SERVIÇO DE PÓS-GRADUAÇÃO DO ICMC-USP

Data de Depósito: 24.11.1999

Assinatura: _____

Walter D. G. L.

Um Estudo Comparativo de Técnicas de *Pruning* para Redes Neurais Artificiais¹

Marco Antonio Alvarez Vega

Orientador: *Prof. Dr. André Carlos P. de Leon F. de Carvalho*

Monografia apresentada ao Instituto de Ciências Matemáticas e de Computação - ICMC-USP, como parte dos requisitos para obtenção do título de Doutor em Ciências – Área: Ciências de Computação e Matemática Computacional.

São Carlos
Novembro/1999

¹Trabalho realizado com apoio financeiro da FAPESP

Abstract

Artificial Neural Networks offer an efficient solution to a wide variety of practical problems. However, choosing an appropriate topology for an Artificial Neural Network is a difficult task. Generally, the process of choosing the number of free parameters is informal and networks are trained with different complexities and topologies until the one with the best performance be selected. This procedure usually does not generate minimal size networks, which can make their implementation unfeasible. In the present work, a comparative study of Pruning techniques is presented. Such techniques are used to improve the complexity of networks, reducing their size without considerably degrading their generalization ability. A large number of experiments was performed with a set of previously selected techniques. The results were analyzed in order to study the behavior of the Pruning techniques in general, as well as identifying those which provide the best performance.

Resumo

Redes Neurais Artificiais (RNAs) têm proporcionado uma solução eficiente para uma grande variedade de problemas práticos. Infelizmente, a seleção dos parâmetros ideais para o processo de aprendizado, bem como a escolha da topologia adequada, não são tarefas triviais. Geralmente, o processo de escolha do número de parâmetros livres é informal, e as redes são treinadas com diferentes topologias e complexidades até que a de melhor desempenho seja encontrada. Este procedimento nem sempre produz redes de tamanho mínimo, o que em muitos casos inviabiliza a implementação. Nesta dissertação é apresentado um estudo comparativo de diversas técnicas de *Pruning*, as quais têm como objetivo minimizar a complexidade da rede, sem degradar sua capacidade de generalização. Um grande número de experimentos foi realizado, utilizando diversas técnicas previamente selecionadas. Uma análise dos resultados obtidos é também apresentada, indicando o comportamento das técnicas de *Pruning* em geral, e identificando as de melhor desempenho.

Agradecimentos

Meus agradecimentos a Deus, por me conduzir por este caminho e me dar a graça de ter podido vivenciar a experiência de estar vinculado a um programa de mestrado. A minha família pelo apoio brindado a todo momento. Ao meu orientador, Prof. Dr. André de Carvalho, pela dedicação e palavras de estímulo durante todo o mestrado, e por ter me conduzido até o final deste processo. Aos meus amigos, muito obrigado pela ajuda nos mais diversos momentos e aos integrantes do LABIC, que formam um grupo humano maravilhoso.

Sumário

1	Introdução	1
1.1	Motivação e Objetivos	2
1.2	Organização do Texto	5
2	Redes Neurais Artificiais	6
2.1	Motivação Biológica	6
2.2	Porque RNAs?	8
2.3	Modelo de um Neurônio	8
2.4	Processo de Aprendizado	11
2.4.1	Algoritmos de Aprendizado	11
2.4.2	Paradigmas de Aprendizado	12
2.5	Generalização e Overfitting	13
2.6	Topologia	14

2.6.1	Redes Feedforward	14
2.6.2	Redes FeedForward Modulares	16
2.7	Complexidade	16
2.8	Representação de Conhecimento	17
2.9	Considerações Finais	19
3	Multilayer Perceptron e Redes Neurais Construtivas	21
3.1	Redes Multilayer Perceptron	21
3.2	Algoritmo Backpropagation	24
3.3	Quando as Redes Convergem?	27
3.3.1	Topologia	27
3.3.2	Inicialização dos Pesos	28
3.3.3	Parâmetros do Aprendizado	29
3.3.4	Conjuntos de Treinamento e Teste	29
3.4	Redes Neurais Construtivas	30
3.4.1	Algoritmo Cascade Correlation	30
3.4.2	Algoritmo MTiling	34
3.5	Considerações Finais	37

4	Técnicas de Pruning	39
4.1	Motivação	39
4.2	Métodos de Cálculo de Sensitividade	42
4.2.1	Magnitude Pruning	42
4.2.2	Optimal Brain Damage	42
4.2.3	Optimal Brain Surgeon	43
4.2.4	Non-Contributing Units	43
4.2.5	Skeletonization	43
4.2.6	Método de Karnin	45
4.2.7	Algoritmo lprune	46
4.3	Métodos Penalizadores	49
4.3.1	Método de Chauvin	49
4.4	Métodos para Redes Construtivas	52
4.4.1	Pruning Cascade Correlation	52
4.4.2	Pruning MTiling	53
4.5	Considerações Finais	55
5	Descrição dos Experimentos	57

5.1	Introdução	57
5.2	Ferramentas Utilizadas	59
5.2.1	Stuttgart Neural Network Simulator	59
5.2.2	Constructive NN Learning Simulator	60
5.3	Experimentos com Redes MLP	61
5.3.1	Experimentos com Dados de Crédito do Proben1	64
5.3.2	Experimentos com Dados de Crédito do Banestado	67
5.3.3	Experimentos com Dados de Paridade	70
5.4	Experimentos com o Algoritmo MTiling	73
5.5	Análise dos Resultados	74
5.6	Considerações Finais	76
6	Conclusões	77
6.1	Estrutura do Texto	77
6.2	Contribuições e Trabalhos Futuros	79

Lista de Figuras

2.1	Modelo de uma célula nervosa (neurônio).	9
2.2	Modelo de um neurônio artificial.	9
2.3	Gráfico da função sigmóide.	10
2.4	(a) Rede não modular. (b) Rede modular.	17
3.1	Duas classes não linearmente separáveis.	22
3.2	Uma rede MLP comum.	23
3.3	Mínimo local e mínimo global.	26
3.4	Estado inicial da arquitetura Cascade Correlation.	32
3.5	Estado da rede após inserção da primeira unidade.	33
3.6	Estado da rede após inserção da segunda unidade.	33
3.7	Rede MTiling.	35
4.1	Exemplo de neurônios mortos.	54

4.2	Exemplo de neurônios correlacionados.	54
4.3	Exemplo de neurônios redundantes.	55
5.1	Distribuição dos dados.	62
5.2	Média do número de conexões após a poda.	66
5.3	Taxa média de classificação correta.	67
5.4	Média do número de conexões após a poda.	68
5.5	Taxa média de classificação correta.	69
5.6	Média do número de conexões após a poda.	72
5.7	Média do número de neurônios após a poda.	72
5.8	Taxa média de classificação correta.	73
5.9	Média do número de neurônios após a poda.	74
5.10	Resultado médio para os conjuntos de teste.	75

Lista de Tabelas

2.1	Comparação entre formas de processamento.	7
4.1	<i>Skeletonization</i> : algoritmo básico.	45
4.2	Algoritmo <i>lprune</i>	50
5.1	Parte do código utilizado nos experimentos.	63
5.2	Resultado médio para os conjuntos de teste.	67
5.3	Resultado médio para os conjuntos de teste.	70
5.4	Código utilizado para gerar os dados de paridade.	70
5.5	Exemplos de pares entrada-saída.	71
5.6	Resultado médio para os conjuntos de teste.	73

Capítulo 1

Introdução

As Redes Neurais Artificiais (RNAs) têm proporcionado uma solução eficiente para uma grande variedade de problemas práticos. Desafortunadamente, a seleção dos parâmetros ideais para o processo de aprendizado, bem como a escolha da topologia ¹ adequada não são consideradas tarefas triviais. Esta dissertação centra o foco no problema da escolha da topologia inicial. Encontrar esta topologia é sumamente importante, já que a escolha de uma topologia inadequada pode impedir a rede de encontrar uma boa solução. Ou a rede não aprende o problema ou é incapaz de generalizar. Uma rede com a menor complexidade ² capaz de aprender o problema rapidamente e generalizar perfeitamente é considerada ótima. No entanto, é muito difícil, mas não impossível, encontrar manualmente uma topologia ótima para problemas não triviais.

Na presente dissertação é apresentado um estudo de diversas técnicas de *Pruning*, as quais têm como objetivo minimizar a complexidade da rede sem degradar sua capacidade de generalização. Além de estudar o mecanismo por trás destas técnicas foi realizado também um estudo comparativo do desempenho de algumas delas quando aplicadas a Redes *Multilayer Perceptron* (MLP) e Redes Neurais Construtivas (RNCs).

¹A topologia de uma rede é definida pelas unidades da rede e pela forma como estão conectadas.

²A complexidade de uma rede é definida pela quantidade de seus parâmetros livres.

1.1 Motivação e Objetivos

É sabido que o cérebro humano é composto por mais de 100 bilhões de células nervosas (chamadas de neurônios), interconectadas entre si por meio de sinapses e de uma maneira altamente complexa. Assim, grande parte do cérebro pode contribuir no processamento de um dado sinal [Schwartz, 1988]. Acredita-se que esta conectividade é responsável pelas características atribuídas à inteligência. As RNAs foram criadas como uma tentativa de simular as interações entre os neurônios existentes no cérebro. Elas são formadas por várias unidades de processamento, que simulam o comportamento dos neurônios, ligadas através de conexões que simulam as sinapses e podem ser utilizadas para realizar tarefas que envolvam aprendizado.

Um dos modelos de RNAs mais populares é o de redes *Multilayer Perceptron* treinadas com o algoritmo *backpropagation* [Rumelhart et al., 1986a]. Elas têm mostrado bons resultados em uma variedade de aplicações e são muito flexíveis e fáceis de implementar. O algoritmo de aprendizado *backpropagation* utiliza o paradigma de aprendizado supervisionado, no qual a rede aprende a partir de um conjunto de treinamento que consiste em um conjunto de pares de vetores do tipo entrada-saída. O vetor de entrada é aplicado à camada de entrada da rede e o vetor de saída é usado como um “professor” que indica a saída correta ou desejada para cada padrão de entrada. A rede aprende ajustando os pesos de suas conexões até que as diferenças entre a saída desejada e a atual sejam aceitavelmente baixas.

Quando um sistema é treinado com exemplos, é muito importante verificar como este sistema se comporta com exemplos não apresentados durante a fase de treinamento. Na realidade, na maioria dos problemas práticos é impossível apresentar todos os exemplos possíveis durante o treinamento. Por este motivo é imperiosa a necessidade do sistema conseguir generalizar a partir do conjunto de treinamento.

Em RNAs, a habilidade de generalizar é dependente de sua estrutura interna, por isso, o problema de determinar seu tamanho apropriado é crucial. Desafortunadamente, ainda não

existe um roteiro bem definido que guie o projeto de uma estrutura apropriada para um determinado problema. Aqui surge um dilema. já que, tanto as redes grandes como as pequenas apresentam diversas vantagens. Quando a rede possui vários parâmetros livres (pesos e *thresholds*), não somente o aprendizado pode ser mais rápido mas o mínimo local pode ser mais facilmente evitado. Redes grandes podem formar regiões de decisão complexas e exibem um alto grau de tolerância a falhas. Por outro lado, resultados teóricos [Baum and Haussler, 1989] e empíricos mostram que redes com poucos parâmetros exibem uma melhor generalização. Ainda, o conhecimento embutido em pequenas redes é presumivelmente mais fácil de ser interpretado e assim a extração de regras simples pode ser facilitada. Por último, do ponto de vista da implementação, redes pequenas requerem menos recursos, permitindo sua utilização em um universo maior de ambientes.

Na maioria de experimentos com RNAs, a topologia da rede é escolhida de forma empírica. Para resolver o problema de escolher automaticamente o tamanho mais apropriado, três abordagens são propostas. A primeira implica em, inicialmente, criar redes pequenas e gradativamente ir adicionando unidades ou camadas até completar o aprendizado. Estes são os algoritmos construtivos [Fiesler and Beale, 1996]. A segunda engloba os métodos de *Pruning*, os quais são aplicados a redes já treinadas e suficientemente grandes para aprender os dados. O objetivo destes métodos é identificar os elementos redundantes e removê-los da rede. Esta última abordagem combina as vantagens de treinar redes grandes (aprendizado rápido e minimização da incidência de mínimos locais) com as de utilizar redes pequenas (generalização melhorada). A terceira, e última abordagem refere-se à utilização de algoritmos genéticos [Radcliffe, 1991], os quais determinam a estrutura mais apropriada por meio de um processo evolutivo em que cada “indivíduo” é uma RNA.

