t \tag{4.6}$$

Para problemas de classificação, a regra adotada para definir a saída da rede neural foi a regra *winner-takes-all*. Segundo essa regra, um padrão que for apresentado à rede vai pertencer à classe representada pelo neurônio da camada de saída com maior valor (o número de nodos na camada de saída é igual ao número de classes do problema).

Sendo o_k(X) o valor de saída da unidade de processamento de saída k para o padrão X, a classe associada ao padrão X é dada pela equação 4.7.

$$\emptyset(X) = \arg\max\ o_k(\mathrm{X})\ ,\forall\ x\ \in\ P_t\ ,\ \mathrm{k} \in \{1,2,\ldots,N_3\} \tag{4.7}$$

De posse da classe real e da classe fornecida pela rede neural, o erro cometido pela rede para o padrão X pode ser formulado pela equação 4.8.

$$\varepsilon(X) = \begin{Bmatrix} 1\ ,\ se\ \emptyset(X) \neq\ \gamma(X) \\ 0,\ se\ \emptyset(X) =\ \gamma(X) \end{Bmatrix} \tag{4.8}$$

A partir do erro de um padrão individual, o erro de classificação para o conjunto de treinamento P_t (porcentagem de padrões de treinamento classificados de maneira incorreta) é dado pela equação 4.9.

$$E(P_t) = \frac{100}{\#\ P_t} \sum_{X\ \in P_t} \varepsilon(X) \tag{4.9}$$

onde # P_t representa a quantidade de padrões de treinamento.

Por sua vez, a porcentagem de conexões utilizada pela rede é dada pela equação 4.10:

$$\varphi(C) = \frac{100}{N_{max}} \sum_{i=1}^{N_{max}} C_i \tag{4.10}$$

onde N_{max} é o número máximo de conexões presente na topologia.

Em [5], são descritas as cinco maneiras de se avaliar o custo de uma solução abordadas no presente trabalho. O objetivo de todas elas é minimizar

simultaneamente o erro de treinamento e o número de conexões de topologias de redes neurais artificiais válidas.

4.6 Esquema de Esfriamento

Para atualização da temperatura que faz parte do código referente à técnica *simulated annealing*, é utilizado o esquema geométrico de esfriamento (linha 23 do algoritmo 5), onde a nova temperatura é dada pela temperatura atual multiplicada por um fator de redução (*r*), que deve ser menor do que 1, mas próximo de 1 [53] . A temperatura inicial T_0 e o fator de redução são parâmetros de implementação definidos pelo usuário, bem como I_T (máxima quantidade de iterações entre duas atualizações sucessivas da temperatura) e I_{max} (máximo número de iterações permitidas). Assim, a temperatura da i-ésima iteração é dada pela equação 4.11 abaixo:

$$T_i = \begin{cases} rT_{i-1}, & se\ i = kI_T, \quad k = 1,2,\dots,\frac{I_{max}}{I_t} \\ T_i, caso\ contrário & \end{cases} \tag{4.11}$$

4.7 Critérios de Parada

O treinamento é interrompido quando é satisfeito o critério GL_5 do *Proben1* [54] por duas vezes ou se forem atingidas as I_{max} iterações (linha 6 do algoritmo 5).

O critério GL_5 indica a perda de generalização durante o treinamento da rede neural, sendo utilizado para evitar a incidência de *overfitting* [54]. Esse critério é definido como o aumento do erro de validação em relação ao menor erro de validação obtido. Nas equações 4.12 e 4.13 são mostrados, respectivamente, o menor erro de validação obtido e a perda de generalização obtida durante a i-ésima iteração:

$$E_{opt}(i) = \min_{i' \le i} E_{va}(i') \tag{4.12}$$

$$GL_{(i)} = 100 \times \left(\frac{E_{va}(i)}{E_{opt}(i)} - 1\right) \tag{4.13}$$

Nas equações acima, *E* representa o erro quadrático, E_{va} (*i*) representa o erro do conjunto de validação após a i-ésima iteração e $E_{opt}(i)$ representa o menor erro de validação obtido até a i-ésima iteração.

5 EXPERIMENTOS E RESULTADOS

Esse capítulo trará uma breve descrição das bases de dados utilizadas bem como as principais características dos experimentos realizados e dos resultados obtidos. Foram utilizados sete problemas de classificação e três problemas de previsão.

5.1 Bases de Dados

Esta seção traz uma descrição das bases de predição e classificação utilizadas nos experimentos como a sua origem, a quantidade de atributos que formam cada base, dentre outras características.

5.1.1 Base de dados do Nariz Artificial

A base de dados do nariz artificial trata da exposição de um nariz artificial utilizando redes neurais artificiais [55] a odores oriundos de três diferentes safras de um mesmo vinho tinto comercial (Almadém, Brasil - safra 1995, 1996 e 1997) feito à base de uvas do tipo merlot. Com esse protótipo foram realizados testes de sensibilidade no arranjo de seis sensores baseados em filmes de *polipirrol* (polímero condutor, geralmente derivado do petróleo, que conduz eletricidade). Os seis sensores foram preparados com diferentes dopantes, por polimerização *in situ* (no local de origem), em substrato de *ITO* (substrato vítreo condutor).

