ESCOLA POLITÉCNICA
DE PERNAMBUCO

Departamento de
Sistemas
Computacionais

OTIMIZAÇÃO DE REDES NEURAIS PARA PREVISÃO DE SÉRIES TEMPORAIS

Trabalho de Conclusão de Curso

Engenharia da Computação

Adélia Carolina de Andrade Barros
Orientador: Prof. Dr. Adriano Lorena Inácio de Oliveira

Recife, maio de 2005

UNIVERSIDADE
DE PERNAMBUCO

ESCOLA POLITÉCNICA
DE PERNAMBUCO

Departamento de Sistemas Computacionais

OTIMIZAÇÃO DE REDES NEURAIS PARA PREVISÃO DE SÉRIES TEMPORAIS

Trabalho de Conclusão de Curso

Engenharia da Computação

Este Projeto é apresentado como requisito parcial para obtenção do diploma de Bacharel em Engenharia da Computação pela Escola Politécnica de Pernambuco – Universidade de Pernambuco.

Adélia Carolina de Andrade Barros
Orientador: Prof. Dr. Adriano Lorena Inácio de Oliveira

Recife, maio de 2005

Adélia Carolina de Andrade Barros

OTIMIZAÇÃO DE REDES NEURAIS PARA PREVISÃO DE SÉRIES TEMPORAIS

Resumo

Este trabalho apresenta um estudo sobre a otimização simultânea da arquitetura e pesos de redes Multi-Layer Perceptron para aplicações de previsão de séries temporais. Com este propósito, foram utilizados dois métodos de otimização global: *Tabu Search* e *Simulated Annealing*. Combinados a estes métodos, usou-se o treinamento com o algoritmo RPROP, numa abordagem híbrida ainda não utilizada para problemas temporais.

A abordagem utilizada procura combinar as principais vantagens dos métodos de otimização global com as dos métodos de convergência local: métodos de otimização são bastante eficientes na busca do espaço global enquanto métodos de convergência fazem uma busca local mais refinada.

A metodologia utilizada neste trabalho foi proposta por Yamazaki [27]. No referido trabalho, técnicas de otimização foram empregadas com sucesso em problemas de classificação. No entanto, tais técnicas não haviam sido aplicadas à previsão de séries temporais até então.

Os resultados mostram que as redes otimizadas segundo os modelos aqui propostos foram bastante eficientes quando comparadas ao conhecido modelo de previsão de séries, MLP com janelas de tempo.

Abstract

This work presents a study about simultaneous optimization of Multi-Layer Perceptron architectures and weights for time series forescasting applications. For this purpose, two global optimization methods are used: Tabu Search and Simulated Annealing. Combined with these methods, the RPROP algorithm is applied, in a hybrid training approach which has not been used for time-based problems yet.

The used approach tries to combine the main advantages from the global optimization methods with the ones obtained with the local convergency methods: optimization methods works efficiently in searching for the global space while the convergency methods make a more accurate search.

In this work, the methodology is proposed by Yamazaki [27]. In the referred work, optimization techniques were successfully applied in classification problems. However, these techniques have never been applied for time-series prediction until now.

The results show that the nets optimized by the proposed models are very efficient when compared with the well-known model for predicting series, the Time Lagged Feedforward Network.

Sumário

Índice de Figuras 5

Índice de Tabelas 6

1 Introdução 8

2 Multi-Layer Perceptron 10

2.1 Arquitetura 12
2.2 Treinamento de Redes MLP 14
2.3 Aplicações 15

3 Previsão de Séries Temporais 16

3.1 Processamento Temporal 16
3.2 Atraso no Tempo 17
3.2.1 MLP com janela de tempo 17
3.2.2 Time Delay Neural Networks 18
3.3 Redes Recorrentes 19
3.3.1 Redes de Elman e Jordan 19
3.4 Aplicações 20

4 Otimização Global de Redes Neurais Artificiais 21

4.1 Otimização global de arquiteturas e pesos 21
4.2 Simulated Annealing 23
4.3 Tabu Search 24

5 Otimização Simultânea de Arquiteturas e Pesos de Redes MLP 25

5.1 Metodologias de treinamento com otimização 25
5.1.1 Metodologia de treinamento com *Simulated Annealing* 26
5.1.2 Metodologia de treinamento com *Tabu Search* 28
5.2 Metodologia de treinamento sem otimização 29
5.3 Pré-processamento das bases de dados 29

6 Experimentos e Resultados 31

6.1 Experimentos com a série NYWater 31
6.1.1 Topologia e algoritmo de treinamento 32
6.1.2 Divisão da base de dados em treinamento validação e teste 32
6.1.3 Experimentos com Multy-Layer Perceptron (MLP) 32
6.1.4 Experimentos com Simulated Annealing (SA) 34
6.1.5 Experimentos com Tabu Search (TS) 37
6.2 Experimentos com a série ElecNew 40
6.2.1 Topologia e algoritmo de treinamento 40
6.2.2 Experimentos com Multy-Layer Perceptron (MLP) 41

6.2.3 Experimentos com Simulated Annealing (SA) 43
6.2.4 Experimentos com Tabu Search (TS) 45

7 Conclusões e Trabalhos Futuros 52

7.1 Conclusões 52
7.2 Trabalhos futuros 53

Índice de Figuras

Figura 1. Ilustração de um problema linearmente separável 10
Figura 2. Ilustração de um problema não linearmente separável 11
Figura 3. Neurônio Artificial 11
Figura 4. Aprendizado supervisionado 12
Figura 5. Exemplo de rede Multi-Layer Perceptron 13
Figura 6. Fluxo das fases de propagação e retro-propagação do algoritmo *backpropagation* 14
Figura 7. Exemplo de MLP com janela de tempo 3 18
Figura 8. Ilustração de uma Rede de Elman 19
Figura 9. Ilustração de uma Rede de Jordan 20
Figura 10. Diferença entre mínimos locais e mínimo global 22
Figura 11. **(a)** MLP totalmente conectada e sua respectiva matriz de conectividade; **(b)** MLP parcialmente conectada e sua respectiva matriz de conectividade. 26
Figura 12. Série NYWater normalizada entre 0 e 1 31
Figura 13. Comparação entre os valores previstos pela melhor rede MLP treinada com o algoritmo RPROP e os valores originais da série, a cada ano do conjunto de teste. 34
Figura 14. Comparação entre os valores previstos pela melhor rede MLP treinada com o algoritmo de otimização formado por SA e RPROP e os valores originais da série, a cada ano do conjunto de teste. 36
Figura 15. Comparação entre os valores previstos pela melhor rede MLP treinada com o algoritmo de otimização formado por TS e RPROP e os valores originais da série, a cada ano do conjunto de teste. 38
Figura 16. Comparação entre os valores originais da série NYWater e os valores previstos pelas melhores redes MLP treinadas com os algoritmos RPROP (sem otimização), SA (híbrido) e TS (híbrido), a cada ano do conjunto de teste. 39
Figura 17. Série ElecNew normalizada entre 0 e 1 40
Figura 18. Comparação entre os valores previstos pela melhor rede MLP treinada com o algoritmo RPROP e os valores originais da série, a cada mês do conjunto de teste. 42
Figura 19. Comparação entre os valores previstos pela melhor rede MLP treinada com o algoritmo de otimização formado por SA e RPROP e os valores originais da série, a cada mês do conjunto de teste. 45
Figura 20. Comparação entre os valores previstos pela melhor rede MLP treinada com o algoritmo de otimização formado por TS e RPROP e os valores originais da série, a cada mês do conjunto de teste. 48
Figura 21. Comparação entre os valores originais da série ElecNew e os valores previstos pelas melhores redes MLP treinadas com os algoritmos RPROP (sem otimização), SA (híbrido) e TS (híbrido), a cada mês do conjunto de teste. 50

Índice de Tabelas

Tabela 1. Entradas e saídas desejadas para o treinamento de MLPs 17
Tabela 2. Resultados obtidos para a série NYWater com redes MLP treinadas com o algoritmo RPROP 33
Tabela 3. Valores originais e previstos pela melhor rede MLP treinada com o algoritmo RPROP, a cada ano do conjunto de teste 34
Tabela 4. Resultados obtidos para a série NYWater com redes MLP treinadas com o sistema de otimização formado por SA e RPROP 35
Tabela 5. Valores originais e previstos pela melhor rede MLP treinada com o algoritmo de otimização formado por SA e RPROP, a cada ano do conjunto de teste 36
Tabela 6. Resultados obtidos para a série NYWater com redes MLP treinadas com o sistema de otimização formado por TS e RPROP 38
Tabela 7. Valores originais e previstos pela melhor rede MLP treinada com o algoritmo de otimização formado por TS e RPROP, a cada ano do conjunto de teste 39
Tabela 8. Valores originais da série NYWater e os valores previstos pelas melhores redes MLP treinadas com os algoritmos RPROP (sem otimização), SA (híbrido) e TS (híbrido), a cada ano do conjunto de teste 40
Tabela 9. Resultados obtidos para a série ElecNew com redes MLP treinadas com o algoritmo RPROP 42
Tabela 10. Valores originais e previstos pela melhor rede MLP treinada com o algoritmo RPROP, a cada mês do conjunto de teste 43
Tabela 11. Resultados obtidos para a série ElecNew com redes MLP treinadas com o sistema de otimização formado por SA e RPROP 44
Tabela 12. Valores originais e previstos pela melhor rede MLP treinada com o algoritmo de otimização formado por SA e RPROP, a cada mês do conjunto de teste 46
Tabela 13. Resultados obtidos para a série ElecNew com redes MLP treinadas com o sistema de otimização formado por TS e RPROP 47
Tabela 14. Valores originais e previstos pela melhor rede MLP treinada com o algoritmo de otimização formado por TS e RPROP, a cada mês do conjunto de teste 49
Tabela 15. Valores originais da série ElecNew e os valores previstos pelas melhores redes MLP treinadas com os algoritmos RPROP (sem otimização), SA (híbrido) e TS (híbrido), a cada mês do conjunto de teste. 51

Agradecimentos

À minha mãe Maria de Fátima Camboim de Andrade Barros e avó Leda Camboim de Andrade pelo amor e suporte em todos os momentos da minha vida.

Ao meu orientador, Prof. Dr. Adriano Lorena Inácio de Oliveira, por ter me ajudado sempre durante todos os anos em que trabalhamos juntos e influenciado a minha decisão por esta área de pesquisa.

Aos colegas de iniciação científica do NUPEC, em especial à Gabriel Azevedo que trabalhou mais diretamente comigo.

Aos colegas Arthur Lins, George Cabral e Gabriel Alves que contribuíram com materiais que ajudaram no término deste trabalho.

A Ana Falcão, Marcelo Nunes, Nívia Quental e Pedro Gomes, amigos queridos que tiveram paciência para ouvir os meus resmungos.

Aos professores do curso de Engenharia da Computação da Universidade de Pernambuco pelo conhecimento transmitido ao longo dos cinco anos de convivência e pelo esforço empregado para a realização do curso.

A todos os amigos do curso de Engenharia da Computação da Escola Politécnica de Pernambuco que sempre estiveram comigo durante toda a jornada acadêmica e sempre farão parte da minha vida.

Às amigas do Clube da Lulu que me proporcionaram 5 maravilhosos anos de convivência.

Capítulo 1

Introdução

Uma rede neural artificial (RNA) [11] é um processador maciçamente paralelo distribuído constituído de unidades de processamento simples, que tem a propensão natural para armazenar conhecimento experimental e torná-lo disponível para uso. Como no cérebro, o conhecimento é adquirido pela rede através de um processo de aprendizagem. Forças de conexão entre neurônios, conhecidas como pesos sinápticos, são utilizadas para armazenar o conhecimento adquirido.

Como dito na definição acima, o conhecimento da rede é armazenado em suas conexões através dos pesos, o que mostra quão relevantes são as conexões no processo de aprendizagem de uma RNA. Uma rede treinada com um número muito baixo de conexões pode não tirar o máximo proveito de seu potencial. Por outro lado, uma rede com um número excessivo de conexões pode se adaptar a ruídos e prejudicar seu treinamento.

Outro fator que influencia o treinamento de uma RNA é a inicialização do valor de seus pesos. Para um dado problema, diferentes inicializações de pesos vão fornecer diferentes resultados.

Apesar de ser fundamental para o sucesso da aplicação, a escolha da arquitetura e pesos de uma rede neural, geralmente, é feita de modo empírico, o que não traz a certeza de um bom resultado.

O problema de otimização pode ser formulado como um problema de busca no espaço, onde cada ponto representa uma arquitetura. Dada uma função de custo, como, por exemplo, diminuir o erro de treinamento e a complexidade da rede (número de conexões), a solução é encontrada através da busca do ponto cujo custo seja mínimo.

Para encontrar o ponto de mínimo global da função de custo é preciso percorrer a função com algum algoritmo de busca. À medida que este percorre a função, é comum que vários pontos de mínimo locais sejam encontrados - varia de acordo a função analisada. Logo, o algoritmo deve ser capaz de analisar se aquele ponto de mínimo é ou não o ponto global, essa é uma tarefa bastante complicada.

Dois métodos de otimização que podem ser usados para lidar com RNAs são o Simulated Annealing (SA) [14] e o Tabu Search (TS) [9][10]. Ambos os métodos são especialmente eficientes já que fogem de mínimos locais buscando sempre a solução global.

Neste trabalho é descrita uma proposta usando SA e TS para a otimização simultânea de arquitetura e pesos de redes MLP (Multi-Layer Perceptron) [25] para a melhoria de problemas de previsão de séries temporais, um tema ainda pouco abordado na área. O objetivo é elaborar um estudo comparativo entre MLPs treinadas com métodos de otimização e MLPs convencionais, não submetidas à otimização. Este trabalho foi inspirado nos trabalhos [26][27], onde SA e TS

foram utilizados com sucesso no problema da classificação de odores por um nariz artificial, e [5], onde é apresentado um sistema para detecção de fraudes em folhas de pagamento baseado em previsão de séries temporais. Sabe-se que um grande número de problemas reais pode ser modelado usando séries temporais. Previsão é uma das técnicas empregadas com sucesso em áreas como detecção de fraudes em folhas de pagamento [20] e em sistemas contábeis [16].

Este trabalho de conclusão de curso está organizado em 7 capítulos no total. Neste capítulo, o contexto no qual este trabalho está inserido, a motivação e os objetivos do mesmo são apresentados juntamente com uma breve citação dos trabalhos relacionados.

O Capítulo 2 revisa alguns conceitos básicos sobre as redes MLP, sua arquitetura e treinamento com o seu mais famoso algoritmo, o *backpropagation*. Ainda no Capítulo 2, são mostradas e brevemente discutidas algumas aplicações que utilizaram redes MLP.

No Capítulo 3 são revisadas algumas técnicas de previsão de séries temporais. Primeiro, são discutidas as técnicas que introduzem atraso no tempo: TDNN e janela de tempo. Em seguida, são apresentadas técnicas que introduzem atraso em sua topologia: redes de Elman e Jordan. No fim do Capítulo, são citados alguns trabalhos relacionados ao tema previsão de séries temporais.

No Capítulo 4, é discutido o tema "otimização global de redes neurais artificiais" com enfoque para a otimização simultânea de arquiteturas e pesos. Os dois algoritmos de otimização, *simulated annealing* e *tabu search*, utilizados neste trabalho são também apresentados neste capítulo.

O Capítulo 5 traz a metodologia utilizada tanto para os treinamentos realizados sem otimização quanto para aqueles que utilizaram técnicas de otimização. O destaque do capítulo é a utilização de MLPs com janela de tempo, *simulated annealing* e *tabu search* para previsão de séries temporais e o pré-processamento realizado sobre as séries analisadas.

O Capítulo 6 mostra como foram realizados os experimentos, apresenta os resultados em si e a análise comparativa dos mesmos para duas séries estudadas.

Finalmente, no Capítulo 7, são discutidas as conclusões, principais contribuições, limitações e propostas para futuros trabalhos.

Capítulo 2

Multi-Layer Perceptron

As redes *Multi-Layer Perceptron* (MLP) [25], como o nome sugere, são formadas por múltiplas camadas de neurônios *Perceptron* [24], unidades de processamento computacionais. As redes MLP têm como principal característica a capacidade de lidar com problemas não linearmente separáveis, ao contrário das redes *Perceptron* de única camada.

Problemas linearmente separáveis são aqueles cuja solução pode ser obtida através de uma reta ou hiperplano (para problemas n-dimensionais). A reta ou hiperplano é responsável por dividir o espaço de soluções em regiões ou classes. A Figura 1 exemplifica um problema linearmente separável, nela é ilustrado um caso onde é possível, por meio de uma reta, dividir a solução em duas classes A e B.

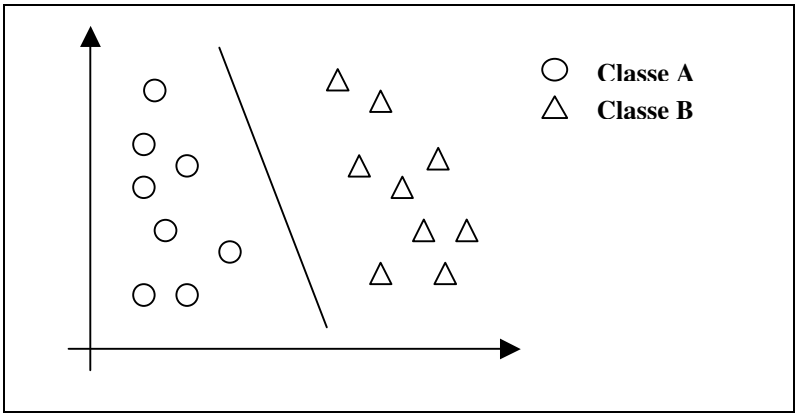


Figura 1. Ilustração de um problema linearmente separável

Porém, sabe-se que na maioria dos casos reais não é possível fazer essa divisão de classes com uma única reta e sim com um conjunto de retas ou regiões espaciais mais complexas (ver Figura 2).

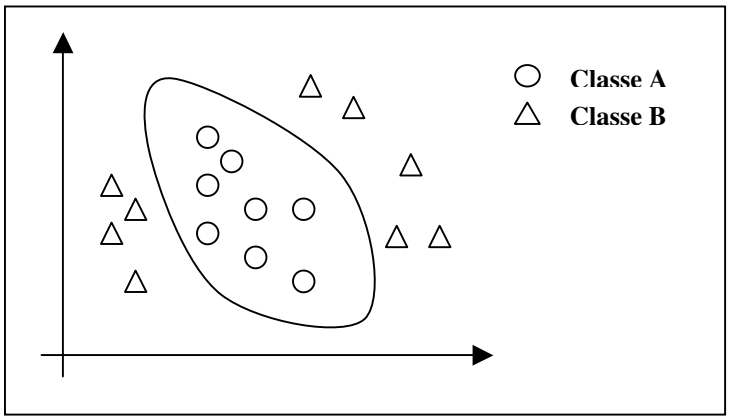


Figura 2. Ilustração de um problema não linearmente separável

Uma vez esclarecido o termo "linearmente separável", torna-se mais fácil entender a razão do uso de redes MLP ter se tornado bastante popular a partir da época de sua criação: elas são aplicáveis a um número maior de problemas reais. Nos dias de hoje ainda é muito comum o uso de redes MLP frente às demais topologias de redes neurais existentes. Vale a pena ressaltar que muitas destas são variações de redes MLP como, por exemplo, redes Jordan [13], Elman [7] e TDNN (*Time Delay Neural Network*) [12].