Os métodos de *Pruning* existentes podem ser classificados em dois grandes grupos. Os que fazem uso de **termos de penalidade** e os que realizam **cálculos de sensibilidade**. Os do primeiro grupo modificam a função que computa o erro da rede de forma que a adaptação dos pesos efetuada pelo algoritmo *backpropagation* penalize as conexões consideradas irrelevantes, dirigindo o valor de seus pesos para 0 durante o treinamento. Conexões com pesos abaixo de

um valor determinado podem ser removidas. Já os do segundo grupo calculam a sensibilidade do erro em relação à remoção de uma conexão. Aquelas que produzirem as menores sensibilidades podem ser removidas. Nos métodos pertencentes a este último grupo, o treinamento e a eliminação das unidades irrelevantes são processos executados em fases completamente separadas. Estes procedimentos, aplicados após o treinamento, não interferem no processo de aprendizado, mas alguns requerem algum período de “re-treino” para manter o desempenho original da rede (em termos de taxas de erro).

A inexistência de um trabalho que compare o desempenho obtido por um número razoável de métodos de *Pruning*, aliado à ausência de evidências que comprovem que um dos métodos de *Pruning* propostos na literatura garanta a obtenção da topologia considerada ótima [Thimm and Fiesler, 1995], justifica a realização de um estudo comparativo entre eles. A eficiência de cada método pode ser estimada segundo três critérios importantes: tamanho final da rede, capacidade de generalização e tempo de treinamento.

No presente trabalho, além de estudar o comportamento de redes MLP e Redes Neurais Construtivas, diversas técnicas de *Pruning* disponíveis na literatura foram estudadas. Cinco delas foram selecionadas para uma análise mais detalhada, envolvendo testes com 3 conjuntos de dados diferentes (duas são aplicações reais).

Os resultados obtidos foram analisados e discutidos de forma a identificar aqueles métodos que tenham um rendimento aceitável. Nos capítulos finais da presente dissertação são apresentados estes resultados acompanhados de uma análise geral.

Adicionalmente, as técnicas de *Pruning* podem ser aplicadas a redes treinadas com algoritmos construtivos. Entre os algoritmos de aprendizado para Redes Neurais Construtivas encontrados na literatura destacam-se o algoritmo *Cascade Correlation* [Falhman and Lebiere, 1990] e o algoritmo *MTiling* [Yang et al., 1996b]. Estes, foram objeto de estudo neste trabalho e ao mesmo tempo um conjunto de dados foi escolhido para realizar experimentos com o algoritmo *MTiling*. O objetivo é verificar a aplicabilidade de técnicas de *Pruning* em Redes Neurais Construtivas.

1.2 Organização do Texto

O texto desta dissertação está organizado da seguinte forma em seis capítulos:

No capítulo 2 é feita uma introdução às Redes Neurais Artificiais, abordando pontos importantes como Motivação para seu aparecimento, Complexidade, Topologias, Funcionamento, Generalização e *Overfitting*. Estes conceitos são fundamentais para o entendimento e cumprimento dos objetivos propostos por este trabalho.

No capítulo 3 é apresentada uma descrição detalhada das redes MLP, do algoritmo *backpropagation* e dos algoritmos construtivos *Cascade Correlation* e *MTiling*, os quais foram utilizados durante o desenvolvimento deste trabalho.

No capítulo 4 são descritas dez técnicas de *Pruning*, sendo nove de cálculo de sensibilidade (duas para redes construtivas e sete para redes MLP) e uma que utiliza termos de penalidade. Algumas destas técnicas serão implementadas e testadas, sendo os resultados dos testes utilizados para realizar avaliar o desempenho das técnicas em questão.

No capítulo 5, são apresentados os resultados dos experimentos realizados com as técnicas de *Pruning*. Além de descrever as bases de dados utilizadas, são mostrados gráficos e tabelas correspondentes ao desempenho e eficácia das mesmas.

Finalmente no sexto e último capítulo são apresentadas algumas considerações finais, tais como conclusões, contribuições do presente trabalho e sugestões para futuros trabalhos na área.

Capítulo 2

Redes Neurais Artificiais

Este capítulo fornece uma breve introdução às Redes Neurais Artificiais. Inicialmente, serão expostos os fundamentos biológicos que motivaram o aparecimento das RNAs e o modelo computacional estabelecido. Depois, serão abordados os principais tópicos relacionados às RNAs como: representação de conhecimento, aprendizado, arquiteturas e classificação de padrões. Finalmente, serão abordados temas como *overfitting* e capacidade de generalização de RNAs.

2.1 Motivação Biológica

O estudo de RNAs foi motivado ao ficar claro que o cérebro humano processa informações de uma maneira completamente diferente dos computadores digitais convencionais [Haykin, 1994]. Computadores convencionais (inspirados na máquina de Von-Neumann) resolvem problemas de forma sequencial e monoprocessada, levando vantagem ao realizar cálculos matemáticos, os quais são computados com muita precisão e rapidez. Por outro lado, o cérebro humano tem outras habilidades e consegue realizar tarefas que são muito complexas para os computadores, como por exemplo: reconhecimento de padrões, percepção do ambiente, análise

de sons, imagens, etc. Essas tarefas são facilmente realizadas pelo cérebro humano devido à sua natureza massivamente paralela, que lhe permite armazenar conhecimento e processar informações muito rapidamente. O cérebro humano é capaz de organizar os neurônios de forma que possam efetuar tarefas independentes e simultaneamente. Na Tabela 2.1 podem ser observadas algumas das principais diferenças entre a forma de processamento do cérebro humano e dos computadores convencionais.

Computador	Cérebro
Sequencial e centralizado	Paralelo e distribuído
Sem tolerância a falhas	Tolerante a falhas
Alta velocidade	Baixa velocidade
Ambiente bem definido, restrito	Ambiente pouco definido, irrestrito

Tabela 2.1: Comparação entre formas de processamento.

Ao nascer, o cérebro possui uma estrutura própria e a capacidade de poder criar suas próprias “regras” (experiência). Esta experiência é desenvolvida ao longo do tempo, mas principalmente, nos 2 primeiros anos de vida, onde estima-se que, em torno de 1 milhão de sinapses são formadas por segundo. As sinapses são importantes unidades estruturais e funcionais encarregadas de mediar as interações entre os neurônios.

De uma forma geral, uma RNA é uma máquina criada para modelar a forma como o cérebro humano realiza uma determinada tarefa, esta rede pode ser implementada utilizando componentes eletrônicos ou simulada em *software* em um computador. Em [Haykin, 1994] pode-se encontrar a seguinte definição para RNAs:

Uma rede neural é um processador distribuído, massivamente paralelo que tem a capacidade natural de armazenar conhecimento e deixá-lo disponível para uso posterior. Ela se assemelha ao cérebro humano em dois aspectos:

- 1. Conhecimento é adquirido pela rede por um processo de aprendizado*
- 2. Conexões entre os neurônios são usadas para armazenar conhecimento*

2.2 Porque RNAs?

As principais características de RNAs são sua capacidade de processamento altamente paralelo e a facilidade de poder **aprender** e logo **generalizar** a partir do conhecimento adquirido. Generalizar refere-se à capacidade de dar respostas razoáveis a entradas não apresentadas durante a etapa de treinamento (aprendizado). Essas características inerentes à forma de processamento, dotam as RNAs da possibilidade de resolver problemas extremamente complexos. Embora seja muito importante reconhecer que ainda há um longo caminho a ser percorrido para conseguir construir um computador que imite o comportamento do cérebro humano, é possível identificar diversas vantagens na utilização de RNAs, entre as quais se destacam:

1. A não linearidade presente nos neurônios, que proporciona à rede tratar de problemas em que o mecanismo gerador dos sinais de entrada é inerentemente não linear.
2. A plasticidade, que permite à RNA adaptar seus parâmetros internos às condições propiciadas pelo ambiente externo.
3. A capacidade de armazenar informação contextual, visto que o conhecimento é distribuído por toda a estrutura da rede, de forma que cada neurônio é potencialmente afetado pela atividade global de todos os outros neurônios. Consequentemente, a informação contextual é tratada com naturalidade por uma RNA.
4. A tolerância a falhas, que é muito importante em implementações em *hardware*.
5. A facilidade para ser implentada em circuitos VLSI.

2.3 Modelo de um Neurônio

Embora a funcionalidade do cérebro não seja atualmente conhecida na sua totalidade, um modelo simples pode ser desenvolvido. A pesquisa em RNAs iniciou-se para tentar aproveitar

as vantagens oferecidas pelo cérebro e simulá-las em computadores convencionais, permitindo desta forma a solução de problemas que ainda não podiam ser resolvidos, da forma desejada, pelos métodos tradicionais.

Um neurônio é a unidade de processamento fundamental para a operação de uma Rede Neural, daí a necessidade de criar um modelo eficiente para ele. No cérebro, cada neurônio está conectado a outros por meio de sinapses, que geralmente ligam um dendrito a um axônio. A Figura 2.1 mostra a estrutura de um neurônio biológico típico. Cada neurônio recebe estímulos (sinais de entrada) pelos dendritos. Quando a soma destes estímulos atinge um valor mínimo, o neurônio é **ativado** e transmite sua saída pelo axônio para outros neurônios.

Figura 2.1: Modelo de uma célula nervosa (neurônio).

Conforme visto, para poder construir RNAs é necessário modelar o neurônio. A Figura 2.2 mostra um possível modelo, no qual podem ser identificados 3 elementos básicos, que são descritos a seguir:

Figura 2.2: Modelo de um neurônio artificial.

1. Um conjunto de sinais de entrada I_j , provenientes de outros neurônios ou diretamente do padrão de entrada. Cada um destes sinais é caracterizado por um peso w_{ij} . Especificamente, cada sinal I_j proveniente da entrada j e conectado a um neurônio i é multiplicado por um peso w_{ij} . Os sinais de entrada do modelo representam os estímulos recebidos pelos dendritos do neurônio biológico. Os pesos simulam o comportamento das sinapses, podendo aumentar ou diminuir o estímulo.
2. Um somador, encarregado de efetuar a soma (net_i) dos sinais de entrada ponderados pelos seus respectivos pesos (como pode ser visto na Eq. 3.2). Estas operações constituem uma combinação linear.
3. Uma função de ativação (μ_i) aplicada ao resultado da Eq. 3.2, para limitar a amplitude do sinal de saída a um determinado intervalo finito. Geralmente, este intervalo é dado por $[0, 1]$ e $[-1, 1]$ e a função de ativação é não linear. A função mais frequentemente utilizada é a função sigmoide, mostrada na Figura 2.3.

Figura 2.3: Gráfico da função sigmóide.

O valor de saída da função de ativação é utilizado como entrada em outros neurônios ou pode ser considerado como o valor de saída final de uma rede.

2.4 Processo de Aprendizado

Entre todas as propriedades de uma RNA, aquela considerada de grande significância é a sua capacidade que possuem de aprender a partir de seu próprio ambiente e de melhorar seu desempenho por meio dele. O processo de aprendizado é geralmente um processo iterativo de adaptação aplicado aos parâmetros livres da rede (pesos e *thresholds*), em que conhecimento é armazenado a cada iteração. Haykin em [Haykin, 1994] define **aprendizado** no contexto de RNAs com as seguintes palavras:

Aprendizado é o processo no qual os parâmetros livres de uma RNA são adaptados por meio de um processo contínuo de estímulo exercido pelo ambiente no qual a rede está embutida. O tipo de aprendizado é determinado pela maneira como as mudanças dos parâmetros ocorrem.

Segundo Haykin, esta definição implica na seguinte sequência de eventos: a RNA é estimulada por um ambiente, logo, sofre mudanças ocasionadas por um estímulo e finalmente responde de uma maneira diferente, relacionada às mudanças ocorridas em sua estrutura interna.