A base de dados foi montada a partir da equipe multidisciplinar do Projeto Aroma, formada por membros dos departamentos de Informática, Física e Química Fundamental da Universidade Federal de Pernambuco [56].

O processo para aquisição dos padrões de odores do nariz artificial envolveu três etapas distintas:

i. Coleta da linha de base (com duração de 10 minutos). Os sensores são expostos a um gás inerte;

ii. Coleta dos valores de resistência do arranjo de sensores, frente ao gás analisado (com duração de 40 minutos);
iii. Purga ou limpeza dos sensores com gás Nitrogênio (com duração média de 5 minutos).

5.1.2 Base de dados Diabetes

De nome original "*Pima Indians Diabetes Database*", essa base pertence ao *National Institute of Diabetes and Digestive and Kidney Diseases*. A base diabetes contém dados pessoais e resultados de exames médicos de pacientes, usados para a classificação da chance de um índio Pima ser diabético ou não. Todos os pacientes que formam a base de dados são mulheres com idade mínima de 21 anos e pertencentes à linhagem de Índios Pima que vivem próximos a Phoenix, Arizona, EUA. A base foi obtida do repositório UCI [57] de base de dados para aprendizagem de máquina.

A base é formada por 768 padrões (pacientes) com 8 atributos contínuos, que correspondem a informações médicas:

i. Número de vezes que a paciente ficou grávida;
ii. Concentração de glicose no plasma em teste de tolerância a glicose oral de 2 horas;
iii. Pressão sanguínea mínima (diastólica) (mm/Hg);
iv. Medida das dobras na pele do tríceps (mm);
v. Aplicações de 2 em 2 horas de soro com insulina (um U/ml);
vi. Índice de massa corpórea (peso em Kg/ (altura em m)^2);
vii. Idade da paciente em anos.

A base de dados informa se as pacientes observadas apresentam sinais de diabetes de acordo com critérios definidos pela Organização Mundial de Saúde (*World Health Organization*), isto é, se 2 horas após a aplicação de plasma glicose existem pelo menos 200 mg/dl em qualquer exame correspondendo a duas classes binárias.

A análise dos dados foi realizada a partir de um algoritmo que gerou resultados no intervalo [0,1]. Os valores contínuos dos intervalos foram transformados em valores binários 0 e 1 posteriormente sendo:

i. Valor 0 (zero): indica que o paciente não possui diabetes - 500 registros (65,1%);
ii. Valor 1 (um): o paciente possui diabetes - 268 registros (34,9%).

5.1.3 Base de dados Íris de Fisher

Também proveniente do UCI [57], esta base de dados contém 50 instâncias de cada classe, onde cada classe representa um tipo de Íris (Setosa, Virgínia e Versicolor). Uma classe é linearmente separável das outras duas e as outras duas não são linearmente separáveis uma da outra. A base contém 4 atributos e 150 padrões e foi originalmente apresentada por Fisher [58] ao representar os princípios da análise discriminante. A base de dados possui 150 registros, onde cada registro é formado por seis atributos:

i. Número que representa o tipo da espécie de flor;
ii. Nome da espécie (Íris setosa, Íris virgínia e Íris versicolor);
iii. Largura da pétala;
iv. Comprimento da pétala;
v. Largura da sépala;
vi. Comprimento da sépala.

5.1.4 Base de dados Tireóide

Esta base contém informações relacionadas à disfunção da tireóide. O problema se resume a determinar se um paciente tem uma tireóide funcionando normalmente, com níveis excessivos de hormônio tireóide (hipertireóide) ou níveis insuficientes desse mesmo hormônio. Há 7200 casos na base, sendo 3.772 do ano de 1985 e 3.428 do ano de 2006 [59]. A classe dos pacientes com níveis excessivos do hormônio representa 2,3% (166 casos) dos exemplos, a classe hipotireóide representa 5,1% (386 casos) das observações, enquanto que o grupo de pacientes com níveis normais do hormônio representa 92,6% (6666 casos) dos padrões restantes. A base de dados é muito desbalanceada e é considerada um problema de difícil resolução para métodos convencionais de classificação. A base possui 21 atributos com 15 variáveis binárias e 6 contínuas, usadas para classificar o paciente em uma das três possíveis classes. Essa base também é proveniente do repositório UCI [57].

5.1.5 Base de dados Reconhecimento de Vinho

Os dados que formam essa base são provenientes da análise química de vinhos oriundos da mesma região da Itália, produzidos com diferentes tipos de uvas

[60]. As análises determinaram as quantidades de 13 componentes encontrados em cada um dos três tipos de vinho. Cada padrão é formado por 13 atributos e três classes. Esta base também foi obtida do repositório UCI [57].