Em redes MLP, cada neurônio ou nodo da rede é alimentado por um conjunto de entradas e um conjunto de pesos sinápticos cada qual associado a uma das entradas. Estes conjuntos são, então, submetidos a um processamento e o fruto deste processamento é o resultado ou saída do neurônio. A Figura 3 é a representação gráfica de um neurônio artificial do tipo MCP [18]. Através da observação da figura, identificamos a disposição das entradas, os valores X_1 até X_n, e de seus pesos, valores W_1 até W_n. O processamento se dá em duas etapas, na primeira é realizada a soma ponderada das entradas com seus respectivos pesos e na segunda o resultado da soma ponderada é fornecido como entrada para uma função de ativação [8]. Só então um resultado é obtido, o qual está representado na figura em questão por Y e seu cálculo na Equação 1.

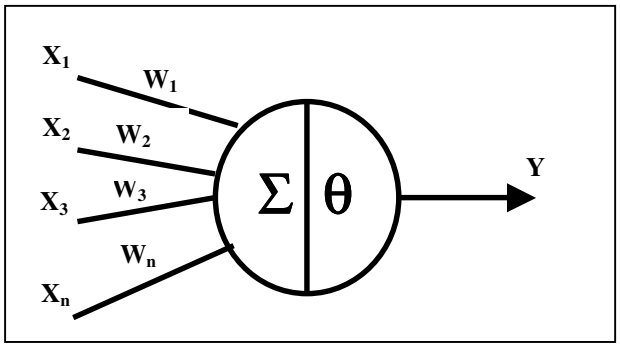


Figura 3. Neurônio Artificial

$$Y_j = \sum_{i=0}^{n} X_j . W_{ij} \qquad \textbf{(Equação 1)}$$

Tendo em mãos a saída do nodo, pode-se calcular o seu erro, ou seja, a diferença da saída obtida de fato para a saída que se deseja obter. A descrição formal do erro pode ser vista na

Equação 2, onde *e* simboliza o erro, *d* representa a saída desejada e Y a saída calculada pela rede. O treinamento (ajuste dos pesos) é conduzido de modo a diminuir este erro.

$$e = d - Y \qquad \textbf{(Equação 2)}$$

A fórmula para ajuste de pesos através da correção de erros é apresentada na Equação 3.

$$W_{i+1} = W_i - \eta e X_i \qquad \textbf{(Equação 3)}$$

Segundo a Equação 3, o ajuste dos pesos é equivalente ao produto do erro *e* pelo valor da entrada associada X_i e a constante η, chamada de taxa de aprendizado. Ainda sobre a Equação 3, W_i representa o valor atual do i-ésimo peso e W_{i+1} o valor do peso ajustado. Esse método de aprendizado, onde a saída desejada é conhecida, é chamado de aprendizado supervisionado. Diz-se que existe um professor que mostra à rede qual a resposta correta para aquele conjunto de entradas. A Figura 4 demonstra graficamente o processo de correção de erros característico do aprendizado supervisionado. Nessa Figura, pode-se perceber o papel do Professor no treinamento: é ele quem mostra à rede o valor da saída desejada.

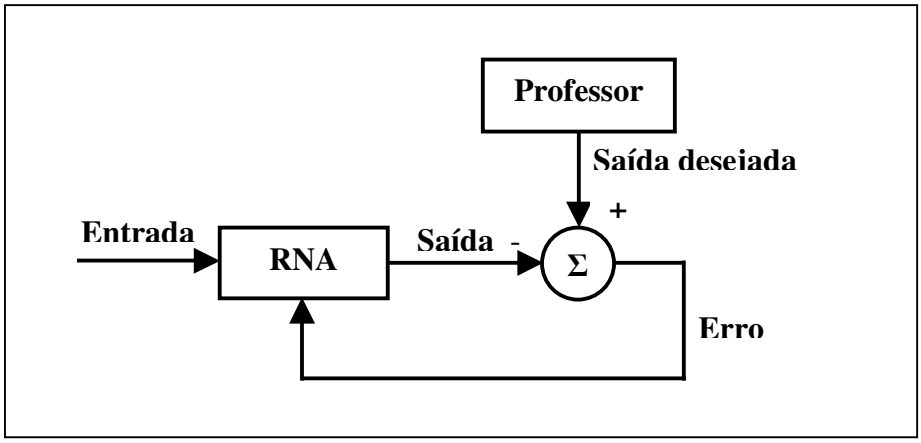


Figura 4. Aprendizado supervisionado

O grande problema no treinamento das redes MLP está no ajuste dos pesos dos neurônios das suas camadas intermediárias. Como seria calculado o erro destes nodos, já que, não existe uma saída desejada para nodos intermediários? Para solucionar este problema, foi proposto um método baseado em gradiente descendente [25], o *backpropagation* ou retro-propagação de erros.

Este capítulo tem por finalidade descrever o funcionamento das redes MLP (Multi-Layer Perceptron), suas características básicas, seu treinamento através do seu principal algoritmo de treinamento, o backpropagation e algumas aplicações nas quais foram utilizadas redes MLP.

2.1 Arquitetura

A estrutura de uma MLP consiste, basicamente, de uma camada de entrada, conhecida como camada sensorial, uma ou mais camadas intermediárias e uma camada de saída. A Figura 5 é a

ilustração de uma rede MLP de 4 camadas, sendo uma de entrada, duas intermediárias e uma de saída.

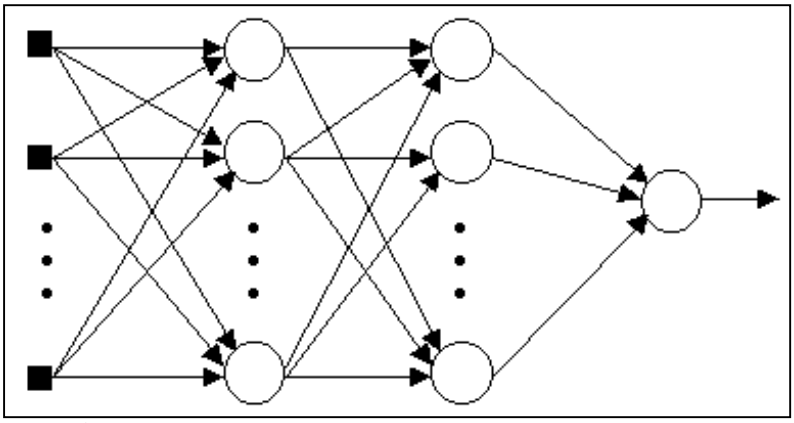


Figura 5. Exemplo de rede Multi-Layer Perceptron

Conforme discutido no princípio deste capítulo, redes MLP possuem um poder computacional muito maior do que redes de uma única camada, pois, elas podem tratar com dados não linearmente separáveis. A arquitetura da MLP (mais precisamente o número de camadas da mesma) é que vai decidir o quão poderosa computacionalmente é a rede. Isto é justificado pelo fato de que o processamento de cada neurônio da camada *j* é realizado sobre a combinação dos processamentos realizados pelos neurônios da camada anterior *j-1*. Ao percorrer as camadas da rede da primeira à última, aumentamos o nível de complexidade do processamento (as funções implementadas são cada vez mais complexas), uma vez que a última camada é alimentada com o resultado do processamento das demais camadas. Para uma rede com duas camadas intermediárias, o seguinte processamento ocorre em cada uma das camadas:

- **primeira camada intermediária**: nesta camada, os neurônios traçam retas no espaço de padrões de treinamento.
- **segunda camada intermediária**: os neurônios combinam as retas traçadas pela camada anterior em regiões convexas.
- **camada de saída:** as regiões convexas formadas pelos neurônios da segunda camada intermediária são combinados de modo a formar regiões com formato abstrato.

Percebe-se que quanto maior o número de camadas de uma MLP, mais complexa é a região espacial gerada no espaço de padrões de treinamento. Visando a obter o número de camadas intermediárias necessárias para a implementação das classes de uma rede neural artificial, surgiram algumas pesquisas importantes [5]. Tais pesquisas mostraram que:

- uma camada intermediária aproxima qualquer função contínua; e
- duas camadas intermediárias aproximam qualquer função matemática, sendo portanto suficientes para a implementação de qualquer problema de classificação.

Com relação ao número de neurônios usados em cada camada, estes são definidos empiricamente e apesar de existirem alguns métodos que se propõem a ajudar na decisão de quantos nodos utilizar por camada, não existe um método que justifique matematicamente a escolha. Sabe-se, no entanto, que o número de neurônios de cada camada, assim como o número

de camadas da rede, sua função de ativação, o número de conexões, o valor inicial dos pesos, entre outros, são fatores importantes na arquitetura da rede, pois, influenciam no resultado gerado [3][26][27].

2.2 Treinamento de Redes MLP

Existem basicamente duas classificações para os algoritmos de treinamento de redes MLP, eles podem ser estáticos, aqueles que alteram apenas o valor dos pesos da rede durante o treinamento, ou dinâmicos, que alteram também a estrutura da rede (camadas, nodos, conexões).

O algoritmo mais conhecido para treinamento de redes de múltiplas camadas é o *backpropagation* [12], ou retropropagação. Esse nome é justificado pelo fato do erro ser retropropagado durante o treinamento. O algoritmo backpropagation é um algoritmo estático e supervisionado, já que, para cada entrada é conhecida a saída esperada. A aprendizagem, então, ocorre pela correção de erros através do ajuste de pesos.

Basicamente, a aprendizagem por retropropagação de erros consiste em duas fases:

1. fase *forward* ou de propagação e
2. fase *backward* ou de retropropagação.

Na fase 1, um conjunto de sinais é submetido à camada de entrada e seu efeito é propagado através da rede passando por todas as camadas, até que, na saída, um resultado é fornecido. Nesse passo não há mudança no valor dos pesos sinápticos.

Na fase 2, ocorre o aprendizado por meio de ajuste de pesos para a correção do erro. O erro é calculado ao final da fase 1 pela subtração do resultado fornecido pela rede com o resultado conhecido (Equação 2) e propagado para trás, ou seja, no sentido inverso ao da propagação. As duas fases são mostradas na Figura 6.

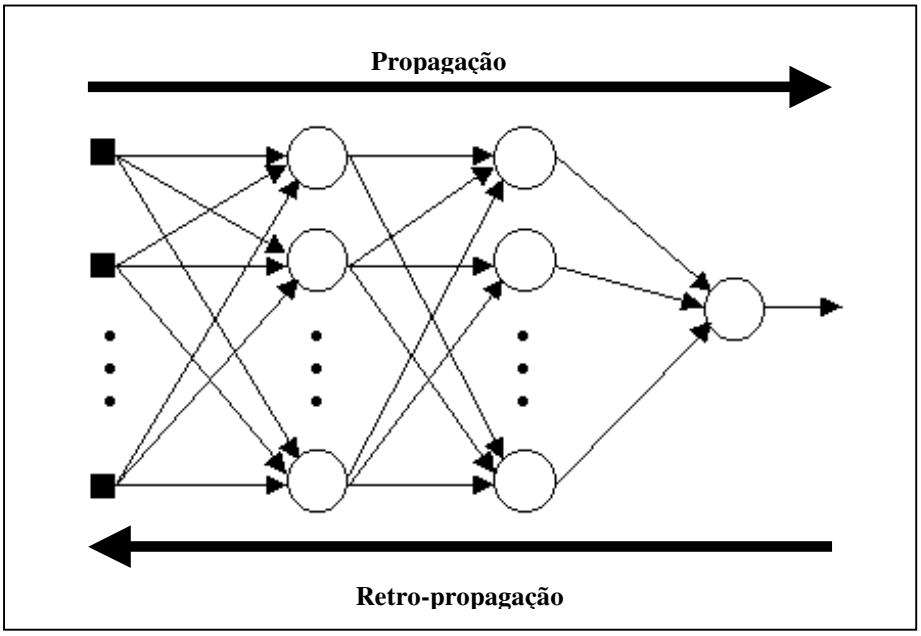


Figura 6. Fluxo das fases de propagação e retro-propagação do algoritmo *backpropagation*

O algoritmo backpropagation é mostrado a seguir:

```
1  Inicializar os pesos
2  Repetir até que algum critério de parada seja satisfeito:
     2.1   Para cada entrada:
           2.1.1 Encontrar a saída da rede através da fase de propagação.
           2.1.2 Comparar a saída da rede com a saída desejada.
           2.1.3 Ajustar os pesos através da fase de retropropagação.
```

2.3 Aplicações

Redes MLP têm sido empregadas em uma grande variedade de aplicações, tais como:

- Reconhecimento de caracteres: em [30], é apresentada uma aplicação baseada em rede backpropagation para reconhecimento de caracteres manuscritos;
- Previsão de índices de bolsas de valores: em [31], um modelo de redes neurais MLP é usado para realizar a previsão da bolsa de valores de Tókio;
- Detecção de fraudes em folhas de pagamento: em [20] utilizou-se redes MLP para prever os valores de duas verbas de uma folha de pagamento do governo e dessa forma obter as séries das verbas isentas de fraude. Qualquer desvio significativo das séries previstas indica fortemente uma fraude;
- Diagnóstico médico: o trabalho [32] compara os resultados obtidos a partir de algumas técnicas estatísticas e algumas baseadas em redes MLP quando aplicadas ao problema de diagnosticar se a paciente sobrevive ou não ao câncer de mama.

Capítulo 3

Previsão de Séries Temporais

Previsão de séries temporais é o processo de prever valores futuros de uma série temporal a partir do conhecimento de seus valores passados. Dentre as técnicas utilizadas para previsão de séries temporais estão aquelas baseadas em diferentes arquiteturas de Redes Neurais Artificiais (RNAs), como Multi-Layer Perceptron (MLP), FIR [31], TDNN [12] e Elman [7], entre outras. Este capítulo se destina a descrever o processo de prever séries temporais realizado por RNAs, assim como algumas técnicas, tais como TDNN, janela de tempo, Elman e Jordan, e também mostrar exemplos de aplicações onde a previsão de séries por RNAs foi aplicada.

3.1 Processamento Temporal

O tempo é um elemento que faz parte do cotidiano dos seres humanos e muitos dos problemas reais podem ser definidos em função dele. Isto faz deste um parâmetro importante na área de reconhecimento de padrões, tais como reconhecimento de voz, detecção de movimentos, processamentos de sinais, etc. As estruturas computacionais convencionais ainda não são ricas o suficiente para lidar com reconhecimento de padrões que mudam com o tempo, ou padrões dinâmicos, pois, não são capazes de representá-lo adequadamente [3]. Para isto são necessárias novas estruturas capazes de representar a evolução temporal dos padrões.

As redes MLP tradicionais não possuem a estrutura adequada para lidar com padrões dinâmicos, portanto, são necessários alguns ajustes em seu treinamento. Uma maneira de adaptar o treinamento é transformar os dados originalmente dinâmicos em dados estáticos através do uso de janelas de tempo.

No entanto, o uso de janelas de tempo não é o meio mais apropriado para realizar processamento temporal. O ideal seria fazer uso de estruturas adequadas a este tipo de processamento. Novos modelos arquiteturais foram desenvolvidos especificamente para o problema, é o caso das arquiteturas FIR, TDNN e Elman. Tais arquiteturas são usadas em diversas aplicações como, por exemplo, previsões financeiras [16][20].

As arquiteturas dinâmicas devem possuir memória para que os processamentos realizados em momentos anteriores sejam considerados [7]. Existem basicamente duas maneiras de prover memória a uma RNA, são elas:

1. Introduzir atraso de tempo, como fazem as técnicas de Janela de Tempo, TDNN (*Time Delay Neural Network*) [12] e FIR (*Finite Impulse Response*) [31].
2. Utilizar arquiteturas de redes recorrentes, como redes de Elman [7] e redes de Jordan [13].

3.2 Atraso no Tempo

3.2.1 MLP com janela de tempo

As redes MLP foram originalmente concebidas para executar tarefas de natureza estática, não foram, portanto, idealizadas para tratar problemas temporais. O método de janela de tempo foi a primeira adaptação da rede MLP treinada com *backpropagation* (ou algoritmos derivados deste) para processamento dinâmico.

Para adaptar as redes MLP à tarefas dinâmicas, já que estas não apresentam uma arquitetura específica para lidar com esse tipo de problema, é necessário introduzir o atraso no tempo através dos padrões de entrada. A esta técnica que introduz atraso nos padrões de entrada chamamos janela de tempo.

Quando queremos usar redes neurais MLP para previsão de séries temporais, definimos como entrada um conjunto de valores passados ordenados no tempo e como saída o valor posterior a esta seqüência [11]. Para tanto, é necessário realizar um processamento na série original, que é feito através do modelo descrito em [2]:

$$Z^{\wedge}(t) = f(Z(t-1), ..., Z(t-p)) \qquad \textbf{(Equação 4)}$$

onde Z(t) é o ponto da série correspondente ao tempo t, p é o tamanho da janela de tempo e Z^(t) é a nova série gerada. Percebe-se que a série gerada terá p pontos a menos que a série original.

Tomemos como base a série Z(1), Z(2), Z(3), Z(4), ..., Z(n), com o tempo variando de 1 a *n*. Durante o treinamento, a idéia é fornecer como entradas valores sucessivos da série, por exemplo, os valores de Z(1), Z(2) e Z(3), e definir que a saída desejada é o valor Z(4). Na segunda iteração, os valores das entradas seriam Z(2), Z(3) e Z(4) e a saída desejada Z(5) e assim sucessivamente para o restante dos pontos da série (ver Tabela 1). O número de valores passados usados como entradas é chamado de ordem de linha atraso ou janela. No exemplo citado, usamos 3 valores para prever o próximo, logo, a janela foi de ordem 3. A arquitetura mostrada na Figura 7 é um exemplo de MLP que poderia ser aplicado a esse problema.

Tabela 1. Entradas e saídas desejadas para o treinamento de MLPs

Entradas	Saída desejada
Z(1), Z(2), Z(3)	Z(4)
Z(2), Z(3), Z(4)	Z(5)
Z(3), Z(4), Z(5)	Z(6)
Z(4), Z(5), Z(6)	Z(7)
...	...
Z(n-3), Z(n-2), Z(n-1)	Z(n)

Ainda sobre a Figura 7, observa-se que o número de unidades de entrada varia de acordo com o tamanho da janela adotado (no exemplo citado o tamanho da janela foi 3). Como acontece

em redes MLP convencionais, os nós da camada de entrada não realizam nenhum processamento sobre os padrões de entrada. As camadas intermediárias podem ou não utilizar funções lineares, comumente são usadas funções não-lineares, tais como a *tangente hiperbólica* e *sigmóide logística*. Quanto às unidades da camada de saída (no exemplo citado, existe apenas uma), é apropriado que façam uso de funções de ativação lineares, pois, elas vão gerar o resultado previsto pela rede e não é interessante que este valor seja limitado entre 0 e 1, como ocorre, por exemplo, quando a função *sigmóide logística* é usada.

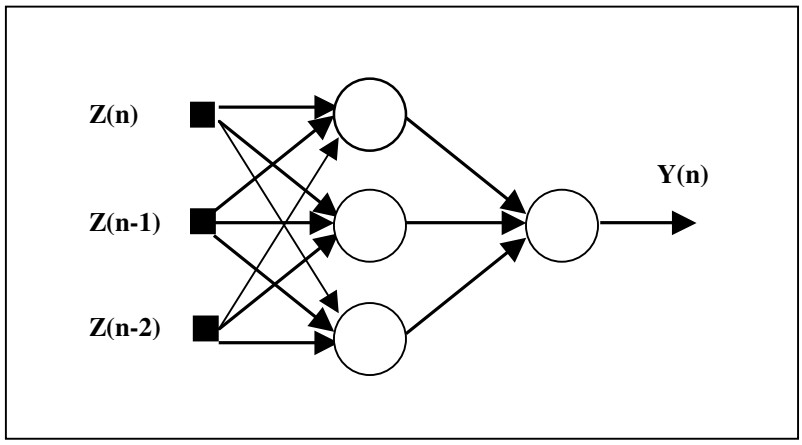


Figura 7. Exemplo de MLP com janela de tempo 3

Note que o processamento temporal é realizado apenas nos padrões de entrada, logo, é necessário eliminar tendência e sazonalidade tornando a série estacionária, de modo que suas propriedades estatísticas, como variância e média, variem pouco com o tempo. Tal passo é conhecido como diferenciação. Séries estacionárias são mais fáceis de prever, uma vez que, suas propriedades estatísticas futuras serão as mesmas do passado.