2.4.1 Algoritmos de Aprendizado

Um algoritmo de aprendizado é um conjunto de regras bem definidas para a solução de um problema de aprendizado. Não há somente um, mas vários algoritmos de aprendizado que podem ser utilizados para treinar RNAs. O que diferencia estes algoritmos é a forma de atualização dos pesos das conexões. Os algoritmos existentes podem ser agrupados em quatro categorias: aprendizado por Correção de Erro, Hebbiano, Competitivo e de Boltzmann (maiores informações podem ser encontradas em [Haykin, 1994]). Na presente dissertação será abordado apenas o aprendizado por correção de erro, uma vez que o mesmo é utilizado por redes MLP e RNCs, que são as redes utilizadas neste trabalho.

Aprendizado por correção de erro

Seja $d_j(n)$ a saída desejada do neurônio j para o n -ésimo padrão e $y_j(n)$ a saída atual de j . Pode-se calcular o erro do neurônio j para este padrão como sendo:

$$e_j(n) = d_j(n) - y_j(n) \quad (2.1)$$

Com os valores $e_j(n)$ de todos os neurônios da camada de saída da rede pode-se calcular a seguinte função de erro:

$$\mathcal{E}(n) = \frac{1}{2} \sum_j e_j^2(n) \quad (2.2)$$

O propósito deste tipo de aprendizado é minimizar o valor da função de erro. Pode-se conseguir este objetivo mudando iterativamente os valores dos pesos das conexões da rede.

Ao plotar os pontos do gráfico **função de erro X pesos**, obtém-se uma superfície denominada **Superfície de Erro**, onde pode-se observar dois casos: o primeiro ocorre quando a superfície possui um mínimo global, e o segundo quando além do mínimo global ela possui mínimos locais. Em ambos os casos, o objetivo dos algoritmos por correção de erro é, a partir de um ponto arbitrário da superfície, (determinado pelos valores iniciais dos pesos) “andar” por ela até achar o mínimo global. No primeiro caso, este objetivo pode ser atingido facilmente, mas no segundo caso existe a possibilidade de cair em um mínimo local e nunca chegar ao mínimo global.

2.4.2 Paradigmas de Aprendizado

A maneira como uma RNA se relaciona com o seu ambiente externo também deve ser considerada no projeto da rede. Um paradigma de aprendizado refere-se ao “modelo” do ambiente

no qual a RNA está operando. Dentro deste contexto, existem os seguintes paradigmas de aprendizado:

- **Aprendizado Supervisionado**, ocorre quando se conta com a presença de um agente externo que cumpre o papel de professor. Conceitualmente, pode-se pensar em um professor, que possui o conhecimento do ambiente, na forma de mapeamentos Entrada-Saída. A rede recebe estes mapeamentos, denominados padrões ou exemplos, processa as Entradas e compara suas respostas com as Saídas desejadas. O erro produzido é utilizado para ajustar os parâmetros da rede.
- **Aprendizado por Reforço**, ocorre quando um crítico externo avalia a resposta fornecida pela rede, direcionando o ajuste dos pesos. O aprendizado é *online* e efetuado por meio de um processo de tentativa e erro que procura maximizar um dado índice de desempenho, denominado “sinal de reforço” [Haykin, 1994]. É baseado em estudos psicológicos sobre o aprendizado de animais.
- **Aprendizado não Supervisionado**, ocorre quando não se conta com a presença de um professor ou crítico, ou seja, quando não há resposta desejada ou sinal de reforço para os padrões de entrada. Este tipo de aprendizado pode ser utilizado com um algoritmo Competitivo ou Hebbiano.

2.5 Generalização e Overfitting

A capacidade de generalização de uma RNA está diretamente relacionada com a forma como ela se comporta com padrões que não lhe foram apresentados durante o treinamento. Uma RNA que possui boa generalização é aquela que, depois de treinada, consegue dar a resposta certa para padrões desconhecidos. A compatibilidade entre a complexidade da rede e do problema (os dados) é um problema comum e em aberto em RNAs. Se a rede tem muitos parâmetros livres (alto grau de liberdade), ela pode facilmente *memorizar* os dados (padrões de treinamento), ou seja, não generaliza. Desta forma, pode responder bem durante

o aprendizado, mas falha com os padrões de teste. Este evento é conhecido como *overfitting*. O problema torna-se maior quando o ruído (presente em qualquer conjunto de dados reais) é levado em conta, assim, a rede pode estar memorizando o ruído. Da mesma forma, se a complexidade da rede é muito menor que a complexidade do problema, a rede pode ser capaz de não conseguir encontrar uma função que aprenda os dados de entrada, fenômeno conhecido como *underfitting*.

Segundo Haykin [Haykin, 1994], a generalização é influenciada por três fatores: tamanho e eficiência do conjunto de treinamento, complexidade do problema e a arquitetura da rede. Redes pequenas têm apresentado bons índices de generalização. Este é um motivo a mais para utilizar técnicas de *Pruning* em RNAs. Mais informações sobre generalização podem ser encontradas em [Lawrence et al., 1996].

2.6 Topologia

A primeira questão a ser resolvida depois de modelar os neurônios, também chamado de unidade de processamento, é definir como eles serão conectados entre si. A forma como a rede é conectada é chamada de **topologia** ou **arquitetura**. A topologia de redes usadas no presente trabalho é a de redes *feedforward*. Este tipo de redes representa uma generalização das redes MLP, as quais formam a categoria de rede mais utilizada em computação neural. Embora muitos algoritmos de treinamento sejam conhecidos por treinarem redes MLP, eles na realidade treinam qualquer rede que tenha a topologia *feedforward*. Esta generalização de redes MLP é muito útil ao lidar com técnicas de *Pruning* e abordagens evolutivas para determinar uma topologia inicial ótima.

2.6.1 Redes Feedforward

As redes *feedforward* formam uma classe especial de RNAs e se caracterizam por propagar o sinal de entrada em apenas uma direção através da rede. Todas as unidades pertencentes

à rede devem fazer parte de uma dada camada. As redes, por sua vez, podem ter 3 tipos diferentes de camadas: de entrada, intermediárias e de saída. Há sempre exatamente uma camada de entrada e uma camada de saída. A camada de entrada recebe o sinal de entrada e sua única tarefa é propagá-lo para as seguintes camadas. Já a camada de saída, computa sua saída depois que todos os sinais de entrada tenham sido propagados pela rede. Todas as unidades que não pertençam às camadas de entrada e de saída são unidades escondidas, as quais são organizadas em camadas intermediárias (ou escondidas). Uma unidade é dita pertencer a uma certa camada seguindo a seguinte definição: *A camada de uma unidade é calculada considerando o maior caminho de conexões¹ existente entre a camada de entrada e a unidade.* Visto que somente são permitidas conexões unidirecionais sem *loops* ou conexões recursivas, é possível afirmar que sempre pode-se determinar o maior caminho entre a camada de entrada e uma unidade da rede.

Uma vez que os sinais de entrada são entregues à camada de entrada, os sinais são propagados pela rede de camada em camada, em que cada unidade atua como descrito anteriormente, até chegar na camada de saída. Neste ponto, os resultados da rede podem ser analisados e a rede tem condições de ajustar seus parâmetros internos para poder resolver o problema. Muitas vezes, isto não é suficiente e a topologia deve ser alterada.

Uma simples definição para a topologia *feedforward* é que qualquer conexão w_{ji} (da unidade j para a unidade i) deve cumprir o critério $j < i$. Outra forma de descrever redes *feedforward* é que podem existir conexões apenas entre unidades em camadas não adjacentes. Se a rede não cumpre os critérios descritos acima, é porque possui algumas conexões ditas de *feedback*. Esta classe de redes é chamada de redes **recorrentes**. A grande maioria de algoritmos de treinamento que trabalham com redes *feedforward* não trabalham com redes **recorrentes** sem que sejam feitas modificações.

¹Um caminho de conexões é dado pelo número de conexões consecutivas.

2.6.2 Redes FeedForward Modulares

A classe de redes *feedforward* pode ser dividida em duas sub-classes: modulares e não modulares. Uma ilustração das mesmas pode ser observada na Figura 2.4. A rede na Figura 2.4 (a) é uma rede não modular, e não contém “sub-redes”. As camadas adjacentes desta rede estão conectadas completamente entre si. Já a rede da Figura 2.4 (b) é uma rede modular, ela contém duas sub-estruturas independentes (A e B), as quais não estão interconectadas. A modularidade é utilizada com o intuito de dividir redes em sub-redes, capazes de resolver partes do problema independentemente. Redes *feedforward* modulares possuem certas vantagens sobre as não-modulares, entre as quais pode-se destacar a baixa complexidade: Se uma rede modular é capaz de aprender os dados de entrada e solucionar o problema, tem a vantagem de, devido à sua baixa complexidade, apresentar um espaço de busca com menos mínimos locais. A busca pela solução pode atingir seus objetivos mais rapidamente quando é usado um algoritmo de aprendizado que usa gradiente descendente, como o *backpropagation*. Um estudo mais detalhado sobre redes modulares pode ser encontrado em [Boers and Kuiper, 1992]. Uma outra vantagem destas redes é a sua capacidade de lidar com problemas de interferência. Estes problemas podem acontecer quando redes não modulares são treinadas para aprender duas partes independentes de um determinado problema [Rueckl et al., 1989]. Uma rede grande o suficiente para aprender o problema não pode ser treinada porque ao aprender uma parte simplesmente “esquece” a outra. Esta rede apresenta um comportamento oscilatório e não é capaz de aprender as duas partes ao mesmo tempo. As redes modulares constituem uma boa solução para este problema, já que elas podem assignar uma sub-rede para aprender uma parte e outra sub-rede para aprender a outra parte.

2.7 Complexidade

A **complexidade** de uma rede é dada pelo seu número de parâmetros livres. Assim, a seleção de uma topologia afeta diretamente a complexidade da rede. Uma rede com duas camadas intermediárias completamente conectadas é capaz, na teoria, de representar qualquer

Figura 2.4: (a) Rede não modular. (b) Rede modular.

mapeamento que outra topologia seja capaz de representar. No entanto, pode ser muito mais difícil treinar esta rede e sua capacidade de generalização pode ser pior que uma com topologia mais simples.

Existem métodos propostos para otimizar esta estrutura. Os mesmos podem ser agrupados em: algoritmos construtivos, algoritmos genéticos e técnicas de *Pruning*. Estes últimos, que são objeto de estudo do presente trabalho, são uma alternativa para a tarefa de controlar a complexidade da rede ajustando o número de seus parâmetros livres.

2.8 Representação de Conhecimento

Segundo Haykin [Haykin, 1994] o conceito de Conhecimento pode ser explicitamente descrito pela seguinte definição:

Conhecimento se refere à informação armazenada ou aos modelos utilizados por uma pessoa ou máquina para interpretar, prever e responder apropriadamente ao ambiente externo.

As características primárias da representação de conhecimento são dadas por: (1) como a informação está disponível primitivamente; e (2) como a informação é codificada fisicamente para ser usada posteriormente. Em aplicações do mundo real utilizando máquinas inteligentes,

pode-se afirmar que uma boa solução depende diretamente de uma boa representação do conhecimento. As RNAs representam uma classe de máquinas inteligentes, e normalmente as possíveis formas de representação de conhecimento, desde as entradas até os parâmetros internos da rede são completamente diversas, o qual torna o desenvolvimento de uma solução satisfatória um verdadeiro desafio ao usar RNAs. A principal tarefa de uma RNA é aprender um modelo do ambiente externo e manter este modelo sempre consistente com o ambiente, de modo a alcançar os objetivos específicos da aplicação.