Abaixo, são mostrados os treze atributos:

i. Teor alcoólico;
ii. Ácido málico;
iii. Cinzas;
iv. Alcalinidade das cinzas;
v. Magnésio;
vi. Fenóis;
vii. Flavanóides;
viii. Fenóis não-flavanóides;
ix. Antocianimas;
x. Intensidade da cor;
xi. Tonalidade da cor;
xii. Taxa de diluição;
xiii. Prolina.

5.1.6 Base de dados Identificação do Tipo de Vidro

O estudo da classificação dos tipos de vidros foi motivado pela investigação criminológica. Na cena do crime, o vidro deixado pode ser usado como uma evidência se for corretamente identificado. Dada essa motivação, a presente base, de título original "Glass Identification", trata justamente da classificação do tipo de vidro. Essa base contém 214 registros, que são formados por 11 atributos (incluindo um identificador (ID) que foi descartado) e pelo atributo classe que identifica o tipo de vidro. As possíveis classes (tipos de vidro) são:

i. Vidro usado para construir janela processado com ar;
ii. Vidro usado para construir janela processado sem ar;
iii. Vidro usado para construir veículo processado com ar;
iv. Vidro usado para construir veículo processado sem ar (não presente na base de dados);
v. Vidro usado na fabricação de louça;
vi. Vidro usado na fabricação de faróis.

Todos os valores dos atributos são contínuos. Abaixo, seguem as informações que compõem cada registro:

i. Número de identificação (1 até 214);
ii. Índice de refração;
iii. Sódio (peso percentual no correspondente óxido);
iv. Magnésio;
v. Alumínio;
vi. Silício;
vii. Potássio;
viii. Cálcio;
ix. Bário;
x. Ferro;
xi. Tipo de vidro.

A base de dados foi obtida do UCI [57].

5.1.7 Base de dados Sobreviventes de Câncer de Mama

De título original "Haberman's Survival", essa base contém casos de estudo conduzidos entre 1958 e 1970 no hospital da universidade de Chicago (EUA) sobre a sobrevivência de pacientes que se submeteram a cirurgia de remoção de câncer de mama. A base é proveniente do repositório UCI [57], possui 306 registros, onde cada registro possui 4 atributos incluindo a classe a qual pertence. O problema se resume a enquadrar um paciente no conjunto dos pacientes que morreram dentre 5 anos após a cirurgia ou no conjunto de pacientes que tiveram uma sobrevida de 5 anos ou mais.

Os atributos que formam cada padrão são:

i. Idade do paciente na época da cirurgia;
ii. Ano da realização da cirurgia (ano - 1900);
iii. Número de nodos positivos periféricos;
iv. Classe a qual o padrão pertence.

5.1.8 Série Caótica *Mackey-Glass*

A série Mackey-Glass é uma série temporal contínua, unidimensional e apresenta oscilações quase periódicas. Tem como objetivo modelar a dinâmica da produção de células brancas do sangue de pacientes com câncer. A série resulta da

integração diferencial caótica com atraso proposta por Mackey e Glass [61], dada pela equação (5.1):

$$\frac{dx}{dt} = -bx(t) + \alpha \frac{x(t-\tau)}{1+x(t-\tau)} \qquad (5.1)$$

A série não apresenta um período definido de forma clara, sendo sensível às condições iniciais e dependente dos valores τ, α e b e x(0), e ainda pode apresentar comportamente caótico. Para a construção da série, fixaram-se os parâmetros $\alpha = 0.2$, b = 0.1, $\tau = 17$, x(0) = 1.2 e x(t) = 0 para t < 0. Foram usados valores no intervalo $t \in$ [0; 2000], onde foram selecionados 1000 pares entrada-saída no intervalo $t \in$ [118;117]. A série Mackey-Glass é reconhecida como uma referência no estudo de aprendizagem de máquina e da capacidade de generalização em diferentes classificadores.

O treinamento foi realizado usando 500 pontos (t = 118 a t = 618). Destes padrões, 250 pontos foram utilizados para validação (t = 618 a t = 868), e a rede neural artificial foi testada com outros 250 pontos (t = 868 a t = 1118). Replicações foram realizadas utilizando os valores da série temporal em diferentes pontos temporais.

5.1.9 Série *Airline Passenger*

O conjunto de dados *airline passenger* foi primeiro utilizado por Brown (1962) e posteriormente por Box e Jenkins (1976). Esta base mede o número total de passageiros em viagens internacionais no período compreendido entre janeiro de 1949 e dezembro de 1960. Os dados apresentam não-linearidade e exibem um comportamento sazonal. A não-linearidade faz dessa base um candidato para medir a efetividade de métodos de previsão [62]. A série é formada por 144 valores numéricos.

5.2 Metodologia de Treinamento

O algoritmo utilizado no treinamento da rede neural foi o algoritmo *backpropagation* com momento [8]. Cada topologia de RNA válida teve seus pesos inicializados com valores diferentes e aleatórios. No total, foram 30 inicializações.

O treinamento é interrompido se o critério GL_5 do *Proben1* [54] for atingido por duas vezes (para evitar que o treinamento seja interrompido precocemente por oscilações iniciais no erro de validação), se for satisfeito o critério de progresso de treinamento do *Proben1* [54] com $P_5(t) < 0.1$ ou se a quantidade máxima de iterações $I_T = 5000$ for alcançada.