3.2.2 Time Delay Neural Networks

Redes com atraso no tempo ou TDNNs (*Time Delay Neural Networks*) são estruturas bem conhecidas para realizar processamento temporal. Trata-se de redes de múltiplas camadas cujos neurônios das camadas intermediária e de saída são replicados ao longo do tempo.

A topologia da rede TDNN é basicamente a de uma rede MLP onde cada conexão possui uma memória associada ou filtro FIR (*Finite Impulse Response*) [12]. O treinamento dessa rede cria, a cada iteração, uma rede estática equivalente através do desdobramento da rede no tempo. Essa etapa do treinamento introduz a característica temporal à rede neural, pois, a cada iteração, os sinais fornecidos na iteração anterior como entrada para as camadas oculta e de saída são considerados.

Ao replicar as camadas, um modelo dinâmico de RNA é criado. Para treinar esse novo modelo, usa-se um algoritmo de aprendizagem supervisionado em que a cada instante de tempo a saída desejada é comparada com a resposta da rede. Um algoritmo destinado ao treinamento dessas redes é o *backpropagation* temporal [3].

3.3 Redes Recorrentes

3.3.1 Redes de Elman e Jordan

As redes de Elman são redes parcialmente recorrentes especialmente criadas para processamento temporal [15]. Estas redes possuem um diferencial em relação às redes MLP: elas possuem uma camada de contexto além das camadas de entrada, intermediárias e saída convencionais. As unidades da camada de contexto são a memória da rede. Através delas a rede introduz um atraso unitário no tempo. O valor de saída das unidades intermediárias são armazenados pelas unidades de contexto e reutilizadas pelas unidades de entrada da rede.

A Figura 8 mostra uma rede de Elman. Nela é destacado o fluxo de ativação das camadas. Em um determinado tempo *t*, o resultado da camada intermediária gerado em *t*-1 e as unidades de entrada correntes são utilizadas como entradas da rede. Na primeira iteração *feedforward* do treinamento estas entradas são propagadas através da rede para produzir uma saída. A rede é então treinada (*backward*) com o algoritmo *backpropagation* padrão (ou variações deste). Após essa primeira iteração, as ativações produzidas pela camada intermediária no tempo *t* são memorizadas pelas unidades de contexto e reutilizadas no tempo *t*+1 do treinamento, como indicado pelo fluxo demonstrado na Figura.

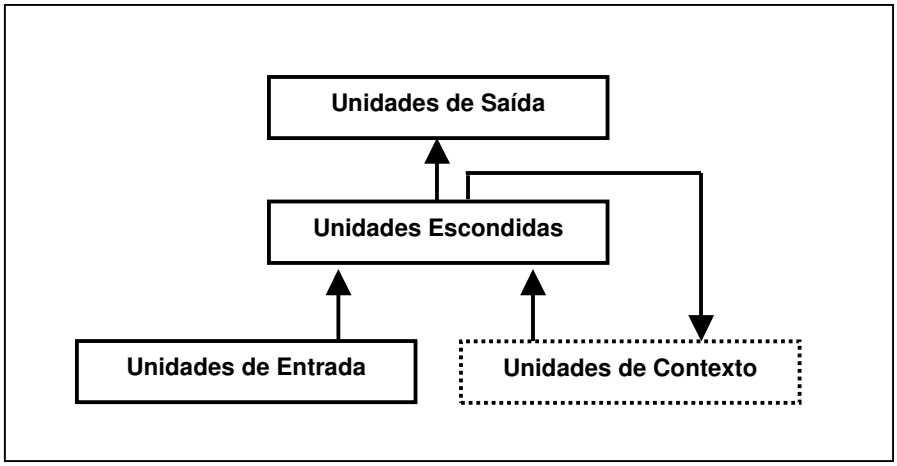


Figura 8. Ilustração de uma Rede de Elman

As unidades de entrada consistem apenas em unidades de armazenamento e não modificam os sinais recebidos como entrada. As unidades de saída utilizam funções de ativação lineares, já que estas vão fornecer os valores previstos e não seria interessante limitar esses valores, como acontece, por exemplo, quando usamos a função sigmoidal logística cujos valores são limitados entre 0 e 1. As unidades intermediárias podem utilizar ou não funções lineares. Por fim, as unidade de contexto atuam apenas como uma memória, realimentando a rede com seus valores passados de ativação da camada intermediária.

Praticamente, tudo o que foi dito em relação às rede de Elman se aplica às rede de Jordan [13] (Figura 9). A diferença entre elas é que as redes de Jordan armazenam os resultados da camada de saída em sua camada de contexto, ao contrário das redes de Elman, que armazenam, na sua camada de contexto, os resultados da sua camada intermediária. Enquanto a recorrência da rede de Elman é feita da camada intermediária para a de entrada, a recorrência da rede de Jordan é feita da camada de saída para a de entrada.

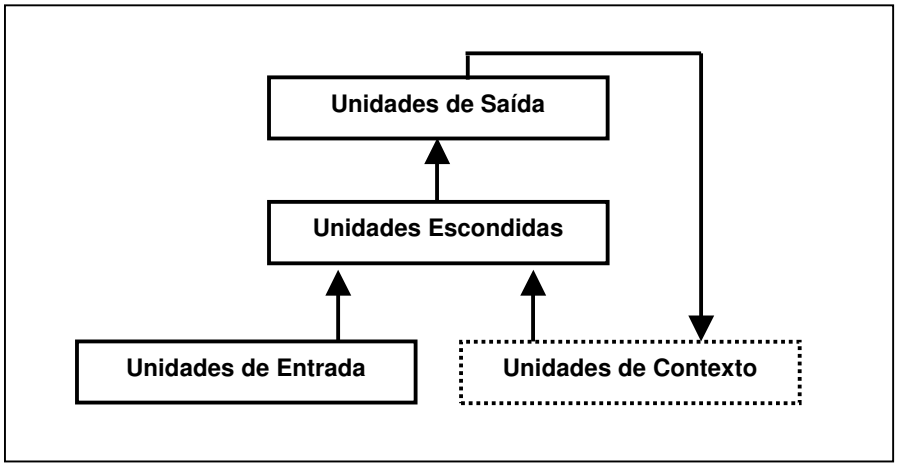


Figura 9. Ilustração de uma Rede de Jordan

3.4 Aplicações

Dentre as várias aplicações tratadas com técnicas de previsão de séries temporais, estão:

- detecção de fraudes em folhas de pagamento[20]. Sistema de suporte à auditoria em folhas de pagamento já descrito no Capítulo2 ;
- detecção de fraudes em cartões de crédito [6];
- previsões financeiras, como, por exemplo, a já citada previsão de índices da bolsa de valores de Tóquio [31].

Capítulo 4

Otimização Global de Redes Neurais Artificiais

Atualmente, existe um grande esforço na área de Inteligência Artificial no sentido de desenvolver sistemas híbridos, aqueles que envolvem mais de uma técnica na resolução de um problema. Esse esforço é motivado pela necessidade de complementar alguns aspectos de determinada técnica com aspectos de outra técnica, cada qual eficaz para um determinado tipo de problema. Existem técnicas que apesar de serem bastante eficientes para alguns casos, apresentam algumas limitações quando aplicadas a outros. É visando superar essas limitações que pesquisadores procuram combinar os pontos fortes de duas ou mais técnicas numa única abordagem híbrida.

Neste trabalho, a ênfase é dada em técnicas de redes neurais artificiais e otimização global. Os métodos de otimização aqui estudados são *Simulated Annealing* [14] e *Tabu Search* [9][10]. Tais métodos juntamente com treinamento de redes MLP foram recentemente aplicados na classificação de odores por um nariz artificial [26][27]. Vale destacar que a abordagem (técnica de otimização e redes neurais MLP) dos trabalhos citados não foi aplicada a problemas de previsão, o que é proposto neste trabalho de conclusão de curso.

4.1 Otimização global de arquiteturas e pesos

Uma rede neural é treinada com a finalidade de minimizar a função de erro através do ajuste dos valores de seus pesos. Como foi visto no Capítulo sobre Multi-layer Perceptron (Capítulo 2), para treinar uma MLP, o algoritmo mais conhecido é o *backpropagation* [12]. Apesar do *backpropagation* ser bem sucedido em várias aplicações, ele apresenta em alguns casos uma deficiência chamada de convergência local, em outras palavras, o algoritmo se prende a mínimos locais da função de erro.

Algoritmos de gradiente descendente, como o *backpropagation*, são considerados métodos de convergência local, pois, utilizam-se de informações sobre o gradiente da função de erro, que são informações locais, para se aproximar de um ponto de mínimo dessa função. Dessa forma, o *backpropagation* é excelente para determinar qual a direção e o valor de ajuste dos pesos adequados para caminhar de um ponto no espaço em direção ao mínimo local iterativamente [27].

Ao contrário do *backpropagation*, os algoritmos de otimização abordados neste trabalho, *tabu search* e s*imulated annealing*, são considerados algoritmos de convergência global, pois, buscam no espaço de erro o ponto de mínimo global baseados apenas em informações sobre a superfície da função [28]. Por não se basear em informações do gradiente para convergir, algoritmos de otimização global são aplicáveis a quaisquer funções, sejam elas diferenciáveis ou não (ao contrário do *backpropagation*). A Figura 10 ilustra a diferença entre os mínimos locais e o mínimo global.

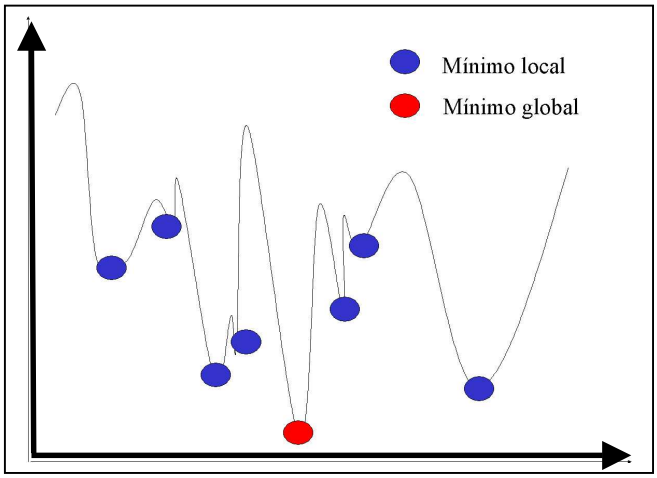


Figura 10. Diferença entre mínimos locais e mínimo global

A união de métodos de otimização global com métodos de convergência local é uma abordagem interessante para a otimização de pesos de uma RNA, uma vez que, o método de otimização age em busca de uma melhor região no espaço global e o método de convergência local faz um ajuste mais refinado dos pesos da rede para que esta chegue ao ponto mínimo daquela região.

Não só o ajuste dos pesos, mas também a topologia escolhida para o treinamento de uma rede é determinante no seu desempenho. Caso a topologia escolhida apresente uma pequena quantidade de nodos e conexões, a rede pode não aprender satisfatoriamente, dada a quantidade de parâmetros ajustáveis. Caso contrário, ou seja, no caso em que o número de nodos e conexões é muito grande, a rede pode acabar se adaptando excessivamente aos padrões específicos de treinamento (*overfitting*) fazendo com que perca sua capacidade de generalização quando submetida a novos padrões.

Apesar de ser fundamental para o sucesso da aplicação, a escolha da topologia é geralmente feita de maneira subjetiva através de métodos de tentativa e erro. A fim de facilitar este dispendioso processo de tentativas, faz-se necessário um meio automático de escolha da melhor topologia. Esta necessidade motiva o estudo de técnicas de otimização de arquiteturas.

O problema de otimização de arquiteturas também pode ser formulado como um problema de busca no espaço, onde cada ponto representa uma arquitetura [28]. Dada uma função de custo, como, por exemplo, o erro de treinamento combinado com a complexidade da rede (número de conexões), a solução é encontrada através da busca do ponto da função cujo custo seja mínimo. Como vimos, o uso de técnicas de otimização global, como as já citadas neste Capítulo, é perfeitamente adaptável para a resolução deste problema de busca no espaço.

É importante salientar que a metodologia para otimização de arquiteturas escolhida deve garantir que apenas redes válidas sejam consideradas, isto é, apenas aquelas com pelo menos uma unidade na camada escondida.

Porém, a otimização da arquitetura apenas, sem o registro de seus pesos, não representa muita coisa, já que, o aprendizado da rede está no valor de seus pesos. Uma maneira de lidar com esta limitação é combinar otimização de arquiteturas e pesos simultaneamente, como propõe este trabalho.

Existem muitos outros parâmetros ajustáveis no treinamento de RNAs e de acordo com o algoritmo escolhido o número de parâmetros pode crescer bastante. No entanto, o foco deste trabalho é a otimização apenas de arquiteturas e pesos.

4.2 Simulated Annealing

O método *simulated annealing* [14] é inspirado nos processos de esfriamento de sólidos que alcançam energia mínima, correspondente a uma estrutura cristalina perfeita, se esfriados de forma suficientemente lenta [21].

Fazendo uma analogia ao processo *annealing*, os sólidos seriam equivalentes às possíveis soluções de rede neural, a energia equivaleria ao custo da solução e a estrutura cristalina perfeita é a representante da solução ótima de rede neural.

O algoritmo *simulated annealing* consiste em, a cada iteração, gerar uma nova solução a partir de uma solução atual. Sempre que uma nova solução é gerada, a sua função de custo é computada para decidir se essa nova solução pode ser aceita como atual. Se o custo da nova solução for menor que o da atual solução, a nova solução é imediatamente aceita como atual. Caso contrário, a nova solução pode ou não ser aceita de acordo com uma certa probabilidade, o *critério de Metrópolis* [19]. De acordo com esse critério, é gerado um número aleatório δ entre 0 e 1. Se $\delta \leq e^{(-\Delta E / T)}$, onde ΔE é a variação do custo e T é o parâmetro chamado temperatura, então a nova solução é aceita como solução atual. Senão a solução é desconsiderada e o processo continua gerando novas redes a partir da solução atual.

Esquemas de esfriamento são responsáveis por especificar um valor de temperatura inicial e também por atualizar o valor da temperatura de acordo com alguma regra. Existem diversas propostas de esquemas de esfriamento na literatura. O esquema de esfriamento mais comumente utilizado é o geométrico, no qual a nova temperatura é calculada pelo valor da temperatura atual multiplicado por um fator de redução [21].

Considerando o conjunto de soluções S como sendo o conjunto de todas as possíveis soluções e f como sendo a função de custo, o algoritmo objetiva encontrar a solução global s, tal que $f(s) \leq f(s')$, $\forall\ s' \in S$. O algoritmo pára após I iterações, definidas pelo usuário. Segue sua descrição [12]:

1. $s_0 \leftarrow$ solução inicial em S
2. Para i=0 até I-1
3. Gera solução s'
4. Se $f(s') < f(s_i)$
5. $s_{i+1} \leftarrow s'$
6. senão
7. $s_{i+1} \leftarrow s'$ com probabilidade $e^{-[\,f(s') - f(s_i)\,]\,/\,T_{i+1}}$
8. Atualiza temperatura
9. Retorna s_I

4.3 Tabu Search

O algoritmo *tabu search* [9][10] também é um algoritmo de busca no espaço. Nele, um conjunto de novas soluções é gerado a partir da solução atual e a melhor (cuja função de custo é a menor) dentre elas é sempre aceita como atual. O fato de a nova solução ser sempre aceita como solução atual, impede que o algoritmo se prenda em mínimos locais

Para evitar ciclos na trajetória de busca, as soluções mais recentemente visitadas são armazenas em uma lista "tabu". Essas soluções são proibidas impedindo que sejam novamente tomadas como solução pelas próximas iterações. Essa tática é interessante, pois, pode haver o retorno à soluções já visitadas, o que implicaria em ciclos na trajetória.

Como não é computacionalmente viável o armazenamento de todas as soluções já visitadas, o que se faz é armazenar apenas um número K de soluções, chamado comprimento ou tamanho da lista tabu. Dessa forma, a lista tabu registra as K últimas soluções encontradas. Quando o tamanho máximo da lista é atingido, a solução mais antiga na lista é removida para dar lugar à nova solução. A lista funciona como uma fila, *first-in-first-out* (FIFO).

Uma observação importante é que o *tabu search* mantém em memória a melhor solução de toda a execução, independentemente de ser a solução atual. Assim, mesmo que ele "passe" a solução global e aceite uma nova solução em seu lugar, a melhor rede é conservada.

Considerando o conjunto de soluções S como sendo o conjunto de todas as possíveis soluções e f como sendo a função de custo, o algoritmo objetiva encontrar a solução global s, tal que $f(s) \leq f(s')$, $\forall\ s' \in S$. O algoritmo pára após I iterações definidas pelo usuário e retorna a melhor solução obtida durante a execução s_{best}. Segue sua descrição:

```
1. s0 ← solução inicial em S
2. sbest ← s0
3. Insere s0 na lista tabu
4. Para i=0 até I-1
5.    Gera V soluções novas
6.    Escolhe a melhor solução s' que não está na lista tabu
7.    si+1 ← s'
8.    Insere si+1 na lista tabu
9.    Se f(si+1) < f(sbest)
10.         sbest ← si+1
11.   Retorna sbest
```

Capítulo 5

Otimização Simultânea de Arquiteturas e Pesos de Redes MLP

Como comentado no Capítulo sobre otimização global de RNAs (Capítulo 4), este trabalho utiliza-se de uma metodologia de otimização simultânea de arquitetura e pesos de redes MLP, para aplicações de previsão de séries temporais. Como explicado naquele Capítulo, cada rede é mapeada em um ponto no espaço de busca, logo, quando encontrada uma solução no espaço, implica que a topologia e o valor dos pesos ótimos foram encontrados. No decorrer deste Capítulo, a metodologia utilizada no treinamento das redes otimizadas, que combina métodos de treinamento global e local, será descrita com maiores detalhes. Essa metodologia foi proposta por Yamazaki em seu trabalho [27].

Também é objetivo deste trabalho comparar os resultados obtidos entre as redes MLP otimizadas e as que não passaram pelo processo de otimização. Este Capítulo mostrará como foram feitos os experimentos com as redes MLP treinadas de modo convencional, não otimizadas.

Além das metodologias de treinamento, é descrita a metodologia empregada no tratamento dos padrões de entrada. Alguns processamentos são necessários para que as bases de dados estejam prontas para serem utilizadas como entradas de um treinamento de RNAs. No que diz respeito às bases de dados temporais, um esforço extra é requisitado de forma a tornar as séries estacionárias, facilitando a previsão.

