UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

RICARDO BASTOS CAVALCANTE PRUDÊNCIO

"Projeto Híbrido de Redes Neurais"

ESTE TRABALHO FOI APRESENTADO À PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO DO CENTRO DE INFORMÁTICA DA UNIVERSIDADE FEDERAL DE PERNAMBUCO COMO REQUISITO PARCIAL PARA OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIA DA COMPUTAÇÃO.

ORIENTADORA: Prof.ª Teresa Bernarda Ludermir

RECIFE, FEVEREIRO/2002

Pós-Graduação em Ciência da Computação

“Projeto Híbrido de Redes Neurais”

Por

Ricardo Bastos Cavalcante Prudêncio

Dissertação de Mestrado

Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
www.cin.ufpe.br/~posgraduacao

RECIFE, FEVEREIRO/2002

Agradecimentos

Escrever agradecimentos serve para relembrar o quanto se é ajudado durante a vida, e concluir que não se consegue nada sozinho. De fato, durante a minha vida, estive sempre cercado de pessoas que me ajudaram bastante.

Primeiramente agradeço a Deus por ter colocado em minha vida pessoas tão especiais. Dentre essas pessoas estão Dona Ana (minha mãe maravilhosa), meu pai, minhas irmãs e meu irmão, enfim, minha família que me deu apoio em todos os momentos. Sem eles eu seria um perdido. Além de me darem muito amor, foram os que mais me incentivaram ao estudo.

Á minha namorada e mulher Flávia que conheci durante o mestrado e que me ajudou de várias formas. Ela me confortou nas horas em que o trabalho não ia bem, agüentou o meu nervosismo inúmeras vezes, me ajudou a combater meus medos injustificados, além de me ajudar diretamente no trabalho com o seu conhecimento. Obrigado Flávia, ter te encontrado foi uma das melhores coisas que aconteceu em minha vida. Você é uma companheira maravilhosa. Te amo muito.

À família de Flávia, que se tornou também minha família, Dona Isa, Sylvia, Jucá, Dona Néia e Dona Denise, somente pra citar os mais próximos. Obrigado pelo carinho que vocês me deram até hoje.

À minha orientadora, professora Teresa, que me acolheu como seu aluno e a quem devo muito de meu trabalho e de minhas recentes conquistas.

Aos meus amigos Ivan, Ryan e Paulyne que vieram de Fortaleza juntamente comigo para fazer mestrado em Recife.

Aos meus amigos do futebol no INOCOP, em ordem alfabética, Admilson, Alessandro, Alexandre, Arakaki, Byron, Dave, Franklin, Fred, Hendrik, Javé, João, Kelvin, Obionor, Thiago, Trinta e Uirá, pelo Futebol no Iconop e por reconhecerem que sou o melhor jogador dentre todos! Além deles, outros amigos que não jogam futebol, mas que não são menos importantes para mim, Akio, Ana Carina (que nos ajudou muito no nosso

começo em Recife), Ana Cristina, Arthur, Ciro, Elisângela, Gustavo, Joaquim, Renato e Rodrigo.

Agradeço também em especial a todos que durante o mestrado me ajudaram com o seu conhecimento, Akio, Eleonora, Estephane, Flávia, Geber, Marcílio, Meuser e Teresa.

Aos professores André Ponce e Marcílio que fizeram parte da minha banca e deram, cada um, excelentes contribuições ao meu trabalho.

Por fim, um agradecimento simbólico aos pesquisadores que contribuíram com os seus trabalhos para a minha dissertação. Isaac Newton dizia que enxergara mais longe porque havia se apoiado no ombro de gigantes.

Recife, fevereiro de 2002.

Resumo

As Redes Neurais Artificiais (RNAs) têm sido aplicadas com sucesso em uma diversidade de problemas do mundo real. Contudo, o sucesso dessas redes para um determinado problema depende muito de um projeto bem realizado. O projeto de redes neurais envolve a definição de vários parâmetros, como, por exemplo, o tipo de rede, a arquitetura, o algoritmo de treinamento utilizado, os parâmetros de treinamento, os critérios de parada, dentre outros. A automatização (total ou parcial) do projeto de RNAs tem como objetivos principais tornar o desempenho das redes menos sensível a decisões erradas de um desenvolvedor inexperiente, além de torná-las acessíveis a usuários não-especialistas em redes neurais. Como solução para o problema da automatização, investigamos o uso de técnicas de Inteligência Artificial que, quando integradas com as redes neurais, resultam em Sistemas Neurais Híbridos (SNHs). Nessa dissertação, apresentamos duas aplicações desses Sistemas Híbridos para a previsão de séries temporais, um problema de relevância fundamental em muitos domínios do mundo real. Primeiramente, propomos um modelo de automatização integrando o Raciocínio Baseado em Casos (RBC) e os Algoritmos Genéticos (AGs). No nosso modelo, o sistema de RBC mantém uma base de casos em que cada caso armazena a descrição de um problema resolvido com redes neurais e a solução aplicada. Diante de um novo problema, uma consulta é feita à base de casos, recuperando as soluções usadas nos problemas mais similares. Essas soluções são inseridas na população inicial dos AGs, que são responsáveis por adaptá-las. Após a execução dos AGs, a solução final poderá ser inserida na base de casos, para auxiliar a solução de problemas futuros. Como estudo de caso, aplicamos o modelo proposto para a otimização da arquitetura de modelos neurais de previsão. As redes geradas pelo modelo apresentaram maior poder de generalização, além de um número menor de conexões de rede. Na segunda aplicação de SNHs, investigamos o uso dos Algoritmos Genéticos durante o aprendizado dos pesos de uma rede neural usada para a previsão de vazões em uma bacia hidrográfica. Nessa aplicação, os AGs foram usados para definir os pesos iniciais da rede para o algoritmo de Levenberg-Marquardt, formando assim um algoritmo de treinamento híbrido. O uso dos AGs aumentou o desempenho do aprendizado, principalmente em relação ao tempo de treinamento. Nessa dissertação, apresentamos as vantagens e limitações dos dois SNHs desenvolvidos, além de indicações de trabalhos futuros.

Abstract

Artificial Neural Networks (ANNs) have been successfully used in a diversity of real-world problems. However, the performance of these networks to solve a particular problem depends on their design. The design of neural networks requires the definition of several parameters, such as the network's model, its architecture, the training algorithm, among others. The (total or partial) automatic design of ANNs aims at turning them less sensitive to wrong decisions of an unexperienced developer, as well as to make them available to non-expert users. As a solution to the automatic design, we investigated the use of Artificial Intelligence techniques which, when combined with ANNs, produce Hybrid Neural System (HNS). In this dissertation, we present two applications of these HNSs to time series prediction problems, which is an important learning problem in several real-world domains. First, we propose a model for automatic design that integrates Case-Based Reasoning (CBR) and Genetic Algorithms (GAs). In our model, the CBR system maintains a case base in which each case stores a description of a problem solved by a neural network together with the applied solution. Given a new problem, a query retrieves from the case base the solutions used in the most similar problems. These solutions are inserted in the initial population of the GAs, which are responsible for their adaptation. Next, the GAs are run and the final solution may be inserted in the case base and used in future problems. As a case study, we applied the proposed model to optimize the architecture of neural prediction models. The networks generated by the models presented a good generalization performance and a low number of network connections. In the second application, we investigated the use of GAs in the weights' training process of a neural network used to river flow prediction. In this application, the GAs were used to define the initial weights to the Levenberg-Marquardt algorithm, composing, this way, a hybrid learning algorithm. The use of GAs improves the learning process mainly in terms of time efficiency. In this dissertation, we present the advantages and limitations of the two developed HNSs, indicating some of the future work.

Índice

Lista de Figuras

Lista de Tabelas

Lista de Abreviações

AGs: Algoritmos Genéticos.

AR: AutoRegressive.

ARMA: AutoRegressive-Moving Average.

BP: BackPropagation.

IA: Inteligência Artificial.

KBANN: Knowledge Based Artificial Neural Networks.

LM: Levenberg-Marquardt.

MA: Moving Average.

MLP: Multi-Layer Perceptron.

MSE: Mean Squared Error.

NARX: Non-Linear AutoRegressive with eXogenous variables.

NARMAX: Non-Linear AutoRegressive-Moving Average with eXogenous variables.

RBC: Raciocínio Baseado em Casos.

RBF: Radial Basis Function.

RNAs: Redes Neurais Artificiais.

SIHs: Sistemas Inteligentes Híbridos.

SNHs: Sistemas Neurais Híbridos.

1 Introdução

Os modelos de Redes Neurais Artificiais (RNAs) têm sido aplicados com sucesso em uma grande diversidade de problemas do mundo real [Widrow et al. 1994][Asakawa, Takagi 1994], dando suporte a diversas tarefas da mineração de dados, como, por exemplo, classificação, previsão, agrupamento, dentre outras [Fayyad et al. 1996], [Almendra et al. 1999]. As RNAs têm um forte poder computacional, são tolerantes a falhas, e são capazes de aprender, generalizar e trabalhar com dados incompletos e com ruído. Segundo [Towell, Shavlik 1994], as RNAs têm se mostrado iguais ou superiores a outras técnicas de aprendizado em uma diversidade de domínios, quando avaliados em termos de sua capacidade de generalização [Shavlik et al. 1991].

Contudo, o sucesso das RNAs para a solução de um determinado problema depende de um projeto bem realizado. Dentre os parâmetros importantes do projeto de redes neurais, podemos citar: o tipo de modelo e a arquitetura de rede, o algoritmo de aprendizado dos pesos, e as taxas de aprendizado. Se o desenvolvedor da RNA tiver pouco conhecimento sobre o problema abordado ou sobre técnicas de construção de redes neurais, ele poderá definir de forma inadequada os parâmetros do projeto, prejudicando de maneira significativa o desempenho da rede.

Nesse trabalho, abordamos o problema da automatização (total ou parcial) do projeto das Redes Neurais Artificiais. O objetivo principal é tornar o projeto das RNAs menos sensível às decisões arbitrárias de um desenvolvedor inexperiente. O projeto otimizado e automático possibilita a utilização das RNAs por usuários finais de uma forma mais direta, sem o intermédio de um especialista na área. Assim, uma solução eficiente para esse problema facilitaria o uso extensivo de redes neurais em problemas do mundo real [Bigus 1996].

Como solução para o problema da automatização, investigamos o uso de técnicas de Inteligência Artificial, como, por exemplo, o Raciocínio Baseado em Casos [Kolodner 1993] e os Algoritmos Genéticos [Holland 1975]. As diferentes aplicações dessas técnicas ao projeto de redes neurais formam *Sistemas Inteligentes Híbridos (SIHs)*. Assim, além de

abordar o problema do projeto automático de redes neurais, estamos interessados no estudo de SIHs, que tem se mostrado um campo de pesquisa promissor [Khebbal, Goonatilake, 1995].

O resto dessa introdução se desenvolve como se segue. Na seção 1.1, apresentamos com mais detalhes o projeto de redes neurais, seguido pela seção 1.2, onde apresentamos uma breve discussão sobre os Sistemas Inteligentes Híbridos. Na seção 1.3, descrevemos o contexto da dissertação, assim como os trabalhos realizados, e concluindo na seção 1.4, descrevemos o conteúdo da dissertação.

1.1 Projeto de Redes Neurais

O projeto de redes neurais pode ser dividido em quatro etapas: a preparação dos dados, a escolha do modelo e da arquitetura de rede, o treinamento dos pesos, e a avaliação da rede treinada. Essas etapas serão brevemente discutidas a seguir.

a) Preparação dos Dados

A etapa de preparação de dados envolve a seleção dos atributos importantes para o problema, a limpeza dos dados e as transformações sobre os dados. O processo de seleção de atributos deve idealmente definir um conjunto mínimo de entradas necessárias para resolver o problema abordado. Fornecer um grande número de entradas a uma rede neural, pode prejudicar o seu desempenho. Esse problema é conhecido na literatura como a *maldição da dimensionalidade* [Bishop 1995].

Em geral, os modelos de redes neurais têm algumas restrições a respeito aos dados de entradas. Uma delas é que os dados das redes devem ser numéricos, assim, quando um problema envolve variáveis categóricas ou texto, transformações devem ser feitas sobre esses dados para torná-los numéricas. Além disso, os modelos de redes neurais podem não funcionar bem com dados numéricos com valor muito alto ou muito baixo. Isso se deve as características das funções de ativação que, em geral, trabalham com valores pequenos. Assim, transformações como a normalização e a padronização dos dados podem melhorar o desempenho da rede.

b) Escolha do Modelo e da Arquitetura

A escolha do modelo de rede depende primeiramente do tipo de problema abordado e das necessidades da aplicação. Cada modelo de rede neural pode ser usado para um determinado grupo de tarefas. Por exemplo, a rede *Multilayer Perceptron* (MLP) [Rumelhart et al. 1986] e a rede *Radial Basis Function* (RBF) [Broomhead, Lowe 1988] são comumente associadas a problemas de reconhecimento de padrões. A rede *Self-Organising Maps (SOM)* [Kohonen 1989], por sua vez, é associada geralmente a problemas de agrupamento. Assim, o tipo de problema abordado indica quais tipos de rede podem ser usadas. Além disso, as necessidades de cada aplicação podem influir na escolha do modelo. Por exemplo, quando o problema tem restrições de tempo, o uso de uma rede RBF, que tem um treinamento mais rápido [Braga et al. 2000], poderá ser mais indicado que o uso de uma MLP.

Entre os parâmetros da arquitetura da rede, temos: número de camadas, número de nodos por camada, tipo de função de ativação dos nodos e os padrões de conectividade. Um dos pontos que pode ser crucial para o desempenho da rede é um bom dimensionamento do número de parâmetros da topologia (número de conexões). Embora um número maior possa dar ao modelo um maior poder computacional, isso pode torná-lo mais sensível aos problemas de *overffiting*, aumentar o número de mínimos locais no espaço de pesos, além de aumentar o tempo de treinamento. Por outro lado, um número muito pequeno de pesos pode não ser o suficiente para modelar adequadamente o problema abordado. As estratégias mais comuns para a definição da topologia, as técnicas construtivas e as de *prunning (ou poda)* [Haykin 1994], são sensíveis a cair em mínimos locais [Russell, Norvig 1995]. Desta forma, o sucesso dessas estratégias depende muito da topologia inicial.

c) Treinamento dos Pesos

Essa é a etapa de definição dos pesos da rede, ou seja, do aprendizado da rede. Um dos pontos mais importantes aqui é a escolha do algoritmo de treinamento. Os algoritmos de treinamento mais usados são os baseados na técnica de gradiente descendente [Battiti 1992]. Entre eles, podemos citar os algoritmos de Backpropagation [Rumelhart et al. 1986] e o de Gradientes Conjugados [Barnard, Cole 1989]. A escolha do algoritmo de

treinamento depende das necessidades do problema. Se o problema tiver restrições de tempo, o mais indicado é o uso de algoritmos de segunda-ordem, os que usam as derivadas segundas do erro, e são, em geral, mais rápidos que o Backprogation [Battiti 1992]. Uma outra abordagem é o uso de algoritmos de otimização estocásticos, como os Algoritmos Genéticos [Montana, Davis 1989], que serão vistos no capítulo 3. Apesar de que, as características de cada algoritmo possam indicar ou impossibilitar o seu uso para determinados problemas, não existem regras definitivas que decidam que algoritmo é mais adequado para cada tipo de problema.

Outro ponto de destaque é que, geralmente, cada algoritmo de aprendizado tem um ou mais parâmetros de treinamento, que devem ser definidos durante a construção da rede. Esses parâmetros podem influir significativamente na eficiência e a convergência do processo de treinamento da rede. Os valores desses parâmetros, em geral, dependem de vários fatores, como o tamanho da rede e as características do conjunto de treinamento. Dentre outros pontos importantes dessa etapa, destacamos a inicialização dos pesos, o número de reinícios do treinamento e os critérios de parada.

d) Avaliação

A avaliação de uma rede neural treinada depende principalmente do tipo de aplicação abordada. A forma mais comum de avaliar redes neurais é através do erro quadrático sobre um conjunto de dados. Entretanto, outros pontos podem ser levados em consideração, dependendo do problema. Em problemas de previsão, por exemplo, não apenas o valor do erro, mas o comportamento do erro no decorrer do tempo é importante. Já em problemas de classificação, o erro de classificação pode ser mais útil. O número de conexões da rede é também relevante, uma vez que influencia na capacidade de generalização da rede e no custo de sua implementação em hardware.

1.2 Sistemas Inteligentes Híbridos

A Inteligência Artificial (IA) é o campo da ciência da computação que tenta reproduzir em máquinas a inteligência humana [Turban 1992]. A década de 50 marcou

tanto o início como o período de maior otimismo na IA. Nesse período, foram definidos os primeiros modelos de redes neurais artificiais [McCulloch, Pitts 1943], programas para jogar xadrez [Shanon 1950], além de resultados animadores com os provadores automáticos de teorema [Newell, Simon 1956]. A expectativa da época era de que, em algumas décadas, os computadores seriam capazes de realizar a mesma quantidade de tarefas que os seres humanos e possivelmente de uma maneira mais eficiente [Russell, Norvig 1995]. Entretanto, a pesquisa em IA mostrou que a tarefa de reproduzir a inteligência era muito mais complexa do que se achava antes. De fato, a inteligência humana tem vários aspectos associados, como o raciocínio, o aprendizado, o senso comum, a capacidade de comunicação, dentre outros. Cada um desses aspectos revela problemas difíceis de resolver.

A fase seguinte da IA foi uma fase de reducionismo, onde cada aspecto associado à inteligência foi tratado separadamente com diversas técnicas. Para o aprendizado de máquina, por exemplo, foram desenvolvidas técnicas como as árvores de decisão [Quinlan 1986], os diversos modelos de redes neurais [Haykin 1994], as redes Bayesianas [Heckerman 1995], dentre outras [Mitchel 1997]. Além do aprendizado de máquina, surgiram também outras áreas de pesquisa distintas dentro da IA, como, por exemplo, o Processamento de Linguagem Natural [Allen 1995] e o estudo dos Sistemas Especialistas [Turban 1992].

As diferentes técnicas de IA foram estudadas e aplicadas a vários problemas a ponto de se ter uma boa noção sobre as vantagens e desvantagens de cada técnica. Atualmente, podemos dizer que estamos em uma terceira fase da IA, em que o objetivo maior é a integração de diferentes técnicas para formar sistemas mais robustos, os chamados *Sistemas Inteligentes Híbridos* (SIHs) ([Khebbal, Goonatilake, 1995],[Khosla, Dillon 1997]). Segundo [Silva 1999], hoje em dia, é mais comum investir no desenvolvimento de idéias maduras do que propor idéias inteiramente novas.

Os principais objetivos do uso de sistemas SIHs são: (1) a resolução de problemas complexos que só poderiam ser resolvidos se tratados com sub-tarefas; e (2) unir as vantagens e superar as limitações individuais de cada técnica. Desta forma, a proposta dos SIHs é a construção de sistemas mais robustos, capazes de trabalhar com vários tipos de representação de conhecimento, dar suporte a vários tipos de inferências, resolver

problemas mais complexos, dentre outras vantagens. Entretanto, temos que destacar que o uso de SIHs podem às vezes gerar mais dificuldades que vantagens. Primeiramente, a implementação e manutenção desses sistemas podem ser mais difíceis, uma vez que sistemas híbridos são intrinsecamente mais complexos que os seus componentes [Gray, Kilgour 1997]. Além disso, como destaca [Braga et al. 2000], os SIHs podem unir, além das vantagens de cada técnica, também os pontos fracos.

1.2.1 Classificação dos Sistemas Inteligentes Híbridos

Em [Khebbal, Goonatilake, 1995], foi proposto um esquema de classificação para sistemas híbridos composta das seguintes classes.

a) Tipo I: *Function Replacing*

Nessa categoria, uma técnica é usada para implementar uma função da outra técnica. Essa forma de hibridismo não acrescenta nenhuma funcionalidade ao sistema inteligente, apenas tenta superar alguma limitação da técnica principal ou otimizar sua execução. Como exemplo, podemos citar o uso de árvores de decisão para indexar a base de casos de um sistema de RBC [Cardie 1993].

b) Tipo II: *Intercommunicating Hybrids*

Essa é a categoria de sistemas híbridos usados para resolver problemas complexos que possam ser divididos em várias subtarefas independentes. Assim, o sistema híbrido é formado por módulos independentes, onde cada um usa uma técnica inteligente para resolver uma das subtarefas do problema principal. Como exemplo, podemos citar o sistema *EITHER* [Mooney, Ourston 1994]**.** Esse sistema implementa módulos independentes de raciocínios indutivo, dedutivo e abdutivo para revisar cláusulas de *Horn*. Cada tipo de raciocínio foi implementado com uma técnica específica.

c) Tipo III: *Polymorphic Hybrids*

Nessa categoria, uma única técnica é adaptada para realizar uma tarefa inerente a outra técnica. A motivação dessa categoria é descobrir novas funcionalidades de uma

técnica e entender como diferentes técnicas podem se relacionar. Como exemplo, podemos citar o uso de redes neurais para raciocínio simbólico [Honavar, Uhr 1995] e [Ajjanagadde and Shastri 1995].

1.2.2 Sistemas Neurais Híbridos

Sistemas Neurais Híbridos são sistemas que combinam dois ou mais subsistemas sendo que um deles é uma rede neural [Braga et al. 2000]. Esses sistemas têm sido estudados em diversos trabalhos como, por exemplo, em [Honavar, Uhr 1994], [Wermter, Sun, 2000] e [Braga et al. 2000].

A classificação dos sistemas neurais híbridos pode seguir a mesma classificação geral descrita na seção anterior. Nos SNHs do tipo I, as redes neurais podem ser hibridizadas de duas formas: (1) uma rede neural é usada para implementar uma tarefa de outra técnica como, por exemplo, na indexação em sistemas de RBC [Malek 1995]; ou (2), uma técnica diferente é usada para definir parâmetros ou mesmo etapas do projeto das redes neurais. Esse tipo de sistema neural híbrido é o foco de nosso estudo, desta forma, vários exemplos desse tipo de sistema híbrido serão vistos ao longo dessa dissertação.

Nos SNHs do tipo II, as redes neurais são usadas como módulos independentes para resolver um subtarefa de uma problema mais complexo. Exemplos desse tipo de SNH podem ser vistos em [Klimasauras 1995], [Scherer, Schlageter 1995]. Nos SNHs do tipo III, as redes neurais são usadas para simular funcionalmente uma outra técnica. Como já citamos, um exemplo desse tipo de sistema híbrido é o uso das redes neurais para efetuar raciocínio simbólico.

1.3 Contexto da Dissertação

Como vimos anteriormente, o projeto das RNAs envolve um grande número de decisões que, se mal tomadas, podem levar a resultados muito inferiores aos que realmente poderiam ser obtidos. Nesse contexto, a automatização do projeto das redes é importante para tornar o desempenho das redes menos dependente das ações do desenvolvedor. Além

disso, o projeto automático facilita o uso de redes neurais em aplicações *on-line* [Bigus 1996].

A automatização do projeto das redes neurais pode ser realizada basicamente através de duas abordagens:

(1) com o uso de conhecimento: nesse caso o conhecimento sobre o domínio de aplicação e sobre as redes neurais pode ser codificado em regras ou aprendido através de técnicas de aprendizado de máquina. Esse conhecimento pode ser usado no projeto de redes neurais de forma estática, ou seja, antes do processo de treinamento, ou de forma dinâmica, inserido em heurísticas de busca aplicadas durante o treinamento das redes.

(2) com o uso de algoritmos de busca cega: quando não há conhecimento disponível sobre o problema abordado, o projeto de redes neurais se resume a um problema de busca cega. Ao contrário da abordagem anterior, essa abordagem pesquisa o espaço de parâmetros usando apenas o conhecimento sobre o desempenho da rede em cada ponto de busca.

Um ponto a destacar aqui é que a automatização do projeto de redes neurais pode ser realizada através do uso de técnicas de Inteligência Artificial. De fato, as duas abordagens para o projeto das redes, o uso de conhecimento e a busca, são temas de estudo da IA. Assim, é natural que as técnicas desenvolvidas dentro da IA sejam aplicadas ao projeto de redes neurais. Como citado anteriormente, essa integração de técnicas gera SNHs do tipo I.

Nessa dissertação, investigamos o uso de duas conhecidas técnicas de IA para automatizar o projeto de RNAs: a metodologia de Raciocínio Baseado em Casos (RBC) e dos Algoritmos Genéticos (AGs). No RBC, um novo problema é resolvido a partir de soluções bem sucedidas usadas em problemas similares do passado. No nosso contexto, cada caso armazena a descrição de um problema resolvido com redes neurais e uma solução bem sucedida (os parâmetros do projeto). Diante de um novo problema, uma consulta é feita à base de casos, recuperando as soluções usadas nos problemas mais similares. Essas soluções são inseridas na população inicial dos AGs, que são responsáveis por adaptá-las ao novo problema. Após a execução dos AGs, a solução final é retornada ao usuário e poderá ser inserida na base de casos, para auxiliar a resolução de novos problemas.

Este trabalho trata o projeto de redes neurais de forma híbrida, partindo da observação de que este problema pode ser visto como um processo de busca auxiliado pelo uso de conhecimento. Nesta proposta, escolhemos o Raciocínio Baseado em Casos para implementar o sistema baseado em conhecimento por ser mais adequado para tratar domínios complexos (em que o conhecimento é pouco formalizado) do que os sistemas baseados em regras [Kolodner 1993]. Esse é o caso do projeto de redes neurais. No que concerne a busca, a vantagem dos AGs é que eles são algoritmos de busca e otimização mais robustos que os algoritmos tradicionais [Lacerda, Carvalho 1999], por serem menos sensíveis a cair em mínimos locais e por serem capazes de otimizar uma classe maior de funções. O modelo proposto foi usado para a definição da arquitetura de redes neurais para problemas de previsão de séries temporais.

Apresentaremos também o uso dos Algoritmos Genéticos durante o aprendizado dos pesos de uma rede neural para um problema de previsão de vazões em bacias hidrográficas. Nessa aplicação, os AGs foram usados para definir os pesos iniciais da rede para o algoritmo de Levenberg-Marquardt ([Levenberg 1944] [Marquardt 1963]). Essa aplicação surgiu durante o curso de mestrado como uma conseqüência do primeiro trabalho. Enquanto o modelo proposto foi aplicado para a definição da arquitetura, no segundo trabalho, o foco foi o aprendizado dos pesos. Em trabalhos futuros, os dois sistemas serão integrados para implementar um sistema mais eficiente.

As duas contribuições da nossa dissertação se restringiram ao problema de previsões de séries temporais, que é um problema de relevância fundamental em diversas aplicações práticas. O uso de modelos de previsão diminui os riscos na tomada de decisões, uma vez que podemos antecipar os possíveis efeitos de uma ação a ser tomada [Montgomery et al. 1990]. De fato, os modelos de previsão de séries temporais têm sido usados em diversos domínios do mundo real como, por exemplo, na área financeira [Oliveira 2000] e de recursos hídricos [Valença 1999a].

1.4 Conteúdo da Dissertação

Além dessa introdução, a dissertação está organizada em mais seis capítulos como se segue:

- **Capítulo 2 - Modelos de Previsão de Séries Temporais:** Nesse capítulo, apresentaremos o problema de previsão de séries temporais e alguns dos modelos de previsão mais conhecidos, como os modelos de Box-Jenkins e os modelos de RNAs. A previsão de séries temporais foi o problema abordado nos dois capítulos de contribuições.

- **Capítulo 3 - Algoritmos Genéticos para o Projeto de Redes Neurais:** Nesse capítulo, apresentamos diferentes formas de integrar os Algoritmos Genéticos ao projeto de redes neurais. Primeiramente, fazemos uma breve apresentação sobre os AGs, seguido da sua aplicação como algoritmo de aprendizado dos pesos e o uso de AGs na definição da topologia de redes neurais. A motivação desse capítulo é a visão do projeto de RNAs como um problema de busca.

- **Capítulo 4 - Estratégias Baseadas em Conhecimento para o Projeto de Redes Neurais:** Nesse capítulo, apresentamos algumas formas de usar conhecimento no projeto de RNAs. Apresentamos as redes neurais baseadas em conhecimento, que usam conhecimento do domínio de aplicação para definir a estrutura e os pesos iniciais da rede, e algumas técnicas que manipulam conhecimento geral sobre redes neurais, com destaque para o Raciocínio Baseado em Casos. A motivação desse capítulo é a visão do projeto de redes neurais como um domínio de conhecimento.