Nos experimentos realizados, a base de dados foi dividida em três conjuntos disjuntos de dados (Treinamento, Validação e Teste). A divisão dos padrões entre os três conjuntos se deu da seguinte forma: 50% dos padrões formaram o conjunto de treinamento, 25% dos padrões formaram o conjunto de validação e os 25% restantes formaram o conjunto de teste. Tal divisão obedece ao relatório [54].

Para tornar o treinamento mais efetivo e rápido, os dados foram normalizados no intervalo [-1,+1]. A normalização dos dados não deixa que a rede tenha uma maior consideração pelos valores de grande magnitude, confundindo o aprendizado da rede. Dessa forma, as técnicas de normalização eliminam esse problema convertendo os dados para intervalos bem definidos.

5.3 Experimentos com a Técnica de Otimização GaTSa

A população inicial foi definida com 10 cromossomos. Não foi utilizado o elitismo. O operador de recombinação gênica utilizado foi o *Uniform Crossover* [51] com uma probabilidade de 20%. O operador de mutação foi o *Gaussian Mutation* [52] com uma probabilidade de 10%.

O esquema de esfriamento utilizado foi a regra de esfriamento geométrico com valor inicial para a temperatura igual a 1 e com fator de redução igual a 0,9.

O método GaTSa com a otimização de função é comparado com o método GaTSa original (sem a otimização de função).O método GaTSa integra as heurísticas das técnicas *Tabu Search*, *Simulated Annealing*, Algoritmos Genéticos e *Backpropagation.* Além disso, realiza-se uma busca construtiva baseada na poda de conexões (otimização do número de neurônios escondidos). Para a realização dos experimentos, a topologia máxima da rede neural para cada conjunto de dados foi de :

- Nariz Artificial ($N_1 = 6$, $N_2 = 10$, $N_3 = 3$, $N_{max} = 90$);
- Íris de Fisher ($N_1 = 4$, $N_2 = 5$, $N_3 = 3$, $N_{max} = 35$);
- Tireóide ($N_1 = 21$, $N_2 = 10$, $N_3 = 3$, $N_{max} = 240$);
- Diabetes ($N_1 = 8$, $N_2 = 10$, $N_3 = 2$, $N_{max} = 100$);
- Mackey-Glass ($N_1 = 4$, $N_2 = 4$, $N_3 = 1$, $N_{max} = 20$);

- Haberman ($N_1 = 3$, $N_2 = 5$, $N_3 = 2$, $N_{max} = 25$);
- Glass ($N_1 = 9$, $N_2 = 5$, $N_3 = 6$, $N_{max} = 75$);
- Wine ($N_1 = 13$, $N_2 = 5$, $N_3 = 3$, $N_{max} = 80$);
- AirlinePassenger ($N_1 = 3$, $N_2 = 4$, $N_3 = 1$, $N_{max} = 16$)

Para todas as topologias, os valores N_1 (número de nodos de entrada) e N_3 (número de nodos de saída) são dependentes do problema. Os valores de N_2 (números de nodos escondidos) foram definidos empiricamente.

Na tabela 5.1, é mostrado o desempenho médio do algortimo GaTSa com e sem a otimização de função. Para efeito de comparação, foram avaliados o erro de classificação do conjunto de teste, o número médio de nodos de entrada, o número médio de nodos escondidos, a porcentagem de conexões da rede neural ao final da otimização (em relação ao número máximo de conexões da topologia máxima permitida) e o SEP (erro percentual quadrático).

Nessa tabela são apresentados os resultados médios obtidos em 10 simulações onde cada simulação compreende 30 execuções diferentes do algoritmo.

Tabela 2 - Desempenho médio das técnicas GaTSa e GaTSa + OF (Otimização Função)

Base de Dados	Valores	GaTSa	GaTSa + OF
Nariz Artificial	Erro Class. (%)	0.7914	14.9042
	Entradas	5.2267	2.2100
	Escondidos	4.4867	2.1533
	Connec. (%)	36.2254	9.8100
Íris de Fisher	Erro Class. (%)	5.2256	26.3333
	Entradas	3.3600	1.5567
	Escondidos	4.1333	1.900
	Connec. (%)	31.8538	11.6700
Tireóide	Erro Class. (%)	7.1509	7.8748
	Entradas	7.1233	1.7300
	Escondidos	2.0833	1.6610
	Connec. (%)	12.6400	2.8200