Toda RNA aprende a partir de um conjunto de exemplos, no caso do aprendizado supervisionado cada exemplo consiste de um par entrada-saída: um sinal de entrada (geralmente um vetor de características) e sua saída desejada correspondente. Desta forma, o conjunto inicial de exemplos representa o conhecimento do ambiente no qual a rede vai operar. Dado este conjunto de exemplos, o projeto de uma RNA pode proceder da seguinte maneira:

1. Uma arquitetura apropriada é selecionada com uma camada de entrada formada por um número de unidades igual ao número de características de cada padrão do conjunto de entrada e uma camada de saída formada por um número de unidades igual ao tamanho de cada padrão de saída. A escolha do número de camadas intermediárias, assim como o número de unidades presentes em cada uma delas ainda é uma questão em aberto. Não existe uma forma bem definida para encontrar uma rede capaz de aprender os dados e ao mesmo tempo oferecer uma topologia com complexidade mínima. O uso de técnicas de *Pruning* tenta, senão resolver, de alguma forma amenizar este problema. Neste primeiro passo, um subconjunto de exemplos é usado para treinar a rede utilizando algum algoritmo de aprendizado. Esta fase é denominada *aprendizado*.
2. O desempenho da rede é medido com um subconjunto de exemplos que não foram apresentados à rede anteriormente, geralmente este conjunto é chamado de conjunto de teste. Esta segunda fase serve para medir a capacidade de *generalização* da rede.

Em qualquer RNA, a representação do conhecimento adquirido é definido pelos valores dos parâmetros livres (pesos e *thresholds*) da rede. Isto representa uma enorme desvantagem em

relação aos algoritmos de aprendizado simbólico visto que a extração de regras precisa ser realizada por um agente externo.

2.9 Considerações Finais

Neste capítulo foram abordados aspectos teóricos genéricos sobre RNAs, dando maior ênfase aos modelos e paradigmas que serão utilizados na investigação de técnicas de *Pruning*. De uma forma geral, pode-se concluir que RNAs são dotadas de um elevado poder computacional devido a sua natureza massivamente paralela e à alta conectividade entre suas unidades de processamento. São inspiradas no comportamento do cérebro, do qual extraem as principais características que fazem parte do processo de aprendizado humano, como a capacidade de generalizar e de aprender a partir do seu ambiente.

O aprendizado, no contexto de RNAs, é definido como o processo pelo qual os parâmetros de uma RNA são adaptados por um processo de estímulo pelo ambiente no qual a rede está embutida [Haykin, 1994]. Assim, a rede é estimulada pelo ambiente, sofre mudanças ocasionadas pelo estímulo e finalmente responde de uma maneira diferente devido às mudanças em sua estrutura interna. Com base nestas definições, existem diferentes algoritmos e paradigmas de aprendizado. Um algoritmo de aprendizado refere-se à maneira pela qual a rede muda sua estrutura interna em resposta ao novo estímulo. Por sua vez, um paradigma de aprendizado descreve a maneira como a rede se relaciona com o seu ambiente externo. Os experimentos a serem realizados neste trabalho serão efetuados utilizando algoritmos por correção de erro combinados com o paradigma de aprendizado supervisionado.

Historicamente, as RNAs surgiram em meio a um entusiasmo geral na comunidade científica, mas paulatinamente foram aparecendo algumas limitações nos modelos existentes que provocaram um descaso e falta de apoio para pesquisas e aplicações na área. Apesar disso, alguns pesquisadores foram persistentes e continuaram trabalhando. Graças a eles, atualmente, o panorama é diferente e o interesse pelas RNAs cresce consideravelmente, o que pode ser

constatado na aplicação delas em cada vez mais áreas e problemas diferentes.

A maioria das aplicações de RNAs geralmente são divididas em 3 categorias: **classificação** (reconhecimento de padrões), onde a rede tenta classificar padrões em categorias predefinidas; **previsão**, onde a rede tenta extrapolar uma série de entradas, e **controle**, onde a rede é usada interativamente para guiar algum processo ou dispositivo externo. As duas primeiras são basicamente casos de aproximação, em que procura-se aproximar alguma função.

Capítulo 3

Multilayer Perceptron e Redes Neurais Construtivas

No presente capítulo serão descritas as redes *Multilayer Perceptron*, bem como o algoritmo de treinamento *backpropagation*. Serão abordados os fatores que podem levar as redes a encontrar a solução desejada ou não. Finalmente, dois modelos de Redes Neurais Construtivas são apresentados, os quais são denominados *Cascade Correlation* e *MTiling*.

3.1 Redes Multilayer Perceptron

O *Perceptron* é a forma mais simples de RNAs utilizadas para classificação de padrões ditos linearmente separáveis. Basicamente, é formado por um neurônio com conexões ajustáveis (ligadas às entradas) utilizando na saída uma função de ativação do tipo *threshold*, como o modelo apresentado na seção 2.3. O algoritmo utilizado para ajustar seus parâmetros internos foi proposto em [Rosembblatt, 1958], onde Rosembblatt provou o **Teorema da Convergência do Perceptron**, o qual dizia que se os padrões utilizados para treinar o *Perceptron* pertencem a classes linearmente separáveis, então o algoritmo converge e produz uma superfície de

decisão na forma de um hiperplano entre as duas classes.

Posteriormente, Minsky e Papert em [Minsky and Papert, 1969] analisaram matematicamente o *Perceptron* e constataram que o mesmo não era capaz de solucionar problemas que não sejam linearmente separáveis. Eles demonstraram que, para este tipo de problemas, o *Perceptron* era incapaz de generalizar a partir dos dados de entrada e acreditavam que não seria possível treinar com eficiência RNAs contendo mais de uma camada. Quando duas classes *A* e *B* não podem ser separadas por um simples hiperplano, é necessário ter uma combinação de hiperplanos mais sofisticada para classificar os dados. Na Figura 3.1 pode ser observado como a classificação de padrões foi aproximada utilizando 2 planos. Para resolver este problema é necessário contar com mais de um neurônio e mais de uma camada.

Figura 3.1: Duas classes não linearmente separáveis.

Somente na metade da década de oitenta, com o desenvolvimento do algoritmo de treinamento *backpropagation* por Rumelhart, Hinton e Williams em [Rumelhart et al., 1986a], foi mostrado que é possível treinar, com bom desempenho, redes com camadas intermediárias, derivando assim no modelo de redes mais utilizado atualmente, as redes *MultiLayer Perceptron* (MLP). Este tipo de redes é capaz de solucionar problemas que não são linearmente separáveis e utilizam um algoritmo de aprendizado por correção de erro com um paradigma de aprendizado supervisionado. As mesmas são assim chamadas porque possuem mais de uma camada (ao contrário da rede *Perceptron*).

As redes MLP podem ter três tipos diferentes de camadas: de entrada, intermediárias (es-

condidas) e de saída. Há exatamente uma camada de entrada e uma camada de saída. A primeira recebe os padrões de entrada e sua única tarefa é propagá-los pela rede para as próximas camadas. Os neurônios da última camada, por sua vez, calculam as saídas depois que o sinal de entrada tenha sido propagado por todas as camadas intermediárias. Cada neurônio das camadas intermediárias calcula sua saída e envia o sinal para neurônios das próximas camadas. A Figura 3.2 mostra um exemplo de uma rede MLP.

Figura 3.2: Uma rede MLP comum.

Entre cada unidade de processamento existem conexões que possuem valores (denominados pesos) associados a elas. Estes valores são adaptados a medida que o treinamento é realizado, possibilitando assim o armazenamento de conhecimento na rede.

Segundo Haykin [Haykin, 1994], as redes MLP possuem três características distintas:

1. O modelo de cada neurônio inclui uma **não linearidade** na sua saída, e esta **não linearidade** é suave (diferenciável em qualquer ponto). Para conseguir saídas não lineares torna-se necessário aplicar uma função de ativação não linear, como por exemplo, a função sigmóide mostrada na Eq. 3.1:

$$y_j = \frac{1}{1 + e^{-net_j}} \quad (3.1)$$

2. Possuem uma ou mais camadas intermediárias (escondidas) que não fazem parte nem da entrada nem da saída da rede. As unidades pertencentes a estas camadas permitem que a rede aprenda tarefas complexas, extraíndo progressivamente as características mais significativas dos padrões de entrada.
3. Apresentam um alto grau de conectividade determinado pelas conexões. Uma mudança na estrutura da rede ocasiona uma mudança nos pesos associados a cada conexão.

O poder computacional destas redes é grande, devido a estas características e à sua capacidade de aprender. Entretanto, estas mesmas características são as responsáveis pela presente “falta” de conhecimento do seu comportamento interno por parte dos pesquisadores. Embora atualmente estas redes tenham sido aplicadas satisfatoriamente para resolver diversos problemas, ainda há muita pesquisa a ser realizada até atingir um domínio teórico consistente e abrangente sobre seu funcionamento.

3.2 Algoritmo Backpropagation

O algoritmo *backpropagation* é um algoritmo de aprendizado supervisionado proposto para o treinamento de redes MLP. Neste algoritmo, um conjunto de padrões de treinamento deve ser apresentado à rede na forma de pares $[X(n), D(n)]$ onde o primeiro é um vetor que descreve o padrão de entrada n e o segundo é o vetor de saída desejada para o padrão n . Os pesos das conexões devem ser inicializados (geralmente com valores aleatórios) e iterativamente adaptados em um processo supervisionado a fim de aprender as características mais importantes dos padrões de treinamento.

O processamento das informações é efetuado em dois passos: no primeiro passo, para cada padrão apresentado à rede, as informações fluem a partir da camada de entrada, pelas camadas intermediárias, até a camada de saída, na qual é produzida a resposta para o padrão. No segundo passo, a resposta da rede é comparada com a resposta desejada para o padrão, sendo calculado o erro. Este erro é retro-propagado a partir da camada de saída até a camada de

entrada e os pesos das conexões da rede são ajustados utilizando a regra delta generalizada (Eq. 3.5) (uma adaptação da regra delta [Widrow and Hoff, 1960] para redes com mais de uma camada).

Ao treinar redes MLP com o algoritmo *backpropagation*, os parâmetros da rede (pesos e *thresholds*) são inicializados com valores aleatórios não muito altos, geralmente pertencentes ao intervalo $[0, 1]$. Posteriormente, os padrões são apresentados iterativamente até concluir o treinamento, que ocorre quando o erro da rede (veja Eq. 2.2) for considerado aceitavelmente baixo, ou após um certo número de iterações ou ciclos. A seguir são apresentados os passos efetuados para o treinamento de um dado padrão n em uma iteração:

1. Dado um padrão denotado por $[X(n), D(n)]$, o vetor $X(n)$ será apresentado à camada de entrada e o vetor $D(n)$ à camada de saída;
2. A partir da primeira camada após a camada de entrada, efetuar a propagação do sinal, camada por camada, computando a entrada total de cada neurônio j da camada l de acordo com a Eq. 3.2:

$$net_j^{(l)}(n) = \sum_i w_{ji}^{(l)}(n) \cdot y_i^{(l-1)}(n) \quad (3.2)$$

em que, $y_i^{(l-1)}(n)$ é a saída do neurônio i conectado à entrada do neurônio j na camada $l - 1$ para o padrão n e $w_{ji}^{(l)}(n)$ é o peso da conexão do neurônio i na camada $l - 1$ para o neurônio j na camada l . As saídas dos neurônios das camadas intermediárias e de saída serão definidas aplicando a cada valor $net_j^{(l)}(n)$ a função sigmóide descrita pela Eq. 3.1. Particularmente, se o neurônio j está na camada de saída, então, deve ser também computado o erro de acordo com a Eq. 2.1;

3. Efetuar a retro-propagação do sinal, computando os gradientes $\delta_j(n)$ dos neurônios a partir da camada de saída até a primeira camada intermediária. Para os neurônios da camada de saída é utilizada a Eq. 3.3 e para os das camadas intermediárias a Eq. 3.4:

$$\delta_j(n) = e_j(n) \cdot y_j(n) \cdot [1 - y_j(n)] \quad (3.3)$$

$$\delta_j^{(l)}(n) = y_j^{(l)}(n) \cdot [1 - y_j^{(l)}(n)] \cdot \sum_k \delta_k^{(l+1)}(n) \cdot w_{kj}^{(l+1)}(n) \quad (3.4)$$

O próximo passo é ajustar os pesos das conexões existentes em cada camada, de acordo com a regra delta generalizada (Eq. 3.5):

$$w_{ji}^{(l)}(n+1) = w_{ji}^{(l)}(n) + \alpha \cdot [w_{ji}^{(l)}(n) - w_{ji}^{(l)}(n-1)] + \eta \delta_j^{(l)}(n) \cdot y_i^{(l-1)}(n) \quad (3.5)$$

onde, η é a taxa de aprendizado e α é a taxa de *momentum*, que são variáveis definidas antes de iniciar o treinamento e podem ser alteradas durante o processo para otimizar o mesmo.