- **Capítulo 5 - RBC e AG para Otimização da Arquitetura:** Nesse capítulo, apresentamos a definição do modelo proposto que integra o Raciocínio Baseado em Casos e os Algoritmos Genéticos para o projeto de RNAs, assim como a sua aplicação para a definição da arquitetura de modelos neurais de previsão.

- **Capítulo 6 - Treinamento Híbrido de Redes Neurais:** Nesse capítulo, apresentamos o uso de um algoritmo híbrido no treinamento de uma rede neural para um

problema de previsão de vazões. Esse algoritmo integrou os Algoritmos Genéticos e o algoritmo de Levenberg-Marquardt.

- **Capítulo 7 - Conclusões:** Nesse capítulo, concluímos a dissertação, fazendo um breve resumo dos resultados obtidos, apontando algumas limitações, além das possibilidades de trabalhos futuros.

2 Modelos de Previsão de Séries Temporais

Diversos fenômenos do mundo real variam no decorrer do tempo. Alguns fenômenos podem ser registrados numericamente para os mais diversos fins. Podemos registrar, por exemplo, o consumo mensal de energia elétrica de uma casa durante um ano para identificar algum padrão de consumo, ou registrar continuamente a temperatura de uma reação química durante um intervalo de tempo para se obter conclusões sobre o processo. Os registros resultantes da observação de um fenômeno ou um processo que varia no tempo são denominados de séries temporais. Em outras palavras, uma série temporal é um conjunto de observações de um fenômeno ordenadas no tempo [Box, Jenkins 1970]. Uma série temporal pode ser contínua, quando é medida sem interrupções no tempo, ou discreta quando é medida em intervalos sucessivos no tempo, em geral, em intervalos regulares.

A análise de uma série temporal é o processo de identificação das características e propriedades importantes da série utilizadas para descrever, em termos gerais, o fenômeno gerador da série. A análise de séries temporais é um processo de indução, uma vez que, a partir de um conjunto de observações, infere características gerais do fenômeno observado. Em [Morettin, Toloi 1987], são mencionados diversos objetivos da análise de séries temporais, dentre os quais destacamos:

- Sumário dos dados: resumo das características da série, como a existência de tendências, sazonalidade e variações, a construção de gráficos explicativos, dentre outras tarefas.
- Estabelecimento de causalidade: consiste em identificar relações de causa e efeito entre séries, como por exemplo, a relação entre séries de venda e séries de propaganda de um produto.
- Classificação: consiste em mapear uma série temporal em uma classe dentro de um conjunto de classes, como por exemplo, a classificação de um eletrocardiograma, dentre as classes, *normal* e *anormal*.

- Previsão: consiste em estimar os valores futuros da série com base nas informações disponíveis no presente.

Segundo [Dorffner 1996], a maior aplicação da análise de séries temporais é a geração de modelos de previsão. Os modelos de previsão de séries temporais são capazes de definir, com um certo grau de confiabilidade, os valores futuros de uma série a partir de informações passadas da própria série e de outras variáveis significantes no problema. O uso de modelos de previsão é fundamental para diminuir os riscos na tomada de decisões, uma vez que a eficácia de uma decisão depende obviamente dos eventos que se seguem à decisão [Montgomery et al. 1990]. Dentre as muitas aplicações de previsão de séries temporais, podemos citar: planejamento de produção, aplicações financeiras, controle de processos e aplicações médicas.

Segundo [Morettin, Toloi 1987], existem dois enfoques para a modelagem e previsão de séries temporais. O primeiro é o uso de uma teoria sobre o domínio de aplicação (econômica, hidrológica, física, etc...) para a construção de um modelo conceitual para o fenômeno gerador da série. Como exemplo, podemos citar o modelo SMAP (*Soil Moisture Accounting Procedure*) para a previsão de vazões em bacias hidrográficas, desenvolvido por [Lopes et al. 1981] e descrito em [Valença 1999a]. Esse modelo procura representar matematicamente o comportamento de uma bacia hidrográfica. A previsão da vazão é feita através de equações que envolvem variáveis do domínio como precipitação, escoamento superficial e evapotranspiração. Os modelos teóricos, como o SMAP, têm a desvantagem de necessitar de um conhecimento aprofundado do domínio para a sua construção, além de ter aplicação limitada ao domínio abordado.

O outro enfoque consiste em construir um modelo de previsão a partir dos dados disponíveis, sem recorrer a uma teoria específica. Nessa abordagem, uma série é modelada através de uma função com parâmetros livres ajustados a partir dos dados da série. Essa abordagem é chamada em [Sjoberg et al. 1994] de modelos caixa-preta, uma vez que os parâmetros dos modelos gerados não têm uma interpretação direta dentro do domínio do problema.

Nas próximas seções, apresentamos duas classes de modelos caixa-preta bastante difundidos na literatura. Na seção 2.2, são discutimos os modelos de Box e Jenkins, que

modelam as séries através de funções lineares, e na seção 2.3, são apresentadas as Redes Neurais Artificiais, que usam funções não-lineares.

2.1 Modelos de Box-Jenkins

Os modelos estatísticos desenvolvidos por Box e Jenkins [Box, Jenkins 1970] na década de 70 desempenham um papel importante no campo de previsão de séries temporais. Apesar da simplicidade desses modelos, eles têm sido aplicados satisfatoriamente em uma diversidade de problemas reais e seus princípios servem até hoje como base para outros modelos [Masters 1995a]. A classe de Box e Jenkins contém dois modelos básicos, o modelo *autoregressivo* e o modelo de *médias-móveis*. Esses modelos podem ser usados individualmente ou podem ser combinados.

Em um modelo autoregressivo puro, a previsão de um valor da série é feita através de uma combinação linear dos valores passados da série. . O modelo AR(*p*) (*Autoregressive model*) pode ser descrito pela equação 2.1.

$$Z(t) = f_0 + f_1 Z(t-1) + ... + f_p Z(t-p) + e(t) \qquad (2.1)$$

Nesse modelo o valor da série Z no tempo t é previsto usando os últimos p valores da série. O parâmetro p é chamado de ordem do modelo. Os valores dos parâmetros f_i são ajustados conforme a série que se deseja modelar, visando diminuir o erro de previsão. A variável e representa o erro de previsão obtido pelo modelo.

Um modelo de médias-móveis puro de ordem q, MA(q) (*Moving Average*), representa uma série a partir de valores passados dos erros de previsão, podendo ser descrito pela equação 2.2.

$$Z(t) = q_0 + q_1 e(t-1) + ... + q_q e(t-q) + e(t) \qquad (2.2)$$

Nesse modelo o parâmetro q indica quantos valores passados do erro de previsão são usados como regressores. Os parâmetros q_i devem ser ajustados conforme a série sendo modelada.

O modelo ARMA(*p*,*q*) (*Autoregressive-Moving Average*) é uma generalização dos modelos anteriores, e é formado por uma soma de um componente autoregressivo e um componente de médias móveis. Esses modelos podem ser escritos pela equação:

$$Z(t) = f_0 + f_1 Z(t-1) + ... + f_p Z(t-p) + q_1 e(t-1) + ... + q_q e(t-q) + e(t) \qquad (2.3)$$

Os modelos de Box e Jenkins têm limitações teóricas por serem apenas modelos lineares, e portanto, incapazes de modelar um grande número de fenômenos. As vantagens desses modelos é que eles são simples, fáceis e rápidos de serem implementados. Aplicações dos modelos de Box e Jenkins em várias séries reais e simuladas podem ser vistas em [Box, Jenkins 1970] e [Pankratz 1983].

2.1.1 Projeto dos Modelos de Box-Jenkins

O projeto de um modelo de previsão, segundo [Box, Jenkins 1970], é um processo iterativo envolvendo as seguintes etapas: identificação, estimação do modelo e avaliação dos resultados obtidos (ver figura 2.1).

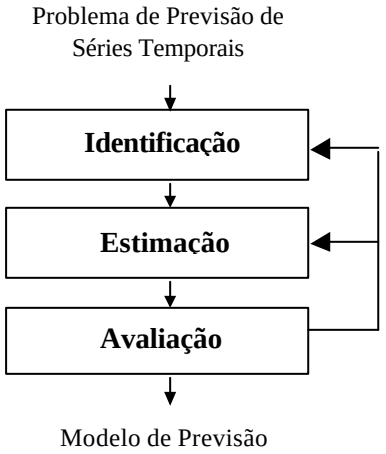

Figura 2.1: Projeto dos modelos de Box-Jenkins
(crédito da figura [Box, Jenkins 1970])

A etapa de identificação nos modelos de Box-Jenkins consiste em definir a ordem do modelo, ou seja, os valores dos parâmetros p e q. Uma das ferramentas mais usadas na identificação desses modelos é a análise das *autocorrelações* e das *autocorrelações parciais*, que são estatísticas das séries, capazes de quantificar dependências no tempo. A estratégia para a identificação aqui é analisar os gráficos dessas estatísticas e tentar identificar algum comportamento que possa associar a série analisada à ordem do modelo. Uma desvantagem dessa estratégia é que ela exige habilidade para analisar os gráficos e inferir a ordem dos modelos. Mais detalhes sobre a identificação através das autocorrelações, assim como exemplos, são encontrados em [Box, Jenkins 1970], [Pankratz 1983] e [Harvey 1993].

A estimação dos modelos é geralmente baseada em alguma técnica que minimize a soma dos erros quadrados [Box, Jenkins 1970]. Como esses modelos têm em geral poucos parâmetros livres, a estimação é geralmente uma etapa rápida.

A forma mais usual para avaliar modelos de previsão é através de alguma medida quantitativa dos erros de previsão, como a média ou a soma dos erros quadráticos. Além disso, outras características dos erros podem ser analisadas como, por exemplo, a normalidade e a aleatoriedade.

2.2 Modelos Neurais de Previsão

Outra abordagem utilizada para a previsão de séries temporais é o uso das Redes Neurais Artificiais (RNAs) [Dorffner 1996]. Em [Drossu, Obradovic 2001], são destacadas duas características importantes das redes neurais que as tornam atrativas para a modelagem de séries temporais. A primeira é o poder das redes neurais como aproximadores universais de funções. Como são modelos não-lineares com um número maior de parâmetros, as RNAs têm o potencial de modelar adequadamente uma quantidade maior de séries em comparação aos modelos lineares clássicos. Outro ponto importante é o relacionamento direto dos modelos de redes neurais com modelos estatísticos. Desta forma, as redes neurais podem se beneficiar com a pesquisa já realizada com os modelos estatísticos clássicos. Em [Dorffner 1996], vemos que alguns dos modelos importantes de

redes neurais podem ser considerados como extensões de modelos estatísticos clássicos, ou ainda como uma forma de implementar modelos estatísticos não-lineares.

Existem diversos modelos de redes neurais que podem ser usados para a previsão de séries temporais. Nas próximas seções, apresentaremos alguns desses modelos dentre os mais conhecidos. Na seção 2.3.1, apresentaremos redes *feedforward*, enquanto que na seção 2.3.2, apresentamos redes recorrentes. Na seção 2.3.3, tecemos uma crítica entre os modelos de Box e Jenkins, apresentados na seção anterior, e as redes neurais.

2.2.1 Redes Feedforward: Multilayer Perceptron e Radial Basis Function

Uma forma de implementar modelos autoregressivos não-lineares é através do uso de redes neurais *feedforward*, onde a camada de entrada recebe os regressores da série e o neurônio da camada de saída retorna a previsão da série em um dado tempo. Nesse caso, para simular um modelo autoregressivo de ordem *p*, a rede neural receberia como entrada os últimos *p* valores da série. A camada de entrada dessas redes recebe o nome de janela de tempo, uma vez que, fornece a cada instante de tempo apenas uma visão parcial da série. Podemos ver na figura 2.2, um exemplo de uma rede *feedforward* com uma camada oculta com três neurônios e com uma janela de tempo de tamanho 4.

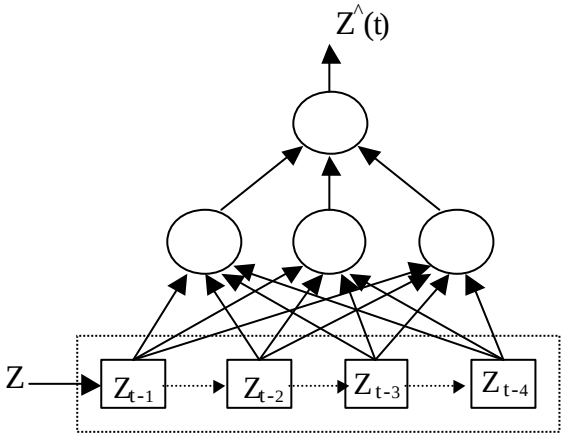

Figura 2.2: Rede feedforward implementando um modelo autoregressivo não-linear de ordem 4

Segundo [Haykin 1994], quando uma rede neural é usada para a tarefa de processamento temporal, ela deve manter de alguma forma um mecanismo de memória. Nas redes *feedforward*, esse mecanismo é definido na camada de entrada da rede, que é a memória dos últimos valores da série a ser prevista.

Existem diferentes termos para designar de uma forma geral os modelos autoregressivos baseados em redes neurais *feedforward*. Em [Haykin 1994], elas são definidas como TLFN (*Time Lagged Feedforward Networks*), devido ao atraso de tempo definido pela janela de tempo. Em [Sjoberg et al. 1994], elas são denominadas de redes NARX (Non-linear AutoRegressive with eXogenous variables), usando a mesma nomenclatura usada para os modelos estatísticos não-lineares.

As redes *feedforward* mais comuns usadas para previsão de séries temporais são as *Multilayer Perceptron* (MLP) [Rumelhart et al. 1986] e as redes *Radial Basis Function* (RBF) [Broomhead, Lowe 1988]. Uma das principais diferenças entre esses dois modelos está no tipo de função implementada nos neurônios. Enquanto que a primeira rede implementa funções sigmóides, a última implementa funções gaussianas. Aplicações das redes MLP na previsão de séries temporais podem ser vistas em [Tang, Fishwick 1993], [Drossu, Obradovic 2001] e [Thiesing, Vornberger 1997], e das redes RBF podem ser vistas em [Hutchinson 1994] e [Nikovski, Zargham 1996].

2.2.2 Redes Recorrentes: Jordan e Elman

As redes neurais podem ser usadas também para implementar modelos ARMA não-lineares. Para implementar um modelo ARMA não-linear de ordem (p,q), as redes armazenam na entrada, além dos p últimos valores da série, os últimos q valores dos erros de previsão. A entrada da rede formada pelos erros de previsão é denominada de camada de contexto. Esses erros são inseridos na camada de contexto através de uma conexão recorrente partindo do neurônio de saída da rede. Na figura 2.3, temos um exemplo de uma dessas redes usando como regressores os dois últimos valores da série prevista e os últimos dois valores dos erros de previsão.

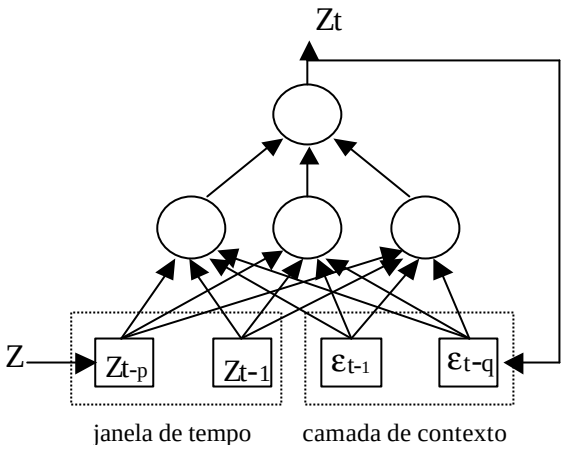

Figura 2.3: Rede Jordan

Segundo [Dorffner 1996], esse tipo de rede é uma simplificação das conhecidas redes Jordan [Jordan 1986]. Outra denominação para esse tipo de rede, vinda da comunidade estatística, é rede NARMAX (Non-linear Autoregressive-Moving Average with eXogenous variables) [Sjoberg et al. 1994]. Aplicações desse tipo de rede recorrente para a previsão de séries temporais podem ser vistas em [Aussem et al. 1994], [Schuster 1996], [Hallas, Dorffner 1998] e [Oliveira 2000].

Outro modelo de rede neural recorrente que é usado freqüentemente para a previsão de séries temporais são as redes Elman [Elman 1990]. A diferença entre a rede Elman e a rede Jordan é que na primeira a conexão recorrente parte da camada intermediária ao contrário das redes Jordan em que a conexão recorrente parte da camada de saída (ver figura 2.4). A camada de contexto das redes de Elman armazena uma cópia das ativações da camada oculta nos tempos anteriores. Aplicações das redes Elman para previsão podem ser vistas em [Pham, Liu 1995], [Koskela et al. 1996], [Stagge, Sendhoff 1997] e [Giles et al. 2001].

Apesar de serem mais gerais e potencialmente mais poderosas que as redes *feedforward*, as redes recorrentes apresentam algumas desvantagens, como por exemplo, o aprendizado mais lento [Koskela et al. 1996]. [Dorffner 1996] salienta que os algoritmos de aprendizado padrões, como o *Backpropagation*, podem não funcionar bem para redes recorrentes. Estudos comparativos entre redes *feedforward* e recorrentes para vários

problemas de previsão podem ser encontrados em [Koskela et al. 1996] e [Hallas, Dorffner 1998].

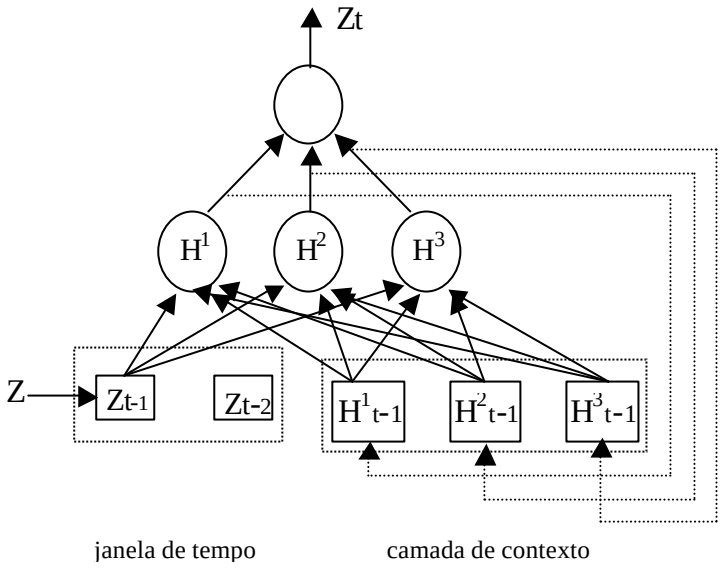

Figura 2.4: Rede Elman

2.2.3 Box-Jenkins versus Redes Neurais

Como visto, as redes neurais têm maior poder computacional se comparadas aos modelos lineares de Box-Jenkins. Enquanto os modelos de Box-Jenkins têm o poder de computar apenas funções lineares, as redes neurais podem computar qualquer função não-linear contínua. Além disso, os modelos de Box-Jenkins podem não se adequar bem às séries não-estacionárias, aquelas em que as características estatísticas, como média e desvio padrão, variam no decorrer do tempo. As redes neurais, pelo contrário, teoricamente não assumem estacionariedade [Dorffner 1996], podendo ser aplicadas a séries não-estacionárias sem o uso de transformações para obter comportamento estacionário (como é o caso dos modelos de Box-Jenkins).

Apesar dos pontos positivos, o uso de RNAs apresentam problemas que não se mostram tão severos nos modelos lineares, como os problemas de *overfitting* e de mínimos locais [Dorffner 1996] e o tempo de construção das redes. Além disso, com uma quantidade pequena de dados, a performance dos modelos neurais pode ser prejudicada [Kreesuradej et

al. 1994]. Essas dificuldades estão diretamente relacionadas ao fato das redes neurais terem mais parâmetros livres que os modelos de Box-Jenkins.

O desempenho das redes neurais como modelos de previsão parece ser mais dependente das decisões tomadas durante o seu projeto. Um exemplo disso pode ser visto comparando os trabalhos de [Sharda, Patil 1990] e [Tang, Fiskwick 1993]. No primeiro trabalho, os autores compararam os modelos de previsão de Box-Jenkins com redes MLP em vários problemas de previsão de séries temporais. Apesar da superioridade teórica das redes neurais, elas obtiveram os melhores resultados em pouco mais da metade das séries analisadas. No segundo trabalho, os autores fizeram novos experimentos, modificando a arquitetura da rede e os parâmetros de aprendizado, com as séries em que as redes neurais obtiveram os piores resultados. Como resultado, em todas as séries analisadas, as redes neurais obtiveram resultados iguais ou superiores aos modelos de Box-Jenkins. Isso mostra que, para esse grupo de séries, o desempenho das redes neurais foi muito dependente das decisões realizadas no seu projeto, nesse caso a escolha da arquitetura e dos parâmetros de aprendizado.

Podemos citar ainda outros trabalhos que atestam a eficácia dos modelos de redes neurais sobre os modelos de Box-Jenkins, como [Kolarik, Rudorfer 1994] e [Drossu, Obradovic 2001].

2.3 Conclusões

Neste capítulo, introduzimos o problema de previsão de séries temporais, e sua relevância em aplicações reais. Dentre os modelos de previsão existentes, apresentamos os modelos de Box e Jenkins e as Redes Neurais Artificiais. Ambos os modelos têm a vantagem de não precisar de um entendimento teórico aprofundado do domínio de aplicação, uma vez que, são gerados a partir de dados. Os modelos de Box-Jenkins têm a vantagem de serem mais simples e rápidos de serem implementados, contudo eles são teoricamente limitados por serem ape0nas modelos lineares. Por outro lado, as Redes Neurais são modelos poderosos capazes de descrever relacionamentos não-lineares complexos. Entretanto, as redes são mais sensíveis a problemas de *overfitting* e mínimos

locais, além de precisarem de um número maior de dados. Em muitos trabalhos as Redes Neurais se mostraram superiores aos modelos de Box-Jenkins. Porém, a superioridade das Redes Neurais é muito dependente de um projeto bem realizado, ou seja, da escolha apropriada de certos parâmetros como número de neurônios da rede e os valores dos parâmetros de treinamento.

A aplicação das redes neurais em problemas do mundo real, como o problema de previsão de séries temporais, pode ser facilitada com o uso de técnicas que tornem o projeto das redes mais eficiente. Nos próximos capítulos, apresentaremos algumas dessas técnicas que podem ser usadas para a otimização do projeto de redes neurais.

3 Algoritmos Genéticos para o Projeto de Redes Neurais

Como já mencionamos anteriormente, o uso eficiente do potencial das redes neurais depende de um projeto bem realizado. Durante a construção de uma rede neural para um problema, vários parâmetros devem ser instanciados, como por exemplo, o número de neurônios na camada oculta, os valores dos parâmetros de aprendizado, os valores dos pesos da rede, as variáveis de entrada da rede, as funções de ativação de cada neurônio, dentre outras. Cada variável tem o seu próprio conjunto de valores possíveis, ou seja, o seu espaço de busca. Assim sendo, o projeto de uma rede neural pode ser visto como um problema de busca e otimização multidimensional, onde cada dimensão é um parâmetro da rede a ser definido. Sob esse ponto de vista, nada mais natural que utilizar um algoritmo de busca e otimização para automatizar o projeto das redes neurais [Balankrishnan, Honavar 1995a].

Em particular, os Algoritmos Genéticos (AGs) [Holland 1975] têm sido largamente usados no projeto de redes neurais, principalmente na definição da topologia, e como algoritmo de treinamento dos pesos. A escolha dos Algoritmos Genéticos se deve ao fato de serem algoritmos de otimização bem mais robustos que os algoritmos de otimização tradicionais, como as técnicas de gradiente, além de serem facilmente implementados. Além disso, os Algoritmos Genéticos são flexíveis o suficiente para atender a diferentes requisitos de várias aplicações, podendo ser usados para o projeto de um grande número de modelos de redes neurais. Apesar de serem usados com maior freqüência na otimização da topologia e dos pesos das redes, os Algoritmos Genéticos podem também ser aplicados para otimizar outros parâmetros do projeto, como as taxas de aprendizado, as funções de ativação, dentre outros.

Nesse capítulo, discutimos como os Algoritmos Genéticos podem ser usados para otimizar o projeto das redes neurais. Na seção 3.1, apresentamos um breve resumo sobre os Algoritmos Genéticos. Na seção 3.2, abordamos o uso de AGs como algoritmo de aprendizado dos pesos de redes neurais e na seção 3.3, abordamos a otimização da topologia de redes neurais com AGs. Na seção 3.4, estão as conclusões. Além das

referências que serão citadas durante esse capítulo, destacamos alguns trabalhos sobre a integração de AGs e Redes Neurais em [Rudnick 1990], [Yao 1995] e [Talko 1999].

3.1 Algoritmos Genéticos

Um problema de busca e otimização se caracteriza por ter um grande número de soluções possíveis, cada uma com um desempenho diferente para resolver o problema. Esse tipo de problema pode ser definido através de um *espaço de busca*, que é o conjunto de todas as soluções possíveis do problema, e uma *função objetivo*, que indica o desempenho de cada solução do espaço de busca para a resolução do problema. Um algoritmo de otimização se propõe a encontrar uma solução para o problema com maior valor de função objetivo possível. Mesmo que não encontre a melhor solução, um bom algoritmo de busca deve pelo menos encontrar uma boa solução para o problema.

Diversos algoritmos de otimização já foram propostos e aplicados aos mais diversos problemas de otimização (ver [Fletcher 1987] e [Hong, Vasaru 2001]). Dentre os algoritmos de otimização mais conhecidos estão os baseados no gradiente [Masters 1995b]. Nesses algoritmos, a cada passo, um ponto no espaço de busca é avaliado. A partir do ponto de busca atual, calcula-se o gradiente local, que indica a direção em que função cresce mais, e com base nessa direção, o novo ponto de busca é gerado. Esse processo é repetido até atingir um critério de parada pré-definido. As desvantagens dessa abordagem é que elas são sensíveis a cair em mínimos locais e são restritas à otimização de funções diferenciáveis. Essas desvantagens limitam drasticamente o uso dos algoritmos baseados em gradiente.

Uma alternativa para os algoritmos baseados em gradiente são os algoritmos estocásticos. Nesses algoritmos, a pesquisa por novos pontos no espaço de busca é feita através de regras de transição probabilísticas. Dentre esses algoritmos, podemos citar o *Simulated Annealing* [Kirkpatrick 1983] e os Algoritmos Genéticos [Holland 1975]. Esses últimos despertaram um grande interesse na comunidade científica [Goldberg 1989] [Davis 1991] [Michalewicz, 1992] [Mitchell 1996].

Os Algoritmos Genéticos (AGs) são algoritmos de busca e otimização inspirados na teoria da evolução das espécies [Darwin 1859]. Na evolução natural, os indivíduos mais adaptados ao ambiente terão maior probabilidade de sobreviver e se reproduzir, passando suas características genéticas para as gerações futuras. Assim, a tendência é que com o passar do tempo esse processo de seleção e reprodução resulte em indivíduos cada vez mais adaptados ao ambiente. Além disso, os indivíduos podem sofrer mutações genéticas que em alguns casos podem também ajudá-lo a sobreviver no ambiente. Os Algoritmos Genéticos são uma abstração desse processo, onde cada *indivíduo* representa uma solução no espaço de busca associada ao valor de sua função objetivo, chamada nesse contexto de *função de aptidão*. Essa solução é armazenada em um *cromossomo* conforme um esquema de representação definido. Em cada geração, os AGs trabalham com uma população de indivíduos. Para gerar uma nova população, é selecionado da população anterior um conjunto de indivíduos para reprodução. Essa seleção deve ser conduzida de forma que, em geral, os indivíduos com maior valor de aptidão tenham maior probabilidade de serem selecionados. A reprodução é realizada através da aplicação dos operadores de *crossover*, que trocam características genéticas armazenadas nos cromossomos dos pais para gerar novos filhos. Esses operadores são aplicados com uma certa taxa de probabilidade. Finalmente, após a reprodução, os operadores de *mutação* são aplicados sobre a população atual. Assim como nos operadores de crossover, a mutação é realizada com uma dada probabilidade. Esse processo é repetido iterativamente até se verificar algum critério de parada pré-estabelecido e a melhor solução gerada é retornada pelo algoritmo. Na figura 3.1, temos um esquema simples de implementação dos AGs.

Segundo [Goldberg 1989], os Algoritmos Genéticos se diferenciam de outros algoritmos de otimização pelas seguintes características:

- AGs trabalham com uma codificação dos parâmetros e não propriamente com os parâmetros: essa característica permite estabelecer esquemas de codificação e operadores genéticos padrões que podem sem aproveitados em diferentes problemas.