Diabetes	Erro Class. (%)	27.0615	33.2066
	Entradas	1.5100	1.0500
	Escondidos	1.200	1.1100
	Connec. (%)	9.0975	6.2100
Mackey-Glass	SEP de Teste	0.7300	7.0390
	Entradas	1.0533	1.0200
	Escondidos	1.0100	1.0333
	Connec. (%)	11.4923	12.0400
Glass	Erro Class. (%)	52.7885	58.8500
	Entradas	7.0300	1.4900
	Escondidos	5.7100	1.5467
	Connec. (%)	28.5700	3.8700
Haberman	Erro Class. (%)	7.9259	25.978
	Entradas	12.1567	1.0000
	Escondidos	5.7733	1.1300
	Connec. (%)	38.9800	11.5100
Wine	Erro Class. (%)	26.1711	23.2074
	Entradas	1.0033	3.8433
	Escondidos	1.0033	2.2500
	Connec. (%)	10.1700	8.8200
AirPassenger	SEP de Teste	0.9348	5.5489
	Entradas	1.0100	1.0000
	Escondidos	1.0067	1.0000
	Connec.(%)	13.9000	10.7300

Comparando os resultados, verifica-se uma piora, para a maior parte das bases, no erro percentual de classificação para o conjunto de teste. Em contrapartida, constata-se que os valores obtidos pelo método GaTSa sem otimização de função referentes à média do número de entradas da rede, à média do número de neurônios escondidos utilizados pela rede e à porcentagem de conexões da rede neural artificial

ficaram muito próximos ou foram maiores (redes com maior número de conexões) do que os valores retornados pelo método com otimização de função.

Para a base Nariz Artificial, o número de neurônios na camada de entrada e escondida, na média, foram menores que os valores obtidos pelo método GaTSa original. Entretanto, o erro percentual de classificação saltou de 0.79% para 14.9%, sinalizando que o uso das novas funções (clolog, probit, radbas e sech) não se mostrou mais eficaz que o uso das funções tradicionais (logsig e tansig).

Para as bases de classificação Tireóide e Diabetes, as taxas de erro percentual de classificação obtidas pelo método GaTSa com otimização de função ficaram muito próximas das taxas obtidas pelo método GaTSa sem otimização de função. Além disso, as redes neurais obtidas pelo método que inclui otimização de função foram menores do que as redes obtidas pelo método GaTSa.

O método GaTSa com otimização de função foi melhor que o método GaTSa, em termos de erro percentual de classificação, apenas para a base de reconhecimento do vinho (*wine*). Entretanto, a topologia utilizada pelo método GaTSa com otimização de função foi maior do que a topologia usada pelo método GaTSa sem a otimização de função, levando a um aumento no tempo de treinamento da rede neural.

Para as duas séries temporais (*Mackey-Glass* e *AirPassenger*), os valores do erro percentual quadrático foram melhores do que as taxas de erro apresentadas pelo método GaTSa com otimização de função. Embora, tenha apresentado topologias de redes maiores.

Para todos os conjuntos de dados abordados nesse trabalho, as topologias de rede neural geradas pelo método GaTSa com otimização de função foram bem menores que as topologias apresentadas pelo método GaTSa original. Para alguns casos, o percentual de conexões da rede neural, na média, ficou entre 2% e 4%. Como são nos pesos que a rede neural armazena o conhecimento adquirido, durante a fase de aprendizagem, acerca do problema tratado, esse número reduzido de conexões não se mostra suficiente para que a rede extraia e armazene características importantes sobre a natureza do problema em questão.

O algoritmo GaTSa tira do projetista a responsabilidade de definir uma topologia máxima para a rede neural. O processo de otimização tem início com uma topologia mínima de uma rede neural válida formada por apenas um neurônio na única camada intermediária. Durante o processo evolucionário, novos nodos podem ser adicionados à camada intermediária. Esse processo de adição de novos nodos à camada escondida é controlado pelo valor da temperatura do método *Simulated*

Annealing. Como a média dos valores dos nodos escondidos obtida pelo método GaTSa com otimização de função se mostra muito baixa, uma medida a ser tomada seria aumentar o valor de temperatura inicial com o intuito de aumentar o espaço de busca a ser explorado. Com um valor inicial baixo, a temperatura estaria limitando o espaço de busca a ser explorado acarretando uma possível piora do método.

Com um espaço de busca maior conseguido através do aumento da topologia das redes neurais pela inserção de novos nodos na camada escondida, uma modificação na tentativa de obter melhores resultados seria aumentar o tamanho da lista tabu do método *Tabu Search* de modo que fosse avaliado um número maior de soluções para a escolha da melhor solução da vizinhança. O comprimento da lista tabu pode estar contribuindo para a piora dos resultados, levando a ocorrência de ciclos na busca ou deixando de explorar regiões promissoras do espaço de busca.

Para a maior parte das bases utilizadas, o uso das novas funções (clolog, probit, sech e radbas) não se mostrou eficaz na resolução dos problemas propostos. Como o desempenho dos modelos conexionistas está fortemente relacionado ao comportamento (natureza) dos dados, pode-se apontar como causa da deterioração dos resultados a não capacidade das novas funções de modelar satisfatoriamente os dados. Uma possível causa que também pode ser levantada é o uso de funções de ativação com diferentes faixas de ativação na camada escondida e de saída.

6 CONCLUSÃO

Neste trabalho, foi proposta uma mudança no algoritmo de otimização global e local GaTSa [5], que realiza a otimização simultânea da topologia e dos pesos das conexões.