5.1 Metodologias de treinamento com otimização

Para implementar as técnicas de otimização simultânea de pesos e arquiteturas de redes MLP, alguns aspectos precisam ser bem definidos:

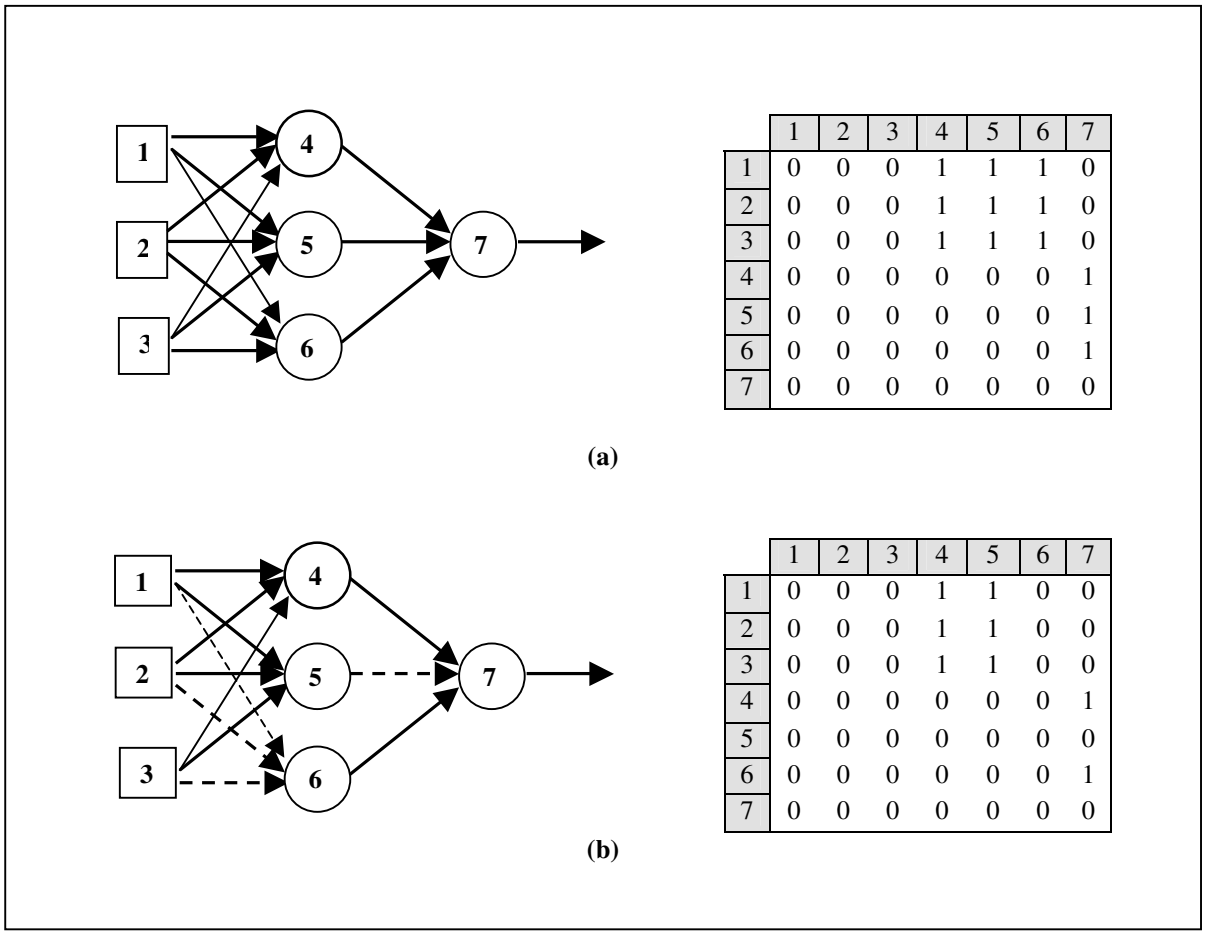
1. como será representada a solução;
2. qual será a função de custo considerada;
3. como funcionará o mecanismo de geração de novas soluções;
4. o esquema de esfriamento adotado (para o *simulated annealing*);
5. quais serão os critérios definidos para a parada do treinamento; e
6. qual será o método de treinamento local.

Tais aspectos são detalhados a seguir tanto para o método *simulated annealing*, quanto para o método *tabu search*.

5.1.1 Metodologia de treinamento com S*imulated Annealing*

As redes utilizadas neste trabalho são as redes MLP de única camada intermédiaria e totalmente conectada, ou seja, existem todas as conexões possíveis entre suas camadas adjacentes. O número de nodos das camadas de entrada, intermediária e de saída são definidos de acordo com o problema atacado. No caso de pré-processamento temporal para arquiteturas MLP com janela de tempo, o número de entradas e o número de saídas vai variar de acordo com o tamanho da janela e o número de pontos futuros da série que se quer prever.

Neste trabalho, cada solução da MLP é definida por dois parâmetros: a matriz de seus pesos (valores de números reais) e a matriz de conectividades (valores binários). A conectividade nada mais é do que a representação de uma conexão existente, que é feita através de um bit de conectividade que vale 1, se a conexão existir, e 0, caso contrário. Dessa forma, se a conexão não existir, seu peso é desconsiderado durante o treinamento. A partir dessa definição do bit de conectividade é possível manipular não só o valor de uma conexão como também sua existência. A Figura 11 apresenta exemplos dessas matrizes.



	1	2	3	4	5	6	7
1	0	0	0	1	1	1	0
2	0	0	0	1	1	1	0
3	0	0	0	1	1	1	0
4	0	0	0	0	0	0	1
5	0	0	0	0	0	0	1
6	0	0	0	0	0	0	1
7	0	0	0	0	0	0	0

	1	2	3	4	5	6	7
1	0	0	0	1	1	0	0
2	0	0	0	1	1	0	0
3	0	0	0	1	1	0	0
4	0	0	0	0	0	0	1
5	0	0	0	0	0	0	0
6	0	0	0	0	0	0	1
7	0	0	0	0	0	0	0

Figura 11. (a) MLP totalmente conectada e sua respectiva matriz de conectividade; **(b)** MLP parcialmente conectada e sua respectiva matriz de conectividade.

A Figura 11 (a) mostra como é representada uma rede neural do tipo MLP *feedforward* totalmente conectada em uma matriz de conectividade entre as unidades. Na matriz, as linhas representam o nodo de origem da conexão e as colunas o nodo de destino. Note que onde há conexão entre o nodo origem e o destino, o valor binário 1 é associado, caso não exista, o valor 0 consta representando esta situação. Ainda na Figura 11, em (b), está exemplificada a mesma rede MLP descrita imediatamente acima, agora com algumas desconexões. Com esta abordagem binária é simples avaliar se uma conexão existe, assim como ativar/desativar conexões.

Percebe-se ainda, através da Figura 11 (b), que com a eliminação de algumas conexões, os neurônios 5 e 6 passaram a não influenciar mais no treinamento de forma que a rede passou a ter apenas o neurônio 4 na camada intermediária. Um cuidado que deve ser tomado é o de manter pelo menos um neurônio ativo na camada intermediária. O algoritmo implementado deve garantir esta condição.

A função de custo adotada neste trabalho foi o EQM (*erro quadrático médio*), que mede o erro quadrático médio entre os valores desejados e os obtidos pela rede. Dessa forma, o algoritmo procura minimizar o EQM para a obtenção da solução.

A Equação 5 demonstra o cálculo de EQM. Nela i é o número de unidades de saída e p o número de padrões.

$$E = \frac{1}{2}\sum_{p}\sum_{i}(d_i^{(p)} - y_i^{(p)})^2 \qquad \textbf{(Equação 5)}$$

O mecanismo de criação de novas redes funciona como descrito a seguir: com base na solução atual, cada conexão terá seu bit de conectividade alterado com uma certa probabilidade, ou seja, o que está conectado, desconecta e vice-versa. Basicamente o que este mecanismo faz é criar um valor aleatório no intervalo [0, 1] e verificar se este valor é menor que a probabilidade estipulada. Se for este o caso, o bit de conectividade é invertido, caso contrário nada é feito. O valor de probabilidade usado neste trabalho foi de 20%[26]. Além de inverter o valor de conectividade, este mecanismo também modifica o valor dos pesos vinculados às conexões. Os valores dos pesos da nova rede são calculados a partir da soma de um valor aleatório (distribuição uniforme) em [-1.0, +1.0] com os pesos da solução atual.

Como foi citado em capítulo anterior, a escolha de um esquema de esfriamento é muito importante na implementação do *simulated annealing*. Um esquema de esfriamento deve definir um valor de temperatura inicial, assim como uma regra para atualização da mesma. Um dos esquemas mais comuns de esfriamento disponíveis na literatura é o esquema de esfriamento geométrico, onde a nova temperatura é calculada pela multiplicação da temperatura atual por um fator de redução [10], que deve ser menor do que 1 e próximo a 1. Neste trabalho, o valor inicial da temperatura foi definido como 1 e o fator de redução 0.9. A cada 10 iterações a temperatura é decrementada.

O algoritmo pára se o critério GL5 descrito no Proben1 [22] for satisfeito ou se o número máximo de 10.000 iterações for atingido. O critério GL5 avalia a capacidade de aprendizado da rede. Sabe-se que o excesso de treinamento ocasiona um fenômeno chamado de *overfitting*, ou a super adaptação dos parâmetros da rede aos padrões de treinamento, o que não é desejável. O que se pretende ao treinar uma rede é que ela seja capaz de aprender com os dados aos quais foi submetida durante o treinamento e possa, a partir de então, generalizar o conhecimento adquirido na classificação de novos padrões. O GL5 interrompe o treinamento caso avalie que a capacidade de generalização caiu mais de 5%.

O GL5 é descrito matematicamente a seguir. Seja E a função de erro, $E_{va}(t)$ o erro de validação no momento t e $E_{opt}(t)$ o menor erro de validação obtido até o momento t, como descrito na Equação 6.

$$E_{opt}(t) = \min_{t' \leq t} E_{va}(t') \qquad \textbf{(Equação 6)}$$

Podemos definir a perda de generalização no momento *t* como sendo o crescimento relativo do erro de validação sobre o erro mínimo, em percentual, como mostra a Equação 7.

$$GL(t) = 100 \cdot \left(\frac{E_{va}(t)}{E_{opt}(t)} - 1 \right) \qquad \textbf{(Equação 7)}$$

Como já mencionado, o critério GL5 pára o treinamento se o limite de 5% for alcançado, ou seja, se GL(*t*) > 5%.

Foi visto que uma técnica de otimização local seria combinada às técnicas de otimização global para o maior refinamento nas buscas. Isto foi implementado da seguinte forma: após a escolha da melhor rede em termos globais, esta seria treinada com um algoritmo local, compondo o que chamamos anteriormente de um sistema híbrido.

Neste trabalho, o algoritmo de aprendizagem local escolhido foi o RPROP (*Resilient Backpropagation*) [23], uma variação do algoritmo *backpropagation* [12]. Logo, o sistema consta de uma primeira etapa onde o *simulated annealing* escolhe globalmente uma rede e de uma segunda etapa onde os pesos são ajustados buscando uma maior aproximação do mínimo local. A escolha do algoritmo RPROP para atuar como método de convergência local é uma contribuição deste trabalho, pois, este ainda não havia sido sido utilizado na abordagem original proposta por Yamazaki [27].

Ficaram então definidos os principais aspectos considerados na implementação do sistema de otimização através de *simulated annealing*. Como veremos a seguir, a maioria deles se repetirá para o também algoritmo de otimização global *tabu search*.

5.1.2 Metodologia de treinamento com *Tabu Search*

Assim como o *simulated annealing*, o algoritmo *tabu search* também foi aplicado a redes MLP totalmente conectadas de apenas uma camada intermediária. Da mesma forma, a função de custo, representação das soluções, mecanismo de geração de novas redes, critério de parada e métodos de treinamento local foram os mesmos descritos acima.

Vimos no Capítulo 4 que o tabu search utiliza-se uma lista tabu de tamanho *K* para evitar ciclos na sua trajetória de busca. Na implementação de tabu search deste trabalho, o tamanho máximo da lista tabu foi definido em 10 [26].

Conforme citado no mesmo Capítulo 4, As soluções armazenadas na lista tabu são proibidas e caso coincida de uma nova solução ser igual a uma das *K* pertencentes à tabu, ela não será aceita. Como neste trabalho serão usados valores reais para a definição dos pesos, é praticamente impossível que dois valores coincidam exatamente. Por isso, usaremos o critério da proximidade citado e implementado em [26]. De acordo com este critério, duas soluções são consideradas iguais se: os bits de conectividade das conexões equivalentes das redes em questão forem os mesmos, e se o valor dos pesos de cada conexão da nova solução estiver dentro do intervalo [*C* + *N*, *C* - *N*], onde *C* é o peso correspondente à mesma conexão na solução da lista tabu e *N* um número real. Foi adotado aqui que *C* estaria no intervalo [-1.0, +1.0] e o valor de *N* adotado foi 0.1.

5.2 Metodologia de treinamento sem otimização

Como descrito no Capítulo 3, uma das técnicas mais populares para previsão de séries temporais é a baseada em redes MLP e chama-se janela de tempo. Conforme discutido naquele Capítulo, redes MLP não foram originalmente criadas com a intenção de processar problemas temporais, sua arquitura não foi modelada com esse propósito. Porém processamento temporal é uma aplicação bastante utilizada nos dias atuais e algumas técnicas baseadas em MLP foram criadas (treinamento com atraso no tempo) e até novos modelos foram propostos para lidar com as limitações das redes MLP tradicionais (Elman, Jordan, etc).

A técnica de janela de tempo destaca-se por ser a mais simples dentre as técnicas de atraso no tempo. Ela introduz o atraso no tempo através dos padrões de entrada. Por sua simplicidade, esta técnica foi adotada para a realização dos experimentos com redes MLP neste trabalho.

Durante o treinamento, são fornecidas como entradas valores sucessivos da série, por exemplo, os meses de janeiro, fevereiro e março e é definido que a saída desejada é o valor do mês de abril. Na segunda iteração, os valores das entradas seriam fevereiro, março e abril e a saída desejada maio e assim por diante para o todos os dados restantes. A janela utilizada neste trabalho foi de ordem 4, ou seja, são usados 4 pontos passados da série para prever o próximo ponto. O tamanho da janela é equivalente ao número de neurônios na camada de entrada, logo, o foram utilizadas 4 unidades de entrada.

O tamanho da janela influencia ainda o tamanho do conjunto de dados previsto, na próxima Seção será visto como isto ocorre mais detalhadamente.

5.3 Pré-processamento das bases de dados

Como já vimos, previsão de séries temporais exige uma série de processamentos iniciais com o conjunto de dados de entrada para introduzir atraso no tempo, tornar a série estacionária e normalizar seus valores. O primeiro processamento necessário é o de definir qual será a janela *p* e sua saída desejada. A partir dessa organização, uma nova série vai ser gerada com um menor número de pontos que a série original. A Equação 8 é a mesma descrita na Equação 4 e ilustra como se dá esse processamento inicial:

$$Z^{\wedge}(t) = f(Z(t\text{-}1), ..., Z(t\text{-}p)) \qquad \textbf{(Equação 8)}$$

nela Z(*t*) é o ponto da série correspondente ao tempo *t*, *p* é o tamanho da janela de tempo e Z^(*t*) é a nova série gerada. Percebe-se que a série gerada terá *p* pontos a menos que a série original.

Como já citado, é importante eliminar tendência e sazonalidade tornando a série estacionária, de modo que as suas informações estatísticas variem pouco com o tempo. Isto facilita bastante a previsão das séries, pois, suas propriedades estatísticas futuras serão as mesmas do passado. Para tornar séries estacionárias, aplica-se a técnica de diferenciação. A partir de diferenciação é possível gerar os correlogramas relativos às séries e, dessa forma, avaliar visualmente se existe tendência ou sazonalidade, ou ainda perceber se a série já é estacionária.

O próximo passo é normalizar os dados entre 0 e 1 segundo a expressão

$$X_{norm} = \frac{X - X_{\min}}{X_{\max} - X_{\min}} \qquad \textbf{(Equação 9)}$$

onde x_{max} e x_{min} correspondem respectivamente aos valores máximo e mínimo de um padrão x da série, e x_{norm} é a série normalizada resultante.

Uma vez realizados os passos acima citados, deve-se dividir a base de dados em conjunto de treinamento, validação e teste, de modo que a ordem temporal seja respeitada, ou seja, os dados mais antigos devem ser usados pra treinamento e os mais recentes para teste [15].

Capítulo 6

Experimentos e Resultados

Este Capítulo se destina a descrever os experimentos realizados relativos a previsão de séries temporais com redes neurais MLP treinadas com o algoritmo RPROP e com as duas abordagens híbridas, já descritas no Capítulo 5, uma envolvendo o uso combinado de *simulated annealing* e RPROP e outra combinando *tabu search* com RPROP. O uso combinado dessas abordagens é uma importante contribuição deste trabalho.

O objetivo deste Capítulo é mostrar que com o uso de métodos de otimização, o desempenho das previsões realizadas pela rede usando as séries estudadas é eficiente. Maiores informações sobre como foram realizados os experimentos e sobre as séries temporais usadas para este estudo serão expostas ao longo deste Capítulo.

6.1 Experimentos com a série NYWater

A série NYWater pode ser encontrada no repositório público de séries temporais, *Time Series Data Library*, cuja URL é http://www-personal.buseco.monash.edu.au/~hyndman/TSDL/. Esta série representa o uso anual de água na cidade de Nova Iorque medida em litros *per capita*. A série normalizada possui 69 valores coletados entre os anos de 1899 e 1967. O tamanho pequeno da série deve ser levado em consideração, pois é difícil realizar previsão com séries tão pequenas. Na Figura 12 é apresentada a série NYWater normalizada entre 0 e 1.

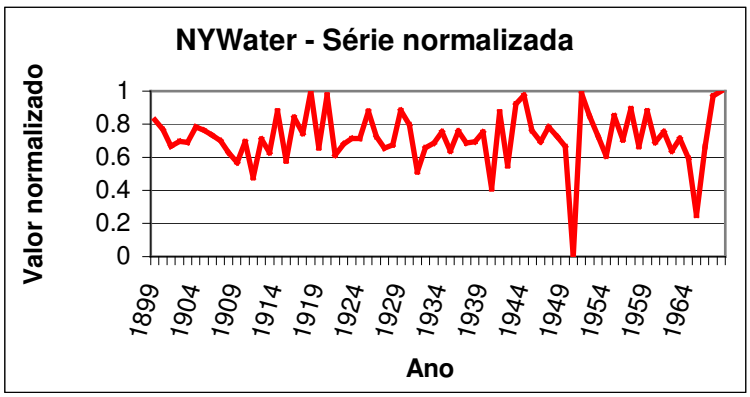


Figura 12. Série NYWater normalizada entre 0 e 1

6.1.1 Topologia e algoritmo de treinamento

Inicialmente, foram utilizadas redes MLP totalmente conectadas com uma camada intermediária contendo 4 unidades tanto para os experimentos com redes MLP treinadas com janela de tempo quanto com as treinadas com algoritmos de otimização. Vale ressaltar que, para esta última abordagem, a topologia da rede pode mudar durante o treinamento.

Como dito anteriormente, o tamanho da janela adotado foi 4, ou seja, a cada iteração são fornecidas 4 entradas para que a rede preveja a entrada de número 5, logo, as camadas de entrada e de saída são formadas, respectivamente, por 4 e 1 unidades. As unidades da camada intermediária utilizaram função de ativação *sigmóide logística* e as de saída usaram função de ativação *linear*, por motivos já explicados no Capítulo 3. As redes possuem todas as conexões *feedforward*.

6.1.2 Divisão da base de dados em treinamento validação e teste

Para o treinamento das redes foram usados os dados normalizados correspondentes à série NYWater. Para o total de 69 anos disponíveis neste conjunto de dados, ao utilizar a ordem de atraso p, o conjunto de dados é reduzido para $69 - p$, como o valor de p foi definido como sendo 4, a nova base de dados passou a conter 65 pontos.