- AGs realizam a busca com uma população de pontos do espaço e não simplesmente com um ponto: muitos algoritmos de otimização trabalham apenas com um ponto no espaço de busca e usam alguma regra de transição para gerar o próximo ponto. Essa

estratégia é mais sensível à escolha do ponto inicial de busca e mais suscetível a cair em mínimos locais. Iniciando com vários pontos esse problema é amenizado. Outra vantagem de se usar muitos pontos é facilitar a implementação em paralelo do algoritmo.

- AGs usam apenas informações da função objetivo e não de derivadas ou outro conhecimento auxiliar: essa característica possibilita o uso dos AGs em uma gama muito maior de problemas, sendo capaz de otimizar funções com objetivos conflitantes. Os métodos de otimização baseados em gradiente, por exemplo, têm a restrição forte de otimizarem apenas funções diferenciáveis.

- AGs usam regras de transição probabilísticas e não regras determinísticas: o uso de regras probabilísticas pode ser útil ao algoritmo para sair de mínimos locais, o que muitos algoritmos que usam regras de transição determinísticas, como Hill Climbing, não são capazes de fazer.

Além dessas características, [Lacerda, Carvalho 1999] apresenta ainda outras vantagens dos Algoritmos Genéticos que os tornam métodos canônicos de otimização bem mais robustos que as técnicas tradicionais como, por exemplo, a capacidade de trabalhar tanto com parâmetros contínuos como discretos, a capacidade de otimizar um grande número de variáveis e a facilidade de implementação e de hibridização com outras técnicas.

```
/* P_atual : Indivíduos da população atual
/* P_interm: Indivíduos da população intermediária
/* Apt : Aptidão dos indivíduos da população atual

P_atual = iniciar_população( );          /*    iniciar a população aleatoriamente
Enquanto critério de parada = Falso              /* Verificação dos critérios de parada
        Apt = função_aptidão (P_atual);          /* Cálculo da aptidão
        P_interm = seleção (P_atual, Apt);       /* Processo de seleção
        P_interm = crossover (P_interm);         /* Operadores de crossover
        P_interm = mutação (P_interm);           /* Operadores de mutação
        P_atual = P_interm;                      /* Atualização da população
Fim
Retorna P_atual
```

Figura 3.1: Esquema de implementação geral dos AGs

3.2 AGs para o Treinamento dos Pesos

As Redes Neurais Artificiais têm sido usadas efetivamente em uma variedade de problemas do mundo real [Widrow et al. 1994]. O sucesso dessa técnica para um domínio particular depende fortemente de um bom algoritmo de treinamento que atenda as necessidades e restrições do problema sendo abordado. O treinamento de redes neurais pode ser visto como um problema de otimização onde o espaço de busca é o espaço de pesos da rede e a função objetivo é uma função do desempenho obtido pela rede para um determinado conjunto de padrões. Os diversos algoritmos de treinamento baseados em gradiente ([Battiti 1992], [Masters 1995b] e [Bishop 1995]), em particular os algoritmos de Backpropagation [Rumelhart et al. 1986] e Gradiente Conjugado [Barnard, Cole 1989], têm sido satisfatoriamente usados em diversos problemas. No entanto, como esses algoritmos realizam buscas locais, eles são mais suscetíveis a caírem em mínimos locais e dependem muito dos pesos iniciais da rede. De acordo com [Masters 1995b], o melhor que podem fazer esses algoritmos, teoricamente, é atingir o mínimo local mais próximo do ponto inicial de busca.

Uma maneira de superar o problema de mínimos locais é reiniciar várias vezes os pesos da rede, executar algoritmo de treinamento e, finalmente, selecionar a rede que obteve os melhores resultados. Contudo, alguns pontos devem ser discutidos quanto ao número de reinícios: (1) quando esse número é muito pequeno, os resultados poderão ser insatisfatórios, dependendo do problema abordado e do algoritmo de treinamento usado; (2) quando o número de reinícios é muito grande, os resultados tendem a ser melhores, entretanto, o tempo de treinamento pode ser tornar inviável, especialmente em algoritmos lentos como o Backpropagation. Esse processo é provavelmente mais plausível se usarmos algoritmos de convergência rápida, como os de segunda-ordem [Battiti 1992]. Assim, um número maior de reinícios dos pesos pode ser feito com um prejuízo menor em relação ao tempo de execução.

Uma alternativa interessante para o treinamento de redes neurais é o uso de algoritmos de otimização estocásticos [Masters 1995b]. Entre os algoritmos estocásticos que já foram usados no treinamento de redes neurais podemos citar o *Simulated Annealing* [Kirkpatrick 1983] e o *Adaptative Random Search* [Caprile et al. 1991] que foram aplicados

respectivamente em [Masters 1995b] e [Battiti, Tecchiolli 1994]. Entretanto, os AGs são os algoritmos estocásticos mais usados para o treinamento de redes neurais. Os AGs são menos sensíveis a cair em mínimos locais que os algoritmos de aprendizagem tradicionais baseados em gradiente, como o *Backpropagation*. Eles conseguem sair de situações de mínimos locais principalmente pelo fato de usarem regras de transição de estado probabilísticas. Dentre os trabalhos que usaram os Algoritmos Genéticos como algoritmo de aprendizado dos pesos, destacamos [Montana, Davis 1989], [Fogel et al. 1990], [Belew et al. 1990], [Kinnebrock 1994] e [Castillo et al. 2000].

Nos Algoritmos Genéticos, cada indivíduo armazena uma configuração de pesos da rede e é avaliado por uma função de aptidão que indica o desempenho da rede para o conjunto de padrões de treinamento. A cada geração, a aptidão de cada indivíduo é calculada usando diretamente os pesos armazenados nos cromossomos. Ao contrário dos algoritmos baseados em gradiente, os novos pesos da rede são definidos através dos operadores genéticos sem a necessidade de calcular derivadas, o que em alguns modelos de redes neurais pode ser dispendioso.

Nas próximas subseções, faremos alguns comentários a respeito dos três pontos básicos para a implementação de um Algoritmo Genético: a representação do problema, os operadores genéticos e a função de aptidão. Além disso, apresentamos o treinamento híbrido de redes neurais.

3.2.1 Representação do Problema

O esquema de representação é crucial para o bom funcionamento de um Algoritmo Genético uma vez que ele determina quão complexo será o espaço de busca, o que diretamente influencia a eficiência do processo evolucionário. Além disso, os operadores genéticos são implementados com base na representação escolhida. No caso dos pesos de uma rede neural, o esquema de representação determina também a precisão com que as informações serão armazenadas. Basicamente, dois tipos de representação podem ser usados para codificar pesos de uma rede neural, a binária e a real.

Na representação binária, os pesos da rede neural são transformados em números binários e armazenados em cadeias de bits de tamanho fixo. Essas cadeias são então concatenadas para formar o cromossomo (ver figura 3.2).

Em [Yao 1995], são apresentadas algumas vantagens da representação binária. Primeiramente, a existência de operadores genéticos padrões para a representação binária. Outra vantagem é a facilidade de implementação dos algoritmos em hardware. Dentre as desvantagens, podemos citar que para problemas muito grandes os algoritmos terão que armazenar e manipular uma quantidade muito grande de bits, o que pode diminuir a eficiência do treinamento. Podemos citar também o problema da precisão que é determinada pelo tamanho da cadeia de bits que armazena cada peso. Nesse caso, para cada aplicação existe o conflito entre a precisão requerida pelo problema e a eficiência computacional do processo evolucionário. Esse conflito deve ser resolvido durante a implementação dos algoritmos.

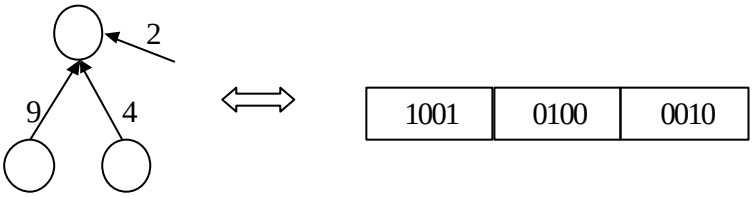

Figura 3.2: Representação binária dos pesos

Embora Algoritmos Genéticos usem tradicionalmente a representação binária em diversos problemas, para o problema da codificação dos pesos de uma rede neural, a representação real têm sido dominante na literatura [Branke 1995]. Nessa representação, os valores reais dos pesos da rede são armazenados diretamente nos cromossomos sem nenhum tipo de transformação (figura 3.3). As vantagens da representação real é que ela diminui o espaço de busca dos pesos e não apresenta problemas de precisão como na representação binária.

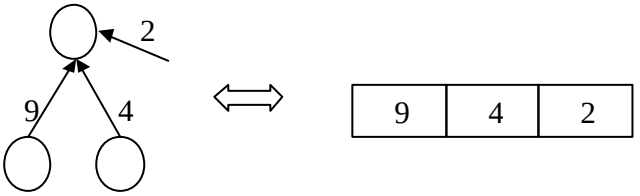

Figura 3.3: Representação real dos pesos

3.2.2 Operadores Genéticos

Os operadores de *crossover*, em geral, causam mudanças mais drásticas nos cromossomos dos filhos do que os operadores de mutação. De fato, enquanto que os operadores de *crossover* trocam blocos inteiros de informação entre cromossomos diferentes, o operador de mutação apenas perturba com uma probabilidade baixa o cromossomo do filho gerado. Essa característica dos operadores de *crossover* pode ter um efeito destrutivo na evolução dos pesos de uma rede neural. Um ponto a observar aqui é que o comportamento das redes neurais é sensível mesmo a pequenas variações dos pesos. Desta forma, duas redes que têm uma boa aptidão podem gerar novas redes com aptidão muito ruim devido ao grande efeito perturbador do operador de *crossover*. Um exemplo desse problema pode ser visto na figura 3.4, em que podemos ver duas redes neurais funcionalmente equivalentes, onde apenas a ordem dos pesos foi trocada. Essas duas redes têm a mesma aptidão, mas cromossomos diferentes. Com a aplicação de um *crossover* de dois pontos, essas redes geram um filho significativamente diferente de ambos os pais. Como o comportamento das redes neurais é sensível a mudanças nos pesos, a aptidão do filho gerado pode ser muito diferente da dos pais. Esse problema é chamado na literatura de problema da permutação [Hancock 1992]. Apesar de alguns trabalhos abandonaram o uso de operadores de *crossover*, usando apenas os operadores de mutação, que causam menor variação nos pesos, [Yao 1995] afirma que ainda é necessário um maior estudo sobre o impacto dos operadores de *crossover* no desempenho dos algoritmos genéticos.

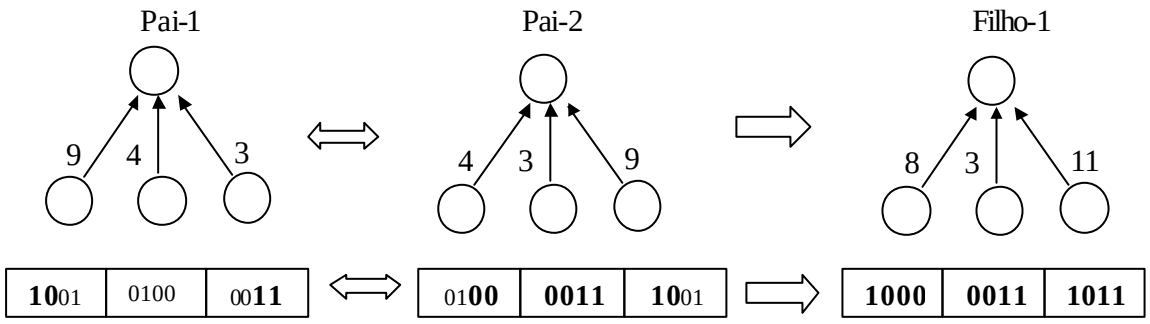

Figura 3.4: Dois pais funcionalmente equivalentes geram um filho significantemente diferente dos pais

Os operadores de mutação, em geral, adicionam pequenas perturbações aleatórias aos pesos da rede. A perturbação mais comum é a Gaussiana [Fogel et al. 1990] [Masters 1995b]. Na perturbação Gaussiana para cada peso da rede, um pequeno número aleatório é definido a partir de uma distribuição Normal $(0, \sigma^2)$ com média 0 e desvio padrão σ constante. Esse número é adicionado ou não ao valor atual do peso conforme a taxa de mutação. A diferença entre os cromossomos do pai e do filho gerado pela mutação depende dos valores definidos para σ e para a taxa de mutação. Esses valores devem ser pequenos o suficiente para não gerarem mudanças muito drásticas nas redes geradas. Na figura 3.5, temos um possível filho gerado pelo Pai-1 da figura 3.4, desta vez com a aplicação de um operador de mutação Gaussiana. Note que as mudanças geradas no cromossomo Filho-2 são menores que as geradas anteriormente com o operador de crossover no Filho-1.

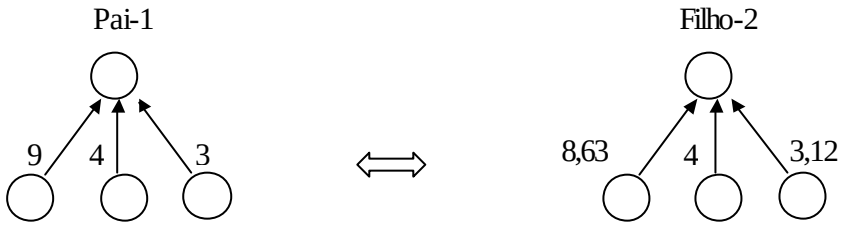

Figura 3.5: Possível filho gerado por um operador de Mutação Gaussiana

Além da distribuição Normal, outras distribuições podem ser usadas para perturbar os pesos das redes como, por exemplo, a distribuição de Cauchy. Para maiores detalhes ver [Masters 1995b].

3.2.3 Função de Aptidão

Como já foi discutido, uma das vantagens do treinamento de redes neurais com AGs é que existe pouca restrição em relação à função otimizada. Como exemplo, a função objetivo não precisa ser diferenciável, o que é o caso dos algoritmos de treinamento tradicionais baseados no gradiente. Desta forma, a função de aptidão dos AGs pode refletir de forma mais precisa as características de cada aplicação. Assim, nos problemas de classificação, uma escolha natural seria usar o erro de classificação. Já nos problemas de regressão, o mais simples é usar o erro quadrático médio.

3.2.4 Treinamento Híbrido de Redes Neurais

O treinamento híbrido de redes neurais combina as vantagens dos AGs e dos algoritmos de treinamento baseados no gradiente, e ao mesmo tempo supera as limitações das duas técnicas. Os AGs são bons em realizar buscas globais, entretanto são ruins para aprofundar a busca localmente. Por outro lado, as técnicas baseadas em gradiente são boas em buscas locais e ruins em buscas globais. No treinamento híbrido, os algoritmos genéticos, enquanto algoritmos de busca globais, são usados para encontrar boas regiões no espaço de pesos e os algoritmos baseados em gradiente realizam uma busca mais aprofundada nessas regiões. Em outras palavras, os algoritmos genéticos são usados para definir os pesos iniciais dos algoritmos baseados em gradiente.

Segundo [Goldberg 1989], existem várias maneiras de unir os algoritmos Genéticos a algoritmos de busca locais, destacando as seguintes: (1) os algoritmos genéticos são executados e em seguida o algoritmo de otimização local aprofunda a busca a partir dos resultados dos algoritmos genéticos e (2) o algoritmo de busca local é inserido no processo de evolução dos algoritmos genéticos. Esses dois procedimentos são apresentados graficamente na figura 3.6. É importante destacar que embora o segundo procedimento possa atingir resultados melhores que o primeiro, ele é mais ineficiente em termos computacionais uma vez que em cada geração o algoritmo de busca local é executado sobre cada indivíduo.

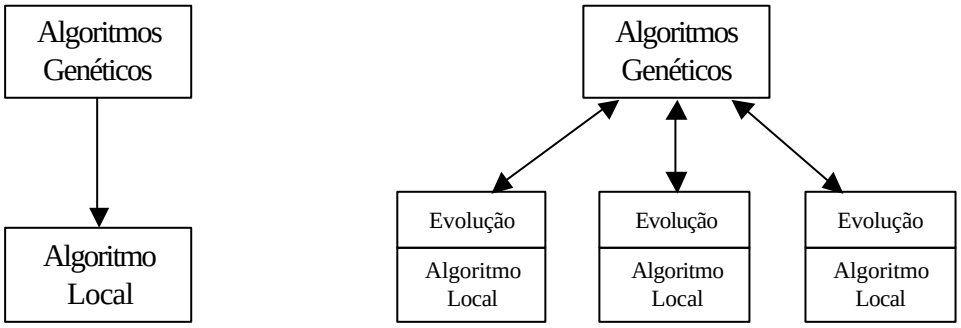

Figura 3.6: Esquemas de hibridização de AGs e algoritmos de busca locais (crédito da figura [Goldberg 1989])

Como se trata de um sistema híbrido, algumas dificuldades podem surgir com o uso do treinamento híbrido de redes neurais. Primeiramente, a implementação é mais difícil uma vez que se trata da implementação e integração de duas técnicas. Além disso, a calibragem de um algoritmo de treinamento híbrido é mais complexa se comparado com o treinamento puro. De fato, tanto os parâmetros do algoritmo de treinamento local como os do algoritmo genético deverão ser definidos.

O treinamento híbrido de redes neurais já foi usado em diferentes trabalhos. Em [Belew et al. 1990], os autores integraram os Algoritmos Genéticos com os algoritmos de Backpropagation e de Gradientes Conjugados e os algoritmos híbridos se mostraram mais eficientes que os seus componentes usados separadamente. Outros trabalhos que usaram algoritmos de treinamento híbridos podem ser encontrados ainda em [Kinnebrock 1994] e [Castillo et al. 2000].

3.3 AGs para Otimização da Topologia

A definição da topologia de uma rede neural tem uma importância vital nos resultados obtidos para um dado problema. Um superdimensionamento da topologia da rede neural, ou seja, a escolha de uma rede com muitos parâmetros livres (pesos ajustáveis), pode levar a problemas de generalização (*overfitting* dos exemplos de treinamento). Já uma topologia muito pequena pode não ser suficiente para resolver o problema em questão. Alguns dos algoritmos mais usados para o aprendizado de redes neurais, como o algoritmo de *Backpropagation*, são aplicados sobre uma topologia de rede fixa definida pelo desenvolvedor em um processo de tentativa e erro. Porém, se o desenvolvedor tiver pouca experiência em usar redes neurais e/ou no problema que deseja resolver, ele eventualmente terá que definir e treinar muitas topologias antes de alcançar bons resultados. Assim, existe uma necessidade óbvia de automatizar esse processo.

Segundo [Haykin 1994], as seguintes estratégias podem ser usadas para definir a topologia de redes neurais:

- Crescimento: a busca é iniciada a partir de uma topologia pequena que é seqüencialmente aumentada seguindo regras deterministicas. Como exemplo podemos citar o algoritmo *Cascade Correlation* [Fahlman, Lebiere 1990].

- Poda: a busca começa a partir de uma topologia grande e as conexões e/ou neurônios desnecessários são identificados e eliminados da rede. Como exemplo citamos o algoritmos de *Optimal Brain Damage* [LeCun et al. 1990].

O problema dessas estratégias é que elas são mais suscetíveis a caírem em mínimos locais, pois realizam uma busca Hill-Climbing no espaço de topologias [Russell, Norvig 1995]. Desta forma, o sucesso dessas estratégias depende muito da topologia inicial.

Uma abordagem alternativa para definir a topologia de redes neurais é com o uso dos Algoritmos Genéticos, que diminuem o risco de mínimos locais pelo fato de avaliar simultaneamente diferentes regiões do espaço de busca ao contrário das estratégias anteriores. Em [Yao 1995], são mencionadas ainda várias características do espaço de busca de topologias de redes neurais que tornam os AGs atrativos para realizar uma otimização nesse espaço. Essas características são:

- Superfície infinitamente grande: uma vez que podem não existir restrições em relação ao tamanho das redes.

- Superfície não-diferenciável: o espaço de topologias é discreto. Essa característica impossibilita o uso de técnicas com gradiente.

- Superfície complexa e com ruído: a avaliação da topologia depende do processo de treinamento que tem uma aleatoriedade associada.

- Superfície deceptiva: duas topologias semelhantes podem ter desempenhos muito diferentes.

- Superfície multimodal: duas topologias muito diferentes podem ter resultados semelhantes.

Existem muitos trabalhos que utilizaram AGs para otimizar a topologia de redes neurais, dentre os quais destacamos [Bornholdt, Graudenz 1992], [Mandischer 1993], [Stepniewsky, Keane 1996], [Yao, Liu 1997] e [Prudêncio, Ludermir 2001a]. Em geral, a evolução da topologia segue o esquema definido na figura 3.7. A cada geração, todos os

cromossomos são decodificados e cada topologia é treinada com um algoritmo de treinamento tradicional a partir de pesos iniciais aleatórios. A função de avaliação é calculada com base nos resultados do treinamento. Nesse ponto, cada topologia já terá o seu valor de aptidão associado. Em seguida, a evolução da topologia é realizada. Esse processo é repetido até atingir um critério de parada pré-definido.

Uma crítica a essa abordagem, é que uma mesma topologia pode ter valor de aptidão diferente dependendo dos resultados do treinamento. De fato, esses resultados dependem de pesos iniciais que são aleatórios. Assim, uma arquitetura potencialmente boa pode ser descartada do processo evolutivo no caso do treinamento receber um conjunto de pesos iniciais ruins. Uma alternativa para atenuar esse problema é treinar cada topologia com várias configurações de pesos iniciais aleatórios e definir como função de aptidão um valor médio dos resultados de treinamento. Entretanto, esse processo requer mais tempo de execução. Outra alternativa é evoluir conjuntamente a topologia e os pesos da rede como em [Koza, Rice 1991] [Liu, Yao 1996]. Nesse caso, cada indivíduo representa uma topologia associada a uma configuração de pesos e teria um único valor de aptidão. Isso elimina totalmente a aleatoriedade na função de aptidão.

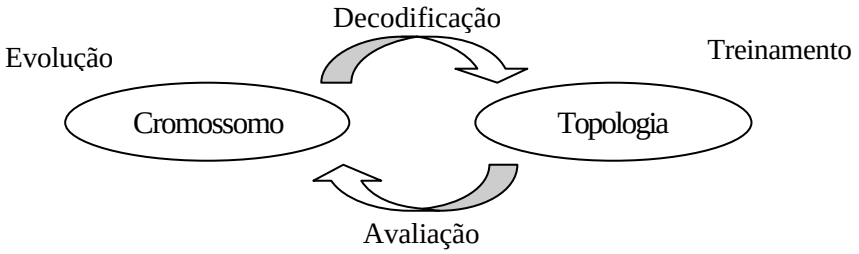

Figura 3.7: Otimização da topologia com AGs
(crédito da figura [Balakrishnan, Honavar 1995a])

3.3.1 Representação do Problema

De acordo com [Balakrishnan, Honavar 1995b], o esquema de representação usado para codificar as topologias é importante porque restringe a classe de arquiteturas envolvidas no problema e determina a complexidade e a eficiência do processo evolucionário. Assim como na codificação dos pesos, a definição dos operadores genéticos depende diretamente do tipo de representação escolhido. A seguir apresentamos os tipos básicos de representação da topologia: a forma direta e a indireta.

Também chamada de forte ou ainda de baixo nível, a representação direta armazena no cromossomo cada conexão da rede. Uma maneira de implementar uma representação direta é definindo uma matriz onde o elemento Cij da matriz C armazena de forma binária a existência de conexão entre os neurônios i e j (ver figura 3.8). O genótipo pode ser gerado pela concatenação das linhas ou colunas da matriz de conexões. Esse tipo de representação é bastante simples de implementar e capaz de codificar tanto redes recorrentes como redes *feedforward*. Restrições no espaço de busca ou conhecimento a priori podem ser aplicadas facilmente na matriz. Outra vantagem é que como a representação direta é binária ela pode se beneficiar dos operadores genéticos padrões da representação binária.

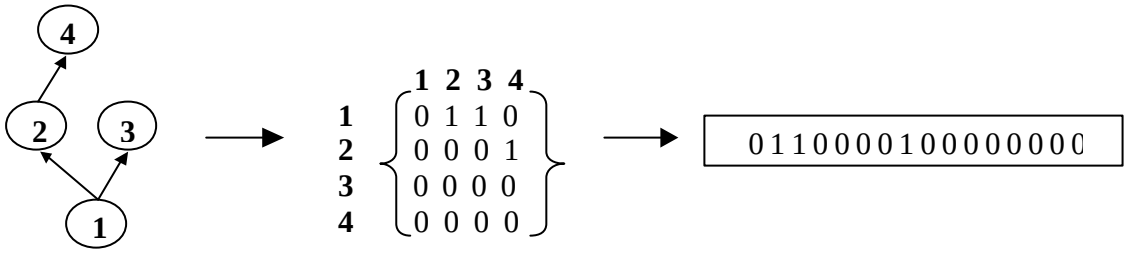

Figura 3.8: Representação direta da arquitetura

Um problema dessa codificação é que o tamanho do cromossomo cresce exponencialmente com o aumento de neurônios na rede. Uma solução para esse problema é restringir o tamanho dos cromossomos, o que pode ter o efeito benéfico de eliminar as redes com muitos parâmetros, mais sensíveis a problemas de *overfitting*.

Em vez de armazenar cada conexão, na representação indireta, fraca ou de alto nível, apenas as características mais importantes são armazenadas, como, por exemplo, o número de camadas, número de neurônios por camada e a forma como os neurônios estão conectados. A representação indireta é mais biologicamente plausível que a direta uma vez que seria impossível para os cromossomos humanos guardarem informações sobre todas as conexões do cérebro.

Por armazenar apenas a estrutura da rede em vez das conexões, o espaço de busca de arquiteturas é reduzido e mais fácil de ser pesquisado. Possivelmente será necessário um número menor de gerações para se encontrar uma boa topologia. Além disso, essa representação ameniza o problema do crescimento do tamanho do cromossomo. Uma

desvantagem dessa representação é que ela, em geral, não é capaz de representar todas as topologias possíveis, gerando apenas as mais regulares [Branke 1995].

3.3.2 Operadores Genéticos

O efeito destrutivo dos operadores de *crossover* também está presente quando se trata de otimizar a topologia. De fato, a modificação da topologia de uma rede neural pode ter um efeito ainda mais drástico no desempenho da rede do que perturbações nos pesos. Por esse motivo, alguns trabalhos, como [Bornholdt, Graudenz 1992] e [Yao, Liu 1997], não adotam esses operadores.

Os operadores de mutação da topologia mais comuns eliminam ou adicionam uniformemente elementos da rede, como conexões ou neurônios [Branke 1995]. A eliminação de elementos da rede é preferível uma vez que encorajam a geração de redes com poucos parâmetros. Em vez de eliminar e adicionar elementos de maneira uniforme, medidas de significância podem ser usadas para eliminar ou inserir elementos que causam menor impacto no desempenho da rede (como em [Yao, Liu 1997]).

3.3.3 Função de Aptidão

Além do que já foi discutido na seção 3.2 sobre função de aptidão, outros critérios podem ser usados para avaliar topologias. Existem critérios que podem ser usados para diminuir o risco de overfitting, penalizando as topologias com muitos parâmetros (conexões) [Moody 1994]. Em geral, esse fator de penalidade leva em conta o número de conexões da rede ou alternativamente o número de neurônios. Podemos ver em [Zhang, Muhlenbein 1993] e [Stepniewsky, Keane 1996], alguns exemplos de funções de aptidão que usam essa idéia.

3.4 Conclusões

Como vimos anteriormente, existem diferentes formas de usar as vantagens dos Algoritmos Genéticos para otimizar o projeto de redes neurais. Nesse capítulo, nós abordamos principalmente o uso dos Algoritmos Genéticos como algoritmos de aprendizado dos pesos e para a otimização da topologia. Para o aprendizado dos pesos, os

Algoritmos Genéticos são capazes de se adaptar as necessidades de diferentes aplicações devido a sua flexibilidade na composição de funções de aptidão. Para a definição da topologia, os Algoritmos Genéticos são alternativas para substituir dispendiosos processos manuais de tentativa e erro e as técnicas construtivas e de poda, que são mais sensíveis ao problema de mínimos locais.

As principais aplicações de AG no projeto de redes neurais são na otimização da topologia e como algoritmos de aprendizado dos pesos. Entretanto existem diversos trabalhos que utilizam AGs em outros aspectos no projeto de redes neurais. Em [Yang, Honavar 1997], os AGs foram usados para selecionar as variáveis de entradas de uma rede neural. Em [Hakkarainen et al. 1996], foram definidos conjuntamente as entradas da rede, o número de neurônios na camada oculta, as funções de ativação, o algoritmo de aprendizado e os parâmetros de treinamento. Em [Whitehead, Choate 1996], os AGs foram usados para definir os centros e as larguras das redes RBF.