Através dessa mudança, o algoritmo GaTSa passou a realizar não somente a otimização dos pesos e da topologia de redes neurais do tipo *multi-layer perceptron* como também passou a realizar a otimização da função de ativação no sentido de automatizar o processo de escolha da função de ativação dos nodos da camada escondida e de saída.

Para isso, foi utilizado um conjunto de possíveis funções de ativação formadas por três funções (logsig, tansig e radbas) oriundas do Matlab [48] e três funções (clolog, sech e probit) provenientes do trabalho de Gomes [49].

As contribuições do presente trabalho foram a constatação de que as funções de ativação sigmóide logística (logsig) e tangente hiperbólica (tansig), utilizadas na

maioria dos trabalhos científicos, mostram-se mais eficazes na resolução dos problemas abordados nesse trabalho e a realização de testes sobre novas bases não exploradas no trabalho original [5].

O uso das novas funções de ativação tanto pelo método GaTSa original como pelo método GaTSa modificado acarretou resultados piores para a maioria das bases.

Como trabalho futuro, seria interessante realizar novos experimentos seguindo as hipóteses levantadas na seção 2.3. Além disso, seria importante explorar a topologia da rede através da mudança dos valores dos parâmetros que podem estar tento um impacto negativo nos resultados como:

- A taxa que controla o aumento da vizinhança. Para que sejam exploradas novas áreas no espaço de busca, aconselha-se aumentar o número de soluções vizinhas;
- Alterar a forma como estão codificados os valores das bases. Nos experimentos realizados, os valores foram normalizados no intervalo [-1,+1].
- Verificar o desempenho do método quando mudados os valores das taxas de mutação e recombinação gênicas.

A questão da codificação dos valores da base pode estar contribuindo para a piora no desempenho do método devido ao fato de estarem sendo utilizadas funções com diferentes faixas de ativação. Dentre as seis funções utilizadas, duas (sech e tansig) possuem faixa de ativação [-1,+1]. As quatro restantes possuem faixa de ativação [0,1]. Com a possibilidade de escolha de uma função para os nodos da camada escondida com faixa [-1,+1] e de outra função para a camada de saída com faixa [0,1], pode haver uma perda de continuidade na geração da superfície de erro da rede neural pelo uso de funções com propriedades distintas nas camadas da rede. Para constatar essa observação, seria importante utilizar funções com a mesma faixa de ativação para formar o conjunto de funções do qual seriam selecionadas as funções para as camadas escondida e de saída da rede.

Outra proposta seria a realização da otimização de parâmetros ajustáveis das funções. No caso da função sigmóide logística e da tangente hiperbólica, esse parâmetro seria o valor b das equações (2.4) e (2.5) que representa o parâmetro de inclinação da curva sigmóide.

Referências

[1] – Widrow, B., Rumelhart, D.E. e Lehr, M.A. “Neural Networks: Application in Industry, Business and Science”, Communications of the ACM, Vol. 37(3), pp.93-105, 1994.

[2] – Jeffrey, J.L. e Vitter, J.S. “Complexity Results on Learning by Neural Nets”, in machine learning, pp. 211-230, 1991.

[3] – Abraham, A. “Meta Learning Evolutionary Artificial Neural Networks”. Neuro-Computing, Vol. 56, pp. 1 – 38, 2004.

[4] Papalambros, P. Y. e Wilde, D. J. “Principles of Optimal Design: Modeling and Computation”, Cambridge University Press, 412 p.

[5] – Zanchettin, C. “Otimização Global em Redes Neurais Artificiais”, Tese de Doutorado, Centro de Informática, Universidade Federal de Pernambuco, Recife, 2008, 146p.

[6] – Braga, A.P., Carvalho, A.P.L. e Ludermir, T.B. “Redes Neurais Artificiais: Teoria e Aplicações”, Rio de Janeiro: Livros Técnicos e Científicos, 262p, 2000.

[7] – McCulloch, W. e Pitts, W. “A Logical Calculus of the Ideas Immanent in Nervous Activity”, *Bulletin of Mathematical Biology*, pp. 115 – 133, 1943.

[8] – Rumelhart, D.E., Hinton, G.E. e Williams, R.J. “Learning Representations by Back-Propagating Errors”, Nature, Vol. 323, pp. 533-536, 1986.

[9] – Haykin, S. “Neural Networks – A Comprehensive Foundation”, New Jersey, Prentice Hall, 842p, 2001.

[10] – Zanchettin, C. “Sistema Neural Híbrido para Reconhecimento de Padrões em um Nariz Artificial”, Dissertação de Mestrado, Centro de Informática, Universidade Federal de Pernambuco, Recife, 2004.

[11] – Kimura, F., Inoue, S., Wakabayashi, T., Tsuruoka, S. e Miyake, S. “Handwritten Numeral Recognition Using Autoassociative Neural Networks”, *IEEE Transactions on Neural Networks*, Vol. 1, pp. 166 - 171, 1998.