Sabe-se que o *backpropagation* utiliza uma técnica de gradiente descendente e que a rede ajusta seus pesos em busca do valor mínimo da função que computa o erro. Existem situações em que a rede incorre em um **mínimo local** que não é necessariamente a solução esperada. Assim, muitas vezes a técnica do gradiente descendente não consegue chegar até o **mínimo global**. Na Figura 3.3 observa-se um exemplo de **mínimo local** e **mínimo global** em uma superfície de erro bidimensional.

Figura 3.3: Mínimo local e mínimo global.

Como mencionado anteriormente, duas variáveis podem ser utilizadas para otimizar o desempenho da rede durante o treinamento, são elas a taxa de aprendizado e o termo *momentum*. Se a taxa de aprendizado utilizada é inicializada com um valor alto, pode-se afirmar que a rede aprenderá mais rapidamente uma vez que as mudanças nos pesos serão mais drásticas. No entanto, se ela for muito alta, ocorrerá uma grande oscilação nos pesos, dificultando a convergência. O ideal é ter a maior taxa de aprendizado possível sem que ocorra esta oscilação. O conceito de *momentum* está relacionado com a idéia de que as mudanças anteriores nos pesos têm influência na direção do ajuste atual. Com *momentum*, uma vez que os pesos começam a ir em uma direção, incentiva-se que eles continuem indo na mesma direção. Ele pode ajudar a rede a não cair em mínimos locais, assim como pode aumentar a velocidade do aprendizado.

O algoritmo *backpropagation* é praticamente padrão para treinar redes MLP, embora hoje existam algumas derivações. Segundo Haykin [Haykin, 1994], o seu poder computacional se baseia nos seguintes pontos: O uso de um método local para adaptar seus parâmetros e de um método eficiente para calcular as derivadas parciais da função de custo em relação a esses parâmetros.

3.3 Quando as Redes Convergem?

Nesta seção são abordados alguns pontos que influenciam o comportamento de uma rede, levando a mesma a encontrar ou não a solução desejada. Estes pontos incluem a topologia, a inicialização dos pesos, os parâmetros de aprendizado do algoritmo *backpropagation* e os conjuntos de treinamento e teste.

3.3.1 Topologia

Uma topologia bem escolhida é muito importante para a rede encontrar a solução desejada. Redes que são muito pequenas não são capazes de aprender o conjunto de treinamento ou

simplesmente o aprendem parcialmente. Já as redes que são muito grandes, devido ao alto grau de liberdade oferecido, podem ter dificuldade para generalizar novos dados. Deve ser observado, no entanto, que não somente o tamanho é crucial para que a rede consiga aprender corretamente e generalizar. Redes do mesmo tamanho podem ser ou não capazes de chegar até a solução desejada dependendo de sua topologia. A topologia considerada ótima é aquela com complexidade mínima que seja capaz de aprender os dados e conseguir generalizar a partir deles. Na realidade, é muito difícil chegar a esta topologia “ideal” manualmente.

3.3.2 Inicialização dos Pesos

A influência da inicialização dos pesos durante o treinamento pode ser ilustrada analisando as seguintes situações extremas:

1. Se os pesos da rede são inicializados (aleatoriamente ou não) de forma que uma solução adequada ao problema seja encontrada, dispensando a necessidade de treinar a rede.
2. Se os pesos da rede levam a uma solução que cai em um mínimo local, impossibilitando ao algoritmo *backpropagation* de encontrar a solução desejada (encontrar um mínimo global).

Além destas situações extremas, pode acontecer que, mesmo com diferentes inicializações uma rede seja capaz de encontrar a solução para o problema apenas com algumas delas. Por este motivo, muitas vezes, torna-se necessário treinar a rede várias vezes (com diferentes inicializações) para conseguir encontrar a solução desejada. A amplitude do intervalo de onde os pesos são selecionados aleatoriamente, também tem uma certa influência no aprendizado da rede. Segundo alguns autores, bons valores são aqueles que estão dentro do intervalo $[-0.5, 0.5]$, os quais pertencem à parte dinâmica da função sigmóide. Esta amplitude pode ser reduzida, para que a inicialização tenha uma influência menor no treinamento, mas pode resultar também em uma longa fase de treinamento até encontrar a solução.

3.3.3 Parâmetros do Aprendizado

Algoritmos de aprendizado diferentes possuem parâmetros de aprendizado diferentes, e estes precisam ser inicializados antes de começar a treinar a rede. Nesta seção são tratados os parâmetros relacionados com o algoritmo *backpropagation*.

O parâmetro de aprendizado mais importante do algoritmo *backpropagation* é a taxa de aprendizado. Este parâmetro influencia o comportamento da convergência e a duração da fase de treinamento. Uma convergência segura (no mínimo até o próximo mínimo local) pode ser atingida estipulando uma taxa de aprendizado com um valor pequeno. Em contrapartida, isto provoca um considerável aumento na duração da fase de aprendizado. Se a taxa de aprendizado é inicializada com um valor muito alto, pode fazer a rede convergir até um certo ponto (o qual é dependente do problema) mais rapidamente, mas também pode impedir a mesma de aprender os dados.

Existe um outro parâmetro muito importante que não faz parte do algoritmo *backpropagation* padrão: é o termo *momentum*. Um valor alto para o *momentum*, geralmente acelera a convergência, mas pode causar consequências indesejáveis quando mudanças rápidas na direção da busca durante o treinamento são necessárias para melhorar o erro da rede.

3.3.4 Conjuntos de Treinamento e Teste

A seleção de conjuntos de treinamento e teste apropriados é muito importante para a obtenção de um bom desempenho. RNAs são utilizadas não para memorizar os dados de treinamento e sim para conseguir generalizar a partir deles. Para poder verificar a capacidade de generalização da rede, um segundo conjunto de dados é necessário - o conjunto de teste. Ambos os conjuntos devem ser criados com muito cuidado, o conjunto de treinamento deve ser representativo, ou seja, deve abranger o problema inteiro. Além do mais, é importante ter certeza que não é possível outra generalização a não ser aquela presente no conjunto de teste.

Quando é utilizado um método de aprendizado *online*, onde os pesos são alterados depois de apresentar cada padrão, a ordem na qual os padrões são apresentados pode ser importante. Por isso, é vantajoso mudar a ordem de apresentação dos padrões a cada ciclo durante o treinamento.

3.4 Redes Neurais Construtivas

Algoritmos de aprendizado construtivos são uma alternativa para a construção incremental de redes (potencialmente mínimas). Geralmente estas redes são inicializadas com poucas unidades e, durante o treinamento, vão crescendo até a obtenção do nível de generalização desejado. Entre os algoritmos mais conhecidos para treinar este tipo de redes, podem-se destacar: o *Cascade Correlation* [Falhman and Lebiere, 1990] e o *MTiling* [Yang et al., 1996b], os quais serão descritos a seguir.

3.4.1 Algoritmo Cascade Correlation

Cascade Correlation é uma arquitetura neural que utiliza um algoritmo de aprendizado supervisionado construtivo para o treinamento de uma rede, chamado *Cascade Correlation*. Ao invés de apenas ajustar os pesos de uma rede de topologia fixa, o algoritmo *Cascade Correlation* começa treinando uma rede com o menor tamanho possível (apenas as camadas de entrada e saída). Enquanto o comportamento almejado não for obtido, continua o treinamento adicionando novas unidades, uma por uma, criando uma estrutura de múltiplas camadas. Este algoritmo possui algumas vantagens sobre os algoritmos anteriores: possui um aprendizado muito rápido, a rede consegue determinar seu próprio tamanho e topologia e não precisa de propagar os sinais de erro para trás pelas conexões da rede, como o *backpropagation*. O desenvolvimento deste algoritmo foi motivado pela existência de certos problemas e limitações no algoritmo *backpropagation*, dentre elas o mais importante é a lentidão com a qual a rede aprende determinadas tarefas [Falhman and Lebiere, 1990]. Segundo os autores,

esta lentidão deve-se principalmente a dois fatores:

1. O método utilizado pelo *backpropagation* para a atualização dos pesos, computa a primeira derivada parcial do erro total em relação a cada conexão (peso) da rede. Dadas essas derivadas é possível utilizar a técnica do gradiente descendente para reduzir o erro a cada passo. Este método funciona, mas é necessário dar um número muito grande de passos pequenos para poder chegar a um mínimo que nem sempre é o mínimo global. Em um sistema de aprendizado, precisa-se de dar os maiores passos possíveis. Desafortunadamente, optar por passos muito grandes pode impedir a rede de chegar a uma solução satisfatória. Para poder escolher um tamanho razoável para os passos a serem dados em busca da convergência é necessário calcular derivadas de ordens mais altas, informação que não está disponível no *backpropagation*.
2. Cada unidade interna da rede tende a se tornar um extrator de características e deveria ter um papel importante na rede como um todo. Esta tarefa é dificultada pelo fato de todas as outras unidades estarem mudando de estado ao mesmo tempo. As unidades escondidas em uma determinada camada estão impossibilitadas de se comunicar com outras unidades diretamente, cada unidade enxerga apenas as entradas provenientes das camadas anteriores e o sinal de erro retropropagado a partir da camada de saída da rede.

Basicamente, a rede *Cascade Correlation* é a combinação de duas idéias: a primeira é a de arquitetura em cascata, na qual as unidades são adicionadas à rede uma por vez e são congeladas imediatamente. A segunda é o algoritmo de aprendizado, que cria e adiciona as novas unidades. Para cada nova unidade, é maximizada a magnitude da correlação entre a saída da nova unidade e o erro residual da rede, que tenta-se eliminar.

Inicialmente, a rede é formada apenas por um *Bias* e pelas unidades de entrada e saída. Um exemplo desta arquitetura inicial pode ser observado na Figura 3.4, onde a rede é formada por três unidades de entrada, duas de saída e não possui unidades escondidas.

Figura 3.4: Estado inicial da arquitetura Cascade Correlation.

Cada unidade de entrada está conectada com todas as unidades de saída por pesos ajustáveis. Existe uma unidade de entrada especial com um sinal permanente de $+1$ denominada *bias*, a qual será conectada com todas as unidades de saída (e unidades escondidas por ventura existentes). A saída das unidades de saída pode ser dada pela soma linear de suas entradas multiplicadas pelos pesos de suas conexões, ou fazendo uso de uma função de ativação não linear.

Unidades escondidas podem ser adicionadas uma por vez, a partir de um conjunto de unidades candidatas, que “concorrem” entre si para somente uma ser escolhida. Esta escolha envolve o cálculo da correlação entre a saída de cada unidade candidata e o erro produzido pela rede, desta forma, a unidade candidata com maior correlação é a escolhida.

Cada uma das unidades adicionadas recebe uma conexão de cada uma das unidades de entrada existentes e de todas as unidades escondidas inseridas anteriormente. Ao mesmo tempo, cada unidade adicionada é conectada com todas as unidades de saída. Os pesos das conexões de entrada das unidades intermediárias são congelados quando elas são adicionadas e somente as conexões das unidades da camada de saída são treinadas repetidamente. O algoritmo inclui novas unidades quando não é notada uma redução significativa no erro depois de um dado período de tempo (isto é controlado por um parâmetro de “paciência” estimado pelo usuário). Estas unidades são adicionadas até que o valor do erro seja aceitavelmente baixo.

Nas Figuras 3.5 e 3.6, pode-se observar a inserção de 2 unidades escondidas. É importante observar nas Figuras que as conexões marcadas com um x são treinadas repetidamente e as marcadas por um quadrado são conexões congeladas.

Figura 3.5: Estado da rede após inserção da primeira unidade.

Figura 3.6: Estado da rede após inserção da segunda unidade.