No problema de previsão de séries temporais, destacamos alguns trabalhos integrando os Algoritmos Genéticos ao projeto de redes neurais, como [Whitehead, Choate 1996], [DeFalco et al. 1998], [Prudêncio, Ludermir 2001a].

Mesmo tornando o projeto das redes neurais mais eficiente, salientamos que o uso dos Algoritmos Genéticos pode trazer consigo algumas desvantagens, como maior dificuldade de implementação. Essas dificuldades são comuns quando se tenta integrar duas técnicas diferentes em um sistema híbrido.

No próximo capítulo, veremos como o projeto de redes neurais pode ser auxiliado com o uso de conhecimento a priori.

4 Estratégias Baseadas em Conhecimento para o Projeto de Redes Neurais

No capítulo 3, vimos como o projeto de redes neurais pode ser auxiliado com o uso dos Algoritmos Genéticos. A aplicação desses algoritmos passa pela visão do projeto de RNAs como um problema de busca, podendo portanto ser tratado com um algoritmo de busca tradicional. Na prática, o projeto de redes neurais não é limitado apenas a um processo de busca cega. Quando existe conhecimento disponível, ele pode e deve ser usado. De fato, a pesquisa e aplicação de redes neurais já geraram uma gama de *expertise* que permitiria abordar o problema do projeto das redes de uma maneira mais voltada para o uso de conhecimento.

Segundo [Hilário et al. 1996], a escolha de uma técnica para o projeto de redes neurais depende da disponibilidade ou não de conhecimento específico do domínio abordado. Quando o domínio de aplicação dispõe de uma teoria aproximadamente correta e completa, esse conhecimento pode ser usado para inicializar redes neurais, principalmente a topologia e os pesos da rede. Em domínios sobre os quais não se dispõe de conhecimento específico, ainda é possível usar o conhecimento disponível sobre os modelos e algoritmos de redes neurais. O conhecimento nesse caso pode ser usado de forma estática, quando os parâmetros do projeto são definidos antes do processo de treinamento através da análise do conjunto de treinamento, ou de forma dinâmica, durante o processo de treinamento analisando o desempenho da rede.

Nesse capítulo, veremos como o projeto de redes neurais pode ser otimizado com o uso de conhecimento a priori. Como o contexto da dissertação é o estudo de sistemas híbridos, apresentaremos algumas técnicas de Inteligência Artificial que manipulam conhecimento simbólico e que podem ser usadas no projeto de redes neurais. Na seção 4.1, veremos como conhecimento do domínio de aplicação pode ser usado para gerar redes neurais baseadas em conhecimento, apresentando um dos algoritmos mais conhecidos para a construção dessas redes, o algoritmo KBANN (*Knowledge Based Artificial Neural Networks*) [Towell et al. 1990]. Na seção 4.2, abordaremos o uso de conhecimento sobre redes neurais, apresentando algumas técnicas usadas para representar e usar esse

conhecimento de forma efetiva no projeto das redes. Enfim, na seção 4.3, concluiremos o capítulo, tecendo uma crítica sobre as técnicas apresentadas.

4.1 Conhecimento sobre o Domínio de Aplicação

Uma maneira de resolver problemas dentro da Inteligência Artificial é representar o conhecimento do domínio do problema de forma a ser compreendido e usado por uma máquina. O conhecimento disponível sobre um domínio pode ser formalizado através de uma linguagem de representação de alto nível (regras proposicionais, lógica de primeira ordem, lógica *fuzzy*, autômatos...) e ser usado para realizar deduções e resolver o problema sendo tratado. Considere, por exemplo, um problema hipotético de classificar um aluno de mestrado entre as classes, *bom*, *regular* e *ruim*. Na figura 4.1, temos um possível conjunto de regras para resolver esse problema. Essas regras formam uma teoria *aproximadamente* correta e completa do domínio abordado, uma vez que outras variáveis e regras podem ser necessárias para o problema e, além disso, as regras concebidas armazenam somente um conhecimento aproximado sobre o conceito que se quer classificar.

Variáveis do domínio	**Regras para classificação**
Média da graduação: média Conceito no mestrado: conceito Número de publicações: # pub	SE média > 7.0 E conceito = A ENTÃO classe = *bom*. SE média < 7.0 E conceito = A ENTÃO classe = *regular* SE média < 7.0 E conceito = B ou C ENTÃO classe = *ruim*. SE média < 7.0 E #pub > 2 ENTÃO classe = *regular*.

Figura 4.1: Conhecimento simbólico

Existem domínios mais complexos em que mesmo uma teoria aproximada é difícil de ser concebida. Nesses domínios, é de grande validade o uso de técnicas de aprendizado como árvores de decisão, redes neurais, programação em lógica indutiva, dentre outras [Mitchel 1997]. Essas técnicas extraem o conhecimento do domínio a partir de um conjunto

de exemplos do problema que se deseja resolver. No caso das redes neurais, o conhecimento é obtido durante o processo de treinamento e representado nos pesos numéricos e na estrutura da rede de forma implícita. A interpretação do conhecimento armazenado em uma rede neural não é óbvia uma vez que o conhecimento não é representado em uma linguagem simbólica de alto nível e compreensível.

Embora as redes neurais não precisem de um conhecimento teórico aprofundado sobre o problema (apenas de exemplos de treinamento) para serem desenvolvidas, quando o domínio de aplicação dispõe de uma teoria aproximadamente correta e completa, o projeto de redes neurais pode ser auxiliado de forma significativa. Esse conhecimento é, em geral, usado no projeto das redes conforme o esquema da figura 4.2, no qual, uma técnica de compilação é usada para transformar o conhecimento disponível do domínio em uma rede neural (topologia e pesos). A rede gerada será o ponto inicial de treinamento e um eventual ajuste da topologia. A idéia aqui é que a rede neural compilada tenha o mesmo comportamento que a teoria diante do problema abordado, ou seja, que simule a teoria. Desta forma, o processo de treinamento será apenas um refinamento do conhecimento inserido na rede. Quanto mais rico for esse conhecimento, maior será o ganho no desempenho da rede em termos de generalização e tempo de desenvolvimento.

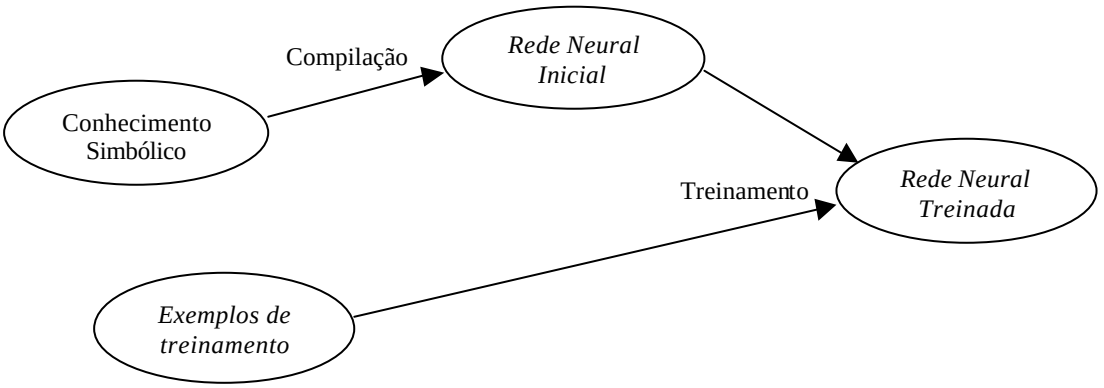

Figura 4.2: Uso de conhecimento do domínio no projeto de redes neurais (crédito da figura [Towell, Shavlik 1994])

As diferentes técnicas de compilação se diferenciam basicamente pela linguagem de representação de conhecimento utilizada (regras, autômatos, gramáticas, etc...) e pelo tipo

de rede gerada (*feedforward* e recorrente). Em [Omlin, Giles 1996], autômatos de estado finito são usados para definir redes recorrentes de segunda ordem. Em [Ivanova, Kubat 1995] e [Kubat 1998], temos técnicas que usam árvores de decisão para inicializar redes *feedforward* (MLP e RBF, respectivamente). Em [Keller, Tahani 1992], os autores implementam redes neurais *feedforward* a partir de regras *fuzzy*. Algoritmos que transformam regras proposicionais em redes feedforward podem ser vistos em [Towell et al. 1990] e [Tresp et al. 1993]. Em [Oliveira, Ludermir 1992], máquinas de Turing são usadas para gerar redes neurais booleanas recorrentes. Podemos ver também implementações de autômatos probabilísticos em redes neurais em [Souto 1999].

As redes neurais geradas a partir de conhecimento do domínio são denominadas genericamente de redes neurais baseadas em conhecimento [Hilário et al. 1996]. Segundo [Agre, Koprinska 1996], o desempenho dessas redes, em geral, tem se mostrado superior se comparado às redes inicializadas randomicamente. A seguir, apresentamos com mais detalhes a técnica de compilação de redes neurais baseadas em conhecimento mais conhecida da literatura, o algoritmo KBANN (*Knowledge Based Artificial Neural Neural Network*) [Towell et al. 1990].

Algoritmo KBANN

O algoritmo KBANN inicializa a topologia e os pesos de uma rede *feedforward* para problemas de classificação a partir de um conjunto de regras proposicionais escritas em cláusulas de *Horn*. A seguir, temos os passos do algoritmo KBANN.

(1) Reescrever as regras: essa primeira etapa serve para tornar o conjunto de regras em uma forma hierárquica semelhante à estrutura hierárquica de redes neurais. Nessa transformação, todas as regras disjuntivas serão representadas por um conjunto de regras cada qual com um só antecedente (ver exemplo na figura 4.3).

(2) Mapear regras em redes neurais: No algoritmo KBANN, os fatos das regras representarão as entradas da rede, e as conclusões finais representarão os neurônios da camada de saída. As conclusões intermediárias, que são deduções realizas até as conclusões finais, são representadas como os neurônios das camadas ocultas. Desta forma, a topologia inicial da rede depende de como as regras são encadeadas. Enfim, os pesos iniciais são

definidos conforme as dependências lógicas entre as variáveis das regras, de forma que o comportamento da rede seja igual ao das regras.

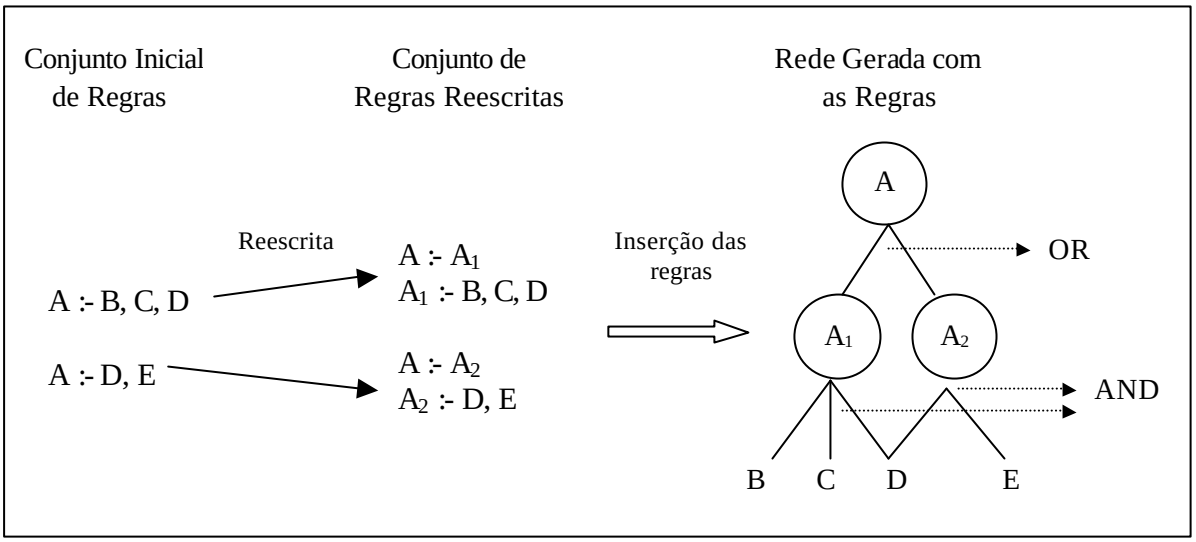

Figura 4.3: Mapeamento de regras em redes no algoritmo KBANN

Um ponto importante na transformação das regras em redes neurais é definir como funções *booleanas* podem ser simuladas na rede. Na figura 4.4, vemos como um neurônio artificial pode simular funções *booleanas* simples como o AND e o OR, a partir de uma definição apropriada dos pesos e do limiar de ativação do neurônio. Na simulação do AND, os pesos associados às entradas binárias são definidos com o valor y. O limiar de ativação é definido com um valor de forma que o neurônio só será ativado se todas as entradas binárias receberem valor 1, desta forma simulando a função AND. Para simular a função OR, o limiar de ativação definido com um valor pequeno o suficiente para ser ativado quando apenas uma das entradas receber valor 1.

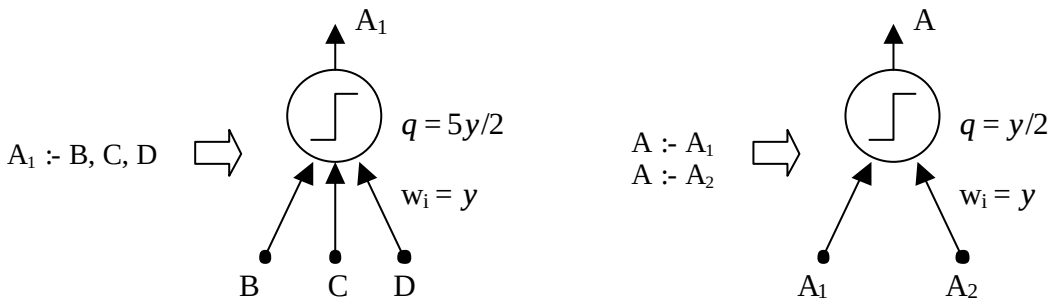

Figura 4.4: Codificação das funções booleanas AND e OR na forma de neurônios

(3) Adicionar novas entradas, neurônios, conexões: essa tarefa serve para adicionar na rede novos conceitos e características que não estavam expressos no conjunto de regras. O artigo que apresenta originalmente o algoritmo KBANN não fornece nenhum mecanismo para modificar a topologia da rede, de uma maneira apropriada (de forma que as regras iniciais não sejam corrompidas e os elementos inseridos tenham interpretação simbólica) Contudo, já existem alguns trabalhos nessa direção (ver [Fletcher, Obradovic 1993] e [Optiz, Shavlik 1993]).

(4) Perturbar os pesos: nessa etapa, pequenas perturbações aleatórias são adicionadas aos pesos da rede. Essa tarefa é importante para evitar problemas causados por simetria [Towell et al. 1990].

O algoritmo KBANN foi aplicado a diversos problemas de classificação como, por exemplo, o reconhecimento de estruturas em DNA [Towell, Shavlik 1994]. As redes geradas por KBANN se mostraram com maior poder de generalização e um desenvolvimento mais rápido do que redes geradas por um processo de tentativa e erro.

4.2 Conhecimento sobre Redes Neurais

Na seção anterior, vimos como o conhecimento específico de um domínio pode auxiliar o projeto de redes neurais. Entretanto, esse tipo de conhecimento resolve apenas uma parte do problema que é a inicialização da rede. Além disso, nem todo domínio dispõe de uma teoria rica o suficiente de forma que a aplicação de uma daquelas técnicas forneça um ganho significativo no desempenho das redes. Quando não se dispõe de uma teoria do domínio, ainda podemos usar o conhecimento sobre as redes neurais para auxiliar o seu projeto. De fato, a pesquisa e aplicação de redes neurais têm gerado conhecimento teórico e prático que podem ser usados efetivamente no projeto das redes. Esse conhecimento pode ser usado de duas formas: (1) estática, quando os parâmetros do projeto são definidos antes do processo de treinamento, (2) ou de forma dinâmica, em que os parâmetros são adaptados durante o processo de treinamento.

Uma das formas de codificar conhecimento e usá-lo para resolver um problema é através de sistemas baseados em regras. Nesses sistemas, o conhecimento é codificado em

regras gerais do domínio e uma máquina de inferência realiza deduções a partir das regras e de instâncias do problema abordado. Para o projeto de redes neurais não existem regras definitivas capazes de tratar o problema de uma maneira mais completa. Entretanto, existem técnicas que podem tratar de pontos isolados do projeto de maneira eficiente. Na seção 4.2.1, veremos uma forma de implementar regras, a lógica *fuzzy*, e o seu uso no projeto redes neurais. Escolhemos apresentar essa linguagem de representação por se tratar de uma linguagem de representação de conhecimento de alto nível de abstração e por ser uma forma menos usual de implementar regras em sistemas especialistas.

Outra forma de implementar sistemas baseados em conhecimento é através do Raciocínio Baseado em Casos (RBC). Nesses sistemas, o conhecimento é armazenado em uma base de casos e novos problemas são resolvidos a partir da adaptação de soluções usadas em casos similares. Na seção 4.2.2, apresentamos o Raciocínio Baseado em Casos, sua aplicabilidade ao projeto de redes neurais e alguns trabalhos relacionados.

4.2.1 Regras Fuzzy

A teoria dos conjuntos *fuzzy* (difusos) foi desenvolvida originalmente em [Zadeh 1965] para fornecer uma ferramenta matemática para lidar com variáveis lingüísticas. Um conjunto *fuzzy* pode ser definido dentro de um universo de discurso X, através de uma função de pertinência que mapeia todos os elementos de X para o intervalo [0,1]. Ao contrário da definição tradicional de conjuntos, onde a função de pertinência é 1 se o elemento pertence ao conjunto e 0 caso contrário, a função de pertinência de um conjunto difuso dá um grau de associação do elemento com o conceito definido pelo conjunto. Como exemplo, a figura 4.5 apresenta a função de pertinência de um conjunto difuso representando o conceito "ALTO" para pessoas. Essa função indica de forma aproximada o "quanto" uma pessoa é alta. Dependendo do conceito que se queira representar no conjunto *fuzzy*, diferentes tipos de funções de pertinência podem ser usados, em geral, funções triangulares e trapezoidais [Galvão 1999].

Uma regra fuzzy é semelhante a uma regra de produção IF-THEN com a diferença que os antecedentes e conseqüentes das regras fuzzy são variáveis lingüísticas associadas a conjuntos difusos. Um domínio pode ser formalizado através de uma base de regras fuzzy e

um procedimento de raciocínio fuzzy, ou raciocínio aproximado, é usado para derivar conclusões a partir das regras [Tzafestas, Blekas 1997].

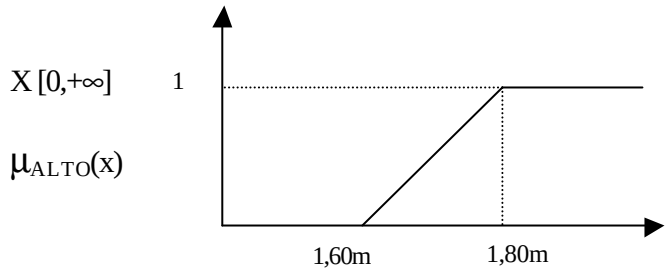

Figura 4.5: Conjunto *fuzzy* representando o conceito "ALTO"

Segundo [Tzafestas, Blekas 1997], um sistema especialista fuzzy tem em geral os seguintes componentes: (1) *módulo de fuzzificação* que transforma as entradas do problema em uma variável na representação fuzzy; (2) *módulo de defuzzificação* que transforma uma saída expressa em lógica fuzzy para um valor solução que possa ser diretamente aplicado ao sistema controlado; (3) *base de conhecimento* que armazena o conhecimento sobre o problema na forma de regras fuzzy; (4) e por fim a *máquina de inferência* que implementa os meios de inferência da lógica fuzzy. Esse esquema de sistema especialista está apresentado na figura 4.6.

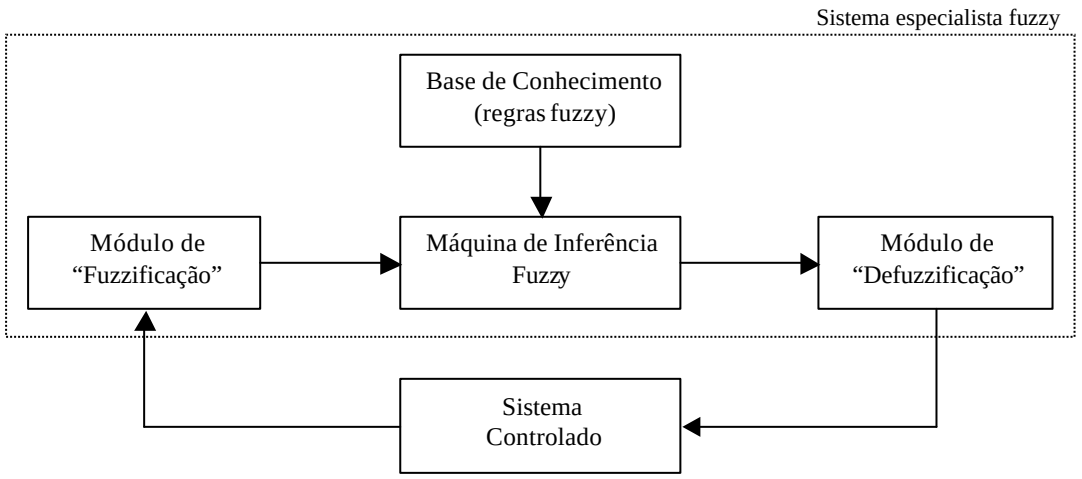

Figura 4.6: Esquema geral para sistemas especialistas Fuzzy (crédito da figura [Tzafestas, Blekas 1997])

Devido a sua capacidade de representar imprecisão e processo de decisão mais flexível, sistemas *fuzzy* têm sido aplicados com sucesso em diversos problemas como em problemas de reconhecimento de padrões e controle [Bezdek, Pal 1992].

Existem várias formas de hibridizar as redes neurais e a lógica *fuzzy* que fogem do escopo dessa dissertação (para maiores informações ver [Kartalopoulos 1995] e [Mitra, Hayashi 2000]). Segundo [Skapura 1996], existem muitas oportunidades de integrar a lógica fuzzy ao projeto de redes neurais, como, por exemplo, em técnicas de poda da topologia, em que a eliminação de conexões seria feita por regras fuzzy. Outra maneira de usar lógica fuzzy, que apresentaremos a seguir, é no controle dos parâmetros de algoritmos de treinamento de redes neurais.

Em geral, cada algoritmo de aprendizado dos pesos de redes neurais tem parâmetros associados. Por exemplo, no algoritmo de *Backpropagation*, temos a taxa de aprendizado e a taxa de momento. O valor dessas taxas é fundamental para a eficiência e convergência do processo de aprendizado. Os parâmetros de aprendizado devem variar durante o aprendizado para acelerar a convergência e evitar instabilidade no algoritmo [Kong, Kosko 1992].

Segundo [Russell, Marks 1999], o treinamento de redes neurais é um processo dinâmico e os diferentes algoritmos usados para adaptar os parâmetros durante o treinamento podem ser vistos como sistemas de controle para acelerar a convergência e evitar instabilidade. Desta constatação, se origina a plausibilidade do uso de lógica fuzzy, uma vez que ela tem sido usada com sucesso em diversos problemas de controle. O controle *fuzzy* do treinamento de redes neurais segue o mesmo esquema definido na figura 4.6. Nesse caso, as regras representam heurísticas de adaptação dos parâmetros de aprendizado e o sistema controlado é o próprio processo de treinamento. Ainda segundo [Russell, Marks 1999], a qualidade final dos resultados do treinamento depende muito mais da qualidade do conhecimento contido nas heurísticas do que no mecanismo usado para implementá-las. Mesmo assim, é destacado que o uso de regras *fuzzy* é muito conveniente devido ao alto nível de abstração dessas regras.

Em [Choi et al. 1992], os autores implementaram regras *fuzzy* para a definição da taxa de aprendizado e da taxa de momento do algoritmo de *Backpropagation*. Nas regras

implementadas são usadas duas variáveis de controle: CE (Change of Error), a aproximação do gradiente e CCE (Change of CE), a aproximação do gradiente de segunda ordem. Para cada uma dessas variáveis, foram criadas cinco variáveis lingüísticas, cada uma associada a um conjunto *fuzzy*: "Negative Big", "Negative Small", "Zero", "Positive Small" e "Positive Big". As variáveis controladas (taxa de aprendizado e taxa de momento) foram associadas a três variáveis lingüísticas: "Negative Small", "Zero" e "Positive Small". Os autores compararam o aprendizado com as regras de adaptação propostas e o algoritmo de *Backpropagation* padrão, e foi constatado que o algoritmo com controle *fuzzy* apresentou convergência mais rápida e um erro quadrático menor.

Dentre outros trabalhos que usaram regras *fuzzy* para a adaptação dos parâmetros de aprendizado, podemos citar ainda [Hertz, Hu 1992] e [Arabshahi et al. 1992].

4.2.2 Raciocínio Baseado em Casos

Uma das formas de implementar sistemas baseados em conhecimento é através de sistemas baseados em regras [Turban 1992]. O conhecimento nesses sistemas é armazenado em uma base de regras gerais do domínio e o raciocínio é realizado por uma máquina de inferência capaz de realizar deduções lógicas a partir das regras do sistema e de instâncias do problema abordado. Um novo problema é resolvido através da aplicação das regras do sistema sobre as variáveis instanciadas do problema.

Apesar do sucesso desses sistemas, eles têm aplicabilidade limitada em domínios muito complexos, em que o conhecimento é pouco formalizado. Nesses domínios, pode ser difícil identificar relacionamentos importantes necessários para compor as regras do sistema. Outra dificuldade é em domínios dinâmicos, onde os relacionamentos entre as variáveis mudam constantemente. Nesse caso, as regras do sistema deverão ser atualizadas constantemente para representar as novas situações do domínio, o que leva a um gasto maior na manutenção do sistema.

Nesse contexto, o Raciocínio Baseado em Casos (RBC) ([Kolodner 1993], [Aadmot, Plaza 1994]) é uma alternativa interessante para implementar sistemas baseados em conhecimento. Em um sistema de RBC, um novo problema é resolvido, adaptando soluções usadas em casos similares resolvidos no passado. Ao contrário dos sistemas

baseados em regras *gerais*, nos sistemas baseados em casos, o conhecimento é armazenado em uma base de casos, contendo várias situações possíveis do problema. Cada caso armazena conhecimento *específico* de uma situação do domínio. O raciocínio é realizado através de 4 etapas: recuperação, reuso, revisão e retenção (ver figura 4.7). Na primeira etapa, os casos mais similares ao caso atual são recuperados da base de casos, em seguida esses casos passam pela etapa de reuso, onde as soluções armazenadas nos casos recuperados são adaptadas para o novo problema. Na etapa de revisão, a solução adaptada é avaliada e eventuais estratégias de reparo são aplicadas à solução, dependendo de sua avaliação. Na última etapa, quando uma solução é aceita, um novo caso poderá ser criado, associando o problema resolvido e a solução proposta. Esse caso poderá ser inserido na base de casos para ser usado em novas situações no futuro.

Segundo [Leake 1996], o Raciocínio Baseado em Casos é uma estratégia de raciocínio eficaz devido a duas observações sobre a natureza do mundo. Primeiramente, problemas similares têm soluções similares, e assim a solução de um problema pode servir como um ponto inicial útil para a resolução do problema similar. O segundo ponto é que os tipos de problemas que um agente encontra tendem a acontecer seguidamente. Desta forma, futuros problemas serão provavelmente similares aos atuais.