Com o objetivo de realizar previsão de séries temporais, é necessário que a divisão da base de dados respeite a ordem temporal da mesma [15]. Como descrito no Capítulo 5, é importante que os dados do conjunto de treinamento sejam os mais antigos, os do conjunto de validação sejam os posteriores e os últimos dados formem o conjunto de testes. Dessa forma, dentre os $69 - p$ valores disponíveis, os primeiros 33 foram destinados a compor o conjunto de treinamento, os 16 valores seguintes foram usados no conjunto de validação e, por fim, os 16 valores restantes formaram o conjunto de teste, respeitando-se a proporção de 50% do total de pontos para treinamento, 25% para validação e 25% para teste [22].

6.1.3 Experimentos com Multy-Layer Perceptron (MLP)

O objetivo dos experimentos realizados com a série NYWater é o de prever o consumo anual de água em litros *per capita* na cidade de Nova Iorque. Dado o consumo de água em anos anteriores, deseja-se obter o consumo do ano imediatamente posterior, por exemplo, dados os anos 1899, 1900, 1901 e 1902, o consumo relativo ao ano 1903 seria previsto.

Aqui descrevemos a metodologia, os parâmetros e os resultados da previsão sobre a série NYWater na abordagem que não utiliza algoritmos de otimização global.

Metodologia de treinamento

O algoritmo utilizado foi o *Resilient Backpropagation* (RPROP) [23]. Para a topologia adotada (MLP totalmente conectada), foram realizadas 30 execuções deste algoritmo com diferentes inicializações de pesos e seus valores variando entre –1,0 e +1,0. As 10 melhores e as 10 piores execuções foram descartadas, portanto, consideramos apenas as 10 execuções restantes para os resultados finais. A avaliação de melhor ou pior rede é feita com base em seu EQM, quanto menor o EQM melhor a capacidade de previsão da rede neural. Mais precisamente, o EQM do conjunto de teste, já que este avalia o desempenho da rede quando submetida a novos padrões, que é o que nos interessa.

Dois critérios de parada foram considerados durante o treinamento, o GL5 do *Proben1* [22] e a parada por número máximo de 10.000 iterações. O treinamento pára antecipadamente se o critério GL5 for alcançado duas vezes, este critério permite que o treinamento seja interrompido

assim que for percebida a perda da capacidade de generalização da rede treinada, evitando, dessa forma, *overfitting*. A metodologia detalhada de treinamento sem otimização foi descrita na Seção 5.2.

Descrição dos parâmetros

Seguem descritos os parâmetros adotados para o treinamento das redes MLP, através dos quais será possível a reprodução destes experimentos.

Topologia: MLP

Algoritmo: RPROP

Camadas escondidas: 1

Quantidade de neurônios escondidos: 4

Quantidade máxima de iterações: 10.000

Número de entradas: 4

Número de saídas: 1

Padrões de treinamento: 33

Padrões de validação: 16

Padrões de teste: 16

Resultado obtidos

O aspecto observado ao fim dos experimentos foi o EQM nos conjuntos de treinamento, validação e teste. A Tabela 2 mostra os valores dos aspectos observados para as 10 execuções consideradas, sua média no conjunto de teste, 0,0306, e desvio padrão 0,000165. Note que a execução que apresentou o melhor resultado foi a de número 2, com EQM de teste igual a 0,030411. Como foi dito, o EQM do conjunto de teste avalia a capacidade de previsão para dados ainda não conhecidos pela rede.

Tabela 2. Resultados obtidos para a série NYWater com redes MLP treinadas com o algoritmo RPROP

Execução	EQM treinamento	EQM validação	EQM teste
1	0,008634	0,050242	0,030471
2	0,008635	0,051121	0,030411
3	0,008573	0,050939	0,030674
4	0,008634	0,050242	0,030471
5	0,007927	0,053366	0,030673
6	0,008474	0,05118	0,030772
7	0,008514	0,051615	0,030846
8	0,010182	0,053064	0,030779
9	0,008624	0,050902	0,03045
10	0,008645	0,050718	0,030453
Média	0,008684	0,051339	0,0306
Desvio Padrão	0,000569	0,001074	0,000165

A Figura 13 mostra a comparação entre o conjunto de teste original da série NYWater e o conjunto de valores previstos no treinamento com janela de tempo, a cada ano. A Tabela 3 confronta os valores previstos com os originais já ilustrados graficamente na Figura 13.

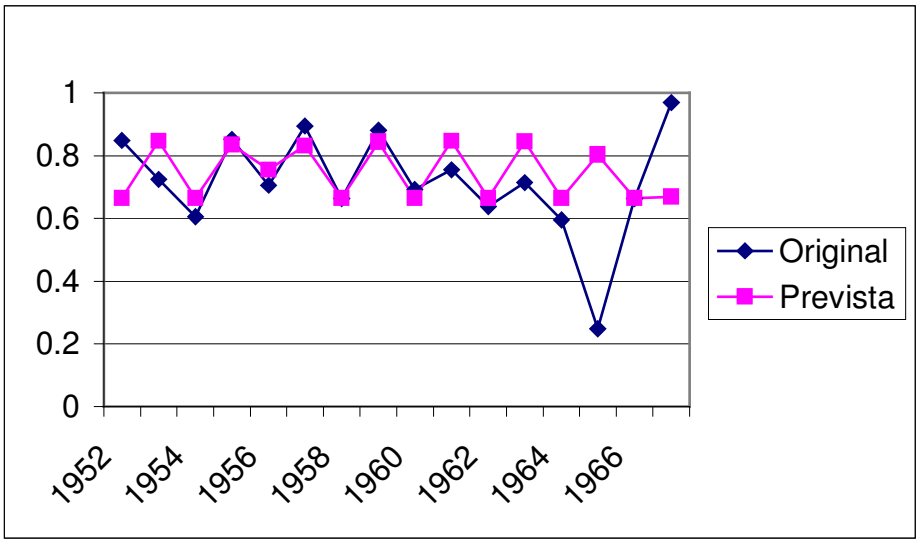


Figura 13. Comparação entre os valores previstos pela melhor rede MLP treinada com o algoritmo RPROP e os valores originais da série, a cada ano do conjunto de teste.

Tabela 3. Valores originais e previstos pela melhor rede MLP treinada com o algoritmo RPROP, a cada ano do conjunto de teste

Ano	Valor original	Valor previsto
1952	0,848059	0,66435
1953	0,72423	0,84654
1954	0,605087	0,66433
1955	0,851406	0,83474
1956	0,706158	0,75438
1957	0,894244	0,83083
1958	0,66332	0,66433
1959	0,881526	0,84372
1960	0,691432	0,66433
1961	0,754351	0,84641
1962	0,638554	0,66433
1963	0,71419	0,84464
1964	0,595047	0,66454
1965	0,247657	0,80269
1966	0,66332	0,66453
1967	0,96988	0,66866

6.1.4 Experimentos com Simulated Annealing (SA)

Nesta Seção, o mesmo problema de previsão da série NYWater é atacado, porém, desta vez, com técnicas de otimização. A técnica aqui utilizada é a já descrita abordagem híbrida que une otimização global, através de *simulated annealing*, e otimização local, realizada pelo RPROP.

Metodologia de treinamento

A técnica empregada foi a de otimização através dos algoritmos *Simulated Annealing* (SA) [14] e RPROP. A topologia inicial das redes foi a MLP com todas as conexões *feedforward.* Ao longo do treinamento é esperado que algumas conexões sejam desconectadas, modificando esta topologia. Como aconteceu no treinamento com RPROP, foram realizadas 30 execuções com esta

abordagem, cada qual com diferentes valores iniciais de pesos, estes no intervalo entre –1,0 e +1,0. Foram descartadas as 10 melhores e as 10 piores execuções e os resultados foram observados apenas para as 10 restantes. Mais uma vez o critério de avaliação foi o EQM do conjunto de teste, por avaliar a reação da rede a novos padrões.

O treinamento híbrido passa por duas fases: a de otimização global e a de otimização local, nesta ordem. Na primeira fase, a rede inicial é treinada apenas com o algoritmo SA para a busca do mínimo global, só depois a segunda fase de busca local acontece, como está descrito na Seção 5.1.1 sobre metodologia de otimização global.

Os critérios de parada adotados para os algoritmos foram o GL5 do *Proben1* [22] e o limite máximo de iterações, para a fase SA o limite foi de 100 iterações e para a fase RPROP o limite foi de 10.000 iterações.

Descrição dos parâmetros

Abaixo estão os parâmetros utilizados no treinamento das MLPs com o sistema de otimização híbrido composto por *simulated annealing* e RPROP.

Topologia: MLP

Algoritmo: Combinação de SA e RPROP

Camadas escondidas: 1

Quantidade de neurônios escondidos: 4

Temperatura inicial: 1

Número de entradas: 4

Número de saídas: 1

Padrões de treinamento: 33

Padrões de validação: 16

Padrões de teste: 16

Quantidade máxima de iterações: 100 para a fase SA e 10.000 para a fase RPROP

Resultados obtidos

Os resultados das 10 execuções consideradas é mostrado na Tabela 4. Através da observação desta Tabela, pode-se perceber que a melhor rede experimentada dentre as 10 executadas foi a rede da quinta execução, cujo EQM foi de 0,024276. Na Tabela 4 também é possível ver os valores de média e desvio padrão das 10 execuções.

Tabela 4. Resultados obtidos para a série NYWater com redes MLP treinadas com o sistema de otimização formado por SA e RPROP

Execução	EQM treinamento	EQM validação	EQM teste
1	0,010303	0,049046	0,026029
2	0,01027	0,048717	0,028513
3	0,011324	0,055382	0,026931
4	0,010857	0,055089	0,028367
5	0,012234	0,056428	0,024276
6	0,011281	0,060255	0,027922
7	0,011814	0,058408	0,028746
8	0,012297	0,05711	0,026863
9	0,011424	0,053789	0,026236
10	0,01089	0,055481	0,028629
Média	0,011269	0,054971	0,027251
Desvio Padrão	0,000712	0,003689	0,001455

O valor do EQM médio do conjunto de teste foi 0,027251, com desvio padrão de 0,001455. Já é possível notar uma sensível melhora com relação aos resultados das previsões sem otimização, o que comprova a nossa teoria inicial de que as redes otimizadas forneceriam melhores resultados que as não otimizadas.

A Figura 14 mostra o gráfico comparativo entre os valores originais e os previstos pela melhor rede. Nota-se que para alguns pontos a previsão foi bastante próxima, em contrapartida, existem pontos, onde a diferença é significativa. É importante lembrar que a quantidade de pontos da série torna a previsão complicada.

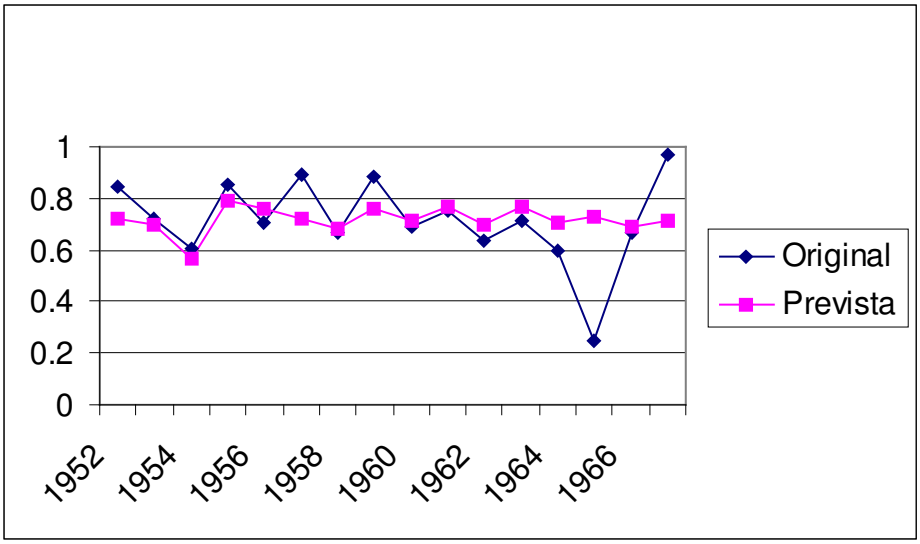


Figura 14. Comparação entre os valores previstos pela melhor rede MLP treinada com o algoritmo de otimização formado por SA e RPROP e os valores originais da série, a cada ano do conjunto de teste.

Tabela 5. Valores originais e previstos pela melhor rede MLP treinada com o algoritmo de otimização formado por SA e RPROP, a cada ano do conjunto de teste

Ano	Valor original	Valor previsto
1952	0,848059	0,72009
1953	0,72423	0,69987
1954	0,605087	0,56258
1955	0,851406	0,79138
1956	0,706158	0,75794
1957	0,894244	0,72009
1958	0,66332	0,68043
1959	0,881526	0,75887
1960	0,691432	0,71415
1961	0,754351	0,77022
1962	0,638554	0,69987
1963	0,71419	0,76695
1964	0,595047	0,70926
1965	0,247657	0,72983
1966	0,66332	0,69157
1967	0,96988	0,7168

6.1.5 Experimentos com Tabu Search (TS)

Além da técnica que utiliza SA, cujos resultados foram demonstrados acima, serão vistos os resultados obtidos com a técnica de otimização híbrida envolvendo TS.

Metodologia de treinamento

A técnica aqui empregada foi a de otimização através dos algoritmos *Tabu Search* (TS) [9][10] e RPROP. A metodologia deste treinamento foi descrita na Seção 5.1.2. com enfoque na técnica empregada.

Foram realizadas 30 execuções deste algoritmo, as redes iniciais e o critério de seleção das execuções consideradas são os mesmos já descritos para o SA.

Os critérios de parada adotados para os algoritmos foram o GL5 do *Proben1* [22] e o limite máximo de iterações. Na fase de otimização global o limite foi de 100 iterações e na fase de otimização local o limite foi de 10.000 iterações.

Descrição dos parâmetros

Abaixo estão os parâmetros utilizados no treinamento das MLPs com o sistema de otimização híbrido composto por *tabu search* e RPROP.

Topologia: MLP

Algoritmo: Combinação de TS e RPROP

Camadas escondidas: 1

Quantidade de neurônios escondidos: 4

Número de vizinhos: 20

Tamanho da lista tabu: 10

Número de entradas: 4

Número de saídas: 1

Padrões de treinamento: 33

Padrões de validação: 16

Padrões de teste: 16

Quantidade máxima de iterações: 100 para a fase TS e 10.000 para a fase RPROP

Resultados obtidos

Os resultados das 10 execuções com *tabu search,* sua média e desvio padrão podem ser vistos na Tabela 6. Para estes experimentos, a média do EQM no conjunto de teste foi de 0,02544 e o desvio padrão 0,001922. A melhor dentre as redes testadas foi a que obteve EQM de valor 0,020634 no conjunto de teste.

A Figura 15 e a Tabela 7 mostram a comparação entre os resultados obtidos nestes experimentos e os resultados desejados para a série NYWater. Também estes resultados foram melhores que os resultados expostos na Seção de resultados do RPROP, quando comparados ao resultados obtidos com o SA também há uma pequena diminuição do erro médio, o que mostra que o treinamento com TS foi um pouco mais eficiente do que o SA para a previsão desta série.

Tabela 6. Resultados obtidos para a série NYWater com redes MLP treinadas com o sistema de otimização formado por TS e RPROP

Execução	EQM treinamento	EQM validação	EQM teste
1	0,011535	0,056113	0,026607
2	0,011361	0,05472	0,026238
3	0,01218	0,058317	0,026612
4	0,012099	0,057575	0,027037
5	0,012286	0,056607	0,025238
6	0,011242	0,05855	0,020634
7	0,011508	0,054854	0,024764
8	0,012123	0,056544	0,026481
9	0,011688	0,057815	0,026545
10	0,012238	0,056212	0,024244
Média	0,011826	0,056731	0,02544
Desvio Padrão	0,000399	0,001333	0,001922

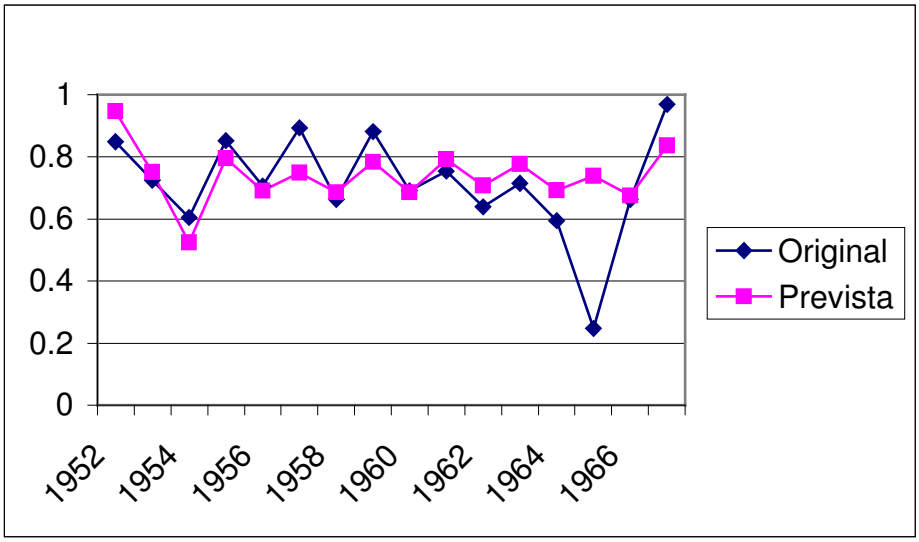


Figura 15. Comparação entre os valores previstos pela melhor rede MLP treinada com o algoritmo de otimização formado por TS e RPROP e os valores originais da série, a cada ano do conjunto de teste.

Tabela 7. Valores originais e previstos pela melhor rede MLP treinada com o algoritmo de otimização formado por TS e RPROP, a cada ano do conjunto de teste

Ano	Valor original	Valor previsto
1952	0,848059	0,94747
1953	0,72423	0,75117
1954	0,605087	0,52456
1955	0,851406	0,79498
1956	0,706158	0,69097
1957	0,894244	0,74911
1958	0,66332	0,68471
1959	0,881526	0,78309
1960	0,691432	0,68502
1961	0,754351	0,79197
1962	0,638554	0,70709
1963	0,71419	0,77622
1964	0,595047	0,69225
1965	0,247657	0,73845
1966	0,66332	0,67569
1967	0,96988	0,83592

Uma melhor forma de visualização das séries previstas pelas 3 abordagens descritas acima (RPROP, SA e TS) pode ser encontrada na Figura 16 e na Tabela 8. Tanto na figura quanto na tabela, estão dispostas todas as 3 séries previstas relativas ao conjunto de teste.

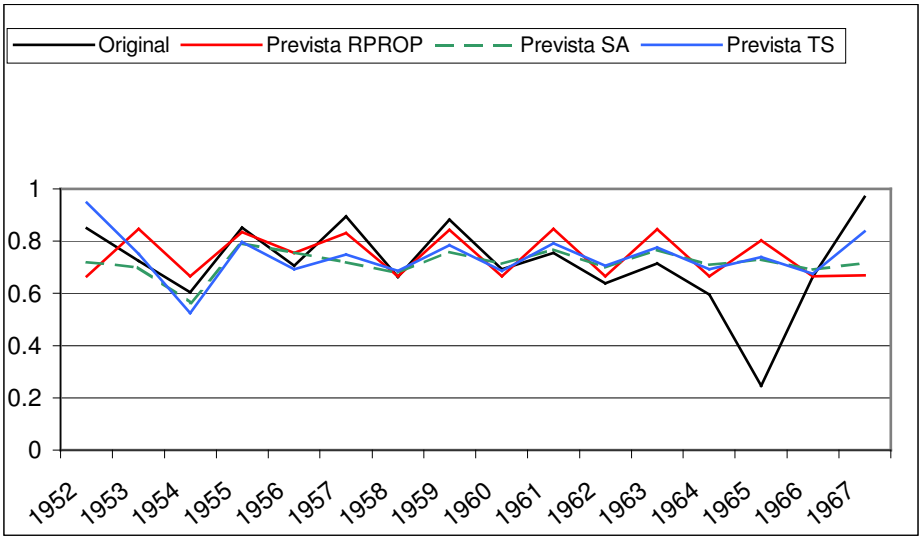


Figura 16. Comparação entre os valores originais da série NYWater e os valores previstos pelas melhores redes MLP treinadas com os algoritmos RPROP (sem otimização), SA (híbrido) e TS (híbrido), a cada ano do conjunto de teste.