O algoritmo *Cascade Correlation* pode ser sumarizado nos seguintes passos:

1. Inicializar a rede com o número de entradas e saídas requeridas;
2. **Treinar** a rede ajustando as conexões dos neurônios da camada de saída até o erro ser aceitável ou até que não haja melhora de desempenho por um determinado tempo. Se o desempenho é o esperado, termina. Se não continua no passo 3;

3. **Gerar** unidades “candidatas”. Cada uma delas é conectada somente às unidades de entrada e às unidades escondidas existentes, com pesos inicializados randomicamente;
4. Tentar maximizar a correlação entre a ativação das candidatas e o erro, **treinando** (ajustando) os pesos das candidatas. Parar quando a correlação deixe de melhorar;
5. Escolher a candidata com maior correlação, **congelar** seus pesos de entrada e **inseri-la** na rede, conectando-a com as unidades de saída;
6. **Voltar** ao passo 2. O processo termina se o erro da rede é menor que um certo valor estipulado antes de iniciar o algoritmo.

3.4.2 Algoritmo MTiling

Em [Mézard and Nadal, 1989], Mézard e Nadal propuseram o algoritmo *Tiling* e mostraram que ele converge sem erros de classificação quando é utilizado um conjunto de padrões separáveis em 2 categorias. Como tarefas de classificação de padrões, na prática, lidam com padrões pertencentes a M classes (para $M > 2$), é de muito interesse que algoritmos construtivos como o *Tiling* lidem com mais de 2 classes. Este contexto motivou o aparecimento do algoritmo *MTiling* proposto por Yang, Parekh e Honavar em [Yang et al., 1996b].

O algoritmo *Tiling* constrói redes de neurônios organizados em camadas. A camada inferior recebe um vetor de N entradas através de seus N neurônios. Os neurônios nas próximas camadas recebem as entradas dos neurônios da camada imediatamente inferior a cada uma delas. Para treinar cada neurônio é escolhido o algoritmo mais conveniente ¹ dependendo dos dados a classificar.

No algoritmo *Tiling*, cada camada possui um **neurônio mestre** que atua como o neurônio de saída para a camada. Se este neurônio é incapaz de classificar os dados de entrada, então **neurônios auxiliares** podem ser adicionados à camada e treinados para assegurar uma

¹Entre os algoritmos mais utilizados destacam-se *Pocket Algorithm*, *Thermal perceptron* e *Barycentric Correction Procedure*, os quais são comparados em [Yang et al., 1996a]

representação fiel dos padrões de treinamento. Este critério de fidelidade declara que dois exemplos de treinamento ligados a classes diferentes não podem produzir saídas idênticas em uma mesma camada. Segundo Mézard, esta fidelidade é uma condição necessária para a convergência em redes organizadas em camadas. A convergência do algoritmo *Tiling* para 0 erros de classificação considera como entrada conjuntos de dados finitos, em que os atributos dos padrões são valorados bipolarmente (i.e. 1 e -1 somente) [Parekh et al., 1997].

O algoritmo *MTiling* é uma extensão ao algoritmo *Tiling* e classifica padrões pertencentes a M classes ($M > 2$). Esta extensão envolve a construção de camadas com M neurônios mestres (um para cada uma das classes). Grupos de um ou mais neurônios auxiliares são treinados para que a camada corrente cumpra o critério de fidelidade. Na Figura 3.7 pode-se observar a construção de uma rede *MTiling*.

Figura 3.7: Rede *MTiling*.

Os passos básicos do algoritmo são descritos a seguir:

1. Treinar uma camada de M neurônios, cada um deles conectado a N entradas;
2. Se os **neurônios mestres** da camada corrente conseguem alcançar o desempenho de classificação esperado, então pare;

3. Caso contrário, se a camada corrente não cumpre o critério de fidelidade, então adicionar **neurônios auxiliares** para torná-la fiel, como explicado a seguir. Senão ir ao passo 4
 - (a) Entre todas as saídas não fiéis na camada de saída corrente, identificar aquela que possua o maior número de padrões associados a ela. (Cada saída é um vetor de $M + k'$ elementos, onde k' é o número total de **neurônios auxiliares** adicionados à camada anteriormente);
 - (b) Formar um subconjunto de padrões para treinar os **neurônios auxiliares** a partir dos padrões que geraram as saídas identificadas no passo 3a;
 - (c) Adicionar um grupo de k ($1 \leq k \leq M$) **neurônios auxiliares**, onde k é o número de classes representadas no conjunto de padrões identificado no passo 3b, e treiná-los;
 - (d) Repetir os últimos três passos até que a representação dos padrões na camada corrente seja fiel;
4. Treinar uma nova camada de M **neurônios mestres** cujas entradas estão conectadas a cada neurônio de sua camada inferior e voltar ao passo 2.

A convergência do algoritmo *MTiling* para nenhum erro de classificação é provada em duas partes. Primeiro, é provado que uma representação fiel pode ser atingida por qualquer conjunto de dados finito e não contraditório (possivelmente contendo atributos com valores reais) utilizando um número finito de **neurônios auxiliares**. Posteriormente é mostrado que com cada camada adicional o número de erros de classificação é reduzido pelo menos em um padrão. Uma descrição mais detalhada desta prova de convergência pode ser encontrada em [Yang et al., 1996b].

3.5 Considerações Finais

Um dos modelos de RNAs mais difundido atualmente é o de redes MLP treinadas com o algoritmo *backpropagation*. Embora apresentem uma alta capacidade de aprendizado, existe ainda uma “falta” de conhecimento sobre seu comportamento interno. Neste ponto, é importante salientar que não existe uma maneira exata e formal de criar redes MLP com a melhor capacidade de generalização. Atualmente, muitas pesquisas estão direcionadas à determinação da topologia (conectividade de elementos internos) que implique na melhor capacidade de generalização e que possua a menor complexidade em termos de número de elementos.

A influência de alguns parâmetros no aprendizado e a capacidade de generalização de RNAs foi apresentada, entre os quais podem-se destacar: a topologia, a inicialização dos pesos, a taxa de aprendizado, o termo *momentum* e os conjuntos de treinamento e teste. A análise mostrou que estes parâmetros devem ser estipulados de uma forma apropriada e considerando o problema, para que a rede seja capaz de resolvê-lo.

Algoritmos construtivos têm sido propostos na literatura como alternativa para a construção automática de redes de menor tamanho. Dois algoritmos foram abordados neste capítulo, o *Cascade Correlation* [Falhman and Lebiere, 1990] e o *MTiling* [Yang et al., 1996b]. O primeiro apareceu com o objetivo de “reparar” certas limitações e problemas observados no algoritmo *backpropagation* ² e é muito difundido. Entre suas principais vantagens, podem ser citadas: a rapidez do treinamento, a não necessidade de retropropagar o erro pela rede para adaptar seus parâmetros e o fato de poder determinar seu próprio tamanho e topologia. O algoritmo *MTiling*, por sua vez, é uma generalização do algoritmo *Tiling* [Mézard and Nadal, 1989], para problemas com mais de duas classes.

As RNCs possuem características (inerentes à maneira de moldar sua estrutura) que as tornam poderosas e eficientes para resolver problemas de classificação. Considerando este panorama, optou-se por avaliar sua eficiência ao serem submetidas a métodos de *Pruning*. Desta

²Em [Falhman and Lebiere, 1990] pode-se encontrar uma descrição mais detalhada destas limitações.

forma, resulta interessante realizar uma comparação com o desempenho de redes MLP, no contexto de otimização de RNAs.

No próximo capítulo serão abordadas algumas das principais técnicas de *Pruning* existentes na literatura, tanto para redes MLP e RNCs. O principal objetivo destas técnicas é reduzir o tamanho da rede sem que haja uma grande perda na capacidade de generalização da mesma.

Capítulo 4

Técnicas de Pruning

Este capítulo apresenta uma introdução a algumas das técnicas de *Pruning* propostas na literatura e utilizadas com frequência em aplicações reais. Inicialmente, serão abordados tópicos relacionados ao porquê da sua necessidade e como estas técnicas podem contribuir para melhorar o desempenho da rede e, seguidamente, serão descritas algumas das mais difundidas.

4.1 Motivação

Apesar dos avanços das pesquisas em RNAs, existem ainda alguns aspectos que precisam ser melhor definidos. Um destes aspectos envolve a escolha da topologia apropriada de rede para poder resolver um dado problema.

Aqui surge um dilema, porque tanto redes grandes como pequenas possuem suas próprias vantagens. Entre as vantagens das redes grandes podem ser citadas:

- O aprendizado torna-se mais rápido, uma vez que as características dos dados de entrada podem ser aprendidas por unidades diferentes;

- O mínimo local é mais facilmente evitado quando a rede possui muitos parâmetros (pesos e/ou unidades) [Rumelhart et al., 1986b]. Particularmente, um estudo teórico apresentado em [Yu, 1992] mostrou que, para dados uniformemente distribuídos, quando o número de unidades escondidas é igual ao número de exemplos de treinamento (menos 1), a superfície de erro não apresenta mínimos locais;
- A tolerância a falhas pode ser maior:
- O espaço da solução não é restrito, dando lugar a resolução de problemas mais complexos.

Por outro lado, redes pequenas também possuem certas vantagens, entre as quais podem se destacar:

- A generalização pode ser melhor [Baum and Haussler, 1989], redes grandes tendem a **memorizar** os padrões de treinamento;
- A extração de conhecimento é mais fácil e as regras geradas podem ser mais simples [Towell et al., 1991];
- A implementação exige menos recursos computacionais, tanto de hardware, como de software.

Não existe um procedimento padrão que assegure a obtenção da melhor topologia, ou que obtenha a rede com uma boa capacidade de generalização. Os métodos mais frequentemente utilizados para isto são:

- **Early Stopping**. Este método não altera a topologia, mas procura encontrar a rede que melhor generaliza os dados. Para tal, é necessário dividir os dados disponíveis para treinamento em dois conjuntos diferentes (treinamento e validação). O procedimento é simples. Periodicamente, durante o treinamento, padrões do conjunto de validação

são apresentados à rede e um erro de validação é computado. O treinamento pára quando este erro começa a aumentar. Entre as vantagens deste método, destacam-se a rapidez e a facilidade para ser manipulado, uma vez que a única intervenção do usuário é a escolha do conjunto de validação. Maiores referências podem ser encontradas em [Finnoff et al., 1993] e [Prechelt, 1998].

- **Algoritmos Genéticos.** A escolha da melhor topologia é feita por meio de um processo evolutivo e adaptativo. Esta abordagem nasce da idéia de que o cérebro foi, durante muito tempo, minuciosamente modelado através da genética e da seleção natural. Uma explanação mais detalhada pode ser encontrada em [Yao, 1993].
- **Redes Construtivas.** São redes que são inicialmente criadas com poucas unidades e parâmetros, e gradativamente vão crescendo até completar o aprendizado. Um dos algoritmos mais utilizados para o desenvolvimento de RNCs é o *Cascade Correlation* [Fahlman and Lebiere, 1990]. Maiores referências sobre outros algoritmos construtivos podem ser encontradas em [Fiesler and Beale, 1996].

Como uma alternativa a estas abordagens, existem os métodos de *Pruning*, que, ao contrário das redes construtivas, começam com redes grandes e gradativamente eliminam suas unidades ou conexões (segundo heurísticas que determinam quais são as menos relevantes para a solução) até achar uma topologia adequada e eficiente. Reed em [Reed, 1993], classifica os algoritmos de *Pruning* em dois grandes grupos: os métodos de cálculo de sensibilidade e os métodos penalizadores. Os algoritmos do primeiro grupo estimam a sensibilidade da função de erro à remoção de elementos da rede. Aqueles que provocarem a menor variação no erro, são removidos. Os algoritmos do segundo grupo modificam a função de erro de forma que, durante o treinamento, os pesos considerados não necessários assumem valores cada vez mais próximos de zero. Assim, os pesos podem ser removidos quando eles se tornarem menores que um certo *threshold*.

Em geral, os métodos de sensibilidade modificam uma rede já treinada, ou seja, a rede é treinada, as sensibilidades são calculadas e, finalmente, as conexões são removidas. Por

outro lado, os métodos penalizadores removem as conexões (com pesos próximos a zero) durante o treinamento, uma vez que a função de erro é modificada de forma que o algoritmo de treinamento utilizado aproxime os pesos desnecessários a zero.

4.2 Métodos de Cálculo de Sensitividade

Os métodos descritos a seguir fazem parte do grupo de algoritmos que calculam a sensibilidade da função de erro à remoção de elementos da rede.