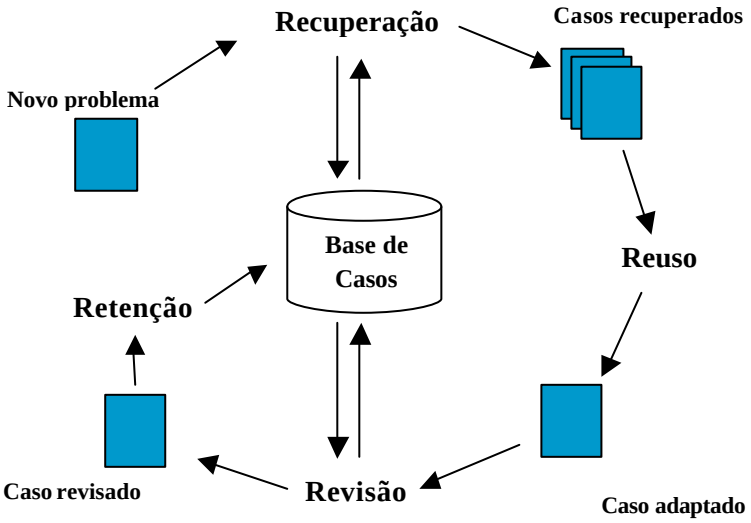

Figura 4.7: Ciclo de Resolução de Problemas do Raciocínio Baseado em Casos (crédito da figura [Aadmot, Plaza 1994])

No contexto do projeto de redes neurais, cada caso associa um problema resolvido pelas redes associado a uma solução bem sucedida. Essa solução armazena os parâmetros do projeto, como o tipo de rede utilizado, a arquitetura, o algoritmo de treinamento, dentre outros. Podemos destacar alguns pontos positivos do uso do Raciocínio Baseado em Casos para automatizar o projeto de redes neurais:

- RBC não necessita de uma formalização precisa do domínio como os sistemas baseados em regras: definir regras globais para o projeto de redes neurais é uma tarefa difícil devido à complexidade do problema. De fato, segundo [Michaud, Rubio 1996], não existem regras definitivas para o projeto de redes neurais.

- RBC é capaz de definir os parâmetros do projeto de uma forma mais completa, uma vez que não existe nenhuma restrição sobre o tipo de solução que pode ser armazenada em um caso.

- RBC se adapta a mudanças no domínio: isso facilitaria o uso das redes neurais para problemas em domínios dinâmicos.

Um dos primeiros trabalhos a usar uma base de casos para o projeto de redes neurais foi [Michaud, Rubio 1996]. Nesse trabalho, os autores montaram um pequeno conjunto de casos com problemas de classificação, para sugerir parâmetros do projeto de redes *Multilayer Perceptron*. Os parâmetros definidos pelo sistema implementado foram: o número de neurônios na camada oculta, o intervalo dos pesos iniciais, a taxa de aprendizado e a taxa de momento do algoritmo de *Backpropagation*, o número máximo de épocas de treinamento e o número de reinícios dos pesos. Os casos da base serviram como ponto inicial para heurísticas de busca. Nesse trabalho, os autores ainda não usam o termo Raciocínio Baseado em Casos, entretanto compartilham as mesmas idéias dessa metodologia. Em [Sovat, Carvalho 2001], os autores propõem o uso de Raciocínio Baseado em Casos também para problemas de classificação. Neste trabalho, os autores prevêem uma iteração com o usuário ao contrário do trabalho anterior.

Em [Prudêncio, Ludermir 2001b] os autores usam uma base de casos para definir a arquitetura de redes NARMAX para previsão de séries temporais. As soluções retornadas pela base de casos serviram como pontos iniciais de busca de um algoritmo genético. Nesse trabalho, os algoritmos genéticos tiveram melhor desempenho quando inicializados com as

soluções recuperadas da base. Esse último trabalho será apresentado com mais detalhes no capítulo 5.

4.3 Conclusões

Nesse capítulo, apresentamos como conhecimento pode ser usado para auxiliar o projeto de redes neurais. Destacamos aqui duas fontes de conhecimento: (1) sobre o domínio do problema abordado, e (2) sobre os modelos de redes neurais e algoritmos de treinamento. No primeiro tópico, apresentamos técnicas que compilam redes neurais a partir de teorias representadas em linguagens simbólicas. Vimos que essas técnicas podem trazer um ganho no desempenho das redes se comparadas à sua definição aleatória. No segundo tópico, destacamos algumas técnicas que codificam e usam o conhecimento geral sobre redes neurais. Apresentamos a lógica *fuzzy*, como uma forma de implementar sistemas baseados em regras, e o Raciocínio Baseado em Casos.

Vimos que o Raciocínio Baseado em Casos pode ser vantajoso sobre os sistemas baseados em regras quando trata de um domínio pouco entendido. Esse é o caso do projeto de redes neurais onde é difícil estabelecer regras para definição dos seus parâmetros. Na próxima seção, apresentaremos um modelo para automatizar o projeto de redes neurais que envolve o uso de RBC.

5 RBC e AG para a Definição da Arquitetura de Redes Neurais

Como já foi dito anteriormente, as Redes Neurais Artificiais (RNAs) apresentam diversas vantagens que as tornam bem sucedidas em uma diversidade de problemas. Dentre essas características podemos citar: forte poder computacional, tolerância a falhas e capacidade de trabalhar com dados incompletos e com ruído. Entretanto, o desempenho de uma rede neural para um determinado problema está diretamente relacionado a um projeto de rede bem realizado. De fato, os resultados de uma rede neural podem ser prejudicados pelas decisões arbitrárias de um desenvolvedor com pouca experiência. Desta forma, o projeto automático das redes neurais é importante para otimizar o desempenho das redes.

Nos capítulos 3 e 4, vimos algumas técnicas que podem auxiliar o projeto das redes neurais, partindo de dois pontos de vista diferentes: (1) o projeto de redes neurais como um problema de busca, daí a aplicabilidade dos Algoritmos Genéticos, e (2) o projeto de redes neurais como um problema resolvido com o uso de conhecimento. Entretanto, na prática, o projeto de redes neurais envolve os dois enfoques: ele é um problema de busca que pode ser iniciada, restringida ou direcionada com o uso de conhecimento.

Dentro desse contexto, propomos um modelo para a automatização do projeto de redes neurais integrando a metodologia de Raciocínio Baseado em Casos e os Algoritmos Genéticos. No nosso modelo, a busca é realizada pelos Algoritmos Genéticos e a inicialização da busca é feita pelo Raciocínio Baseado em Casos.

Esse capítulo será apresentado como se segue. Na seção 5.1, temos a descrição geral do modelo proposto, em seguida, na seção 5.2, definiremos as suas etapas de implementação. Na seção 5.3, apresentamos a aplicação do modelo proposto para a definição da arquitetura de redes neurais para previsão de séries temporais. Finalmente, na seção 5.4, concluímos o capítulo.

5.1 Descrição do Modelo

Nesse trabalho, propomos um modelo para automatizar o projeto das RNAs usando a metodologia de Raciocínio Baseado em Casos e os Algoritmos Genéticos. Cada caso associa um tipo de problema resolvido por redes neurais (classificação, previsão, etc...) e uma solução que foi previamente usada para o problema. A solução armazena pontos importantes do projeto, como o modelo de rede usado, a arquitetura da rede, o algoritmo de aprendizado, os parâmetros de treinamento, dentre outros. Diante de um novo problema, uma busca é feita na base de casos recuperando as soluções usadas nos problemas mais similares ao problema abordado. Essas soluções servem como ponto inicial de busca dos Algoritmos Genéticos, que é responsável pela tarefa de adaptação do Raciocínio Baseado em Casos. Após a execução dos Algoritmos Genéticos, a solução retornada pode ainda ser revisada pelo usuário e, em seguida, inserida na base de casos e usada no futuro em novos problemas.

O nosso modelo automatiza o projeto de redes neurais de forma híbrida, usando um sistema baseado em conhecimento e um algoritmo de busca. De fato, quando um desenvolvedor projeta uma rede neural manualmente, ele usa o seu conhecimento para definir, da melhor maneira possível, os parâmetros do projeto. Ele propõe uma solução inicial, e partir desse ponto, modifica os parâmetros, visando melhorar ainda mais os resultados, ou seja, ele realiza uma busca. Desta forma, para automatizar esse processo, seria natural combinar um sistema baseado em conhecimento e um algoritmo de busca e otimização. A vantagem de usar RBC como sistema baseado em conhecimento é que ele é mais adequado que os sistemas baseados em regras, em domínios pouco formalizados, como é o caso do projeto de redes neurais. A vantagem dos AGs como algoritmos de busca é que eles são menos sensíveis a problemas de mínimos locais. Outro ponto positivo nessa integração é que tanto RBC como os AGs, podem ser usados para definir os parâmetros do projeto de redes neurais de uma forma mais completa, uma vez que tanto o cromossomo dos AGs, como os casos do RBC, podem armazenar parâmetros do projeto de maneira irrestrita.

A idéia de usar os Algoritmos Genéticos a partir de soluções retornadas de uma base de casos esteve presente em outros trabalhos na resolução de diferentes problemas, como o

problema do Caixeiro Viajante, projeto de circuitos e jogos [Louis, Li 1997], [Louis, Johnson 1997] e [Ramsey, Grefessette 1993]. Até o presente momento, não encontramos nenhum trabalho que combinasse as duas técnicas para o problema do projeto de redes neurais.

Nas próximas seções, apresentamos os pontos importantes da implementação do modelo proposto. Na nossa visão, os Algoritmos Genéticos são os responsáveis pela tarefa de adaptação do Raciocínio Baseado em Casos. Desta forma, descreveremos as etapas do modelo proposto sob o ponto de vista do Raciocínio Baseado em Casos, incluindo os AGs na etapa de *Reuso*.

5.1.1 Aquisição de Casos

Essa etapa consiste em compor uma base inicial de casos. Aqui, podemos destacar as seguintes tarefas:

- Identificação: consiste em definir que variáveis e características do problema e da solução serão usadas para representar e indexar os casos. No nosso contexto, um problema pode ser caracterizado através das características dos dados de treinamento e de variáveis do domínio de aplicação. No problema de classificação, por exemplo, podemos usar a quantidade de padrões, desvio-padrão das amostras de cada classe, e o índice de correlação de cada classe [Sovat, Carvalho 2001]. No problema de séries temporais podemos citar outras características como o tamanho, a média, a variância e as autocorrelações das séries, a presença de sazonalidade e tendência, o espectro da série dentre outras. Segundo [Wettschereck et al. 1997], a escolha dos atributos pode ser automatizada por um algoritmo de busca. Além das características do problema, o caso deve armazenar também parâmetros da solução ou um subconjunto desses parâmetros, como que tipo de rede foi usada para o problema, que tipo de algoritmo de treinamento, a arquitetura de rede, dentre outros. Na figura 5.1, apresentamos um possível caso dentro do domínio de séries temporais.
- Representação: Um ponto a definir aqui é a estrutura de representação dos casos. Outro ponto importante é como a base de casos será organizada. Dependendo do número de casos da base, estruturas de indexação hierárquicas poderão facilitar o processo de busca.

- Coleta dos casos: Os casos poderão ser extraídos da literatura de redes neurais (livros, anais de conferência, relatórios técnicos) [Sovat, Carvalho 2000], definidos experimentalmente, ou definidos de forma automática como, por exemplo, com um algoritmo de busca.

Caso 1

PROBLEMA		**SOLUÇÃO**	
Série Temporal	[0.2, 0.4, 0.3, ..., 1.2]	**Modelo**	Rede NARMAX
Média da Série	0.625	**Arquitetura**	janela de tempo=2
Desvio Padrão	0.12		camada de contexto=1
Autocorrelações	[1, 0.9, 0.76, 0.5,.., 0.1]		camada oculta=4
Sazonalidade	Não	**Algoritmo**	Backpropagation
Tendência	Sim	**Taxa de aprendizado**	0.2

Figura 5.1: Exemplo de caso no domínio de séries temporais

5.1.2 Etapas do Raciocínio Baseado em Casos

A resolução de problemas no Raciocínio Baseado em Casos consiste nas seguintes etapas: recuperação, reuso, revisão e retenção, apresentadas a seguir.

a) Recuperação

O ponto mais importante dessa tarefa é definir uma medida de similaridade entre problemas de séries temporais. Uma maneira de fazer isso poderá ser através da distância euclidiana entre os atributos descritores das séries.

b) Reuso Utilizando AGs

O reuso das soluções armazenadas nos casos consiste em treinar as redes neurais com os dados do problema, usando os parâmetros definidos nos casos. A etapa de reuso de um sistema de RBC geral prevê uma etapa de adaptação, onde as soluções retornadas da base são modificadas de forma a se adequar melhor ao problema sendo tratado. No nosso modelo, as soluções retornadas da base são adaptadas pelos Algoritmos Genéticos. Um cromossomo nos AGs devem armazenar em sua estrutura os mesmos parâmetros da solução armazenada no caso. Se o caso armazenar apenas soluções parciais (subconjunto dos parâmetros do projeto de redes neurais), os parâmetros restantes poderão ser definidos de

várias formas como aleatoriamente, pelo próprio usuário, ou por alguma regra pré-definida. (Por exemplo, no caso da figura 5.1, não temos nenhuma informação sobre como os pesos da rede devem ser inicializados).

Quando os parâmetros da solução são todas variáveis numéricas, então o cromossomo pode usar diretamente a mesma representação do caso, caso contrário, transformações devem ser feitas para transformar variáveis textuais e categóricas em variáveis numéricas. Na figura 5.2, vemos como a solução armazenada no caso da figura 5.1 pode ser transformada em um cromossomo, usando a representação real.

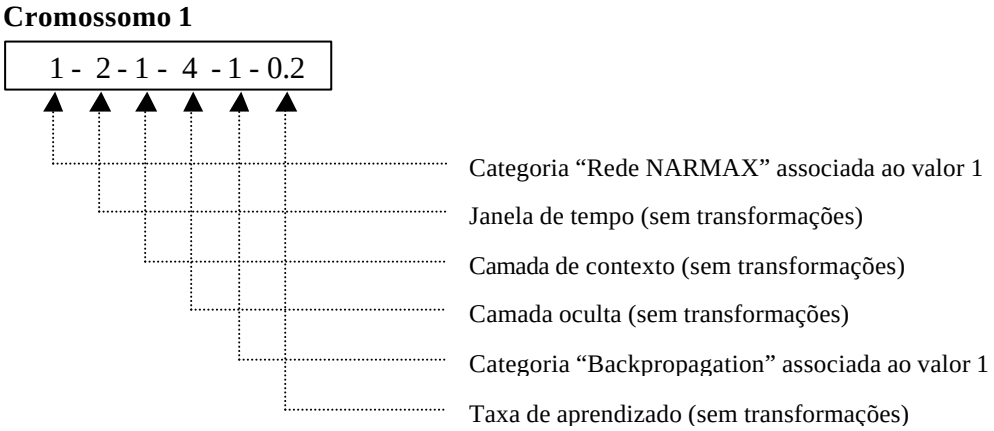

Figura 5.2: Representação do caso da figura 5.1 em um cromossomo

Os operadores genéticos serão os responsáveis por modificar os parâmetros armazenados nas soluções a fim de pesquisar novos pontos no espaço busca. A implementação desses operadores dependerá do tipo de parâmetro da solução armazenado no cromossomo. Enfim, a função de aptidão mede o desempenho da solução para o problema abordado, sendo assim específica para cada aplicação.

c) Revisão

Nessa etapa, a solução proposta pelos AGs pode ser reavaliada, e estratégias de reparo podem ser aplicadas antes da solução ser inserida na base. A avaliação poderá ser feita pelo usuário, ou pelo próprio sistema com o auxílio de novos dados do problema. Dependendo da avaliação, novas soluções poderão ser indicadas, ou pelo próprio usuário, ou automaticamente por um algoritmo de busca.

Após a aceitação da solução, o sistema poderá trabalhar de forma automática visando encontrar soluções ainda melhores. Isso deverá ser feito enquanto o sistema não estiver sendo requisitado pelo usuário. A motivação aqui é tentar gerar soluções mais próximas possível da solução ótima.

d) Retenção

Nessa etapa, devemos definir se o caso será armazenado na base e fazer a manutenção da base. Essa etapa dependerá de quão diferente é o novo caso dos casos já armazenados na base. Para ser inserido na base, um caso deve conter uma situação nova ainda não armazenada na base.

5.2 Estudo de Caso

Nessa seção, apresentamos um estudo de caso para a definição da arquitetura das redes NARMAX [Sjoberg et al. 1994] (ver seção 2.2) para a previsão de séries temporais. Os modelos utilizados são modelos *univariados*, ou sejam usam apenas informações da própria série para a sua previsão. A arquitetura dessas redes teve os seguintes parâmetros otimizados: o tamanho da janela de tempo, o tamanho da camada de contexto e o número de neurônios da camada oculta.

Na figura 5.3, podemos ver a arquitetura do protótipo implementado. O módulo de *Processamento* (PROC) é responsável pelo pré-processamento da série temporal de entrada. O módulo *Gerenciador da Base de Casos* (GBC) é responsável pela manutenção da base de casos e recuperação dos casos. Desta forma, esse módulo implementa as etapas de recuperação e retenção do Raciocínio Baseado em Casos. No módulo *Algoritmo Genético* (AG), estão implementados os AGs para a otimização da arquitetura dos modelos NARMAX. O módulo de *Treinamento* (TR) implementa as redes NARMAX e o algoritmo de treinamento de Levenberg-Marquardt [Marquardt 1963] para os pesos das redes. Os módulos AG e TR implementam a etapa de reuso do Raciocínio Baseado em Casos.

O módulo GBC recebe como entrada a série temporal processada pelo módulo PROC e retorna um número pré-definido de casos, selecionados com base nas suas

similaridades com a série processada. O número de casos recuperados é igual ao tamanho da população dos AGs. Em seguida, essas arquiteturas são inseridas na população inicial de um algoritmo genético implementado no modulo AG. Cada arquitetura é treinada pelo módulo TR, responsável pelo aprendizado dos pesos e pela definição da aptidão de cada arquitetura. A solução final será a melhor arquitetura gerada pelo módulo AG durante o processo de evolução, em termos do erro quadrático médio sobre o conjunto de validação. Em seguida, um novo caso é criado e inserido na base associando a série processada e a arquitetura final. O novo caso é então disponibilizado para uso no futuro, a fim de sugerir soluções adequadas para novos problemas. No protótipo implementado, não houve nenhuma revisão na solução gerada pelos AGs. Cada solução é sempre inserida diretamente na base de casos.

O protótipo inteiro foi implementado em Matlab 5.0 [Mathworks 1996]. As redes NARMAX, e o algoritmo de Levenberg-Marquardt foram implementados usando o *toolbox Nnsysid* (*Neural Network System Identification*) para Matlab 5.0 [Norgaard 1997].

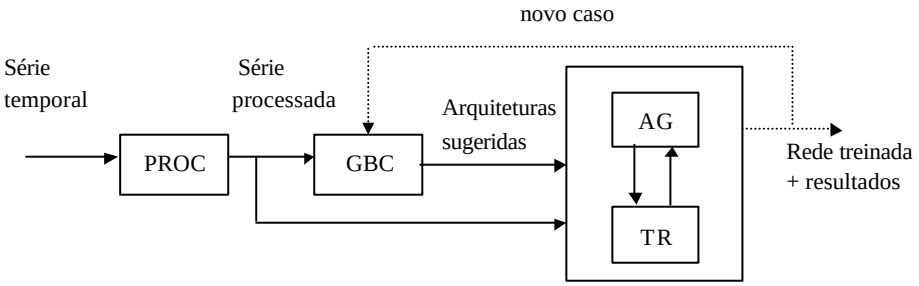

Figura 5.3: Arquitetura do protótipo implementado

5.2.1 Aquisição dos Casos

- Identificação: Além dos dados da própria série, as autocorrelações da série também foram consideradas no protótipo.

- Representação: Cada série temporal foi representada como um vetor numérico e armazenado em um arquivo de forma seqüencial. A solução armazena também de forma vetorial os valores dos parâmetros otimizados.

- Coleta dos casos: A base inicial de casos foi criada com 47 casos. Para gerar os casos, escolhemos 47 séries temporais da literatura (ver Apêndice 1) e aplicamos os AGs para definir a arquitetura para cada série. Nessas execuções, o número de cromossomos por geração foi 4 e o número de gerações por execução foi 5. Assim, para cada série, 20 arquiteturas foram definidas e treinadas e a de menor erro de validação foi tomada como a solução armazenada no caso. A taxa de mutação foi de 40%. Na primeira geração dos AGs, os valores da janela de tempo e da camada de contexto foram definidos aleatoriamente dentro do intervalo de [0;12] e o número de neurônios na camada oculta foram definidos dentro do intervalo [1;5]. Mais detalhes sobre a implementação dos AGs serão vistos a seguir.

5.2.2 Implementação dos módulos

a) Módulo PROC

A função desse módulo é transformar uma série temporal não estacionária em uma série estacionária, ou seja, que não apresentam tendência e sazonalidade**.** Em [Kolarik, Rudorfer 1994]**,** os autores alcançaram melhores resultados experimentais quando eliminaram a tendência e sazonalidade das séries.

Para eliminar a tendência das séries, aplicamos operadores de diferença definidos conforme a equação 5.1.

$$Z^d(t) = Z(t) - Z(t-1) \qquad (5.1)$$

onde Z é a série sendo analisada e Z^d é a série diferenciada. Esses operadores são aplicados sucessivamente sobre a série resultante até que a tendência seja eliminada. Para identificar quando esse processo deve ser interrompido, implementamos o teste de seqüências, ou teste de Wald-Woldwitz [Morettin, Toloi 1987], que é um teste estatístico usado para verificar a presença de tendência em séries temporais.

A eliminação da sazonalidade das séries é feita de forma análoga, usando o operador de diferenças sazonais, conforme a equação 5.2.

$$Z^d(t) = Z(t) - Z(t-s) \qquad (5.2)$$

onde Z é a série sendo analisada, Z^d é a série diferenciada e s é o período de sazonalidade da série. Ainda não implementamos nenhum teste estatístico para verificar automaticamente a presença de sazonalidade. Por enquanto, estamos fazendo essa verificação visualmente, através da análise do gráfico das séries.

b) Módulo GBC

Dada um novo problema, o módulo GBC mede a similaridade do problema atual com todos os problemas armazenados na base e retorna os mais similares. O número de casos retornados da base é igual ao número de cromossomos da população dos AGs. A medida de similaridade implementada foi a distância euclidiana entre as 10 primeiras autocorrelações amostrais de cada série. A autocorrelação de ordem k, *r(k)*, mede a dependência entre os valores da série no tempo *t* e no tempo *t-k* e pode ser estimado pela equação:

$$r(k) = \frac{1}{N}\sum_{1}^{N}(Z(t) - m) * (Z(t-k) - m) \qquad (5.3)$$

onde Z é a série temporal sendo analisada, N é o número de valores da série e m é a média da série. As autocorrelações de mais baixa ordem são mais significativas para descrever séries temporais do que as de mais alta ordem [Box, Jenkins 1970]. As 10 primeiras autocorrelações já são suficientes para revelar padrões de comportamento nas séries.

As autocorrelações amostrais são estatísticas das séries que medem dependências no tempo, comumente utilizadas na definição dos modelos de Box e Jenkins ([Box, Jenkins 1970] e [Pankratz 1983]) e também no projeto de redes neurais ([Sjoberg et al 1994] e [Norgaard 1997]). Como visto na seção 2.1, para se determinar se um modelo de Box-Jenkins é adequado para uma série, uma estratégia é comparar o comportamento teórico de modelos conhecidos com o comportamento das autocorrelações amostrais. O modelo escolhido será o que apresentar o comportamento mais similar (ver [Box-Jenkins] para mais detalhes). Podemos dizer que, de certa forma, a comunidade estatística vem usando

implicitamente as mesmas idéias do Raciocínio Baseado em Casos para inferir os modelos de Box-Jenkins. Em parte, o que estamos fazendo aqui é implementando computacionalmente esse mecanismo para os modelos de redes neurais.

c) Módulo AG

A seguir descreveremos os pontos mais importantes da implementação dos algoritmos genéticos: representação, operadores genéticos, mecanismo de seleção e a função de aptidão.

Representação dos Pesos

A topologia da rede foi representada de forma indireta. Cada cromossomo consiste em um vetor armazenando os valores reais dos parâmetros otimizados, como na figura 5.4. O primeiro elemento do cromossomo corresponde ao tamanho da janela de tempo da rede, o segundo elemento ao tamanho da camada de contexto e, finalmente, o terceiro elemento do cromossomo corresponde ao número de neurônios da camada oculta. Após a decodificação do cromossomo, uma rede totalmente conectada é criada.

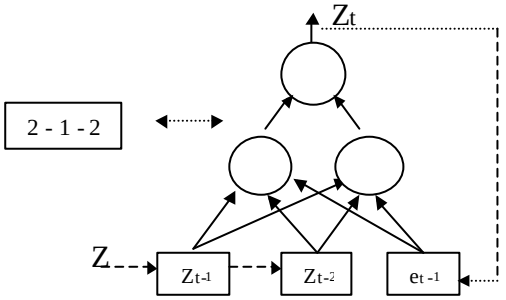

Figure 5.4: Exemplo de codificação

Operadores Genéticos

O operador de *crossover* não foi implementado devido ao seu possível efeito destrutivo na otimização de redes neurais (ver seção 3.2.2). Como operador genético, implementamos apenas um operador de mutação que adiciona ou remove uma unidade dos valores atuais dos parâmetros. Considere, por exemplo, o cromossomo representado na figura 5.4. O primeiro elemento do cromossomo armazena atualmente o valor 2. Os

possíveis valores para esse elemento na próxima geração, caso a mutação seja aplicada, são 1 e 3, correspondendo respectivamente à remoção e à adição de uma unidade ao seu valor atual. O elemento permanecerá com valor 2, caso o operador de mutação não seja aplicado. Desta forma, o primeiro elemento tem como valores possíveis para a próxima geração o conjunto 1, 2 e 3. De forma análoga, o segundo elemento tem como valores possíveis o conjunto 0, 1 e 2, e o terceiro elemento o conjunto 1, 2, e 3. No total, existem 3 x 3 x 3, ou seja, 27 configurações possíveis para esse cromossomo na próxima geração, correspondendo às combinações dos valores possíveis de cada elemento. O que irá influir na definição dos próximos cromossomos é o valor da taxa de mutação. Quanto mais alto for esse valor, mais diferentes serão os novos cromossomos em relação aos cromossomos da população anterior.

Seleção

Existem basicamente duas formas de implementar o mecanismo de seleção nos Algoritmos Genéticos: seleção proporcional à aptidão e seleção por torneio. A seleção proporcional à aptidão pode gerar convergência prematura uma vez que os indivíduos com menor aptidão são rapidamente descartados [Lacerda, Carvalho 1999]. Desta forma, o espaço de busca pode não ser bem explorado. Por esse motivo implementamos a seleção por torneio, que no nosso caso foi torneio de 3. Nessa seleção, três indivíduos são escolhidos aleatoriamente da população e o indivíduo de maior aptidão é selecionado. Esse processo é repetido até formar uma nova população com o mesmo número de indivíduos da população anterior. Além disso, foi implementado o elitismo, onde o melhor indivíduo de uma geração é copiado diretamente para a geração seguinte.

Função de Aptidão

A função de aptidão dos AGs foi o erro quadrático médio (Mean Squared Error, MSE) do conjunto de validação, após o treinamento realizado pelo módulo TR.

d) Módulo TR

O treinamento é realizado com o algoritmo de Levenberg-Marquardt a partir de pesos aleatórios, definidos dentro do intervalo uniforme de [-0.5; 0.5]

Divisão e Transformação dos Dados

Quando uma série temporal é recebida pelo módulo TR, ela é padronizada, ou seja, diminuída pela média e dividida pelo desvio padrão da série, e em seguida, é dividida igualmente nos conjuntos de treinamento, validação e teste.

Critérios de Parada

Os critérios de parada do treinamento usados foram os mesmos sugeridos em [Pretchel 1994]. O treinamento foi feito com validação cruzada e os seguintes critérios de parada foram usados:

i) GL_{10} (*Generalization Loss*): o treinamento para quando o erro de validação atual ultrapassa em 10% o valor do menor erro de validação, o que indica que houve uma perda de 10% no desempenho de generalização.

ii) Número máximo de 500 ciclos de treinamento.

iii) P_5 (*Progress*): a cada 5 ciclos de treinamento é verificado se o erro sobre o conjunto de treinamento diminuiu significativamente. Essa verificação é feita conforme a equação 5.4, onde o parâmetro $E_{tr}(t)$ corresponde ao erro sobre o conjunto de treinamento no t-ésimo ciclo de treinamento. Se nesse ciclo, o valor de $P_5(t)$ for menor que 0.1, então o processo de treinamento é interrompido.

$$P_5(t) = 1000 \cdot \left(\frac{\sum_{i \in 1,\ldots,5} E_{tr}(t-i+1)}{5 \cdot \min_{i \in 1,\ldots,5} E_{tr}(t-i+1)} - 1 \right) \tag{5.4}$$

5.2.3 Experimentos

Para avaliar o desempenho de nosso protótipo, extraímos três séries temporais da literatura (ver Apêndice 1) e definimos as redes neurais para cada uma delas usando dois procedimentos: (1) usando o protótipo implementado, com os AGs inicializados a partir dos casos; e (2) usando apenas o módulo AG com inicialização aleatória. Desta forma, visamos avaliar a contribuição das soluções recuperadas da base para auxiliar a busca nos

Algoritmos Genéticos. Os parâmetros dos AGs (tamanho da população, número de gerações e taxa de mutação) e o espaço de busca explorado (intervalos de inicialização de cada parâmetro otimizado) foram os mesmos usados durante a criação da base inicial de casos (ver seção 5.2.1). Cada um dos dois procedimentos foi executados 5 vezes com o mesmo número de gerações e a média dos erros obtidos para os três conjuntos de dados estão apresentados nas tabelas 5.1, 5.2 e 5.3.