Tabela 8. Valores originais da série NYWater e os valores previstos pelas melhores redes MLP treinadas com os algoritmos RPROP (sem otimização), SA (híbrido) e TS (híbrido), a cada ano do conjunto de teste

Ano	Valor original	Valor previsto		
		RPROP	SA	TS
1952	0,848059	0,66435	0,72009	0,94747
1953	0,72423	0,84654	0,69987	0,75117
1954	0,605087	0,66433	0,56258	0,52456
1955	0,851406	0,83474	0,79138	0,79498
1956	0,706158	0,75438	0,75794	0,69097
1957	0,894244	0,83083	0,72009	0,74911
1958	0,66332	0,66433	0,68043	0,68471
1959	0,881526	0,84372	0,75887	0,78309
1960	0,691432	0,66433	0,71415	0,68502
1961	0,754351	0,84641	0,77022	0,79197
1962	0,638554	0,66433	0,69987	0,70709
1963	0,71419	0,84464	0,76695	0,77622
1964	0,595047	0,66454	0,70926	0,69225
1965	0,247657	0,80269	0,72983	0,73845
1966	0,66332	0,66453	0,69157	0,67569
1967	0,96988	0,66866	0,7168	0,83592

6.2 Experimentos com a série ElecNew

A série ElecNew está disponível no repositório público de séries temporais, *Time Series Data Library*, cuja URL é http://www-personal.buseco.monash.edu.au/~hyndman/TSDL/ . Esta série representa a geração total de eletricidade pelas indústrias geradoras dos Estados Unidos. A série normalizada possui valores mensais no período de janeiro de 1986 a outubro de 1996, totalizando 130 pontos. Na Figura 17 é apresentada a série ElecNew normalizada entre 0 e 1.

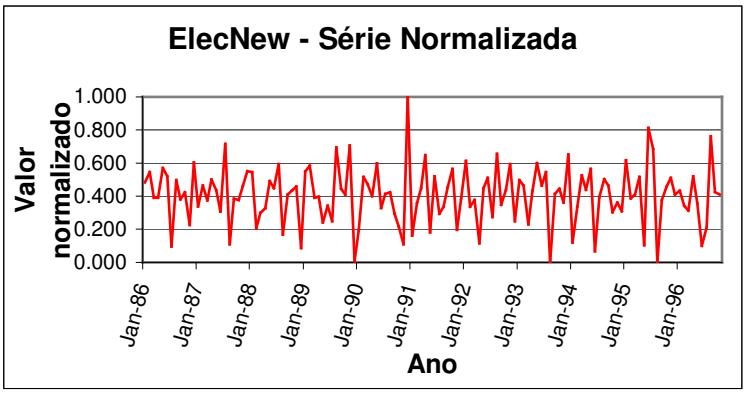


Figura 17. Série ElecNew normalizada entre 0 e 1

6.2.1 Topologia e algoritmo de treinamento

Para a realização de todos os treinamentos com a série ElecNew, utilizou-se redes MLP *feedforward* totalmente conectadas com uma camada intemediária e 4 nodos nessa camada. Como já comentado neste Capítulo, após passar pelo processo de otimização global, a topologia

da rede deve ser alterada, gerando uma nova rede com conexões e valores de pesos iniciais modificados dos originais.

Como o tamanho da janela de treinamento especificado neste trabalho foi 4 e o número de pontos a ser previsto foi 1, a rede MLP que se adequa à essas características deve possuir 4 unidades na camada de entrada e 1 unidade na camada de saída.

As unidades da camada intermediária utilizaram função de ativação *sigmóide logística* e as de saída usaram função de ativação *linear*, por motivos já explicados no Capítulo 3.

Divisão da base de dados em treinamento validação e teste

Para o treinamento das redes foram usados os dados normalizados correspondentes à série ElecNew. Para o total de 130 meses disponíveis neste conjunto de dados, ao utilizar a ordem de atraso p como sendo 4, o conjunto de dados é reduzido para 130 – p, ou seja, 126 pontos.

Para realizar previsão de séries temporais, a divisão da base de dados precisa seguir a ordem temporal original da mesma [15]. Como descrito no Capítulo 5, é importante que os dados do conjunto de treinamento sejam os mais antigos, os do conjunto de validação sejam os posteriores e os últimos dados formem o conjunto de testes. Assim, dentre os 126 valores disponíveis, os 64 primeiros formaram o conjunto de treinamento, os 31 valores seguintes foram usados no conjunto de validação e, por fim, os 31 valores restantes formaram o conjunto de teste, seguindo aproximadamente a proporção de 50% do total de pontos para treinamento, 25% para validação e 25% para teste [22].

6.2.2 Experimentos com Multy-Layer Perceptron (MLP)

O que se quer prever a partir da série ElecNew é a quantidade de energia elétrica gerada mensalmente nos Estados Unidos. Dada a geração de energia em meses anteriores, queremos prever consumo do mês imediatamente posterior, por exemplo, dados os meses janeiro de 1995, fevereiro de 1995, março de 1995 e abril de 1995, obteremos o mês de maio de 1995 previsto.

Aqui é descrita a metodologia, os parâmetros e os resultados da previsão sobre a série ElecNew na abordagem que utiliza apenas o algoritmo RPROP convencional para treinamento, sem métodos de otimização.

Metodologia de treinamento

A metodologia de treinamento para a série ElecNew foi basicamente a mesma descrita para a série NYWater. Utilizou-se o *Resilient Backpropagation* (RPROP) [25]. Para a topologia MLP inicial, foram realizadas 30 execuções deste algoritmo com diferentes inicializações de pesos e seus valores variando entre –1,0 e +1,0, descartando-se as 10 melhores e as 10 piores execuções.

O treinamento pára se o número máximo de 10.000 iterações for alcançado ou se o critério de parada GL5 do *Proben1* [22] for atingido duas vezes.

Descrição dos parâmetros

Topologia: MLP

Algoritmo: RPROP

Camadas escondidas: 1

Quantidade de neurônios escondidos: 4

Quantidade máxima de iterações: 10.000

Número de entradas: 4

Número de saídas: 1

Padrões de treinamento: 64

Padrões de validação: 31

Padrões de teste: 31

Resultado obtidos

A Tabela 9 mostra os valores do EQM nos conjuntos de treinamento, validação e teste, sua média e desvio padrão, para as 10 execuções consideradas. Para efeito de comparação posterior, é importante observar a média do EQM do conjunto de teste, cujo valor foi 0,03182521, e desvio padrão 0,001157594. Note que a execução que apresentou o melhor resultado foi a de número 3, com EQM de teste igual a 0,030201.

Tabela 9. Resultados obtidos para a série ElecNew com redes MLP treinadas com o algoritmo RPROP

Execução	EQM treinamento	EQM validação	EQM teste
1	0,0220059	0,0182341	0,0307819
2	0,021982	0,0183189	0,030735
3	0,0219255	0,0182732	0,030201
4	0,0209281	0,0177301	0,0318912
5	0,0204395	0,0177089	0,0325279
6	0,0211233	0,0177968	0,0331529
7	0,0219225	0,0179766	0,0308837
8	0,0228524	0,0182998	0,0336891
9	0,0209161	0,0176372	0,0318407
10	0,0210223	0,0179717	0,0325487
Média	0,02151176	0,01799473	0,03182521
Desvio Padrão	0,000733196	0,000269276	0,001157594

A Figura 18 compara o conjunto de teste original da série ElecNew e o conjunto de valores previstos no treinamento sem otimização, a cada mês. A Tabela 10 confronta os valores previstos com os originais já ilustrados graficamente na Figura 18.

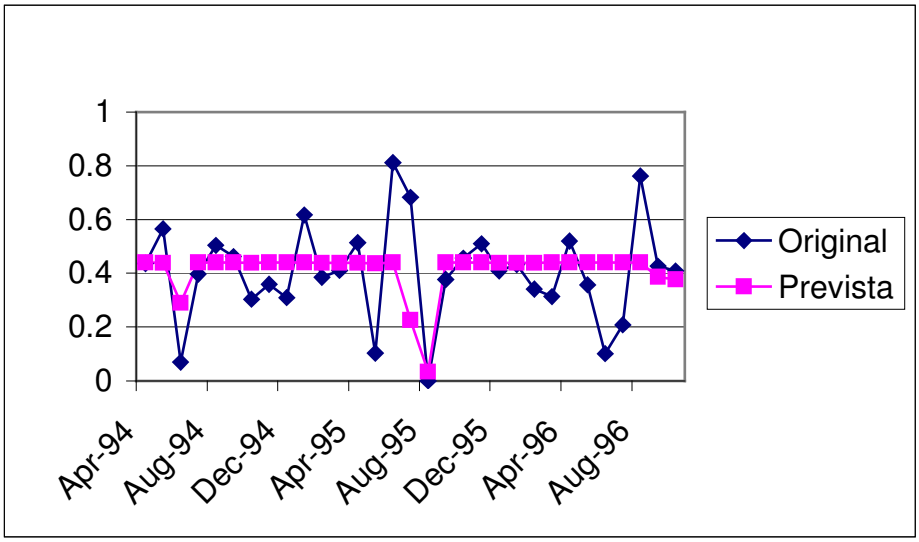


Figura 18. Comparação entre os valores previstos pela melhor rede MLP treinada com o algoritmo RPROP e os valores originais da série, a cada mês do conjunto de teste.

Tabela 10. Valores originais e previstos pela melhor rede MLP treinada com o algoritmo RPROP, a cada mês do conjunto de teste

Ano	Valor original	Valor previsto
Apr-94	0,437204	0,43953
May-94	0,565708	0,43942
Jun-94	0,068984	0,28998
Jul-94	0,396979	0,43953
Aug-94	0,504368	0,43953
Sep-94	0,463233	0,43953
Oct-94	0,303968	0,43913
Nov-94	0,359847	0,43953
Dec-94	0,310157	0,43953
Jan-95	0,617037	0,43953
Feb-95	0,385511	0,43845
Mar-95	0,410994	0,43814
Apr-95	0,514743	0,43937
May-95	0,10375	0,43707
Jun-95	0,812159	0,43953
Jul-95	0,683473	0,22608
Aug-95	0	0,0328
Sep-95	0,376593	0,43953
Oct-95	0,45668	0,43953
Nov-95	0,510557	0,43953
Dec-95	0,409174	0,43926
Jan-96	0,432472	0,43823
Feb-96	0,341645	0,43924
Mar-96	0,313797	0,43953
Apr-96	0,520204	0,43953
May-96	0,356389	0,43953
Jun-96	0,101201	0,43953
Jul-96	0,209319	0,43953
Aug-96	0,761376	0,43953
Sep-96	0,426465	0,38764
Oct-96	0,408628	0,37703

6.2.3 Experimentos com Simulated Annealing (SA)

Aqui, serão mostrados detalhes sobre os experimentos realizados sobre a série ElecNew utilizando o modelo híbrido de treinamento composto por *simulated annealing* e RPROP.

Metodologia de treinamento

A técnica aqui empregada foi a de otimização através dos algoritmos *Simulated Annealing* (SA) [14] e RPROP. Como topologia inicial, adotou-se a MLP *feedforward* totalmente conectada. Após a fase de otimização global é esperado que algumas conexões sejam desconectadas, gerando uma nova topologia que servirá de entrada para a otimização local.

Como aconteceu no treinamento com RPROP, foram realizadas 30 execuções com esta abordagem, cada qual com diferentes valores iniciais de pesos, estes no intervalo entre –1,0 e +1,0. Foram descartadas as 10 melhores e as 10 piores execuções e os resultados foram observados apenas para as 10 restantes. Mais uma vez o critério de avaliação foi o EQM do conjunto de teste, por avaliar a reação da rede a novos padrões.

Os critérios de parada e a seleção das execuções através de seu EQM se deu da mesma forma que na série anterior.

Maiores detalhes sobre como a abordagem híbrida envolvendo *simulated annealing* funciona podem ser encontrados na Seção 5.1.1 sobre metodologia de otimização global.

Descrição dos parâmetros

Abaixo estão os parâmetros utilizados no treinamento das MLPs com o sistema de otimização híbrido composto por *simulated annealing* e RPROP.

Topologia: MLP

Algoritmo: Combinação de SA e RPROP

Camadas escondidas: 1

Quantidade de neurônios escondidos: 4

Temperatura inicial: 1

Número de entradas: 4

Número de saídas: 1

Padrões de treinamento: 64

Padrões de validação: 31

Padrões de teste: 31

Quantidade máxima de iterações: 100 para a fase SA e 10.000 para a fase RPROP

Resultado obtidos

Os resultados dos experimentos está exposto na Tabela 11. A melhor das redes experimentadas, foi a obtida com a segunda execução com EQM de teste igual a 0,025475. A Tabela mostra ainda a média dos valores de EQM para todos os conjuntos (treinamento, validação e teste) e o desvio padrão. Dando continuidade ao nosso critério de avaliação do EQM do conjunto de teste, a média do mesmo foi de 0,027806. Houve uma melhora com relação aos experimentos realizados com o RPROP descritos anteriormente onde o valor do EQM foi 0,03182521.

Tabela 11. Resultados obtidos para a série ElecNew com redes MLP treinadas com o sistema de otimização formado por SA e RPROP

Execução	EQM treinamento	EQM validação	EQM teste
1	0,021859	0,020589	0,027231
2	0,023577	0,020929	0,025475
3	0,0231	0,018664	0,027536
4	0,02698	0,024769	0,028591
5	0,024256	0,020182	0,028634
6	0,021983	0,018825	0,028242
7	0,022244	0,019009	0,028042
8	0,022417	0,019879	0,02878
9	0,02297	0,022299	0,028145
10	0,022649	0,023094	0,027387
Média	0,023203	0,020824	0,027806
Desvio Padrão	0,001517	0,002006	0,000979

A comparação entre as séries original e a gerada pela melhor rede produzida está ilustrada na Figura 19 e os valores reais obtidos confrontados com os desejados estão dispostos na Tabela 12.

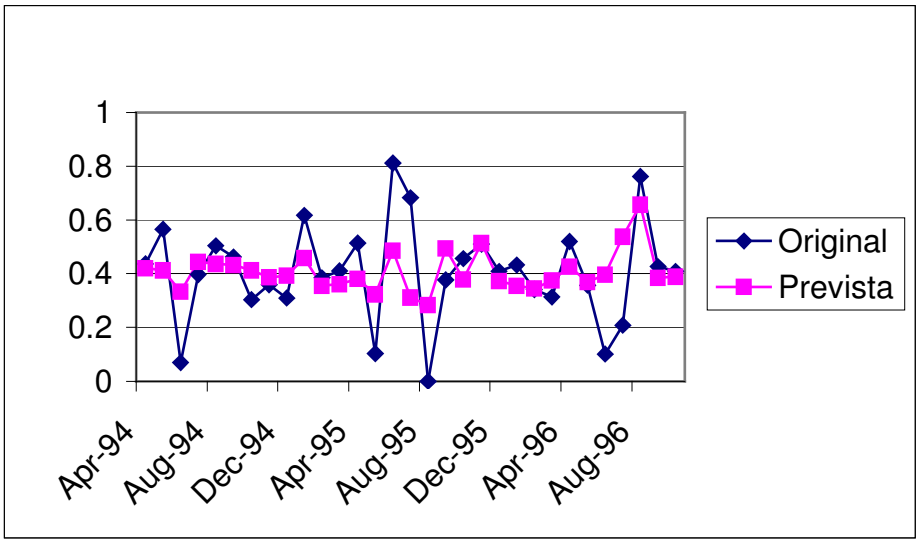


Figura 19. Comparação entre os valores previstos pela melhor rede MLP treinada com o algoritmo de otimização formado por SA e RPROP e os valores originais da série, a cada mês do conjunto de teste.

6.2.4 Experimentos com Tabu Search (TS)

Nesta Seção são mostrados e descritos os experimentos realizados com a abordagem híbrida composta por *tabu search* e RPROP sobre a série ElecNew.

Metodologia de treinamento

A técnica aqui empregada foi a de otimização através dos algoritmos *Tabu Search* (TS) e RPROP. Maiores detalhes sobre esta técnica podem ser obtidos na Seção 5.1.2.
Foram realizadas 30 execuções deste algoritmo. As redes iniciais, o critério de seleção e os critérios de parada das execuções consideradas são os mesmos já descritos para o SA.

Tabela 12. Valores originais e previstos pela melhor rede MLP treinada com o algoritmo de otimização formado por SA e RPROP, a cada mês do conjunto de teste

Ano	Valor original	Valor previsto
Apr-94	0,437204	0,41971
May-94	0,565708	0,41293
Jun-94	0,068984	0,3329
Jul-94	0,396979	0,44368
Aug-94	0,504368	0,43622
Sep-94	0,463233	0,43331
Oct-94	0,303968	0,41256
Nov-94	0,359847	0,38655
Dec-94	0,310157	0,39236
Jan-95	0,617037	0,45773
Feb-95	0,385511	0,3544
Mar-95	0,410994	0,36123
Apr-95	0,514743	0,38027
May-95	0,10375	0,3236
Jun-95	0,812159	0,48674
Jul-95	0,683473	0,31084
Aug-95	0	0,28433
Sep-95	0,376593	0,49307
Oct-95	0,45668	0,37858
Nov-95	0,510557	0,51354
Dec-95	0,409174	0,37289
Jan-96	0,432472	0,35496
Feb-96	0,341645	0,34554
Mar-96	0,313797	0,37568
Apr-96	0,520204	0,42669
May-96	0,356389	0,36875
Jun-96	0,101201	0,39638
Jul-96	0,209319	0,53862
Aug-96	0,761376	0,65581
Sep-96	0,426465	0,38445
Oct-96	0,408628	0,38855

Descrição dos parâmetros

Abaixo estão os parâmetros utilizados no treinamento das MLPs com o sistema de otimização híbrido composto por *tabu search* e RPROP.

Topologia: MLP

Algoritmo: Combinação de TS e RPROP

Camadas escondidas: 1

Quantidade de neurônios escondidos: 4

Número de vizinhos: 20

Tamanho da lista tabu: 10

Número de entradas: 4

Número de saídas: 1

Padrões de treinamento: 64

Padrões de validação: 31

Padrões de teste: 31

Quantidade máxima de iterações: 100 para a fase TS e 10.000 para a fase RPROP

Resultado obtidos

A Tabela 13 contém os resultados das 10 execuções observadas, média e desvio padrão do EQM dos conjuntos de treinamento, validação e teste. Nestes experimentos, o valor médio do EQM de teste para todas as 10 execuções foi de 0,02592784 e o desvio padrão 0,00070636.

Os resultados mostram que a otimização das redes realizada com tabu search melhora o resultado final para a série ElecNews em comparação aos resultados dos treinamentos sem otimização. Relembrando, o EQM de teste resultante dos experimentos sem otimização foi 0,03182521.

Comparando os resultados das redes treinadas por *tabu search* com aquelas treinadas por simulated anneling (média do EQM de teste igual a 0,027806), também notamos que há diminuição do erro médio, como esperado.

A Figura 20 e a Tabela 14 mostram as séries obtida e a desejada (ElecNew original) relativas ao conjunto de teste. Pode-se perceber visualmente a melhoria na previsão da maioria dos pontos da série em comparação às previsões com os outros algoritmos estudados neste trabalho.