4.2.1 Magnitude Pruning

Este é considerado o método de *Pruning* mais simples. Após cada ciclo, a conexão com o menor peso é selecionada e removida. Assim, a sensibilidade de uma conexão é apenas o valor absoluto do seu peso. Embora este método seja muito simples, raramente apresenta resultados muito piores do que aqueles obtidos pelos métodos mais sofisticados

4.2.2 Optimal Brain Damage

Este método estima um valor aproximado (utilizando uma série de Taylor) para a alteração do erro da rede quando alguma conexão é removida. Para simplificar o cálculo, alguns termos da equação inicial (série de Taylor) são removidos de acordo com certas condições assumidas pelo autor como verdadeiras, restando apenas o cálculo dos elementos da diagonal da matriz Hessiana (segunda derivada da função de erro). Em [Cun et al., 1990] pode-se encontrar uma descrição mais detalhada desta técnica.

4.2.3 Optimal Brain Surgeon

Métodos baseados na magnitude dos pesos eliminam aquelas conexões com os menores valores para seus pesos. Isto pode representar um problema em algumas situações. OBD e OBS são métodos que utilizam um critério que analisa o resultado da função de erro. OBS é caracterizado por ser um método mais sofisticado que o OBD. Este método calcula a matriz Hessiana completa de uma forma iterativa, o que leva a uma melhor aproximação do erro cometido pela rede. O principal problema observado com esta técnica é a sua lentidão e a demanda de uma grande quantidade de memória.

4.2.4 Non-Contributing Units

Este método utiliza algumas medidas estatísticas para encontrar unidades que não contribuam para a resposta fornecida pela rede. Lembrando que a rede é subdividida em camadas, a saída de cada unidade é observada a cada ciclo e são removidas aquelas que:

- não variam sua saída;
- apresentam a mesma saída que outra unidade na mesma camada;
- sempre apresentam um valor oposto à saída de uma outra unidade na mesma camada.

Este método corresponde à primeira parte de outro método proposto por Sietsma e Dow em [Sietsma and Dow, 1991].

4.2.5 Skeletonization

O método *skeletonization*, proposto por Mozer e Smolensky em [Mozer and Smolensky, 1989], estima quais unidades são menos importantes para o desempenho da rede e as remove durante

o treinamento. Segundo os autores, uma medida de relevância ρ_i é associada a cada unidade. Esta medida pode ser estimada conforme a Eq. 4.1:

$$\rho_i = E_{\text{without unit } i} - E_{\text{with unit } i} \quad (4.1)$$

onde $E_{\text{with unit } i}$ e $E_{\text{without unit } i}$ são os erros computados pela rede com a unidade i e sem ela, respectivamente. O cálculo desta medida, apesar de ser simples, tem associado uma alta complexidade. Um algoritmo que faça isso para todas as unidades teria complexidade da ordem $O(np)$, para n unidades e p padrões de treinamento. Uma alternativa computacionalmente mais eficiente seria aproximar ρ_i introduzindo um termo adicional α_i para cada unidade, da seguinte forma:

$$y_j = \alpha_j \cdot f\left(\sum_i w_{ji} \cdot y_i\right) \quad (4.2)$$

onde, y_j é a saída da unidade j , w_{ji} é o valor do peso da conexão da unidade i para a unidade j e f é a função de ativação sigmóide. Este termo adicional é um coeficiente associado a toda unidade e tem por finalidade multiplicar sua saída, amplificando ou atenuando a mesma. Se $\alpha_i = 0$, então a unidade i não tem influência no desempenho da rede e se $\alpha_i = 1$, a unidade se comporta normalmente (influencia o desempenho). A importância de uma unidade é aproximada pelo valor $\hat{\rho}_i$, dado pela Eq. 4.3:

$$\hat{\rho}_i = -\left.\frac{\partial E^\ell}{\partial \alpha_i}\right|_{\alpha_i=1} \quad (4.3)$$

Esta equação pode ser computada pelo próprio algoritmo *backpropagation*. Quando $\hat{\rho}_i$ for menor que um certo *threshold*, a unidade i pode ser removida. A equação utilizada para medir o erro durante o treinamento é a soma dos quadrados dos erros (ver Eq. 2.2). Para medir a relevância ρ_i é utilizada a Eq. 4.4

$$E^\ell = \sum |d_{pj} - y_{pj}| \quad (4.4)$$

porque ela proporciona uma melhor estimativa de relevância quando o erro é pequeno. Na prática, $\frac{\partial E^\ell}{\partial \alpha_i}$ oscila fortemente e segundo Mozer e Smolensky uma medida mais estável e que consequentemente proporciona melhores resultados é descrita pela Eq. 4.5.

$$\hat{\rho}_i(t+1) = 0.8 \cdot \hat{\rho}_i(t) - 0.2 \cdot \frac{\partial E^\ell(t)}{\partial \alpha_i} \quad (4.5)$$

O procedimento geral para a aplicação deste método pode ser sumarizado pelos passos descritos na Tabela 4.1.

1. Treinar a rede até conseguir resultados satisfatórios 2. Computar $\hat{\rho}$ para cada unidade 3. Remover as unidades com os menores valores para $\hat{\rho}$ 4. Repetir os passos 1-3 um determinado número de vezes, geralmente dado pelo desempenho obtido a cada passo

Tabela 4.1: *Skeletonization*: algoritmo básico.

4.2.6 Método de Karnin

Karnin, em [Karnin, 1990], propôs um algoritmo que estima a sensibilidade da função de erro à exclusão de cada conexão e poda os pesos que provoquem a menor sensibilidade. Esta estimativa é feita de tal forma que não requer nenhuma alteração da função de erro. Assim, as tarefas de treinar a rede e determinar as possíveis conexões a serem eliminadas são completamente independentes. Isto significa que o processo de aprendizado está livre da

interferência de processos externos. A sensibilidade estimada para a remoção do peso w_{ji} é dada por:

$$S_{ji} = - \sum_{n=0}^{N-1} \frac{\partial E}{\partial w_{ji}} \cdot \Delta w_{ji}(n) \cdot \frac{w_{ji}^f}{w_{ji}^f - w_{ji}^i} \quad (4.6)$$

onde N é o número total de ciclos, w_{ji}^f é o valor final do peso depois do treinamento e w_{ji}^i é o valor inicial do peso. Todos esses valores estão disponíveis durante o treinamento, assim, a expressão é fácil de calcular e não necessita de uma fase específica para isso. Quando Δw_{ji} é calculado pelo algoritmo *backpropagation* padrão, a sensibilidade é dada por:

$$S_{ji} = - \sum_{n=0}^{N-1} [\Delta w_{ji}(n)]^2 \frac{w_{ji}^f}{\eta \cdot (w_{ji}^f - w_{ji}^i)} \quad (4.7)$$

Em termos de complexidade, o único passo extra a ser adicionado é o cálculo da Eq. 4.7 ou, se o termo *momentum* for utilizado, o cálculo da Eq. 4.6. Um vetor de sensibilidades (de tamanho igual ao número de conexões) guarda os valores acumulados necessários para o cálculo de S_{ji} .

Depois do treinamento, cada conexão tem a sua sensibilidade S_{ji} estimada (as quais foram calculadas por um processo executado concorrentemente com o processo de aprendizado) e as que tiverem menor S_{ji} podem ser removidas. Cabe ressaltar que se todas as conexões de saída ou de entrada de um neurônio são removidas então ele poderá ser removido também.

4.2.7 Algoritmo lprune

Segundo Prechelt, em [Prechelt, 1995], o ponto chave para aplicar *Pruning* é o cálculo da importância aproximada de cada parâmetro. A solução mais simples seria determinar a importância como sendo proporcional à magnitude do peso. Exemplos de métodos mais sofisticados são *optimal brain damage* (OBD) e *optimal brain surgeon* (OBS). O método OBD

[Cun et al., 1990] utiliza uma aproximação para a segunda derivada da função de erro em relação a cada peso para determinar a importância do mesmo. OBS [Hassibi and Stork, 1993] evita a aproximação e computa a segunda derivada quase exatamente. Ambos os métodos possuem a desvantagem de precisar treinar a rede até chegar ao erro mínimo antes da aplicação do algoritmo de *Pruning*. Para muitos conjuntos de dados, isto introduz um *overfitting* que não pode ser reparado pelo mesmo algoritmo de *Pruning* [Prechelt, 1995].

Finnoff *et al.*, em [Finnoff et al., 1993], apresentaram um teste estatístico T (ver Eq. 4.8) baseado na probabilidade de um peso assumir o valor 0 durante o treinamento.

$$T(w_i) = \log \left(\frac{\left| \sum_p w_i - \eta \left(\frac{\partial E}{\partial w_i} \right)_p \right|}{\eta \sqrt{\sum_p \left(\left(\frac{\partial E}{\partial w_i} \right)_p - \overline{\left(\frac{\partial E}{\partial w_i} \right)} \right)^2}} \right) \quad (4.8)$$

Ao contrário do OBD e OBS, esta medida não assume que o valor mínimo para o erro tenha sido encontrado, ela pode ser calculada a qualquer hora durante o treinamento. Na Eq. 4.8, os somatórios são sobre todos os n exemplos do conjunto de treinamento, η é a taxa de aprendizado e a barra horizontal denota o valor médio entre todos os exemplos. De acordo com este teste, uma conexão é removida se sua probabilidade de assumir o valor 0 é alta. Esta abordagem faz parte de um método chamado *autoprune* [Finnoff et al., 1993].

Prechelt em [Prechelt, 1995], apresentou um método denominado *lprune*, baseado no *autoprune*. Este método determina quantas unidades deveriam ser removidas a cada iteração durante a aplicação do algoritmo de *Pruning*, observando a evolução dos pesos e do *overfitting*.

Considerando E a função de erro e $E_{tr}(t)$ o valor médio de E sobre o conjunto de treinamento, medido até o ciclo t . $E_{va}(t)$ é o erro sobre o conjunto de validação sendo utilizado para determinar o *overfitting*. $E_{te}(t)$ é o erro sobre o conjunto de teste, que caracteriza a qualidade resultante do treinamento. $E_{opt}(t)$ é o valor mínimo de $E_{va}(t)$ até o ciclo t . Define-se a perda de generalização (GL) no ciclo t , como o aumento relativo de $E_{va}(t)$ sobre $E_{opt}(t)$

(veja Eq. 4.9). Esta perda de generalização, segundo Prechelt, caracteriza diretamente a quantidade de *overfitting* da rede. Um alto valor de GL pode ser uma razão para parar o treinamento ou podar a rede.

$$GL(t) = 100 \cdot \left(\frac{E_{va}(t)}{E_{opt}(t)} - 1 \right) \quad (4.9)$$

Para poder medir o progresso do treinamento, é introduzida a noção de “faixa de treinamento”, como sendo uma sequência de k ciclos. O “progresso” medido depois de cada faixa é dado por:

$$P_k(t) = 1000 \cdot \left(\frac{\sum_{t' \in t-k+1 \dots t} E_{tr}(t')}{k \cdot \min_{t' \in t-k+1 \dots t} E_{tr}(t')} - 1 \right) \quad (4.10)$$

isto é, “quão maior era o valor médio do erro durante a faixa que o valor mínimo na mesma faixa?”.

Um critério importante diz “pare ou pode quando o erro de generalização aumentou em s faixas sucessivas”:

- UP_s satisfeito depois do ciclo t se e somente se UP_{s-1} foi satisfeito depois do ciclo $t - k$
- UP_1 satisfeito depois do primeiro ciclo t “fim de faixa” com $E_{va}(t) > E_{va}(t - k)$

O nome UP denota “quando o erro aumenta” (*error when up*). Este critério é responsável por determinar quando acontecerá a poda. Segundo este critério, o próximo passo da poda deverá ocorrer não quando a taxa GL é alta, mas sim quando ela aumenta.