	Média MSE Treinamento	Média MSE Validação	Média MSE Teste
RBC-AG	126131,15	186623,70	149631,48
AG	104408,96	198346,50	160808,85
Ganho%	- 20,80	+ 5,91	+ 6,95

Tabela 5.1: Média dos Erros – Série temporal 1

	Média MSE Treinamento	Média MSE Validação	Média MSE Teste
RBC-AG	14102,08	41050,44	85318,92
AG	11508,00	42436,19	96287,59
Ganho%	- 22,54	+ 3,27	+ 11,39

Tabela 5.2: Média dos Erros – Série temporal 2

	Média MSE Treinamento	Média MSE Validação	Média MSE Teste
RBC-AG	1355,28	1559,11	1359,75
AG	1474,61	1686,90	1411,46
Ganho%	+ 8,09	+ 7,58	+ 3,66

Tabela 5.3: Média dos Erros – Série temporal 3

Para as três séries de teste, observamos que o uso da base de casos trouxe um ganho nos erros de validação, que foi em média de 5,91%, 3,27% e 7,58% para as séries 1, 2 e 3 respectivamente. O bom desempenho de generalização foi confirmado quando analisamos os erros obtidos para os conjuntos de teste. O ganho médio obtido no conjunto de teste para as séries 1, 2 e 3 foram respectivamente de 6,95%, 11,39% e 3,66 %.

	RBC-AG	AG	Ganho %
Série 1	6,2	11,2	44,64
Série 2	13,6	21,0	35,24
Série 3	5,0	30,2	83,44

Tabela 5.4: Média do número de conexões.

Outro ponto de destaque diz respeito ao número de conexões. Na tabela 5.4, podemos observar a média do número de conexões das redes obtidas pelos dois procedimentos para as três séries de teste. O uso da inicialização com os casos proporcionou a geração de redes com um menor número de conexões, com um ganho médio de 44,64%, 35,24% e 83,44% para as séries 1, 2 e 3 respectivamente. Essa é uma característica desejada uma vez que redes com poucas conexões são menos sensíveis a *overffiting* dos dados e gastam menos recursos computacionais. Um ponto a ressaltar aqui é que esse conhecimento foi implicitamente adquirido e armazenado na base de casos, durante a etapa de aquisição.

Embora sejam resultados encorajadores, eles são ainda preliminares. O sistema foi implementado apenas para problemas univariados, onde cada série foi prevista usando apenas informações da própria série (valores passados e valores passados dos erros de previsão). A medida de similaridade usou apenas as autocorrelações, entretanto, outras características podem ser usadas como, por exemplo, o tamanho e a variância da série. Cada característica pode ter uma relevância diferente dentro do problema. Assim, um ponto a investigar no futuro, além da definição de novas características, é a definição do peso de cada uma delas.

Além disso, o sistema implementado focou em apenas uma parte do problema que é a definição da arquitetura. Todas as outras definições do projeto como o tipo de algoritmo de aprendizado utilizado, as taxas de aprendizado, as transformações sobre os dados, dentre outras, não foram abordadas ainda.

5.3 Conclusões

Neste capítulo, propomos o uso da metodologia de Raciocínio Baseado em Casos e dos Algoritmos Genéticos para automatizar o projeto de redes neurais. O modelo proposto parte do ponto de vista de que o projeto de redes neurais é um problema de busca auxiliada com o uso de conhecimento. Nesse modelo, os Algoritmos Genéticos são usados para a

tarefa de adaptação dos casos recuperados da base, que são inseridos na população inicial dos AGs.

O modelo proposto foi aplicado a um estudo de caso no domínio de previsão de séries temporais. Usamos o modelo para otimizar parâmetros da arquitetura de redes NARMAX para a previsão de séries temporais. Nesse estudo de caso, montamos uma base com 47 casos que foi usada para definir a arquitetura das redes para a previsão de três séries extraídas da literatura. Os resultados obtidos, ainda que preliminares, são encorajadores. Nas três séries testadas, observamos um ganho de desempenho quando a base de casos foi utilizada para inicializar os AGs em comparação com a inicialização aleatória. Esses resultados foram apresentados na *8th International Conference on Neural Information Processing*, que aconteceu em novembro de 2001 em Shangai, China [Prudêncio, Ludermir 2001b]. Além do bom desempenho de generalização, observamos que o modelo proposto gerou redes com um número menor de conexões, o que pode ser uma característica vantajosa em várias aplicações, por exemplo, naquelas que necessitam de implementar as redes neurais em dispositivos de *hardware*.

Vários trabalhos futuros podem ser sugeridos a partir desse trabalho. Primeiramente, no estudo de caso abordado, vários pontos podem ser melhorados como, por exemplo, a escolha dos atributos das séries, a medida de similaridade e os parâmetros dos Algoritmos Genéticos. Além disso, o modelo pode ser estendido para outros parâmetros e modelos de redes neurais, e não apenas para a arquitetura de redes NARMAX. O modelo ainda pode ser aplicado em outros problemas resolvidos por redes neurais como em problemas de classificação.

No próximo capítulo, apresentamos uma outra aplicação de sistemas híbridos para a previsão de séries temporais, onde os Algoritmos Genéticos foram usados para auxiliar o aprendizado dos pesos de uma rede neural para a previsão de vazões.

6 Treinamento Híbrido de Redes Neurais

Nesse capítulo, apresentamos a aplicação de um sistema neural híbrido em um problema do mundo real. O problema abordado foi o de previsão de vazões e o sistema híbrido integra os Algoritmos Genéticos durante o processo de treinamento das redes. No sistema híbrido implementado, os AGs serão responsáveis por definir os pesos iniciais da rede durante o treinamento com o algoritmo de Levenberg-Marquardt (LM). Nesse capítulo, realizamos vários experimentos para verificar dois pontos: (1) a eficácia do algoritmo de LM para o problema abordado e (2), as vantagens e desvantagens do uso do treinamento híbrido comparado ao treinamento puro de redes neurais.

Esse capítulo se desenvolve como se segue. Na seção 6.1, fazemos uma breve apresentação sobre o problema abordado. Em seguida, na seção 6.2, apresentamos os resultados dos experimentos realizados com o treinamento puro de redes neurais comparando o algoritmo de Levenberg-Marquardt e o algoritmo de Backpropagation. Na seção 6.3, apresentamos o treinamento híbrido de redes neurais, integrando os AGs e o algoritmo de Levenberg Marquardt, e finalmente, na seção 6.4, concluímos o capítulo.

6.1 Descrição do Problema

Nesse capítulo, estamos interessados no problema do treinamento de uma rede neural para um problema de previsão de vazões. Os dados disponíveis são referentes às vazões médias mensais da bacia de Guarapiranga entre o período de janeiro de 1963 a dezembro de 1986, correspondendo a um total de 300 vazões. No Apêndice 2, apresentamos com mais detalhes o problema da previsão de vazões, assim como os dados de vazões usados nesse capítulo.

A rede usada para a modelagem dos dados foi uma rede NARX [Sjogerg et al. 1994] (ver seção 2.2) com uma janela de tempo de tamanho 12 (devido a sazonalidade anual da série) e com 5 neurônios na camada oculta.

6.2 Treinamento com Levenberg-Marquardt

Nessa seção, apresentamos os resultados do treinamento da rede neural apresentada na seção anterior usando o algoritmo de Levenberg-Marquadt ([Levenberg 1944] [Marquardt 1963]). O algoritmo de Levenberg-Marquardt (LM) é um algoritmo de otimização de parâmetros usado especificamente para a minimização da função de soma dos erros quadrados. O algoritmo de LM é um algoritmo de segunda ordem, ou seja, que utiliza informações sobre a derivada segunda da função para a atualização dos pesos. O algoritmo de LM é, em geral, bem mais rápido que o algoritmo de *Backpropagation*, entretanto ele pode não funcionar muito bem com redes com muitos parâmetros. Isso decorre do fato de que a memória exigida por esse algoritmo é proporcional ao quadrado do número de pesos da rede [Sarle 2001]. A eficácia desse algoritmo para o nosso problema foi realmente verificada através de alguns experimentos comparando-o com o algoritmo de Backpropagation. A última etapa de experimentos com o algoritmo de LM avalia o procedimento de reinicialização aleatória dos pesos (esses experimentos serão comparados posteriormente com a inicialização realizada com os AGs).

Na seção 6.2.1, descrevemos como foi realizado o processo de treinamento, e na seção 6.2.2, apresentamos os experimentos realizados.

6.2.1 Processo de Treinamento

A seguir descrevemos os principais pontos do processo de treinamento da rede neural: a inicialização dos pesos, a divisão e transformação dos dados, os critérios de parada e a forma de avaliação.

a) Inicialização dos Pesos

Os pesos foram inicializados aleatoriamente dentre do intervalo [-0.5; 0.5] com distribuição uniforme. Esse intervalo de inicialização já foi sugerido em outros trabalhos como, por exemplo, [Fogel et al. 1990]. A inicialização dos pesos define o espaço de busca do algoritmo de aprendizado.

b) Divisão e Transformação dos Dados

Assim como as séries usadas no capítulo anterior, a série de vazões foi dividida igualmente nos conjuntos de treinamento, validação e teste: os primeiros 100 valores compõem o conjunto de treinamento, os próximos 100 valores, o conjunto de validação, e os restantes 100 valores foram usados como conjunto de teste. Os dados também foram padronizados, ou seja, diminuídos da média e divididos pelo desvio padrão.

c) Critérios de Parada

Os critérios de parada do treinamento usados foram os mesmos usados no capítulo anterior (ver seção 5.2.2):

(i) GL_{10}: medindo a perda de generalização

(ii) Número máximo de 500 iterações.

(iii) P_5: medindo o progresso de treinamento.

d) Avaliação do Treinamento

O processo de treinamento foi avaliado pelo erro MSE (Mean Squared Error) sobre conjunto de validação. Esse erro dá uma estimativa do desempenho de generalização da rede treinada. Assim, estima-se que quanto menor o erro sobre o conjunto de validação, melhor será o desempenho da rede para outros conjuntos de teste.

6.2.2 Experimentos

Na primeira etapa de experimentos, realizamos testes comparando o algoritmo de Levenberg-Marquardt ao algoritmo de *Backpropagation*. Ambos os algoritmos estão implementados em Matlab 5.0 [Mathworks 1996]. O algoritmo LM foi implementado pelo *Toolbox NNSYSID* (*Neural Network System Identification*) [Norgaard 1991] e o algoritmo Backpropagation foi implementado pelo *Neural Network Toolbox* [Demuth, Beale 1993].

Os primeiros testes nos levaram a escolher o algoritmo de LM para uma segunda etapa de experimentos que avaliou o processo de reinicialização aleatória dos pesos.

Os experimentos apresentados nessa seção, assim como todos os experimentos desse capítulo, foram realizados em um computador Pentium com 30Mb de RAM.

a) Experimentos – I Etapa

Inicialmente, realizamos experimentos para determinar o desempenho do algoritmo de LM para diferentes taxas de aprendizado. O algoritmo LM foi executado 50 vezes para cada taxa utilizada. A média do erro MSE (Mean Squared Error) para cada conjunto de dados, além do tempo total de execução estão apresentados na tabela 6.1. Na tabela 6.2, temos as melhores redes, ou seja, as de menor erro de validação, para cada taxa de aprendizado utilizada.

Taxa	Média MSE Treinamento	Média MSE Validação	Média MSE Teste	Tempo total de execução
25,0	22,6291	39,6155	36,8114	8 min
20,0	23,1553	40,3117	36,1047	4 min
10,0	**22,4772**	**39,4333**	**36,1513**	**2,2 min**
5,0	21,7521	40,7823	37,3277	3,5 min
2,5	21,9538	41,5193	38,0547	2,5 min
2,0	21,2383	41,7065	37,9741	2 min
1,5	22,3953	42,1890	38,6376	2 min
1,0	21,1240	42,8008	39,3120	3 min
0,5	26,8521	44,1764	42,8448	2 min
0,1	31,2238	45,7083	45,3388	2 min

Tabela 6.1: Resultados do Alg. LM com diferentes taxas de aprendizagem

Taxa	MSE Treinamento	MSE Validação	MSE Teste
25,0	22,6884	35,5392	36,3001
20,0	22,9924	36,6465	37,8672
10,0	**25,1568**	**35,1928**	**35,7513**
5,0	23,7375	37,1106	36,9741
2,5	22,8015	37,4982	38,3464
2,0	21,3190	37,8354	35,6375
1,5	23,2399	37,7696	36,2919
1,0	20,9321	38,2827	37,0765
0,5	23,7488	37,6700	35,1218
0,1	24,8910	38,3209	39,0354

Tabela 6.2: Treinamento com menor erro de validação – algoritmo LM

Podemos ver que o desempenho do algoritmo varia conforme a taxa de aprendizado, que pode tornar o aprendizado mais, ou menos sensível a cair em mínimos locais. De fato, em todas as taxas existe uma diferença entre as médias dos erros e os melhores erros obtidos, contudo em algumas taxas, como na taxa de 0,1, essa diferença é maior (16% sobre o conjunto de validação). Nesses experimentos, o algoritmo com taxa com 10,0 obteve os melhores resultados para o conjunto de validação, mas um resultado intermediário no tempo total de execução.

Realizamos alguns experimentos com o algoritmo de Backpropagation (BP) para comparar o seu desempenho com o algoritmo de LM. Assim como no algoritmo de LM, variamos a taxa de aprendizado do algoritmo de BP, com 50 execuções para cada uma delas. Os valores da média do erro para cada conjunto de dados, além do tempo total de execução estão resumidos na tabela 6.3. Na tabela 6.4, temos as redes com menor erro de validação para cada taxa.

Taxa	Média MSE Treinamento	Média MSE Validação	Média MSE Teste	Tempo total de execução
0,005	23,4123	40,4222	36,2359	5,3min
0,01	23,7701	39,9404	36,4886	8,3min
0,02	**26,1521**	**39,8326**	**37,7448**	**5min**
0,05	30,9618	45,5501	44,2277	4min
0,10	42,3896	53,1191	54,3303	3,5min
0,15	46,6309	56,6464	56,4171	3,5min
0,20	49,0627	57,8601	57,6891	4min
0,25	53,1993	60,5063	61,7320	2,5min

Tabela 6.3: Resultados do Alg. BP com diferentes taxas de aprendizagem

Taxa	MSE Treinamento	MSE Validação	MSE Teste
0,005	24,6338	36,1174	34,5771
0,01	22,2077	36,3770	35,1571
0,02	**22,7582**	**34,7209**	**35,1771**
0,05	21,7997	39,0294	35,1999
0,10	37,0365	41,7615	40,8156
0,15	39,5921	43,0565	45,6957
0,20	35,5161	43,8345	43,2688
0,25	40,3668	44,7087	47,7539

Tabela 6.4: Treinamento com menor erro de validação – algoritmo BP

Os algoritmos de LM e BP se mostraram equivalentes para o problema abordado. A melhor taxa de aprendizado do algoritmo BP (0,02) apresentou um erro médio no conjunto de validação de 39,8326 que é muito próximo com o valor obtido com a melhor taxa do algoritmo de LM (10,0) que foi 39,4333. O comportamento da melhor rede obtida por cada algoritmo também fora semelhantes. Enquanto a melhor rede gerada pelo algoritmo BP obteve um erro de validação de 34,72 o algoritmo de LM obteve um erro de 35,19, que são valores próximos. O maior diferencial em favor do algoritmo de LM está no tempo de treinamento. Podemos ver que, em geral, o tempo de execução desse algoritmo é menor que o BP. Comparando os algoritmos em suas melhores taxas, vemos que o tempo de execução do algoritmo LM foi em torno de duas vezes menor que o algoritmo BP.

A partir desses experimentos, concluímos que o algoritmo LM é perfeitamente aplicável ao nosso problema, com a vantagem de ser bem mais eficiente em termos do tempo de execução.

b) Experimentos – II Etapa

Para diminuir os efeitos do problema de mínimos locais, um procedimento comum durante o projeto de uma rede neural é executar o algoritmo de treinamento um certo número vezes com pesos iniciais aleatórios e, em seguida, selecionar dentre as várias redes treinadas a que obteve o melhor desempenho para o problema. Na segunda etapa de experimentos com o algoritmo de LM, simulamos esse procedimento com três números de reinícios aleatórios de pesos: 50, 100 e 200. Esses três procedimentos aleatórios são chamados daqui para frente de: *Aleatório(50)*, *Aleatório(100)* e *Aleatório(200)*. O procedimento *Aleatório(50)* executa o algoritmo LM com 50 pesos iniciais aleatórios e retorna a rede com menor erro de validação dentre as 50 redes geradas. Os procedimentos *Aleatório(100)* e *Aleatório(200)* trabalham de forma análoga, entretanto, com o número de 100 e 200 reinícios de pesos, respectivamente.

Cada um dos três procedimentos aleatórios foi testado com as três taxas de aprendizado usadas na primeira etapa de testes: a taxa de 10,0, que obteve em média os menores erros de validação, a taxa de 0,1, que obteve o pior desempenho, e a taxa de 2,0 que foi um valor com desempenho intermediário. Os três procedimentos aleatórios foram

executados 10 vezes para cada taxa de aprendizado. Os valores da média do erro para cada conjunto de dados e o tempo total de execução, obtidos pelas taxas 0,1, 2,0 e 10,0 estão resumidos nas tabelas 6.5, 6.6 e 6.7 respectivamente.

Taxa=10,0	Média MSE Treinamento	Média MSE Validação	Média MSE Teste	Tempo total de execução
Aleatório(50)	22,7077	35,8479	34,9855	30 min
Aleatório(100)	21,9447	35,4148	34,4138	68 min
Aleatório(200)	22,7092	35,5149	35,6331	120 min

Tabela 6.5: Procedimentos aleatórios com a taxa de 10,0

Taxa=2,0	Média MSE Treinamento	Média MSE Validação	Média MSE Teste	Tempo total de execução
Aleatório(50)	22.2724	37,3105	36,2031	23 min
Aleatório(100)	22.6491	36,7750	36,3558	47 min
Aleatório(200)	22.8958	36,5335	37,5623	94 min

Tabela 6.6: Procedimentos aleatórios com a taxa de 2,0

Taxa=0,1	Média MSE Treinamento	Média MSE Validação	Média MSE Teste	Tempo total de execução
Aleatório(50)	21,7584	38,2040	42,9316	19 min
Aleatório(100)	22,7988	37,8256	42,7544	38 min
Aleatório(200)	23,7467	37,5871	40,9037	76 min

Tabela 6.7: Procedimentos aleatórios com a taxa de 0,1

Apesar de que com a taxa de aprendizado de 10,0 os erros obtidos tenham sido menores que com as outras taxas, o tempo total de execução foi o maior observado. Já a taxa de 0,1 obteve os piores erros, contudo com um custo de tempo de execução bem mais baixo. Com a taxa de 2,0, o algoritmo teve desempenho intermediário tanto com os erros obtidos como no tempo de execução.

6.3 Treinamento Híbrido – AG e LM

Nos experimentos da seção anterior, vimos que para se obter redes com menores erros foi preciso usar uma taxa de aprendizado que diminuiu o desempenho do treinamento em relação ao tempo de execução. Nessa seção, investigamos uma alternativa para otimizar

o treinamento usando os Algoritmos Genéticos para iniciar os pesos da rede em vez de reiniciar aleatoriamente.

Cada indivíduo dos algoritmos genéticos armazena uma configuração de pesos que servem como ponto inicial para o treinamento com o algoritmo de LM. A cada geração, o algoritmo LM é executado sobre cada indivíduo e a função de aptidão é obtida com base no erro do conjunto de validação. Em seguida, os pesos treinados são evoluídos e servirão como pesos iniciais para treinamento na geração seguinte. Esse processo é repetido até atingir um critério de parada pré-definido. Nessa forma de treinamento híbrido, as configurações de pesos que atingiram os melhores mínimos locais em uma geração são selecionadas, e perturbadas pelos operadores genéticos. Essa perturbação poderá permitir que o algoritmo, na próxima geração, saia da posição de mínimo local atingida na geração anterior. Desta forma, um treinamento em uma geração é um refinamento do treinamento da geração seguinte.

Na seção 6.3.1, apresentamos os detalhes de implementação dos Algoritmos Genéticos e, na seção 6.3.2, discutimos os experimentos com o algoritmo híbrido.

6.3.1 Implementação dos AGs

A seguir apresentaremos os detalhes de implementação dos pontos mais importantes dos Algoritmos Genéticos.

a) Representação dos Pesos

Os pesos foram codificados usando a representação real, apresentada na seção 3.2.1. Assim, o cromossomo representativo da rede é composto pela concatenação dos valores reais dos pesos da rede. Na primeira geração, os pesos foram definidos aleatoriamente dentre do intervalo [-0.5; 0.5] com distribuição uniforme. Essa inicialização de pesos foi a mesma usada durante os experimentos com o treinamento puro. Desta foram, tanto o treinamento puro, como o treinamento híbrido, estão pesquisando o mesmo espaço de busca. Na figura 6.1, temos um exemplo de um cromossomo dessa representação.

b) Operadores Genéticos

Definimos apenas um operador de mutação Gaussiana, onde um número tirado de uma distribuição Normal (0, σ^2) é adicionado ao valor corrente do peso. O valor de σ deve ser pequeno o suficiente para não causa instabilidade no algoritmo. O valor usado nos testes foi de 0.3, que é um valor menor que os limites do intervalo de inicialização. O valor da taxa de mutação foi de 0.1. Esses valores foram definidos experimentalmente, através de uma etapa prévia de experimentos com AGs usados isoladamente.

Devido ao possível problema do efeito destrutivo do *crossover*, citado na seção 3.2.2, preferimos não implementar esse tipo de operador.

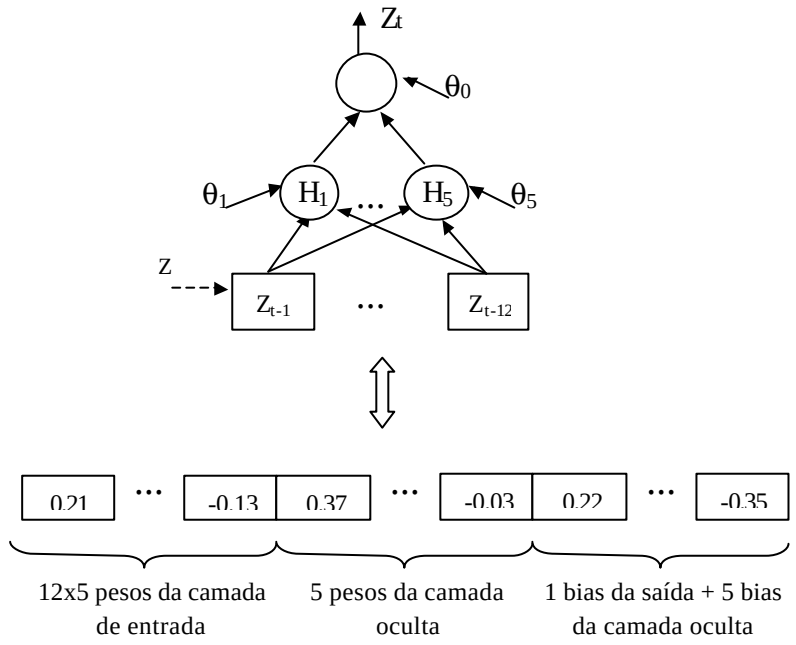

Figura 6.1: Cromossomo que armazena configuração dos pesos

c) Função de Aptidão

A função de aptidão dos AGs é a mesma função utilizada para avaliar o desempenho do treinamento nos experimentos com o algoritmo de LM, o erro MSE (*Mean Squared Error*) do conjunto de validação.

d) Critério de Parada

O critério de para escolhido foi o número máximo de gerações. Desta forma, a cada execução do algoritmo híbrido, o número de treinamentos efetuados é igual ao número de indivíduos por geração multiplicado pelo número de gerações.

e) Seleção

Na seção 5.2.2, mencionamos que a seleção proporcional à aptidão pode causar convergência prematura [Lacerda, Carvalho 1999]. Por esse motivo a seleção implementado foi a seleção por torneio de 3 com elitismo.

6.3.2 Experimentos

Os testes dessa seção foram realizados para comparar os procedimentos de inicialização aleatória e o de inicialização com Algoritmos Genéticos. Os experimentos com o reinício aleatório foram apresentados na seção 6.2.2. Naquela seção, foram usados três números diferentes de inicializações, 50, 100 e 200, com três taxas de aprendizado, 0,1, 2,0 e 10,0. Os três procedimentos aleatórios, *Aleatório(50)*, *Aleatório(100)* e *Aleatório(200),* foram executados 10 vezes para cada taxa de aprendizado.

Para fazer uma comparação justa, realizamos testes com os Algoritmos Genéticos, com os mesmos valores de taxas de aprendizado e número de reinícios empregados anteriormente. Para o número de 50 reinícios, os AGs foram executados com o tamanho de população igual a 5 e um número de 10 gerações, o que chamamos de procedimento *AG(5x10).* Para 100 reinícios, usamos 10 como tamanho da população e 10 como número de gerações, chamando de *AG(10x10)*. E finalmente para 200 reinícios, usamos a população de tamanho 10 com o número de 20 gerações, o que chamamos de *AG(10x20)*. Os procedimentos *AG(5x10)*, *AG(10x10)* e *AG(10x20)* foram executados 10 vezes e os resultados médios dos erros de treinamento, validação e teste, além do tempo total de execução, foram comparados, respectivamente, com os resultados obtidos pelos procedimentos *Aleatório(50)*, *Aleatório(100)* e *Aleatório(200)*. Esses resultados para cada taxa de aprendizado utilizada estão apresentados nas tabelas 6.8, 6.9 e 6.10.

Taxa = 10,0	Média MSE Treinamento	Média MSE Validação	Média MSE Teste	Tempo total de execução
Aleatório(50)	22,7077	35,8479	34,9855	30 min
Aleatório(100)	21,9447	35,4148	34,4138	68 min
Aleatório(200)	22,7092	35,5149	35,6331	120 min
AG(5x10)	21,4619	35,6716	36,2181	13 min
AG(10x10)	21,9556	34,6803	36,1797	24 min
AG(10x20)	22,9866	34,0481	36,8073	47 min

Tabela 6.8: Procedimento aleatório versus AGs com taxa de aprendizado de 10,0

Taxa = 2,0	Média MSE Treinamento	Média MSE Validação	Média MSE Teste	Tempo total de execução
Aleatório(50)	22,2724	37,3105	36,2031	23 min
Aleatório(100)	22,6491	36,7750	36,3558	47 min
Aleatório(200)	22,8958	36,5335	37,5623	94 min
AG(5x10)	24,1478	36,4207	39,0691	12 min
AG(10x10)	24,6531	35,3425	38,1270	23 min
AG(10x20)	22,6433	34,1983	40,6142	49 min

Tabela 6.9: Procedimento aleatório versus AGs com taxa de aprendizado de 2,0

Taxa = 0,1	Média MSE Treinamento	Média MSE Validação	Média MSE Teste	Tempo total de execução
Aleatório(50)	21,7584	38,2040	42,9316	19 min
Aleatório(100)	22,7988	37,8256	42,7544	38 min
Aleatório(200)	23,7467	37,5871	40,9037	76 min
AG(5x10)	22,7350	36,4482	40,5466	12 min
AG(10x10)	22,3057	35,5361	40,8520	23 min
AG(10x20)	23,3401	35,2380	40,4153	45 min

Tabela 6.10: Procedimento aleatório versus AGs com taxa de aprendizado de 0,1

Para todas as combinações de taxa de aprendizado e número de reinícios o algoritmo híbrido obteve menor erro para o conjunto de validação e, portanto, com melhor avaliação. Entretanto, verificamos que as redes geradas com o algoritmo híbrido obtiveram erro menor para o conjunto de teste apenas com a taxa de 0,1. Nessa taxa o algoritmo de LM havia se apresentado mais sensível ao problema de mínimos locais. Nas outras taxas, apesar de sempre gerarem redes com melhor avaliação, o algoritmo híbrido se mostrou inferior para o conjunto de teste. Isso pode sugerir que a forma de avaliar a rede, baseando-

se apenas no erro do conjunto de validação, não foi uma boa escolha ou que os conjuntos de dados não foram bem selecionados ou processados. Mas o fato importante aqui é que o algoritmo híbrido foi capaz de gerar as melhores redes conforme a medida de desempenho pré-estabelecida.