[12] – Nakayaa, K., Hirano, A. e Fukumura, K.I. “On Generalization of Multilayer Neural Network Applied to Predicting Protein Secondary Structure”, *IEEE Transactions on Neural Networks*, Vol. 2, pp. 1209-1213, 2004.

[13] – Rosenblatt, F. “The Perceptron: a Probabilistic Model for Information Storage and Organization in the Brain”, Psychological Review, Vol. 65, n.6, pp. 386-408, 1958.

[14] – Minsky, M. e Papert, S. “Perceptrons: an Introduction to Computational Geometry”. MIT Press, Massachusetts, 1969.

[15] – Yao, X. “Evolving Artificial Neural Networks”, proceedings of the IEEE, Vol. 87, n.9, pp. 1423 – 1447, 1999.

[16] – Cybenko, G. “Approximation by Superpositions of a Sigmoid Function”. Mathematics of Control, Signals and Systems, Vol. 2, pp. 303 – 314, 1989.

[17] – Cybenko, G. “Continuos Valued Neural Networks with Two Hidden Layers are Sufficient”. Technical report, Department of Computer Science, Tufts University, 1988.

[18] – Kordos, M. e Duch, W. “A Survey of Factors Influencing Mlp Error Surface”. Control and Cybernetics, Vol. 33, pp. 611 – 631, 2004.

[19] – Duch, W. and Jankowski, N. “Transfer Functions: Hidden Possibilities for Better Neural Networks”. 9*th* European Symposium on Artificial Neural Networks, pp. 81-94, 2001.

[20] – Hebb, D.O. “The Organization of Behavior”. Wiley, 1949.

[21] - Moller, M.F. "A Scaled Conjugate Gradient Algorithm for Fast Supervised Learning". Neural networks, Vol. 6, n.4, pp. 525-533,1993.

[22] - Fletcher, R e Reeves, C.M. "Function Minimization by Conjugate Gradients", Computer Journal, Vol. 7, pp.149-154, 1964.

[23] - Haggan, M.T. e Menhaj, M.B. "Trainning Feedforward Networks with the Marquardt Algorithm", Transaction Neural Networks, Vol. 5, n. 6, pp. 989-993, 1994.

[24] – Bartlett, P.L. “For Valid Generalization, the Size of the Weights is more Importante than the Size of the Network”. Advances in Neural Information Processing Systems, The MIT Press, Vol. 9, pp.134-140.

[25] - Rumelhart, D. E. e McClelland, J.L. “Parallel Distributed Processing, Vol.1: Foundations. The MIT Press, 1986.

[26] – Bigus, J. P. “Data Mining with Neural Networks: Solving Business Problems from Application Development to Decision Support”, McGraw-Hill, New York, 1996.

[27] – Khebbal, S. e Goonatilake, S. “Intelligent Hybrid Systems: Issues, Classifications and Future Directions”, Intelligent Hybrid Systems, pp. 1 – 20, Wiley, London, 1995.

[28] – Holland, H.J. “Adaptation in Natural and Artificial Systems”. University of Michigan Press, 1975.

[29] – Kirkpatrick, S., Gellat, D.C. e Vecchi, M.P. 1983 “Optimization by Simulated Annealing”, Science, n. 220, pp. 671-680.

[30] – Glover, F. e Laguna, M. (1997). "Tabu Search", Kluwer academic publishers, Boston, 408p.

[31] – Darwin, C. “On the Origin of Species by Means of Natural Selection”. John Murray, London, 1859.

[32] – Murray, D. “Tuning Neural Networks with Genetic Algorithms”, AI Expert, Vol. 9, pp. 21-27, 1994.

[33] – Pham, D.T. e Karaboga, D. “Intelligent Optimisation Techniques”, Springer-Verlag, 238p, 2000.

[34] – Goldberg, D.E. “Genetic Algorithms in Search, Optimization and Machine Learning”. Addison-Wesley Longman Publishing Co., 1989.

[35] – Palmes, P.P, Hayasa, T. e Usiu, S. “Mutation-Based Genetic Neural Networks” IEEE Transactions on Neural Networks, Vol. 16, n.3, pp. 587-600, 2005.

[36] – Baker, J. E. “Adaptive Selection Methods for Genetic Algorithms”, Proceedings of the First International Conference on Genetic Algorithms and their Applications, Lawrence Erlbaum Associates, Hillsdale, NJ, pp.101-111.

[37] – Metropolis, N., Rosenbluth, A.W., Rosenbluth, M.N., Teller, A.H., e Teller, E. “Equation of State Calculations by Fast computing Machines”, The Journal of Chemical Physics, Vol. 21, n.6, pp. 1087-1092.

[38] – Boese, K.D e Kahng, A.B., “Simulated Annealing of Neural Networks: the “Cooling” Strategy Reconsidered”, Proceedings IEEE International Symposium on Circuits and Systems, pp. 2572-2575, 1993.

[39] – Glover, F. “Future Paths for Integer Programming and Links to Artificial Intelligence”, Computation Operations Research, Vol. 13, pp. 533-549.