Tabela 13. Resultados obtidos para a série ElecNew com redes MLP treinadas com o sistema de otimização formado por TS e RPROP

Execução	EQM treinamento	EQM validação	EQM teste
1	0,023043	0,022366	0,0266824
2	0,022168	0,01927	0,025594
3	0,023672	0,020722	0,0267915
4	0,022405	0,022264	0,0253216
5	0,023077	0,020038	0,0252778
6	0,024667	0,021739	0,0266243
7	0,023766	0,017542	0,0258332
8	0,021681	0,020008	0,0267594
9	0,020995	0,020996	0,0250965
10	0,02231	0,020245	0,0252977
Média	0,022778	0,020519	0,02592784
Desvio Padrão	0,001086	0,001461	0,00070636

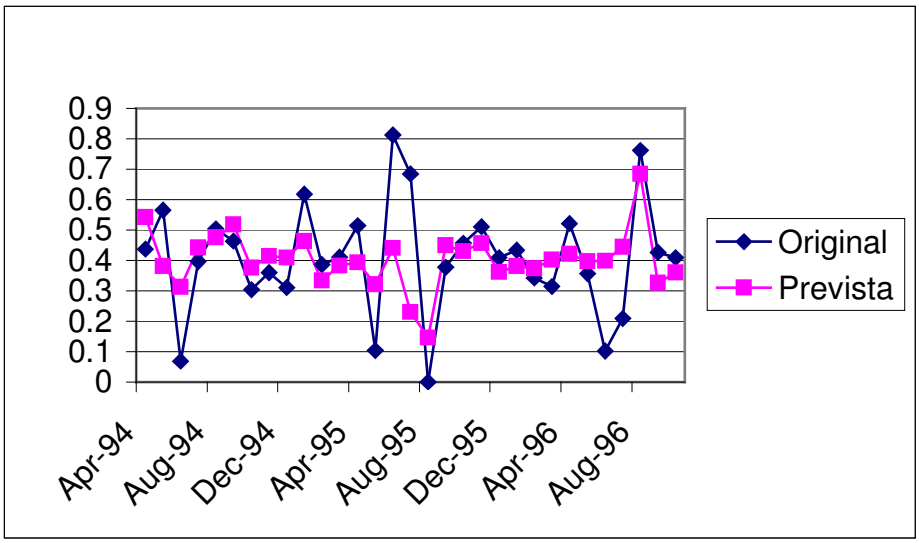


Figura 20. Comparação entre os valores previstos pela melhor rede MLP treinada com o algoritmo de otimização formado por TS e RPROP e os valores originais da série, a cada mês do conjunto de teste.

Tabela 14. Valores originais e previstos pela melhor rede MLP treinada com o algoritmo de otimização formado por TS e RPROP, a cada mês do conjunto de teste

Ano	Valor original	Valor previsto
Apr-94	0,437204	0,542
May-94	0,565708	0,38
Jun-94	0,068984	0,31208
Jul-94	0,396979	0,44188
Aug-94	0,504368	0,47518
Sep-94	0,463233	0,5172
Oct-94	0,303968	0,3746
Nov-94	0,359847	0,41413
Dec-94	0,310157	0,4082
Jan-95	0,617037	0,46334
Feb-95	0,385511	0,33401
Mar-95	0,410994	0,38231
Apr-95	0,514743	0,39275
May-95	0,10375	0,32057
Jun-95	0,812159	0,44109
Jul-95	0,683473	0,22962
Aug-95	0	0,14503
Sep-95	0,376593	0,4495
Oct-95	0,45668	0,42921
Nov-95	0,510557	0,4561
Dec-95	0,409174	0,36127
Jan-96	0,432472	0,38135
Feb-96	0,341645	0,3739
Mar-96	0,313797	0,40139
Apr-96	0,520204	0,42076
May-96	0,356389	0,397
Jun-96	0,101201	0,39885
Jul-96	0,209319	0,4444
Aug-96	0,761376	0,68344
Sep-96	0,426465	0,32716
Oct-96	0,408628	0,35955

A Figura 21 e a Tabela 15 confrontam os resultados obtidos para todas as previsões do conjunto de teste realizadas com as 3 abordagens acima descritas. Nelas estão as séries previstas pelo RPROP, SA e TS de forma a facilitar a comparação entre os resultados.

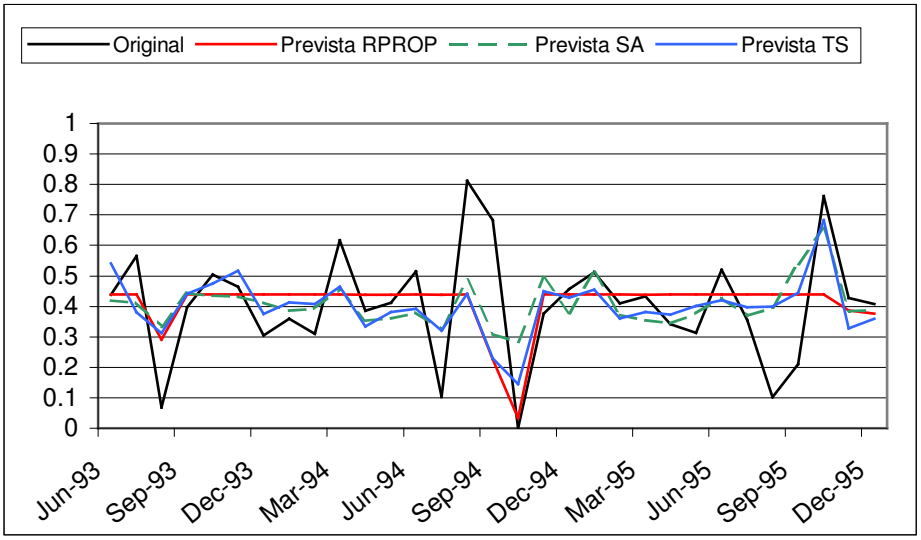


Figura 21. Comparação entre os valores originais da série ElecNew e os valores previstos pelas melhores redes MLP treinadas com os algoritmos RPROP (sem otimização), SA (híbrido) e TS (híbrido), a cada mês do conjunto de teste.

Tabela 15. Valores originais da série ElecNew e os valores previstos pelas melhores redes MLP treinadas com os algoritmos RPROP (sem otimização), SA (híbrido) e TS (híbrido), a cada mês do conjunto de teste.

Ano	Valor original	Valor previsto		
		RPROP	SA	TS
Apr-94	0,437204	0,43953	0,542	0,41971
May-94	0,565708	0,43942	0,38	0,41293
Jun-94	0,068984	0,28998	0,31208	0,3329
Jul-94	0,396979	0,43953	0,44188	0,44368
Aug-94	0,504368	0,43953	0,47518	0,43622
Sep-94	0,463233	0,43953	0,5172	0,43331
Oct-94	0,303968	0,43913	0,3746	0,41256
Nov-94	0,359847	0,43953	0,41413	0,38655
Dec-94	0,310157	0,43953	0,4082	0,39236
Jan-95	0,617037	0,43953	0,46334	0,45773
Feb-95	0,385511	0,43845	0,33401	0,3544
Mar-95	0,410994	0,43814	0,38231	0,36123
Apr-95	0,514743	0,43937	0,39275	0,38027
May-95	0,10375	0,43707	0,32057	0,3236
Jun-95	0,812159	0,43953	0,44109	0,48674
Jul-95	0,683473	0,22608	0,22962	0,31084
Aug-95	0	0,0328	0,14503	0,28433
Sep-95	0,376593	0,43953	0,4495	0,49307
Oct-95	0,45668	0,43953	0,42921	0,37858
Nov-95	0,510557	0,43953	0,4561	0,51354
Dec-95	0,409174	0,43926	0,36127	0,37289
Jan-96	0,432472	0,43823	0,38135	0,35496
Feb-96	0,341645	0,43924	0,3739	0,34554
Mar-96	0,313797	0,43953	0,40139	0,37568
Apr-96	0,520204	0,43953	0,42076	0,42669
May-96	0,356389	0,43953	0,397	0,36875
Jun-96	0,101201	0,43953	0,39885	0,39638
Jul-96	0,209319	0,43953	0,4444	0,53862
Aug-96	0,761376	0,43953	0,68344	0,65581
Sep-96	0,426465	0,38764	0,32716	0,38445
Oct-96	0,408628	0,37703	0,35955	0,38855

Capítulo 7

Conclusões e Trabalhos Futuros

7.1 Conclusões

A principal contribuição deste trabalho é a aplicação da metodologia proposta por Yamazaki [27] descrita no Capítulo 5 adaptada a problemas de previsão de séries temporais, o que torna este trabalho inédito. Foram apresentadas duas técnicas de otimização global, *simulated annealing* e *tabu search*, duas abordagens híbridas envolvendo as técnicas e o conhecido algoritmo de otimização local r*esilient backpropagation* (RPROP).

O RPROP apresenta a vantagem de convergir mais rapidamente que o *backpropagation*. O uso do algoritmo RPROP também é um diferencial deste trabalho com relação ao de Yamazaki, que utilizou o *backpropagation* como método de convergência local.

A aplicação das metodologias de treinamento híbridas mostrou-se mais eficiente que o treinamento realizado apenas com o uso do RPROP. No Capítulo 6, foram mostrados os resultados dos treinamentos executados com a abordagem híbrida para duas séries temporais, nele é possível comparar os resultados entre as abordagens com otimização e sem otimização. Para ambas as séries, as redes otimizadas obtiveram melhor desempenho que as redes que não passaram pelo processo de otimização.

Ainda com base nos resultados obtidos no Capítulo 6, é possível também comparar as abordagens híbridas entre si. Verificou-se que os treinamentos com *tabu search* foram, em média, sempre mais eficientes que os treinamentos com *simulated annealing*.

Além disso, foi abordado um problema bastante comum na área científica (previsão de séries temporais), a fim de contribuir um pouco mais com esta área. Vale ressaltar que existem diversas aplicações de natureza temporal, o que torna este trabalho aplicável a um grande número de problemas reais.

A metodologia de otimização utilizada para a previsão neste trabalho já foi usada com sucesso em problemas de classificação em [26][27], consolidando ainda mais a sua eficiência na otimização das redes neurais artificiais.

7.2 Trabalhos futuros

Como sugestão de continuidade para este trabalho, está a aplicação das metodologias aqui descritas e experimentadas a novos conjuntos de dados, incluindo outras séries temporais, para assegurar ainda mais sua eficiência em aplicações de processamento temporal.

Na metodologia exposta surgiram novos parâmetros ajustáveis tais como a temperatura no *simulated annealing*, o número de vizinhos gerados e o valor de *N* (ver Seção 5.1.2) no *tabu search*. Neste trabalho, tais parâmetros foram fixados durante os experimentos, no entanto, eles são fatores importantes do treinamento com otimização global. É interessante realizar novos experimentos variando esses parâmetros e, assim, estudar sua influência nos resultados.

Observou-se que os algoritmos de otimização alteram o estado de algumas conexões da topologia inicial, desconectando-as, e, dessa forma, estruturas de redes ótimas são criadas. Porém, o único critério de avaliação do custo das redes adotado foi o *erro quadrático médio* do conjunto de teste. Viu-se que a geração otimizada de topologias e pesos iniciais de fato melhora a eficiência da rede, diminuindo o valor do *erro quadrático médio* computado. Porém, não foi observado um importante fator que é a complexidade da rede resultante, ou seja, o número de conexões que restaram nas redes otimizadas. Redes pouco complexas representam um custo computacional menor. O fator complexidade poderia ser considerado também como função de custo em experimentos futuros.

Outra proposta é a de estudar a viabilidade da aplicação das metodologias de otimização em outros modelos de redes neurais específicos para processamento temporal, tais como redes de Elman [7] e Jordan [13].

Bibliografia

[1] BOESE, K. D. and KAHNG, A. B., "Best-So-Far vs. Where-You-Are: Implications for Optimal Finite- Time Annealing", *Systems and Control Letters*, 22(1), pp. 71-78, Jan. 1994.

[2] *BOX, G. E.; JENKINS, G. M.*, Time Series Analysis: Forecasting and Control, *Holden-Day, San Francisco, CA, 1970.*

[3] *BRAGA, A. P.; CARVALHO, A. P. L. F.; LUDERMIR, T. B.* "Redes Neurais Artificiais Teoria e Aplicações". Livros Técnicos e Científicos Editora, Rio de Janeiro, 2000.

[4] BURKE, H. et All. Comparing the prediction accuracy os artificial neural networks and other statistical models for breast cancer survival. In G. Tesauro, D. S. Touretzky, T. K. Leen, editors, *Neural Information Processing Systems 7.* MIT Press, 1995.

[5] *CYBENKO, G.,* Approximation by superpositions of a sigmoid function. *Mathematics of Control, Signals and Systems,* 1989.

[6] *DORRONSORO, J. R.; GINEL, F.; SANCHEZ, C.; (1997):* Neural fraud detection in credit card operations. *IEEE Transactions on Neural Networks.*

[7] *ELMAN, J.L.,* Finding Structure in time. *Cognitive science,*14:179-211, 1990.

[8] Faq-learning: http://www.faqs.org/faqs/by-newsgroup/comp/comp.ai.neural-nets.html.

[9] *GLOVER, F. (1986)*: Future paths for integer programming and links to artificial intelligence. *Computers and Operation Research, 13, 533-549.*

[10] *HANSEN, P. (1986):* The steepest ascent mildest descent heuristic for combinatorial programming. *Conf. on Numerical Methods in Combinatorial Optimisation, Capri, Italy.*

[11] *HAYKIN, S*. Neural Networks: A Comprehensive Foundation. Prentice Hall, 2nd edition.

[12] *JAIN, A. K.; MAO, J.; MOHIUDDIN, K. M.* Artificial Neural Networks: A Tutorial. *IEEE Computer, Março 1996, 31-44*.

[13] *JORDAN, M. I.,* Attractor dynamics and parallelism in a connectionist sequential machine. *In Proceedings of The Eight nnual Conference of The Cognitive Science Society,*pages 531-546, Amherst, 1986. Hillsdale: Erlbaum.

[14] *KIRKPATRICK, S; GELLAT JR., C. D. e VECCHI, M. P. (1983)*: Optimization by simulated annealing. Science, 220, 671-680.

[15] *KOSKELA, T.; LEHTOGANGAS, M.; SAARINEN, J. e KASHI, K. (1996):* Time series prediction with multi-layer perceptron, FIR and Elman neural networks. *In Proceedings of the World Congress on Neural Networks, pages 491-496, San Diega, USA.*

[16] *KOSKIVAARA, E.* Artificial Neural Network Models for Predicting Patterns In Auditing. *Journal of Operational Research Society*, 51(9):1060-1069, setembro 2000.

[17] LE CUN, Y. et All, Handwritten digit recognition with a back-propagation network," in Advances in Neural Information Processing Systems (D. Touretzky, ed.), vol. 2, (Denver 1989), pp. 396-404, Morgan Kaufmann, San Mateo, 1990.

[18] MCCULLOCH, W. S. and PITTS, W. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115-133. 1943.

[19] METROPOLIS, N.; ROSENBLUTH, A. W.; ROSENBLUTH, M. N.; TELLER, A. H. and TELLER, E., "Equation of state calculations by fast computing machines", *J. of Chem. Phys.*, Vol. 21, No. 6, pp. 1087-1092, 1953.

[20] *OLIVEIRA, A. L. I; AZEVEDO, G.; BARROS, A.; SANTOS, A.L.M*. Sistema de Suporte a Auditoria de Folhas de Pagamento Baseado em Redes Neurais. In *Anais do IV Encontro Nacional de Inteligência Artificial,* 2003.

[21] *PHAM, D.T. e Karaboga, D*., "Introduction", Intelligent Optimisation Techniques *(Edited by D.T. Pham and D. Karaboga), pp. 1-50, Springer- Verlag, 2000.*

[22] *PRECHELT, L.(1994).* Proben 1 – a set of neural networks benchmark problems and benchmarking rules. *Technical report 21/94, Universität Karlsruhe, Germany.*

[23] *RIEDMILLER, M. and BRAUN, H. (1993).* A direct adaptive method for faster backpropagation learning: The RPROP algorithm. *In Proceedings of the IEEE International Conference on Neural Networks (ICNN 93).*

[24] *ROSENBLAT, F.* The Perceptron: A probabilistic model for information storage and organization in the brain. P`sycol. Rev.. 65:386-408. 1958.

[25] *RUMELHART, D. E.; HINTON, G. E. e WILLIAMS, R. J. (1986):* Learning internal representations by error propagation. *In: D.E. RUMELHART and J.L. MC-CLELLAND (Eds.): Parallel Distributed Processing Vol 1. MIT Press, Cambridge, 318-362.*

[26] *YAMAZAKI, A.; LUDERMIR, T. B.* Neural Network Training with Global Optimization Techniques*, International Journal of Neural Systems, Vol. 13, N. 2, pp. 77-86, April 2003*

[27] *YAMAZAKI, A*, "Uma Metodologia para Otimização de Arquiteturas e Pesos de Redes Neurais", Tese de Doutorado, Centro de Informática, Universidade Federal de Pernambuco, Recife-PE, *Março de 2004*.

[28] *YAO, X*. Evolving artificial neural netwoks. *Proceedings of the IEEE,* September 1999.

[29] YODA, M.. Predicting the Tokyo stock market. Trading on the edge: neural, genetic and fuzzy systems for chaotic financial markets, 66-79. John Wiley & Sons, 1994.

[30] *WAIBEL, A., et al.,* Phoneme Recognition using time-delay neural networks," *IEEE Trans. ASSP, pp. 328-339, Vol. 37, No. 3, March, 1989*.

[31] *WAN, E.,* Finite Impulse Response Neural Networks with Applications in Time-Series Prediction," *Ph.D. Dissertation, Stanford University, Stanford CA, Nov., 1993*.

[32] *ZHANG, G.; PATUWO, B. E.; HU, M. Y. (1998):* Forecasting with artificial neural networks: the state of the art. *International Journal Forecasting 14, 35-62*.