Outro fato observado em todas as combinações é que o algoritmo híbrido foi bem mais eficiente em termos de tempo de execução. As combinações que tornaram o algoritmo híbrido menos eficiente em termos de tempo, em relação ao aleatório, foram as que usaram a taxa de 10. Contudo, mesmo usando a taxa mais lenta, o algoritmo híbrido foi mais rápido que o algoritmo puro quando usou taxas mais rápidas. O bom desempenho do algoritmo híbrido em relação ao tempo de execução se deve ao fato de que a cada geração, o algoritmo de LM apenas aprofunda o treinamento já feito na geração anterior, aproveitando o caminho que já foi percorrido em buscas anteriores.

6.4 Conclusões

Nesse capítulo, apresentamos um problema de previsão de vazões modelado com uma rede NARX. Para o treinamento dessas redes, investigamos o uso de um algoritmo de aprendizado de pesos híbrido onde os Algoritmos Genéticos definem os pesos iniciais da rede para posterior execução do algoritmo de Levenberg-Marquardt. Nos experimentos realizados, o algoritmo híbrido definiu redes com menor erro de validação. O erro de validação foi usado como medida de avaliação do treinamento, assim podemos afirmar que o algoritmo híbrido foi melhor dentro do objetivo do problema. Além disso, para este problema em particular, o algoritmo híbrido se mostrou bem mais eficiente em termos de tempo de execução.

O problema do algoritmo híbrido é que um número maior de parâmetros deve ser definido. Além da taxa de aprendizado do algoritmo de Levenberg-Marquardt, os Algoritmos Genéticos contêm parâmetros que devem ser definidos adequadamente. Um dos trabalhos futuros é definir esses parâmetros de forma automática. Outros pontos a investigar no futuro é o uso de outras medidas de avaliação das redes, o uso do algoritmo híbrido em outras topologias e o estudo do impacto dos operadores de crossover.

7 Conclusões

Nesse trabalho, investigamos o uso de técnicas de Inteligência Artificial para automatizar o projeto de Redes Neurais Artificiais. Os sistemas gerados a partir dessa integração são sistemas neurais híbridos onde uma técnica de IA implementa uma etapa ou tarefas do projeto das redes. Durante a dissertação, apresentamos várias formas de integrar técnicas de IA ao projeto das RNAs, e além disso, realizamos duas aplicações para o projeto das redes em problemas de previsão de séries temporais. Na primeira aplicação, definimos a arquitetura de redes NARMAX para a previsão de séries temporais usando o Raciocínio Baseado em Casos e os Algoritmos Genéticos. Os resultados obtidos com o modelo proposto foram promissores. Na segunda aplicação, usamos os Algoritmos Genéticos para o treinamento de uma rede NARX para um problema de previsão mais específico, a previsão de vazões. O algoritmo de treinamento híbrido, que integrou os AGs ao algoritmo de LM, apresentou bons resultados quando comparado com o treinamento convencional.

A seguir apresentamos um resumo das principais contribuições do nosso trabalho (seção 7.1), e alguns trabalhos futuros (seção 7.2). As considerações finais serão feitas na seção 7.3.

7.1 Resumo das Contribuições

A seguir apresentamos um resumo das principais contribuições do nosso trabalho durante o mestrado.

- Resumo das técnicas de IA usadas para o projeto automático de RNAs: Durante a nossa dissertação, apresentamos várias técnicas de IA que foram usadas no projeto das RNAs. O capítulo 3, que apresentou o uso dos Algoritmos Genéticos, servirá como uma nova fonte, em português, sobre esse tema. No capítulo 4, apresentamos estratégias baseadas em conhecimento para o projeto das RNAs. Esse capítulo foi montado sobre o

esquema definido em [Hilário et al. 1996], contudo, na nossa dissertação, esse foi dado um enfoque no uso das técnicas de IA e em sistemas híbridos.

- Uma nova abordagem para o projeto de redes neurais: na nossa dissertação, propomos um modelo de automatização híbrido integrando o Raciocínio Baseado em Casos e os Algoritmos Genéticos. Essas duas técnicas já foram combinadas para resolver outros problemas, contudo, ainda não vimos nenhum trabalho integrando as duas técnicas para o projeto de RNAs. Além disso, encontramos poucas referências do uso de RBC no projeto das redes. Desta forma, essa dissertação servirá como uma fonte de consulta adicional sobre essa integração.

- Montagem da base de casos: o modelo proposto no capítulo 5 foi usado para definir a arquitetura de redes neurais para a previsão de séries temporais. Nessa aplicação, foi montada uma base de casos associando 50 séries temporais, conhecidas no campo de previsão, a arquiteturas de redes NARMAX. Essa base poderá ser usada no futuro para sugerir a arquitetura dessas redes para novas séries temporais.

- Integração dos Algoritmos Genéticos ao Algoritmo de Levenberg-Marquardt: os AGs já foram integrados em outros trabalhos com algoritmos de treinamento tradicionais, especialmente o algoritmo de Backpropagation. No nosso trabalho, usamos um algoritmo menos tradicional que foi o Levenberg-Marquardt. Para o nosso problema esse algoritmo foi mais eficiente que o Backpropagation.

7.2 Limitações e Trabalhos Futuros

O modelo proposto no capítulo 5 foi usado para definir apenas a arquitetura de um tipo de rede neural. Entretanto, outros parâmetros do projeto poderiam ser armazenados nos casos, estendendo também a outros modelos de redes neurais. Outro ponto a investigar no futuro é o uso de novos atributos para medir a similaridade dos casos. Na medida de similaridade proposta, apenas as autocorrelações da série foram usadas. Entretanto, outras características como o tamanho da série e a variância podem ser relevantes. O modelo proposto tem uma característica teórica que ainda não foi investigada adequadamente: a

capacidade de aprendizado. Um trabalho para o futuro é saber como o modelo se comporta à medida que novos casos são inseridos na base. Para fazer essa investigação, será preciso um número maior de casos. O modelo proposto também poderá ser usado em outros tipos de problema como, por exemplo, em problemas de classificação.

Uma das desvantagens dos sistemas híbridos é que, em geral, eles são mais complexos do que seus componentes. De fato, o uso dos AGs no projeto de redes neurais leva ao problema de como definir os parâmetros do Algoritmo como a taxa de mutação, o número de gerações e tamanho da população. No caso do treinamento híbrido além desses parâmetros, o operador de mutação Gaussiana dependia de mais um parâmetro (a variância da distribuição Gaussiana). Pouco enfoque foi dado no nosso trabalho, no sentido de definir de forma automática esses parâmetros. Um dos trabalhos futuros é investigar como esses parâmetros podem ser definidos sem perda de tempo com etapas prévias de experimentos.

7.3 Considerações Finais

O projeto automático de redes neurais, assim como os sistemas neurais híbridos, são dois temas importantes que ainda precisam de um estudo aprofundado. Nessa dissertação, procuramos dar a nossa contribuição nas duas áreas, apresentando os seus pontos de intersecção. Esperamos que essa dissertação possa servir, no futuro, como uma boa fonte de consulta e estudo para pessoas interessadas nas áreas de redes neurais e sistemas híbridos.

Apêndice A: Base de Casos

Neste apêndice, apresentamos os dados armazenados na base de casos do capítulo 5. A base de casos foi montada a partir de 47 séries temporais extraídas da literatura. Em cada caso, apresentamos a origem da série, as transformações usadas para eliminar a tendência e a sazonalidade da série, e a arquitetura usada para prever a série transformada. Observamos que os operadores de diferenciação e diferenciação sazonal foram aplicados na ordem em que estão apresentados nos casos. Além da base de casos, apresentamos também as três séries usadas para teste no capítulo 5, assim como as transformações usadas e a melhor arquitetura para cada uma delas.

(I) Base de Casos

Casos 1 a 7
Fonte das séries: [Box, Jenkins 1970]
==========

Caso 1:
Série original: SERIES A CHEMICAL CONCENTRATION READINGS
Transformações: uma diferenciação
Arquitetura: 01-01-01
==========

Caso 2:
Série original: SERIES B IBM COMMON STOCK CLOSING PRICES: DAILY, 17TH MAY 1961-2ND
Transformações: uma diferenciação
Arquitetura: 08-04-01
==========

Caso 3:
Série original: SERIES B' IBM COMMON STOCK CLOSING PRICES:DAILY,29.6.59-30.6.60
Transformações: uma diferenciação
Arquitetura: 00-11-01
==========

Caso 4:
Série original: SERIES C CHEMICAL PROCESS TEMPERATURE READINGS: EVERY MINUTE
Transformações: duas diferenciações
Arquitetura: 08-00-01
==========

Caso 5:
Série original: SERIES D CHEMICAL PROCESS VISCOSITY READINGS
Transformações: uma diferenciação
Arquitetura: 10-12-01
==========

Caso 6:
Série original: SERIES E WOELFER SUNSPOT NUMBERS:YEARLY (N=100) 1770-1869
Transformações: duas diferenciações

Arquitetura: 05-02-02
==========

Caso 7:
Série original: SERIES G INTERNATIONAL AIRLINE PASSENGERS:MONTHLY TOTALS
Transformações: uma diferenciação e uma diferenciação sazonal
Arquitetura: 12-00-02
==========

Casos 8 a 15
Fonte das séries: [Pankratz 1983]
==========

Caso 8:
Série original: QUARTERLY CHANGE IN BUSINESS INVENTORIES (BILLIONS) 1955-1969
Transformações: uma diferenciação
Arquitetura: 00-06-03
==========

Caso 9:
Série original: PERSONAL SAVING AS % OF DISPOSABLE INCOME 1955-1979
Transformações: uma diferenciação
Arquitetura: 11-06-03
==========

Caso 10:
Série original: MONTHLY BITUMINOUS COAL PRODUCTION IN U.S.A. 1952-1959
Transformações: uma diferenciação
Arquitetura: 03-01-02
==========

Caso 11:
Série original: NEW PRIVATE HOUSING PERMITS (QUARTERLY 1947-1967)
Transformações: duas diferenciações
Arquitetura: 08-00-07
==========

Caso 12:
Série original: QUARTERLY FREIGHT (CLASS I RAILROADS,U.S.A.,BILLIONS TON-MILES) 1965-1978.
Transformações: duas diferenciações
Arquitetura: 04-00-02
==========

Caso 13:
Série original: WEEKLY CLOSING PRICE OF AT&T COMMON SHARES 1979
Transformações: uma diferenciação
Arquitetura: 00-12-01
==========

Caso 14:
Série original: REAL-ESTATE LOANS (BILLIONS) (MONTHLY : JAN.1973-OCT.1967)
Transformações: duas diferenciações
Arquitetura: 00-09-03
==========

Caso 15:
Série original: % TIME THAT PARTS FOR INDUSTRIAL PROJECT AVAILABLE WHEN NEEDED (WEEKLY)
Transformações: uma diferenciação
Arquitetura: 02-01-01
==========

Casos 16 a 27
Fonte das séries: [Anderson 1976]

==========

Caso 16:
Série original: SERIES A DAILY DRY BULB TEMPERATURES (DEG.F) BEN NEVIS FEB.1-AUG.18 1884
Transformações: uma diferenciação
Arquitetura: 02-09-02
==========

Caso 17:
Série original: SERIES B ANNUAL RAINFALL (IN.) NOTTINGHAM CASTLE 1867-1939
Transformações: nenhuma
Arquitetura: 00-06-01
==========

Caso 18:
Série original: SERIES D I.C.I. CLOSING PRICES 25 AUG '72-19 JAN '73 (FINANCIAL TIMES)
Transformações: uma diferenciação
Arquitetura: 09-06-05
==========

Caso 19:
Série 19: SERIES E WOMEN UNEMPLOYED (1000'S) U.K. 1ST OF MONTH JAN.'67-JULY.'72
Transformações: duas diferenciações
Arquitetura: 06-00-01
==========

Caso 20:
Série original: SERIES F SIMULATED SERIES Z(T) = 0.9.Z(T-1) + A(T); A(T)~IN(0,1)
Transformações: uma diferenciação
Arquitetura: 00-05-05
==========

Caso 21:
Série original: SERIES G SIMULATION OF Z(T) = (1 - 0.6B)A(T) ; A(T)~N(0,1)
Transformações: nenhuma
Arquitetura: 06-00-03
==========

Caso 22:
Série original: SERIES H SIMULATED SERIES
Transformações: uma diferenciação
Arquitetura: 06-11-01
==========

Caso 23:
Série original: SERIES J SIMULATED SERIES
Transformações: nenhuma
Arquitetura: 03-00-04
==========

Caso 24:
Série original: SERIES L DOW JONES UTILITY INDEX AUG 28-DEC 18 '72 (WALL STREET JOURNAL)
Transformações: duas diferenciações
Arquitetura: 10-00-05
==========

Caso 25:
Série original: SERIES P PASSENGER MILES (MIL) FLOWN DOMESTIC U.K. JUL.'62-MAY '72
Transformações: uma diferenciação e uma diferenciação sazonal
Arquitetura: 06-00-01
==========

Caso 26:
Série original: SERIES Q MONTHLY SALES OF COMPANY X JAN '65 - MAY '71 C.CHATFIELD
Transformações: uma diferenciação e uma diferenciação sazonal

Arquitetura: 02-12-02
==========
Caso 27:
Série original: SERIES R MEAN MONTHLY AIR TEMPERATURE (DEG.F) NOTTINGHAM CASTLE 1920-1939
Transformações: uma diferenciação sazonal
Arquitetura: 02-08-02
==========

Casos 28 a 33
Fonte das séries: [Abraham, Ledolter 1983]
==========
Caso 28:
Série 28: SERIES 1 QUARTERLY IOWA NONFARM INCOME (1948-1979)
Transformações: duas diferenciações
Arquitetura: 04-05-02
==========
Caso 29:
Série original: SERIES 2 QUARTERLY GROWTH RATES OF IOWA NONFARM INCOME(1948-1979)
Transformações: uma diferenciação
Arquitetura: 06-10-03
==========
Caso 30:
Série original: SERIES 3 MONTHLY AV. RESIDENTIAL ELECTRICITY USAGE IOWA CITY 1971-1979
Transformações: uma diferenciação sazonal
Arquitetura: 01-00-03
==========
Caso 31:
Série 31: SERIES 4 MONTHLY CAR SALES IN QUEBEC 1960-1968
Transformações: uma diferenciação e uma diferenciação sazonal
Arquitetura: 04-03-01
==========
Caso 32:
Série original: SERIES 5 MONTHLY TRAFFIC FATALITIES IN ONTARIO 1960-1974
Transformações: uma diferenciação e uma diferenciação sazonal
Arquitetura: 01-09-01
==========
Caso 33:
Série original: SERIES 6 MONTHLY U.S. HOUSING STARTS (PRIVATELY OWNED 1-FAMILY) 1965-1975
Transformações: uma diferenciação e uma diferenciação sazonal
Arquitetura: 08-00-02
==========

Casos 34 a 46
Fonte das séries: [Montgomery et al. 1990]
==========
Caso 34:
Série original: CHEMICAL PROCESS VISCOSITY DATA
Transformações: nenhuma
Arquitetura: 02-01-03
==========
Caso 35:
Série original: WEEKLY DEMAND FOR A PLASTIC CONTAINER
Transformações: uma diferenciação

Arquitetura: 01-01-04
==========
Caso 36:
Série original: DEMAND FOR A DOUBLE-KNIT POLYESTER FABRIC
Transformações: nenhuma
Arquitetura: 13-09-01
==========
Caso 37:
Série original: WEEKLY SALES OF A CUTTING TOOL
Transformações: u ma diferenciação
Arquitetura: 00-03-04
==========
Caso 38:
Série original: MINUTES OF USAGE PER DAY OF A COMPUTER TERMINAL
Transformações: uma diferenciação
Arquitetura: 03-00-05
==========
Caso 39:
Série original: WEEKLY DEMAND FOR CRANKSHAFT
Transformações: nenhuma
Arquitetura: 08-05-01
==========
Caso 40:
Série original: MONTHLY DEMAND FOR A SPARE PART
Transformações: uma diferenciação
Arquitetura: 10-08-03
==========
Caso 41:
Série original: CHEMICAL PROCESS READINGS EVERY TWO MINUTES
Transformações: nenhuma
Arquitetura: 06-10-01
==========
Caso 42:
Série original: MONTHLY CHAMPAGNE SALES (IN 1000'S)
Transformações: uma diferenciação e uma diferenciação sazonal
Arquitetura: 04-02-01
==========
Caso 43:
Série original: WEEKLY SALES FOR A NOVELTY ITEM (P.37-38 : MONTGOMERY)
Transformações: uma diferenciação
Arquitetura: 08-08-03
==========
Caso 44:
Série original: FOUR WEEKLY DEMAND FOR THERMOSTATS (P.32-33 : MONTGOMERY)
Transformações: nenhuma
Arquitetura: 01-04-04
==========

Casos 45 a 47
Fonte das séries: [O'Donovan 1983]
==========
Caso 45:
Série original: ANNUAL CHANGES IN THE EARTH'S ROTATION, DAY LENGTH (SEC*10**-5) 1821-1970
Transformações: uma diferenciação
Arquitetura: 06-08-01

==========

Caso 46:
Série original: SERIES 5.5 DAILY DRY BULB TEMPERATURES (DEG.F) BEN NEVIS FEB.1-AUG.18 1884
Transformações: uma diferenciação
Arquitetura: 02-02-02

==========

Caso 47:
Série original: SERIES 6.3 MONTHLY EMPLOYEES WHOLES./RETAIL WISCONSIN '61-'75 R.B.MILLER
Transformações: uma diferenciação e uma diferenciação sazonal
Arquitetura: 05-11-02

==========

(II) Séries de Teste
Fonte: [Makridakis, Wheelwright 1989]

==========

Série 1: LENEX CORPORATION SHIPMENT OF RADIOS JAN'67-DEC'71
Transformações: uma diferenciação

==========

Série 2: CFE SPECIALTY WRITING PAPERS MONTHLY SALES
Transformações: uma diferenciação e uma diferenciação sazonal

==========

Série 3: DER STERN WEEKLY SALES OF WHOLESALERS '71-'72
Transformações: uma diferenciação

==========

Apêndice B: Previsão de Vazões

Segundo [Tucci 1993], uma bacia hidrográfica é uma área de captação da água de chuva capaz de transformar o volume de água armazenado em um escoamento regular no tempo. A vazão mede a capacidade de escoamento ou o fluxo de água na bacia. A previsão de vazões de uma bacia pode ser útil para a manutenção de sistemas hidrológicos e para o melhor gerenciamento dos recursos hídricos. A previsão das vazões máximas é útil para o controle de inundações e planejamento de obras. A previsão de vazões mínimas pode ser útil para planejar o abastecimento de água e prever condições críticas. A previsão das vazões médias auxilia o atendimento à demanda de energia, uma vez que essas previsões indicam o potencial de geração de energia no futuro [Valença 1999a].

Uma forma de prever vazões é através de modelos físicos onde o comportamento da bacia e de suas vazões é definido por equações matemáticas que representam e relacionam as características físicas da bacia, como precipitação, evaporação, infiltração dentre outras. Segundo [Valença 1999a], esses modelos, em geral, têm uma calibragem mais complicada e requerem mais experiência na área de recursos hídricos. Como exemplos de modelos físicos podemos citar o modelo SMAP [Lopes et al. 1981] e o modelo MOHTSAR [Viana 1986].

Como visto no capítulo 2, uma forma de se modelar fenômenos temporais sem o uso das características físicas do domínio é através dos modelos caixa-preta [Sjoberg et al. 1994]. Dentre esses modelos, citamos os modelos de Box-Jenkins e as Redes Neurais Artificiais. Aplicações dos modelos de Box-Jenkins para a previsão de séries temporais podem ser encontradas em [Kadowaki et al. 1997], [Valença 1999a] e [Barreto, Andrade 2000]. Dentre os trabalhos que usaram as redes neurais para a previsão de vazões, podemos citar [Kadowaki et al. 1997], [Valença, Ludermir 1998], [Valença 1999a], [Valença 1999b] e [Prudêncio, Ludermir 2001a]. Comparações entre as redes neurais e os modelos de Box-Jenkins para a previsão de vazões em diferentes bacias brasileiras podem ser encontradas em [Kadowaki et al. 1997] e [Valença 1999a].

No capítulo 6, aplicamos o treinamento híbrido de redes neurais para um problema de previsão de vazões. Os dados usados são referentes às vazões médias da bacia de Guarapiranga entre o período de janeiro de 1963 a dezembro de 1986. Esses dados, apresentados a seguir, foram cedidos pelo engenheiro da Chesf (Companhia Hidrelétrica do São Francisco), o Dr. Mêuser Jorge Silva Valença, que usou esses dados anteriormente em [Valença 1999a].

==========

Dados de Vazão: Bacia de Guarapiranga

11.50 17.70 25.90 8.87 6.87 6.37 5.90 8.29 9.80 15.50 9.07 17.10 23.30 16.10 6.50 4.28 3.40 3.67 2.87 3.99
2.47 6.27 8.37 5.19 3.40 10.90 4.78 6.29 6.38 5.67 5.49 3.70 6.29 6.50 7.68 13.00 24.30 16.00 13.70 11.20
13.70 7.18 9.48 6.09 6.60 12.30 12.00 17.60 20.40 18.70 19.50 18.20 13.20 6.10 5.68 7.39 9.88 14.20 12.20
17.20 14.60 29.50 20.10 12.00 6.68 12.50 7.39 5.68 9.49 9.18 17.30 13.10 17.10 9.50 16.60 11.30 9.07 7.10
5.26 6.50 4.98 7.47 4.40 9.07 6.68 8.60 9.89 6.10 4.18 5.98 3.17 3.47 3.47 14.80 25.50 10.30 23.50 37.30
23.00 12.90 8.65 7.77 6.52 7.32 9.77 8.70 6.56 8.89 11.00 11.70 15.10 9.42 7.79 10.40 7.15 5.47 5.52
10.00 6.65 10.20 21.70 17.90 9.69 8.78 4.95 3.14 5.01 6.52 8.03 21.10 10.60 7.35 26.00 18.60 10.20
13.60 8.33 6.31 8.24 6.11 9.13 10.30 11.50 13.10 26.60 8.88 19.60 11.10 5.74 10.80 6.04 3.82 4.64 7.95
7.37 16.20 18.80 21.00 17.50 7.32 6.27 5.06 6.96 4.49 5.00 8.13 11.50 19.70 46.10 39.70 22.60 23.90
18.50 15.00 19.40 15.50 18.40 14.20 12.30 13.60 27.70 11.50 10.60 16.10 9.40 8.20 5.80 4.80 8.20 9.10
8.40 15.90 13.40 14.00 27.80 6.20 9.20 8.40 7.80 4.20 6.10 4.20 17.20 15.40 12.40 10.50 8.20 10.00
8.90 5.20 5.90 7.40 13.40 12.00 14.60 12.60 18.00 19.60 11.90 9.60 4.70 5.90 5.80 5.60 6.00 7.20 7.10
15.50 24.10 8.90 9.90 14.50 8.80 7.10 7.30 4.90 3.70 10.20 10.70 9.80 15.30 20.10 11.40 11.90 6.40
17.10 7.90 6.90 4.30 9.20 11.90 21.50 22.00 36.30 26.90 24.60 21.30 40.30 13.10 10.50 24.50 27.00 19.10
21.90 22.60 13.20 9.50 12.00 10.00 5.20 5.50 9.40 11.80 5.10 5.90 9.40 17.20 20.70 14.60 10.30 7.40
5.70 3.60 3.20 6.20 2.90 5.30 3.70 5.57 23.09 24.40 10.54 10.50 4.80 6.29 6.49 5.46 6.43 8.65 17.36

==========

Referências

[Aadmot, Plaza 1994] A. Aadmot & E. Plaza, Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches, *AI Communications*, Vol. 7, 39-59, 1994.

[Abraham, Ledolter 1983] B. Y. Abraham & J. Ledolter, *Statistical Methods for Forecasting*, John Wiley & Sons, New York, NY, 1983.

[Agre, Koprinska 1996] G. Agre & I. Koprinska, Case-Based Refinement of Knowledge-Based Neural Networks, *Proceedings of the International Conference* "Intelligent Systems: A Semiotic Perspective", Vol. II, pp. 221-226, Gaithersburg, MD, 1996.

[Ajjanagadde, Shastri 1995] V. Ajjanagadde & L. Shastri, Reasoning with Rules and Variables in Neural Networks, *Intelligent Hybrid Systems*, pp. 209-220, John Wiley & Sons, London, 1995.

[Allen 1995] J. Allen, *Natural Language Understanding*, Benjamins/Cummins Publishing Co. Inc., Redwood City, CA, 1995.

[Almendra et al. 1999] C. C. Almendra, I. R. Teixeira & R. B. C. Prudêncio, Descoberta de Conhecimento em Banco de Dados Usando Redes Neurais, *Anais do II Encontro Nacional de Inteligência Artificial* (*II ENIA*), pp. 555-563, Rio de Janeiro, Brasil, 1999.

[Anderson 1976] O.D. Anderson, *Time Series Analysis and Forecasting: the Box-Jenkins Approach*, Butterworth, London, 1976.

[Arabshahi et al. 1992] P. Arabshahi, J. J. Choi, R. J. Marks & T. P. Caudell, Fuzzy Control of Backpropagation, *Proceedings of the First IEEE International Conference on Fuzzy Systems*, pp. 967-972, San Diego, CA, 1992.

[Asakawa, Takagi 1994] K. Asakawa & H. Takagi, Neural Networks in Japan, *Communications of the ACM*, Vol. 37(3), pp.106-112, 1994.

[Aussem et al. 1994] A. Aussem, F. Murtagh & M. Sarazin, Dynamical Recurrent Neural Networks and Pattern Recognition Methods for Time Series Prediction: Application to Seeing and Temperature Forecasting in the Context of ESO's VLT Astronomical Weather Station, *Vistas in Astronomy*, Vol. 38, pp. 357-374, 1994.

[Balakrishnan, Hovavar 1995a] K. Balakrishnan & V. Honavar, Evolutionary Design of Neural Architectures: Preliminary Taxonomy and Guide to Literature, *Technical Report CS TR95-01*, Department of Computer Science, Iowa State University, 1995.

[Balakrishnan, Honavar 1995b] K. Balakrishnan, K. & V. Honavar, Properties of Genetic Representations of Neural Architectures, *Proceedings of the World Congress on Neural Networks*, pp. 807-813, Washington D.C, 1995.

[Barnard, Cole 1989] E. Barnard & R. A. Cole, A Neural-Net Training Program based on Conjugate-Gradient Optimization, *Technical Report CSE 89-014*, Oregon Graduate Institute, Beaverton, OR, 1989.

[Barreto, Andrade 2000] G. A. Barreto & M. G. Andrade, Estimação Paramétrica de Modelos Auto-Regressivos via Estatística Bayesiana e Simulação de Monte Carlo, *Anais do XIII Congresso Brasileiro de Automática* (CBA'2000), pp. 92-97, Florianópolis, Brasil, 2000.

[Battiti 1992] R. Battiti, First and Second-Order Methods for Learning Between Steepest Descent and Newton's Method, *Neural Computation*, Vol. 4, pp. 141-166, 1992.

[Battiti, Tecchiolli 1994] R. Battiti & G. Tecchiolli, Learning with First, Second, and No Derivatives: a Case Study in High Energy Physics, *Neurocomputing*, Vol. 6, pp. 181-206, 1994.

[Belew et al. 1990] R. K. Belew, J. McInerney & N. N. Schraudolph, Evolving Networks: Using the Genetic Algorithm with Connectionist Learning, *Technical Report CS90-174*, University of California, San Diego, CA, 1990.

[Bezdek, Pal 1992] J.C. Bezdek & S.K. Pal, *Fuzzy Models for Pattern Recognition*, IEEE Press, New York, NY, 1992.

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

[Bishop 1995] C. Bishop, *Neural Networks for Pattern Recognition*, Oxford University Press, Oxford, 1995.

[Box, Jenkins 1970] G. E. Box & G. M. Jenkins, *Time Series Analysis: Forecasting and Control*, Holden-Day, San Francisco, CA, 1970.

[Bornholdt, Graudenz 1992] S. Bornholdt & D. Graudenz, General Asymmetric Neural Networks and Structure Design by Genetic Algorithms, *Neural Networks*, Vol. 5, pp. 327-334, 1992.