[40] – Hansen, P. “The Steepest Ascent Mildest Descent Heuristic for Combinatorial Programming”, Presented at the congress on Numerical Methods in Combinatorial Optimization, Italy, 1986.

[41] – Osman, I.H. “Metastrategy Simulated Annealing and Tabu Search algorithms for the Vehicle Routing Problem”, Ann. Operational Research, Vol. 41, n. 1-4, pp. 421-451, 1993.

[42] – Li, W.D., Ong, S.K., e Nee, A.Y.C., “Hybrid Genetic Algorithms and Simulated Annealing Approach for the Optimization of Process Plans for Prismatic Parts”, International Journal of Production Research, Vol. 40, n.8, pp. 1899-1922, 2002.

[43] – Du, X., Li, Y., Chen, W., Zhang, Y e Yao, D. “A Markov Random Field Based Hybrid Algorithm with Simulated Annealing and Genetic Algorithm for Image Segmentation”, Advances in Natural Computation, Springer Berlin, Heidelberg, Vol. 4221, pp. 706-715.

[44] – Lin, Y.C, Su, K.L e Chang, W.C. “Mixed – Integer Evolutionary Optimization of Artificial Neural Networks”, *IEEE Transactions on Neural Networks*, 4th International Conference on Innovative Computing, Information and Control, pp. 532 - 535, 2009.

[45] – Yao, X. e Liu, Y. "Evolutionary Design of Artificial Neural Networks with Different Nodes”, Proceedings of the International Conference on Evolutionary Computation, pp.670-675, 1996.

[46] – Ferentinos, K.P. “Biological Engineering Applications of Feedforward Neural Networks Designed and parameterized by Genetic Algorithms”, Neural Networks, Vol. 18, n.7, pp. 934-950, 2005.

[47] – Rumelhart, D. E., Weigend, S.A., "Predicting the Future: A Connectionist Approach", International Journal of Neural Systems, pp. 193-209.

[48] – <Mathworks. Matlab. Mathworks, Inc., 2007. http://www.mathworks.com>. Acessado em: 07/07/2010.

[49] - Gomes, G.S.S. "Novas Funções de Ativação em Redes Neurais Artificiais *Multilayer Perceptron*", Tese de Doutorado, Centro de Informática, Universidade Federal de Pernambuco, Recife, 2010, 159p.

[50] – Baker, J.E. "Reducing Bias and Inefficiency in the Selection Algorithm", Proceedings of the Second International Conference on Genetic Algorithms and their Appliaction, Lawrence Erlbaum Associates, pp.14-21.

[51] – Sywerda, G. "Uniform Crossover in Genetic Algorithms", Proceedings of the Third International Conference on Genetic Algorithms, Morgan Kaufmann Publishers Inc, pp. 2 – 9.

[52] – Sexton, R.J., Dorsey, R.E., e Johnson, J.D. "Optimization of Neural Networks: A Comparative Analysis of the Genetic Algorithm and Simulated Annealing", European Journal of Operational Research, Vol. 114, pp. 589-601.

[53] – Downsland, K.A. "Simulated Annealing", In Reeves, C.R. (ed), Modern Heuristic Techniques for Combinatorial Problems, Blackwell Scientific Publications, pp.20-69.

[54] – Prehelt, L. "PROBEN1 – A Set of Neural Network Benchmark Problems and Benchmarking Rules", Technical report 21/94, Universität Karlsruhe, 38p.

[55] – Santos, M.S. "Construção de um Nariz Artificial usando Redes Neurais", Tese de Doutorado, Curso de Pós-Graduação em Ciência da Computação, Centro de Informática, Universidade Federal de Pernambuco, Recife-PE, 120p.

[56] – Souza, J.E.G, Neto, B.B., Santos, F.L., Melo, C.P., Santos, M.S., e Ludermir, T.B. "Polypyrrole Based Aroma Sensor", Synthetic Metals, Vol. 102, pp. 1296 – 1299, 1999.

[57] – <UCI Repository of Machine Learning Databases. http://www.ics.uci.edu/~mlearn/MLRepository.html >. Acessado em: 07/07/2010.

[58] – Fisher, R.A. "The Use of Multiple Measurements in Taxonomic Problems", Annals of Eugenics, n.7, pp. 179-188.

[59] – Quinlan, J. "Simplifying Decision Trees", International Journal of Man-Machine Studies, Vol. 27, pp. 221-234.

[60] – Forina, M. et al. "PARVUS – An Extensible Package for Data Exploration, Classification e Correlation", Institute of Pharmaceutical and Food Analysis and Technologies, Via Brigata Salermo, 16147 Genova, Italy.

[61] – Mackey, M.C. e Glass, L. "Oscillation and Chaos in Physiological Control Systems", Science, Vol.197, pp.287-289.

[62] – Ghiassi, M., Saidane, H., e Zimbra, D.K. “A Dynamic Artificial Neural Network Model for Forecasting Time Series Events”, International Journal of Forecasting, Vol. 21, pp. 341-462, 2005.