Apêndice A

Códigos-fonte

Classes Básicas

```
public class Conexao {

	public double peso;
	public boolean bit;
	private final double N = 0.01;

	public Conexao(double _peso, boolean _bit) {
		this.peso = _peso;
		this.bit = _bit;
	}

	public boolean isBit() {
		return bit;
	}
	public void setBit(boolean bit) {
		this.bit = bit;
	}
	public double getPeso() {
		return peso;
	}
	public void setPeso(double peso) {
		this.peso = peso;
	}
	public boolean isEqual(Conexao conec){
		boolean isEqual = false;
		if(conec.isBit() == this.isBit()){
			if((conec.getPeso()-N <= this.getPeso()) && (this.getPeso() <= conec.getPeso()+N)){
				isEqual = true;
			}
		}
		return isEqual;
	}

}
```

```
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;

public class Rede {
	private Conexao[][] pesoH;
	private Conexao[][] pesoO;
	private Conexao[] biasH;
	private Conexao[] biasO;
	private int numIn;
	private int numHid;
	private int numOut;
	private Rede novaRede;
	private double custo;
	public Rede(int numIn, int numHid, int numOut) {
		this.numIn = numIn;
		this.numHid = numHid;
		this.numOut = numOut;
		this.initConexao();
	}
	public Rede(Rede rede) {
		this.numIn = rede.getNumIn();
		this.numHid = rede.getNumHid();
		this.numOut = rede.getNumOut();
		this.initConexao(rede);
	}
	/**
	 * Método cria e inicializa entre -1 e 1 as Conexoes de uma Rede
	 */
	private void initConexao() {
		pesoH = new Conexao[numIn][numHid];
		pesoO = new Conexao[numHid][numOut];
		biasH = new Conexao[numHid];
		biasO = new Conexao[numOut];
		for (int i = 0; i < numIn; i++) {
			for (int j = 0; j < numHid; j++) {
				pesoH[i][j] = new Conexao(((2 * Math.random()) - 1), true);
			}
		}
		for (int i = 0; i < numHid; i++) {
```

```
                for (int j = 0; j < numOut; j++) {
                        pesoO[i][j] = new Conexao(((2 * Math.random()) - 1), true);
                }
        }
        for (int i = 0; i < numHid; i++) {
                biasH[i] = new Conexao(((2 * Math.random()) - 1), true);
        }
        for (int i = 0; i < numOut; i++) {
                biasO[i] = new Conexao(((2 * Math.random()) - 1), true);
        }
}
private void initConexao(Rede rede) {
        pesoH = new Conexao[numIn][numHid];
        pesoO = new Conexao[numHid][numOut];
        biasH = new Conexao[numHid];
        biasO = new Conexao[numOut];

        for (int i = 0; i < numIn; i++) {
                for (int j = 0; j < numHid; j++) {
                        double cadaPeso = rede.getPesoH()[i][j].getPeso();
                        boolean cadaBit = rede.getPesoH()[i][j].isBit();
                        int prob = (int) (100 * Math.random());
                        if (prob <= 20)
                                cadaBit = !cadaBit;
                        pesoH[i][j] = new Conexao(cadaPeso + ((2 * Math.random()) - 1),
                                        cadaBit);
                }
        }

        for (int i = 0; i < numHid; i++) {
                for (int j = 0; j < numOut; j++) {
                        double cadaPeso = rede.getPesoO()[i][j].getPeso();
                        boolean cadaBit = rede.getPesoO()[i][j].isBit();
                        int prob = (int) (100 * Math.random());
                        if (prob <= 20)
                                cadaBit = !cadaBit;
                        pesoO[i][j] = new Conexao(cadaPeso + ((2 * Math.random()) - 1),
                                        cadaBit);
                }
        }
```

```
		for (int i = 0; i < numHid; i++) {
			double cadaPeso = rede.getBiasH()[i].getPeso();
			boolean cadaBit = rede.getBiasH()[i].isBit();
			int prob = (int) (100 * Math.random());
			if (prob <= 20)
				cadaBit = !cadaBit;
			biasH[i] = new Conexao(cadaPeso + ((2 * Math.random()) - 1),
					cadaBit);
		}
		for (int i = 0; i < numOut; i++) {
			double cadaPeso = rede.getBiasO()[i].getPeso();
			boolean cadaBit = rede.getBiasO()[i].isBit();
			int prob = (int) (100 * Math.random());
			if (prob <= 20)
				cadaBit = !cadaBit;
			biasO[i] = new Conexao(cadaPeso + ((2 * Math.random()) - 1),
					cadaBit);
		}
	}
	public void imprimirRede(String arquivoNome) {
		PrintWriter fostream = null;
		try {
			fostream = new PrintWriter(new FileWriter(arquivoNome));
			fostream.println("<?xml version=\"1.0\" encoding=\"ISO-8859-1\"?>");
			fostream.println("<rede>");
			fostream.println("\t<conexoes>");
			for (int i = 0; i < numIn; i++) {
				for (int j = 0; j < numHid; j++) {
					Conexao o = pesoH[i][j];
					fostream.println("\t<conexao in='" + i + "' hid='" + j
							+ "' out=" bit='" + o.isBit() + "'>"
							+ o.getPeso()
							+ "</conexao>");
				}
			}
			for (int i = 0; i < numHid; i++) {
				for (int j = 0; j < numOut; j++) {
					Conexao o = pesoO[i][j];
					fostream.println("\t<conexao in=" hid='" + i + "' out='"
							+ j + "' bit='" + o.isBit() + "'>" + o.getPeso()
```

```
                                                    + "</conexao>");
                        }
                }
                fostream.println("\t</conexoes>");
                fostream.println("\t<biases>");

                for (int i = 0; i < numHid; i++) {
                        Conexao o = biasH[i];
                        fostream.println("\t<bias hid='" + i + "' out=" bit='"
                                        + o.isBit() + "'>" + o.getPeso() + "</bias>");
                }
                for (int i = 0; i < numOut; i++) {
                        Conexao o = biasO[i];
                        fostream.println("\t<bias hid=" out='" + i + "' bit='"
                                        + o.isBit() + "'>" + o.getPeso() + "</bias>");
                }

                fostream.println("\t</biases>");
                fostream.println("</rede>");
        } catch (IOException e) {
                e.printStackTrace();
        } finally {
                fostream.close();
        }
}
public double calculaCusto(DataSet ds, String funcAtivacao) {
        double mse = 0; //mse parcial
        double MSE = 0; //mse final
        double[] net = new double[numHid];
        double[] saidaH = new double[numHid];
        double[] netO = new double[numOut];
        double[] saidaO = new double[numOut];
        int tam = ds.getNPadroes();
        double[] output = new double[tam];

        for (int n = 0; n < ds.getNPadroes(); n++) {
                //inicializa variavel net com zero
                for (int i = 0; i < numHid; i++) {
                        net[i] = 0;
                }
```

```
for (int i = 0; i < numHid; i++) {
        for (int j = 0; j < numIn; j++) {
                double peso = pesoH[j][i].getPeso();
                double bias = biasH[i].getPeso();
                double entrada = ds.getEntradas()[n][j];

                //verifica se a conexao existe
                if (pesoH[j][i].isBit())
                        net[i] += peso * entrada;
                net[i] += bias;
        }
        //Calculo da saida dos neuronios da camanda escondida
        saidaH[i] = funcao(net[i], funcAtivacao);
}
//inicializa variavel netO com zero
for (int i = 0; i < numOut; i++) {
        netO[i] = 0;
}

//calcula soma ponderada (net) da camada escondida
for (int i = 0; i < numOut; i++) {
        for (int j = 0; j < numHid; j++) {
                double peso = pesoO[j][i].getPeso();
                double bias = biasO[i].getPeso();

                //entradas sao as saidas da camada escondida
                double entrada = saidaH[j];

                //verifica se a conexao existe
                //na ultima camada so existe um neuronio, logo n eh preciso
                // testar se ha conexao
                if (pesoO[j][i].isBit())
                        netO[i] += peso * entrada;
                netO[i] += bias;

        }
        //Calculo da saida dos neuronios da camada de saida usando
        // funcao de ativacao
        saidaO[i] = funcao(netO[i], funcAtivacao);
```

```
                        //Calculo da saida usando funcao identidade
                        //saidaO[i] = netO[i];
                        output[n] = saidaO[i];
                }
                mse = 0;
                for (int i = 0; i < saidaO.length; i++) {
                        mse += (Math.pow((saidaO[i] - ds.getSaidas()[n][i]), 2));
                }
                mse = mse / (double) numOut;
                MSE += mse;
        }
        custo = (MSE / (double) ds.getNPadroes());
        imprimirSaida(output);
        return custo;
}

private static double funcao(double net, String funcao) {
        double result = 0;

        if (funcao.equals("sigmoide")) {
                // para evitar overflow da funcao logistica (ver faq-learning)
                if (net < -45)
                        net = 0;
                else if (net > 45)
                        net = 1;
                //calculo da sigmoide
                result = 1 / (1 + Math.exp(-net));
        } else if (funcao.equals("tangente"))
                result = 1.7159 * Math.tan(2 * net / 3);
        return result;
}

public Conexao[] getBiasH() {
        return biasH;
}

public void setBiasH(Conexao[] biasH) {
        this.biasH = biasH;
}
```

```
	public Conexao[] getBiasO() {
		return biasO;
	}
	public void setBiasO(Conexao[] biasO) {
		this.biasO = biasO;
	}
	public Conexao[][] getPesoH() {
		return pesoH;
	}
	public void setPesoH(Conexao[][] pesoH) {
		this.pesoH = pesoH;
	}
	public Conexao[][] getPesoO() {
		return pesoO;
	}
	public void setPesoO(Conexao[][] pesoO) {
		this.pesoO = pesoO;
	}
	public int getNumHid() {
		return numHid;
	}
	public void setNumHid(int numHid) {
		this.numHid = numHid;
	}
	public int getNumIn() {
		return numIn;
	}
	public void setNumIn(int numIn) {
		this.numIn = numIn;
	}
	public int getNumOut() {
		return numOut;
	}
	public void setNumOut(int numOut) {
		this.numOut = numOut;
	}
}

	public boolean isEqual(Rede rede) {
		boolean isEqual = true;
```

```
                Conexao[][] ch = this.getPesoH();
                for (int i = 0; i < this.getNumIn(); i++) {
                        for (int j = 0; j < this.getNumHid(); j++) {
                                isEqual = isEqual && ch[i][j].isEqual(rede.getPesoH()[i][j]);
                        }
                }
                if (isEqual) {
                        Conexao[][] co = this.getPesoO();
                        for (int i = 0; i < this.getNumHid(); i++) {
                                for (int j = 0; j < this.getNumOut(); j++) {
                                        isEqual = isEqual
                                                        && co[i][j].isEqual(rede.getPesoO()[i][j]);
                                }
                        }
                }
                return isEqual;
        }
}

import java.util.LinkedList;
public class Tabu {
        private static LinkedList lista = new LinkedList();
        public static void inserir(Object o) throws IndexOutOfTabuException {
                if (lista.size() < 10) {
                        lista.addLast(o);
                } else if (lista.size() == 10) {
                        lista.removeFirst();
                        lista.addLast(o);
                } else {
                        IndexOutOfTabuException e = new IndexOutOfTabuException();
                        throw e;
                }
        }
        public static void remover() {
                lista.removeFirst();
        }

        public static boolean ehTabu(Rede minhaRede) throws IndexOutOfTabuException {
                boolean eh = false;
                for (int i = 0; i < lista.size(); i++) {
                        Rede r = (Rede) Tabu.getElemento(i);
```

```
                    if(minhaRede.isEqual(r)){
                        System.out.println("TABU");
                        return true;
                    }else {
                        Tabu.inserir(minhaRede);
                    }
                }
                return eh;
        }

        public static Object getElemento(int i) throws IndexOutOfTabuException {
                Object o;
                if (i >= 10) {
                    IndexOutOfTabuException e = new IndexOutOfTabuException();
                    throw e;
                } else {
                    o = lista.get(i);
                }
                return o;
        }
}
```

Classe principal do algoritmo de otimização global SA

```
import java.io.File;
import java.io.IOException;

public class TestaTreina {

	public static void main(String[] args) {
		double print = 0;
		int nosEscondida = 4; //nos da camada escondida
		int nosEntrada = 4;
		int nosSaida = 1;

		Rede redeAtual = new Rede(nosEntrada, nosEscondida, nosSaida);

		String arquivo1 = "treinaEle.pat"; //arquivo de treinamento
		String arquivo2 = "tesEle.pat"; //arquivo de teste
		String arquivo3 = "valEle.pat"; //arquivo de validacao
		String funcao = "sigmoide"; //funcao de ativacao "sigmoide" ou  "tangente"
		int numMaxIteracoes = 10000; //nº maximo de iteraçoes
		int numVizinhos = 10; //nº de rede vizinhas criadas por iteracao
		double mseTreina = 0; //mse do conj. treinamento
		double mseValida = 0; //mse do conj. validacao
		double temperatura = 1;

		redeAtual.imprimirRede("redeInicial.xml");
		DataSet ds = null, ds2 = null, ds3 = null;

		try {
			//carrega entradas do arquivo de treinamento
			ds = new DataSet(arquivo1);
			//carrega entradas do arquivo de teste
			ds2 = new DataSet(arquivo2);
			//carrega entradas do arquivo de validacao
			ds3 = new DataSet(arquivo3);
			// define numero de neuronios da rede
			nosEntrada = ds.getNIn();
			nosSaida = ds.getNOut();

		} catch (IOException e) {
			e.printStackTrace();
		}

		double custoMinimo = redeAtual.calculaCusto(ds, funcao);
		System.out.println(custoMinimo);

		double mseValMinimo = 100;
		double gl = 0;
		int glCounter = 0;
		File file = new File("saida.dat");

		for (int i = 1; i <= numMaxIteracoes && gl <= 5; i++) {
			System.out.println("--------------------------");

			/*
			 * ETAPA DE TREINAMENTO
			 */
```

```
//rede vizinha é criada a partir da atual
Rede redeVizinha = new Rede(redeAtual);
//redeVizinha.imprimirRede("vizinha"+i+".xml");

double cadaCusto = redeVizinha.calculaCusto(ds, funcao);

if (cadaCusto < custoMinimo) {
	custoMinimo = cadaCusto;
	redeAtual = redeVizinha;
} else {
	double random = Math.random();
	double prob = Math.exp(-(cadaCusto - custoMinimo) / temperatura);
	//aceita a pior solucao com uma pequena probabilidade
	if (random <= prob) {
		custoMinimo = cadaCusto;
		redeAtual = redeVizinha;
		System.out.println("**saindo do minimo**");
	}
}

//renomeando o arquivo de saidas geradas durante o treinamento
File newFile = new File("saidaTreina.dat");
if (newFile.exists())
	newFile.delete();
file.renameTo(newFile);

redeAtual.imprimirRede("redeAtual.xml");
/*
 * FIM DA ETAPA DE TREINAMENTO
 */

//Calculo da temperatura. A cada 10 iteracoes a temperatura é decrementada
if (i % 10 == 0) {
	temperatura = temperatura * 0.9;
	//System.out.println("CHANGING TEMPERATURE");
}

/*
 * ETAPA DE VALIDACAO
 */

mseValida = redeAtual.calculaCusto(ds3, funcao);
if (mseValida < mseValMinimo) {
	mseValMinimo = mseValida;
}

//criterio gl5 após 300 iteracoes
gl = 100 * ((mseValida / mseValMinimo) - 1);

if (gl > 5) {
	System.out.println("Parada antecipada, iteracao " + i);
	glCounter++;
}

//renomeando o arquivo de saidas geradas durante a validacao
File newFile2 = new File("saidaValida.dat");
if (newFile2.exists())
	newFile2.delete();
file.renameTo(newFile2);
```

```
                /*
                 * FIM DA ETAPA DE VALIDACAO
                 */

                System.out.println("treina atual: " + cadaCusto
                                + "  treina minimo: " + custoMinimo);
                System.out.println("val atual: " + mseValida + "  val minimo: "
                                + mseValMinimo + "  GL5: " + gl);

                print = cadaCusto;
                System.out.println("iteracao = " + i);
        }

        /*
         * ETAPA DE TESTE
         */

        double mseTeste = redeAtual.calculaCusto(ds2, funcao);
        System.out.println("mse teste: " + mseTeste);

        /*
         * FIM DA ETAPA DE TESTE
         */

        //renomeando o arquivo de saidas geradas durante o teste
        File newFile3 = new File("saidaTeste.dat");
        if (newFile3.exists())
                newFile3.delete();
        file.renameTo(newFile3);

    }
}
```

Classe principal do algoritmo de otimização global TS

```
import java.io.File;
import java.io.IOException;

public class TestaTreinaTabu {

	public static void main(String[] args) {
		String arquivo1 = "treinaEle.pat"; //arquivo de treinamento
		String arquivo2 = "tesEle.pat"; //arquivo de teste
		String arquivo3 = "valEle.pat"; //arquivo de validacao
		String funcao = "sigmoide"; //funcao de ativacao "sigmoide" ou  "tangente"
		int nosEscondida = 4; //nos da camada escondida
		int nosEntrada = 0;
		int nosSaida = 0;
		int numMaxIteracoes = 100; //nº maximo de iteraçoes
		int numVizinhos = 20; //nº de rede vizinhas criadas por iteracao
		double mseTreina = 0; //mse do conj. treinamento
		double mseValida = 0; //mse do conj. validacao
		double temperatura = 1;

		DataSet ds = null, ds2 = null, ds3 = null;

		try {
			//carrega entradas do arquivo de treinamento
			ds = new DataSet(arquivo1);
			//carrega entradas do arquivo de teste
			ds2 = new DataSet(arquivo2);
			//carrega entradas do arquivo de validacao
			ds3 = new DataSet(arquivo3);
			// define numero de neuronios da rede
			nosEntrada = ds.getNIn();
			nosSaida = ds.getNOut();

		} catch (IOException e) {
			e.printStackTrace();
		}

		//cria rede inicial/atual
		Rede redeAtual = new Rede(nosEntrada, nosEscondida, nosSaida);
		redeAtual.imprimirRede("redeInicial.xml");

		//guarda a melhor de todas as redes
		Rede bestRede = redeAtual;

		double custoMinimo = redeAtual.calculaCusto(ds, funcao);

		double mseValMinimo = 100;
		double gl = 0;
		File file = new File("saida.dat");

		for (int i = 1; i <= numMaxIteracoes && gl <= 5; i++) {

			/*
			 * ETAPA DE TREINAMENTO
			 */
```

```
double menorCustoVizinha =100;
Rede melhorRedeVizinha = redeAtual;
for (int j = 0; j < numVizinhos; j++) {
		//rede vizinha eh criada a partir da atual
		Rede redeVizinha = new Rede(redeAtual);
		//redeVizinha.imprimirRede("vizinha"+i+".xml");

		double cadaCusto = redeVizinha.calculaCusto(ds, funcao);

		//escolha da melhor rede dentre as vizinhas
		if (cadaCusto < menorCustoVizinha) {
			menorCustoVizinha = cadaCusto;
			melhorRedeVizinha = redeVizinha;
		}

}
redeAtual = melhorRedeVizinha;


try {
		//guardando a melhor de todas as redes da execucao
		if(menorCustoVizinha<custoMinimo && (!Tabu.ehTabu(redeAtual))){
			custoMinimo = menorCustoVizinha;
			bestRede = redeAtual;
			Tabu.inserir(redeAtual);
		}
} catch (IndexOutOfTabuException e1) {
		e1.printStackTrace();
}
System.out.println("REDE ATUAL: "+menorCustoVizinha
			+" MELHOR REDE:" +custoMinimo);
//renomeando o arquivo de saidas geradas durante o treinamento
File newFile = new File("saidaTreina.dat");
if(newFile.exists())
		newFile.delete();
file.renameTo(newFile);

bestRede.imprimirRede("redeAtual.xml");

/*
 * FIM DA ETAPA DE TREINAMENTO
 */

/*
 * ETAPA DE VALIDACAO
 */

mseValida = bestRede.calculaCusto(ds3, funcao);
if (mseValida < mseValMinimo) {
		mseValMinimo = mseValida;
}

//criterio gl5
gl = 100 * ((mseValida / mseValMinimo) - 1);

if (gl > 5) {
		System.out.println("Parada antecipada, iteracao " + i);
}
```

```
                //renomeando o arquivo de saidas geradas durante a validacao
                File newFile2 = new File("saidaValida.dat");
                if(newFile2.exists())
                        newFile2.delete();
                file.renameTo(newFile2);

                /*
                 * FIM DA ETAPA DE VALIDACAO
                 */

                System.out.println("treina atual: "+ menorCustoVizinha + "  treina minimo: "
                                + custoMinimo);
                System.out.println("val atual: " + mseValida + "  val minimo: "
                                + mseValMinimo + "  GL5: " + gl);
        }

        /*
         * ETAPA DE TESTE
         */

        double mseTeste = bestRede.calculaCusto(ds2, funcao);
        System.out.println("mse teste: " + mseTeste);


        /*
         * FIM DA ETAPA DE TESTE
         */

        //renomeando o arquivo de saidas geradas durante o teste
        File newFile3 = new File("saidaTeste.dat");
        if(newFile3.exists())
                newFile3.delete();
        file.renameTo(newFile3);

        SnnsNetGnerator.generate("redeAtual.xml");
    }
}
```