[Braga et al. 2000] A. P. Braga, A. C. P. L F. Carvalho & T. B. Ludermir, *Redes Neurais Artificiais: Teoria e Aplicações*, LCT, Livros Técnicos e Científicos Editora, Rio de Janeiro, Brasil, 2000.

[Branke 1995] J. Branke, Evolutionary Algorithms for Neural Network Design and Training, *Proceedings 1st Nordic Workshop on Genetic Algorithms and its Applications*, pp. 145-163, Vaasa, Finland, 1995.

[Broomhead, Lowe 1988] D. S. Broomhead & D. Lowe, Multivariable Functional Interpolation and Adaptive Networks, *Complex Systems*, Vol. 2, pp. 321-355, 1988.

[Caprile et al. 1991] B. Caprile, F. Girosi & T. Poggio, A Nondeterministic Method for Multivariate Functions Minimization, *Parallel Architectures and Neural Networks*, pp. 37-44, World Scientific, Singapore, 1991.

[Cardie 1993] C. Cardie, Using Decision Trees to Improve Case-Based Learning, *Proceedings of the Tenth International Conference on Machine Learning*, pp. 25-32, Amherst, MA, 1993.

[Castillo et al. 2000] P.A. Castillo, J. Carpio, J. J. Merelo, V. Rivas, G. Romer & A. Prieto, Evolving Multilayer Perceptrons, *Neural Processing Letters*, Vol. 12, pp.115-127, 2000.

[Choi et al. 1992] J. Choi, P. Arabshahi, R.J. Marks & T.P. Caudell, Fuzzy Parameter Adaptation in Neural Systems, *Proceedings of the International Joint Conference on Neural Networks*, Vol. 1, pp. 232-238, Baltimore, MD, 1992.

[Darwin 1859] C. Darwin, *On the Origin of Species*, John Murray, London, 1859.

[Davis 1991] L. Davis, *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New York, NY, 1991.

[DeFalco et al. 1998] I. DeFalco, A. Iazzetta, P. Natale & E. Tarantino, Evolutionary Neural Networks for Nonlinear Dynamics Modeling, *Lectures Notes in Computer Science*, Vol. 1498, pp. 593-602, 1998.

[Demuth, Beale 1993] H. Demuth & M. Beale, *Neural Network Toolbox for Use with Matlab*, The Mathworks Inc., 1993.

[Dorffner 1996] G. Dorffner, Neural Networks for Time Series Processing, *Neural Network World*, Vol. 6(4), pp. 447-468, 1996.

[Drossu, Obradovic 2001] R. Drossu & Z. Obradovic, Efficient Design of Neural Networks for Time Series Prediction, acessado em 11 de Julho de 2001, URL: http://citeseer.nj.nec.com/183217.html.

[Elman 1990] J. L. Elman, Finding Structure in Time, *Cognitive Science*, Vol. 14, pp. 179-211, 1990.

[Fahlman, Lebiere 1990] S. E. Fahlman & C. Lebiere, The Cascade-Correlation Learning Architecture, *Technical Report CMU-CS-90-100*, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1990.

[Fayyad et al. 1996] U. Fayyad, G. Piatetsky-Shapiro & P. Smyth. *From Data Mining to Knowledge Discovery: An Overview*, *Advances in Knowledge Discovery and Data Mining*, pp. 1-34, AAAI Press, Menlo Park, CA, 1996.

[Fletcher 1987] R. Fletcher, *Practical Methods of Optimization*, John Wiley & Sons, New York, NY, 1987.

[Fletcher, Obradovic 1993] J. Fletcher & Z. Obradovic, Combining Prior Symbolic Knowledge and Constructive Neural Network Learning, *Connection Science*, Vol. 5(3,4), pp. 365-375, 1993.

[Fogel et al. 1990] D. B. Fogel, L. J. Fogel & V. W. Porto, Evolving Neural Networks, *Biological Cybernetics*, Vol. 63, pp. 487-493, 1990.

[Galvão 1999] C. de O. Galvão, Introdução à Teoria dos Conjuntos Difusos, *Sistemas Inteligentes: Aplicações a Recursos Hídricos e Ciências Naturais*, pp.167-191, Ed. Universidade, UFRGS, Porto Alegre, Brasil, 1999.

[Giles et al. 2001] C. L. Giles, S. Lawrence & A. C. Tsoi, Noisy Time Series Prediction Using a Recurrent Neural Network and Grammatical Inference, *Machine Learning*, Vol. 44(1,2), pp. 161-183, 2001.

[Goldberg 1989] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, Reading, MA, 1989.

[Gray, Kilgour 1997] A. Gray & R. Kilgour, *Hybrid Systems FAQ*, URL: http://divcom.otago.ac.nz:800/COM/INFOSCI/SMRL/people/andrew/publications/faq/hybrid/hybrid.htm#Information, 1997.

[Hakkarainen et al. 1996] J. Hakkarainen, A. Jumppanen, J. Kyngäs & J. Kyyrö, An Evolutionary Approach to Neural Network Design Applied to Sunspot Prediction, *Report Series A*, University of Joensuu, Finland, 1996.

[Hallas, Dorffner 1998] M. Hallas & G. Dorffner, A Comparative Study on Feedforward and Recurrent Neural Networks in Time Series Prediction Using Gradient Descent Learning, *Proceedings of 14th European Meeting on Cybernetics and Systems Research*, pp.644-647, Vienna, Austria, 1998.

[Hancock 1992] P. J. B. Hancock, Genetic Algorithms and Permutation Problems: a Comparison of Recombination Operators for Neural Net Structure Specification, *Proceedings of COGANN Workshop (IJCNN)*, pp. 108-122, Baltimore, 1992.

[Harvey 1993] A. C. Harvey, *Time series models*, MIT Press, Cambridge, MA, 1993.

[Haykin 1994] S. Haykin, *Neural Networks: A Comprehensive Foundation*, IEEE Press/Macmillan College Publishing Company, New York, NY, 1994.

[Heckerman 1995] D. Heckerman, A Tutorial on Learning with Bayesian Networks, *Technical Report MSR-TR-95-06*, Microsoft Research, Redmond, Washington, 1995.

[Hertz, Hu 1992] D. B. Hertz & Q. Hu, Fuzzy-Neuro Controller for Backpropagation Networks, *Proceedings of the Simulation Technology and Workshop on Neural Networks Conference*, pp. 570-574, Houston, TX, 1992.

[Hilario et al. 1996] M. Hilario, B. Orsier, A. Rida, and C. Pellegrini., A Knowledge-Based Approach to MLP Configuration, *Proceedings of the International Conference on Neural Networks*, Washington DC, 1996.

[Holland 1975] J. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, MI, 1975.

[Honavar, Uhr 1994] V. Honavar & L. Uhr, *Artificial Intelligence and Neural Networks: Steps Toward Principled Integration*, Academic Press, New York, NY, 1994.

[Honavar, Uhr 1995] V. Honavar & L. Uhr, Integrating Symbol Processing Systems and Connectioninst Networks, *Intelligent Hybrid Systems*, pp. 177-208, Wiley, London, 1995.

[Hong, Vasaru 2001] H. Hong & D. Vasaru, *Survey on Nonlinear Optimization*, 1996, acessado em 20 de Novembro de 2001,
URL: http://citeseer.nj.nec.com/hong96survey.html.

[Hutchinson 1994] J. M. Hutchinson, A Radial Basis Function Approach to Financial Time Series Analysis, *Ph.D. thesis*, Massachusetts Institute of Technology, MA, 1994.

[Ivanova, Kubat 1995] I. Ivanova & M. Kubat, Initialization of Neural Networks by Means of Decision Trees, Knowledge Based Systems, Vol. 8(6), pp. 333-344, 1995.

[Jordan 1986] M. Jordan, Serial Order: a Parallel Distributed Processing Approach, *ICS report 8604*, Institute for Cognitive Science, University of California at San Diego, CA, 1986.

[Kadowaki et al. 1997] M. Kadowaki, S. Soares & M. G. Andrade, Previsão de Vazões Mensais Utilizando Redes Neurais Multicamadas com Algoritmo Backpropagation, *Anais do IV Simpósio Brasileiro de Redes Neurais*, pp. 32-35. Goiana, GO, 1997.

[Kartalopoulos 1995] S. V. Kartalopoulos, *Understanding Neural Networks and Fuzzy Logic: Basic Concepts and Applications*, IEEE Press, New York, NY, 1995.

[Keller, Tahani 1992] J. M. Keller & H. Tahani, Implementation of Conjunctive and Disjunctive Fuzzy Logic Rules with Neural Networks, *International Journal of Approximate Reasoning*, Vol. 6, pp. 221-240, 1992.

[Khebbal, Goonatilake, 1995] S. Khebbal & S. Goonatilake, Intelligent Hybrid Systems: Issues, Classifications and Future Directions, *Intelligent Hybrid Systems*, pp. 1-20, Wiley, London, 1995.

[Khosla, Dillon 1997] R. Khosla & T. Dillon, *Engineering Intelligent Hybrid Multi-Agent Systems*, Kluwer Academic Publishers, Boston, 1997.

[Kinnebrock 1994] W. Kinnebrock, Accelerating the Standard Backpropagation Method Using a Genetic Approach, *Neurocomputing*, Vol. 6, pp. 583-588, 1994.

[Kirkpatrick 1983] S. Kirkpatrick, Optimization by Simulated Annealing, *Science*, Vol. 220, pp. 6761-680, 1983.

[Klimasauras 1995] C. C. Klimasauras, Using Fuzzy Pre-processing with Neural Networks for Chemical Process Diagnostic Problems, *Intelligent Hybrid Systems*, pp. 143-152, Wiley, London, 1995.

[Kohonen 1989] T. Kohonen, *Self-Organization and Associative Memory*, SpringerVerlag, Berlin, Germany, 1989.

[Kolarik, Rudorfer 1994] T. Kolarik & G. Rudorfer, Time Series Forecasting Using Neural Networks, APL Quote Quad, Vol. 25(1), pp. 86-94, ACM Press, New York, NY, 1994.

[Kolodner 1993] J. Kolodner, *Case-Based Reasoning*, Morgan Kaufmann, San Mateo, CA, 1993.

[Kong, Kosko 1992] S.G. Kong and B. Kosko, Adaptive Fuzzy Systems for Backing up a Truck-and-Trailer, *IEEE Trans. Neural Networks*, Vol. 3(2), pp. 211-223, 1992.

[Koskela et al. 1996] T. Koskela, M. Lehtokangas, J. Saarinen & K. Kaski, Time series Prediction with Multilayer Perceptron, FIR and Elman Neural Networks, *Proceedings of the World Congress on Neural Networks*, pp. 491-496, San Diego, CA, 1996.

[Koza, Rice 1991] J. R. Koza & J. P. Rice, Genetic Generation of Both the Weights and Architecture for a Neural Network, *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, Vol II, pp. 397-404, Seattle, WA, 1991.

[Kreesuradej et al. 1994] W. Kreesuradej, D. C. Wunsch II & M. Lane, Time Delay Neural Network for Small Time Series Data Sets, *World Congress on Neural Networks*, Vol II, pp, 248-253, San Diego, CA, 1994.

[Kubat 1998] M. Kubat, Decision Trees Can Initialize Radial Basis Function Networks, *IEEE Transactions on Neural Networks*, Vol. 9, pp. 813-821, 1998.

[Lacerda, Carvalho 1999] E. G. M. Lacerda & A. C. P. L. F. Carvalho, Introdução aos Algoritmos Genéticos, *Sistemas Inteligentes: Aplicações a Recursos Hídricos e Ciências Naturais*, pp. 99-150, Ed. Universidade, UFRGS, Porto Alegre, Brasil, 1999.

[Leake 1996] D. Leake, CBR in Context: The Present and Future, *Case-Based Reasoning, Experiences, Lessons, & Future Directions*, MIT Press, CA, 1996.

[LeCun et al. 1990] Y. LeCun, J. Denker & S. Solla, Optimal Brain Damage, *Advances in Neural Information Processing Systems*, Vol. 2, pp. 598-605, Morgan Kaufmann, New York, NY, 1990.

[Levenberg 1944] K. Levenberg, A Method for the Solution of Certain Non-Linear Problems in Least Squares, *Quarterly Journal of Applied Mathematics*, Vol.2, pp. 164-168, 1944.

[Liu, Yao 1996] Liu, Y. and Yao, X., A Population-Based Learning Algorithm which Learns Both Architectures and Weights of Neural Networks, *Chinese Journal of Advanced Software Research*, Vol. 3(1), pp. 54-65, 1996.

[Lopes et al. 1981] J. E. G. Lopes, B. P. F. Braga & J. G. L. Conejo, Simulação hidrológica: Aplicações de um Modelo Simplificado, *Anais do IV Simpósio Brasileiro de Hidrologia e Recursos Hídricos*, pp. 42-62, 1981.

[Louis, Johnson 1997] S. J. Louis & J. Johnson, Solving Similar Problems Using Genetic Algorithms and Case-Based Memory, *Proceedings of the Seventh International Conference on Genetic Algorithms*, pp. 283-290, San Mateo, CA, 1997.

[Louis, Li 1997] S. J. Louis & G. Li, Augmenting Genetic Algorithms with Memory to Solve Traveling Salesman Problems, *Proceedings of the Joint Conference on Information Sciences*, pp. 108-111, Duke University Press, 1997.

[Makridakis, Wheelwright 1989] S. Makridakis & S. Wheelwright, *Forecasting: Methods for Management*, John Wiley & Sons, New York, NY, 1989.

[Malek 1995] M. Malek, A Connectionist Indexing Approach for CBR Systems, *Lecture Notes in Artificial Intelligence*, Vol. 1010, pp. 520-527, 1995.

[Mandischer 1993] M. Mandischer, Representation and Evolution of Neural Networks, *Proceedings of the International Conference on Artificial Neural Nets and Genetic Algorithms*, pp. 643-649, Innsbruck, Austria, 1993.

[Marquardt 1963] D. Marquardt, An Algorithm for Least-Squares Estimation of Nonlinear Parameters, *SIAM, Journal of Applied Mathematics*, Vol. 11, pp. 431-441, 1963.

[Masters 1995a] T. Masters, *Neural, Novel and Hybrid Algorithms for Time Series Prediction*, John Wiley & Sons, Inc, New York, NY, 1995.

[Masters 1995b] T. Masters, Advanced Algorithms for Neural Networks: a C++ Sourcebook, John Wiley & Sons, Inc, New York, NY, 1995.

[Mathworks 1996] The MathWorks, *MATLAB Language Referenece Manual*, The MathWorks Inc., Natick, MA, 1996.

[McCulloch, Pitts 1943] W. S. McCulloch & W. Pitts, A Logical Calculus of the Ideas Immanent in Nervous Activity, *Bulletin of Mathematical Biophysics*, Vol. 5, pp.115-133, 1943.

[Michalewicz 1992] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, Springer-Verlag, 1992.

[Michaud, Rubio 1996] F. Michaud & R. G. Rubio, Autonomous Design of Artificial Neural Networks by Neurex, *Neural Computation*, Vol. 8, pp. 1767-1786, 1996.

[Mitchel 1997] T. Mitchel, *Machine Learning*, MacGraw Hill, New York, NY, 1997.

[Mitchell 1996] M. Mitchell, *An Introduction to Genetic Algorithms*, MIT Press, 1996.

[Mitra Hayashi 2000] S. Mitra & Y. Hayashi, Neuro-fuzzy Rule Generation: Survey in Soft Computing Framework, *IEEE Transactions on Neural Network*, Vol. 11, pp. 748-768, 2000.

[Montana, Davis 1989] D. J. Montana & L. Davis, Training Feedforward Neural Networks Using Genetic Algorithms, *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pp. 762-767, Detroit, MI, 1989.

[Montgomery et al. 1990] D. C. Montgomery, L. A. Johnson & J. S. Gardiner, *Forecasting & Time Series Analysis*, Mc-Graw-Hill, New York, NY, 1990.

[Moody 1994] J. Moody, Prediction Risk and Architecture Selection for Neural Networks, *From Statistics to Neural Networks: Theory and Pattern Recognition Applications*, pp. 143-156, Springer-Verlag, Berlin, Germany, 1994.

[Mooney, Ourston 1994] R. Mooney & D. Ourston, A Multistrategy Approach to Theory Refinement, *Machine Learning IV: A Multistrategy Approach*, pp. 141-164, Morgan Kaufmann, San Francisco, CA, 1994.

[Morettin, Toloi 1987] P. A. Morettin & C. M. Toloi, Séries Temporais, *Métodos Quantitativos*, Atual, São Paulo, SP, 1987.

[Newell, Simon 1956] A. Newell & H. A Simon, The Logic Theory Machine: A Complex Information Processing System, *IRE Transactions on Information Theory*, Vol. 2, pp. 61-79, 1956.

[Nikovski, Zargham 1996] D. Nikovski & M. Zargham, Comparison of Two Learning Networks for Time Series Prediction, *Proceedings of the Ninth International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (IEA/AIE 96)*, pp. 531-555, Fukuoka, Japan, 1996.

[Norgaard 1997] M. Norgaard, Neural Network Based System Identification Toolbox Version 1.1 For Use with Matlab, *Technical Report 97-E-851*, Department of Automation, Technical University of Demark, 1997.

[O'Donovan 1983] T. M. O'Donovan, *Short Term Forecasting: An Introduction to the Box-Jenkins Approach*, John Wiley & Sons, New York, NY, 1983.

[Oliveira 2000] E. M. J. Oliveira, Previsão da Série Temporal do Índice da Bolsa de Valores de São Paulo Usando Redes Neurais e Estatística, *Dissertação de Mestrado*, Centro de Informática, Universidade Federal de Pernambuco, Recife, Brasil, 2000.

[Oliveira, Ludermir 1992] W. Oliveira & T. B. Ludermir, Turing Machine Simulation by Logical Neural Networks, *Artificial Neural Networks*, Vol. 2, pp. 663-667, Elsevier Science Publishers B. V., North-Holland, UK, 1992.

[Omlin, Giles 1996] C. Omlin & C. Giles, Rule Revision with Recurrent Neural Networks, *IEEE Transactions on Knowledge and Data Engineering*, Vol. 8(1), pp. 183-188, 1996.

[Optiz, Shavlik 1993] D.W. Optiz & J.W. Shavlik, Heuristically Expanding Knowledge-Based Neural Networks, *Proceedings of Thirteenth International Joint Conference on Artificial Intelligence*, pp. 512-517, Chambray, France, 1993.

[Pankratz 1983] A. Pankratz, *Forecasting With Univariate Box-Jenkins Models: Concepts and Cases*. John Wiley & Sons, New York, NY, 1983.

[Pham, Liu 1995] D. T. Pham & X. Liu, *Neural Networks for Identification, Prediction and Control*, Springer, London, 1995.

[Prechelt 1994] L. Prechelt, Proben 1 - A Set of Neural Network Benchmark Problems and Benchmarking Rules, *Technical Report 21/94*, Universitat Karlsruhe, Fakultat fur Informatik, 1994.

[Prudêncio, Ludermir 2001a] R. B. C. Prudêncio & T. B. Ludermir, Evolutionary Design of Neural Networks: Aplication to River Flow Prediction, *Proceedings of the International Conference on Artificial Inteligence and Aplications* (AIA 2001), pp. 56-61, Marbella, Spain, 2001.

[Prudêncio, Ludermir 2001b] R. B. C. Prudêncio & T. B. Ludermir, Design of Neural Networks for Time Series Prediction Using Case-Initialized Genetic Algorithms, *Proceedings of the 8th International Conference on Neural Information Processing* (ICONIP' 01), pp. 990-995, Shanghai, China, 2001.

[Quinlan 1986] J.R. Quinlan, Induction of Decision Trees, *Machine Learning*, Vol. 1. pp. 81-106, MacGraw Hill, New York, NY, 1986.

[Ramsey, Grefenstette 1993] C. Ramsey & J. Grefenstette, Case-Based Initialization of Genetic Algorithms, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pp. 84-91, San Mateo, CA, 1993.

[Rudnick 1990] M. Rudnick, A Bibliography of the Intersection of Genetic Search and Artificial Neural Networks, *Technical Report CS/E 90-001*, Department of Computer Science and Engineering, Oregon Graduate Center, 1990.

[Rumelhart et al. 1986] D. E. Rumelhart, G. E. Hinton & R. J. Williams, Learning Representations by Backpropagation Errors, *Nature*, Vol. 323, pp. 533-536, 1986.

[Russell, Marks 1999] R. Russell & R. Marks, *Neural Smithing*, MIT Press, Cambridge, MA, 1999.

[Russell, Norvig 1995] S. Russell & P. Norvig, *Artificial Intelligence: A Modern Approach*, Prentice Hall Series, Englewood Cliffs, NJ, 1995.

[Sarle 2001] W. S. Sarle, *Neural Network FAQ*, periodic posting to the Usenet newsgroup comp.ai.neural-nets, URL: ftp://ftp.sas.com/pub/neural/FAQ.html, 2001.

[Scherer, Schlageter 1995] A. Scherer & G. Schlageter, A Multi-Agent Approach for the Integration of Neural Networks and Expert Sistems, *Intelligent Hybrid Systems*, pp. 153-174, John Wiley & Sons, London, 1995.

[Schuster 1996] M. Schuster, Learning Out of Time Series with an Extended Recurrent Neural Network, *Neural Network Workshop for Signal Processing*, pp. 170-179, Kyoto, Japan, 1996.

[Shanon 1950] C. E. Shannon, Programming a Computer for Playing Chess, *Philosophical Magazine*, Vol. 41, pp. 256-275, 1950.

[Sharda, Patil 1990] R. Sharda & R. Patil, Neural Networks as Forecasting Experts: An Empirical Test, *Proceedings of the International Joint Conference on Neural Networks*, Vol. 2, pp. 491-494, Washington DC, 1990.

[Shavlik et al. 1991] J. W. Shavlik, R.J. Mooney & G.G. Towell, Symbolic and Neural Learning Algorithms: An Experimental Comparison, *Machine Learning*, 6(2), pp. 111-143, 1991.

[Silva 1999] R. B. A. Silva, Hybrid Systems of Local Basis Functions, *Dissertação de mestrado*, Departamento de Informática, Universidade Federal de Pernambuco, Recife, Brasil, 1999.

[Sjoberg et al. 1994] J. Sjoberg, H. Hjalmarsson, & L. Ljung, Neural Networks in System Identification, *Proceedings of the 10th IFAC Symposium on System Identification*, Vol. 2. pp. 49-72, Copenhagen, Demark, 1994.

[Skapura 1996] D. M. Skapura, *Building Neural Networks*, ACM Press, New York, NY, 1996.

[Souto 1999] M. C. P. Souto, Computability and Learnability in Sequential Weightless Neural Networks, *Ph.D. Thesis*, Imperial College, UK, 1999.

[Sovat, Carvalho 2001] R. B. Sovat & A. C. P. L. F. Carvalho, Escolha e Configuração de Modelos de Redes Neurais Artificiais Utilizando Raciocínio Baseado em Casos, *Anais do Encontro Nacional de Inteligência Artificial (III ENIA)*, Fortaleza, Brasil, 2001.

[Stagge, Sendhoff 1997] P. Stagge & B. Sendhoff, An Extended Elman Net for Modeling Time Series, *Lecture Notes in Computer Science*, Vol. 1327, pp. 427, 1997.

[Stepniewski, Keane 1996] S. W., Stepniewski & A. J. Keane, Topology Design of Feedforward Neural Networks by Genetic Algorithms, *Parallel Problem Solving from Nature*, pp. 771-780, Springer-Verlag, Berlin, Germany, 1996.

[Talko 1999] B. Talko, A Rule-Based Approach For Constructing Neural Networks Using Genetic Programming, *Master's thesis*, Department of Computer Science and Software Engineering, The University of Melbourne, Australia, 1999.

[Tang, Fishwick 1993] Z. Tang & P. A. Fishwick, Feed-forward Neural Nets as Models for Time Series Forecasting, *ORSA Journal of Computing*, Vol. 5(4), pp. 374-386, 1993.

[Thiesing, Vornberger 1997] F. M. Thiesing & O. Vornberger, Sales Forecasting Using Neural Networks, *Proceedings of the International Conference on Neural Networks (ICNN'97)*, Vol. 4, pp. 2125-2128, Houston, TX, 1997.

[Towell et al. 1990] G. Towell, J. Shavlik & M. Noordewier, Refinement of Approximate Domain Theories by Knowledge-Based Neural Networks, *Proceedings of the 8th National Conference on Artificial Intelligence (AAAI-90)*, Vol. 2, pp. 861-866, Boston, MA, 1990.

[Towell, Shavlik 1994] G. Towell & J. Shavlik, Knowledge-Based Artificial Neural Networks, *Artificial Intelligence*, Vol. 70, no. 1-2, pp. 119-165, 1994.

[Tresp et al. 1993] V. Tresp, J. Hollatz & S. Ahmad, Network Structuring and Training Using Rule-Based Knowledge, *Advances in Neural Information Processing Systems*, Vol. 5, pp. 871-878, Morgan Kaufman Publishers, San Mateo, CA, 1993.

[Tucci 1993] C. E. M. Tucci, *Hidrologia: Ciência e Aplicação*, Associação Brasileira de Recursos Hídricos, ABRH, Porto Alegre, Brasil, 1993.

[Turban 1992] E. Turban, *Expert Systems and Applied Artificial Intelligence*, Macmillan Publishing Company, New York, NY, 1992.

[Tzafestas, Blekas 1997] S. G. Tzafestas & K. D. Blekas, Hybrid Soft Computing Systems: A Critical Survey with Engineering Applications, *Technical Report*, National Technical University of Athens, Greece, 1997.

[Valença 1999a] M. J. S. Valença, Analysis and Design of the Constructive Neural Networks for Complex Systems Modeling, *Tese de Doutorado*, Departamento de Infomática, Universidade Federal de Pernambuco, Recife, Brasil, 1999.

[Valença 1999b] M. J. S. Valença, Aplicações de Redes Neurais, *Sistemas Inteligentes: Aplicações a Recursos Hídricos e Ciências Naturais*, pp. 61-96, Ed. Universidade, UFRGS, Porto Alegre, Brasil, 1999.

[Valença, Ludermir 1998] M. J. S. Valença & T. B. L. Ludermir, Self-Organizing Modeling in Forecasting Daily River Flows, *Anais do V Simpósio Brasileiro de Redes Neurais*, pp. 210-214, Belo Horizonte, Brasil, 1998.

[Viana 1986] F. L. Viana, Comportamento Hidrológico das Pequenas Bacias do Nordeste, *Boletim Técnico, Recursos Hídricos, 5*, Departamento de Hidráulica, Universidade Federal do Ceará, Brasil, 1986.

[Wermter, Sun 2000] S. Wermter & R. Sun, An Overview of Hybrid Neural Systems, *Hybrid Neural Systems*, pp. 1-13, Springer-Verlag, Heidelberg, Germany, 2000.

[Wettschereck et al. 1997] D. Wettschereck, D. W. Aha & T. Mohri, A Review and Empirical Evaluation of Feature Weighting Methods for a Class of Lazy Learning Algorithms, *Artificial Intelligence Review*, Vol. 11, pp. 273-314, 1997.

[Whitehead, Choate 1996] B. A. Whitehead & T. D. Choate, Cooperative-Competitive Genetic Evolution of Radial Basis Function Centers and Widths for Time Series Prediction, *IEEE Transactions on Neural Networks*, Vol. 7, no. 4, pp. 869-880, 1996.

[Widrow et al. 1994] B. Widrow, D. E. Rumelhart & M. A. Lehr, Neural Networks: Application in Industry, Business and Science, *Communications of the ACM*, Vol. 37(3), pp. 93-105, 1994.

[Yang, Honavar 1997] J. Yang & V. Honavar, Feature Subset Selection Using Genetic Algorithm, *Proceedings of the Second Annual Conference on Genetic Programming*, pp. 380-385, Stanford, CA, 1997.

[Yao 1995] X. Yao, Evolutionary Artificial Neural Networks, *Encyclopedia of Computer Science and Technology*, Vol. 33, pp. 137-170, Marcel Dekker Inc., New York, NY, 1995.

[Yao, Liu 1997] X. Yao, & Y, Liu, A New Evolutionary System for Evolving Artificial Neural Networks, *IEEE Transactions on Neural Networks*, Vol. 8(3), pp. 694-713, 1997.

[Zadeh 1965] L. A. Zadeh, Fuzzy Sets, *Information and Control*, Vol. 8, pp. 338-353, 1965.

[Zhang, Muhlenbein 1993] B. T. Zhang & H. Muhlenbein, Evolving Optimal Neural Networks Using Genetic Algorithms with Occam's Razor, *Complex Systems*, Vol. 7, pp. 199-220, 1993.