Uma variável λ que é modificada dinamicamente é introduzida com o objetivo de regular a quantidade de parâmetros a serem removidos. O ponto chave para determinar um valor apropriado para esta variável é a amplitude do *overfitting* observado. Quanto maior for o

overfitting, maior deverá ser o valor de λ . Assim, λ pode ser adaptada junto com o crescimento de GL e saturada a algum valor máximo (λ_{max}). Experimentos iniciais mostraram que uma forma ideal para adaptar λ é:

$$\lambda = \lambda(GL) = \lambda_{max} \left(1 - \frac{1}{1 + \frac{GL}{2}} \right) \quad \lambda_{max} = \frac{2}{3} \quad (4.11)$$

O algoritmo *lprune* pode ser visto na Tabela 4.2. Nesta tabela, as constantes 5000, 0.1, 25, 100 e 0.4 constituem valores que determinam um critério de parada adequado durante o processo.

Em suma, este método estima dinamicamente o número de parâmetros a serem removidos e tem se mostrado mais eficaz que métodos com esquemas estáticos (principalmente em redes pequenas) [Prechelt, 1995]

4.3 Métodos Penalizadores

Os métodos descritos na seção anterior tentam encontrar elementos não essenciais, calculando a sensibilidade do erro devido à sua remoção. O método descrito nesta seção modifica a função de custo de forma que o próprio algoritmo *backpropagation* se encarrega de eliminar elementos não essenciais, dirigindo seus pesos para o valor 0.

4.3.1 Método de Chauvin

Em [Chauvin, 1989], Chauvin descreve um método que modifica a função de custo para:

$$C = \mu_{er} \sum_j^O (d_j(n) - y_j(n))^2 + \mu_{en} \sum_j^H e(o_j^2(n)) \quad (4.12)$$

```

REPEAT
  Treinar a rede por um ciclo;
  IF ciclo MOD k = 0 THEN
    Computar  $E_{va}$ ,  $E_{opt}$ , e  $GL$  utilizando o conjunto de validação;
  END;
UNTIL  $GL > 5$ ; (* aplicando early stopping *)
Resetar a rede ao estado que exibia  $E_{opt}$ ;
(* agora começa o treinamento efetuando Pruning *)
REPEAT
  Treinar a rede por um ciclo e computar os valores  $T(w_i)$ ;
  SE ciclo MOD k = 0 THEN
    Computar  $E_{va}$ ,  $E_{opt}$ , e  $GL$  utilizando o conjunto de validação;
    SE  $UP_2(t)$  é satisfeito AND não houve poda nos últimos
    k ciclos THEN
      Podar conexões  $i$  que satisfaçam  $T(w_i) < \lambda(GL)\mu_T$ ;
    END;
  END;
UNTIL  $t > 5000$  OR  $P_k(t) < 0.1$  OR (No mínimo 25 ciclos treinados
desde o último passo de Pruning AND  $GL > 100$  AND  $P_k(t) < 0.4$ )

```

Tabela 4.2: Algoritmo *lprune*.

onde, os somatórios são realizados sobre o conjunto de unidades de saída O e o conjunto de unidades escondidas H . O primeiro termo é a forma como o *backpropagation* padrão computa o erro. O segundo termo avalia a “energia” dispendida pelas unidades escondidas. Os parâmetros μ_{er} e μ_{en} servem para balancear os dois termos. A “energia” dispendida por uma unidade (quanto sua atividade interna varia com os padrões de treinamento) é um indicador da sua importância. Se a unidade varia bastante é provável que represente informação significativa, caso contrário, não guarda muita informação.

A forma como a função e (utilizada pela Eq. 4.12) é definida, determina comportamentos

qualitativamente diferentes. Várias funções são examinadas, as quais possuem a derivada:

$$e' = \frac{\partial e(y^2)}{\partial y_i^2} = \frac{1}{(1 + y^2)^x} \quad (4.13)$$

onde x é um número inteiro. Para $x = 0$, a função e é linear e “energias” altas ou baixas recebem a mesma penalidade, já para $x = 1$, a função e é logarítmica, assim, unidades com “energias” baixas recebem uma penalização maior que as unidades com “energias” altas.

A magnitude dos pesos pode ser adicionada à função de custo:

$$C = \mu_{er} \sum_j^O (d_j(n) - y_j(n))^2 + \mu_{en} \sum_j^H e(y_{ij}^2(n)) + \mu_w \sum_{ij}^W w_{ji}^2 \quad (4.14)$$

onde μ_w é um outro termo de balanço.

Como a derivada do terceiro termo em relação a w_{ji} é $2\mu_w w_{ji}$, pode-se notar que efetivamente este termo conduz os pesos que não são considerados importantes a assumirem o valor 0 e assim poderem ser removidos.

Simulações são descritas em [Chauvin, 1990a] e [Chauvin, 1990b], as quais utilizam a função de custo:

$$C = \mu_{er} \sum_{jn}^{ON} (d_j(n) - y_j(n))^2 + \mu_{en} \sum_{jn}^{HN} \frac{y_j^2(n)}{1 + y_j^2(n)} + \mu_w \sum_{ij}^W \frac{w_{ji}^2}{1 + w_{ji}^2} \quad (4.15)$$

Segundo Chauvin, não foi observado *overfitting* durante o treinamento. Os valores de balanço optados foram $\{\mu_{er} = 0.1, \mu_{en} = 0.1, \mu_w = 0.001\}$. Uma análise mostra que a rede pode ser reduzida a um número ótimo de unidades escondidas independente do seu tamanho inicial.

4.4 Métodos para Redes Construtivas

Estes métodos caracterizam-se por utilizar estimativas de sensibilidade ao invés de termos de penalidade. Isto ocorre porque algoritmos construtivos não se comportam como o algoritmo *backpropagation* que utiliza a técnica de gradiente descendente [Rumelhart et al., 1986a] e retropropaga o erro computado pela função de custo para as camadas intermediárias, ajustando os pesos. A decisão pelo estudo de métodos de *Pruning* para RNCs foi motivada pela possível existência de conexões e/ou unidades em excesso após a construção da rede final.

4.4.1 Pruning Cascade Correlation

Redes treinadas pelo algoritmo *Cascade Correlation* podem ter um grande número de conexões, o qual depende diretamente do número de neurônios existentes na rede. Visto que cada unidade inserida é conectada a todas as unidades existentes na rede, é possível que algumas destas novas conexões sejam redundantes. Em vista disso, é de bom senso utilizar técnicas de *Pruning* para reduzir o tamanho da rede. Métodos de cálculo de sensibilidade parecem ser os mais adequados, já que são aplicados depois do treinamento e porque os algoritmos construtivos não possuem uma função de custo para ser modificada pelos métodos penalizadores. O método de Karnin (ver Seção 4.2.6) parece ser o mais indicado para ser aplicado nas redes treinadas com o algoritmo *Cascade Correlation*, porque não depende da retro-propagação inerente ao algoritmo *backpropagation* (ao contrário de outros métodos, como o *Skeletonization*).

Como o treinamento é cíclico, um dos maiores problemas é saber a hora certa para eliminar as conexões treinadas. Waugh em [Waugh and Adams, 1995] propõe duas opções para o problema de **quando** aplicar *Pruning* a redes treinadas com o algoritmo *Cascade Correlation*:

- podar cada conjunto de candidatas quando o mesmo é treinado e podar a camada de saída quando terminar todo o treinamento.

- podar cada cada conjunto de candidatas e a camada de saída depois de treinar as unidades de saída.

Waugh realizou experimentos utilizando estas duas opções e seus resultados indicam que a aplicação de *Pruning* à rede reduz efetivamente seu tamanho sem diminuir sua capacidade de classificação. Segundo o autor, podar a camada de saída a cada fase de treinamento danifica a rede porque elimina prematuramente conexões que podem ser utilizadas nas próximas fases. A melhor maneira de podar a rede é com a eliminação de conexões depois do treinamento de cada conjunto de unidades, sejam elas candidatas ou de saída.

4.4.2 Pruning MTiling

Como os métodos penalizadores não são adequados para serem aplicados a redes construtivas. Pode-se concluir então que os métodos de cálculo de sensibilidade são os mais indicados para a remoção de elementos redundantes em redes *MTiling*.

No caso de redes *MTiling*, o erro pode ser definido de uma forma mais adequada ao contexto. Assim, para um neurônio i na camada L (que foi treinada e é fiel), a sensibilidade $S(i)$ é definida como:

$$S(i) = \begin{cases} 1 & \text{se eliminar } i \text{ torna } L \text{ não fiel} \\ 0 & \text{caso contrário} \end{cases} \quad (4.16)$$

O procedimento para efetuar a poda deste tipo de redes é aplicado durante o treinamento e invocado após treinar cada camada (uma vez que tenha cumprido o critério de fidelidade). Todos os **neurônios mestres** têm a sua sensibilidade definida para $S(i) = 1$, porque eles não podem ser removidos. Três técnicas simples para identificar os neurônios com sensibilidade $S(i) = 0$ são propostas em [Parekh et al., 1997] e descritas a seguir:

Neurônios Mortos São aqueles neurônios auxiliares que apresentam, exatamente, a mesma saída para todos os padrões no conjunto de treinamento. Eles não afetam a fidelidade da camada corrente. Ver Figura 4.1

Saídas dos neurônios					Classes
-1	-1	+1	-1	-1	I
-1	-1	+1	-1	+1	II
-1	-1	+1	-1	-1	I
-1	+1	-1	-1	-1	II
+1	-1	-1	-1	+1	III

Mestres
Auxiliares
Neurônio morto

Figura 4.1: Exemplo de neurônios mortos.

Neurônios Correlacionados São os pares de neurônios que emitem exatamente a mesma ou exatamente a saída oposta para cada padrão no conjunto de treinamento. Um destes neurônios pode ser eliminado sem afetar a fidelidade da camada. Uma vez eliminados, não serão considerados na busca por outros possíveis pares de neurônios correlacionados. Ver Figura 4.2

Saídas dos neurônios						Classes
-1	-1	+1	-1	+1	-1	I
-1	-1	+1	+1	-1	+1	II
-1	-1	+1	+1	-1	-1	I
-1	+1	-1	-1	+1	+1	II
+1	-1	-1	+1	-1	+1	III

Mestres

Auxiliares

Neurônios correlacionados

Figura 4.2: Exemplo de neurônios correlacionados.

Neurônios Redundantes A busca por estes neurônios envolve a remoção momentânea de cada um dos neurônios auxiliares e verificação da fidelidade nas saídas. Se a camada não cumpre o critério de fidelidade então o neurônio removido é restaurado, caso contrário, ele é redundante e sua sensibilidade é definida como $S(i) = 0$. O neurônio redundante é assim imediatamente eliminado. A busca por outros neurônios redundantes

recomeça no primeiro **neurônio auxiliar**. Este processo de identificação e eliminação de neurônios redundantes continua até que eles não possam mais ser identificados. No pior caso, esta busca faz $O(k^2)$ chamadas à rotina que verifica a **fidelidade** da camada corrente (assumindo que esta camada possui inicialmente k **neurônios auxiliares**). Ver Figura 4.3

Saídas dos neurônios				Classes	
-1	-1	+1	+1	-1	I
-1	-1	+1	+1	+1	II
-1	-1	+1	+1	-1	I
-1	+1	-1	+1	-1	II
+1	-1	-1	-1	+1	III
Mestres			Auxiliares	Neurônio redundante	

Figura 4.3: Exemplo de neurônios redundantes.

Neurônios com $S(i) = 0$ são eliminados e o treinamento continua com a adição de uma nova camada de M **neurônios mestres**. A busca destes neurônios durante o processo de *Pruning* implica em um custo adicional, que consiste em determinar os neurônios com $S(i) = 0$.

4.5 Considerações Finais

Um dos principais problemas na aplicação de RNAs para a resolução de problemas é a escolha da topologia adequada. Diversos métodos para a construção de redes “ótimas” têm sido propostos. Inicializar a rede com um baixo número de elementos e incrementar esta quantidade durante o treinamento é uma abordagem proposta por métodos construtivos. Por outro lado, algoritmos genéticos fazem uso de um processo **evolutivo** e adaptativo para chegar à melhor topologia. Técnicas de *Pruning*, por sua vez, são aplicadas a redes inicialmente grandes o suficiente para aprender os dados de entrada.

Na grande maioria de aplicações de RNAs, o tamanho final da rede exerce um papel pre-

ponderante no seu desempenho. Embora, redes pequenas e grandes possuam suas diversas vantagens, as primeiras parecem estar à frente. É muito mais barato implementar redes pequenas, tanto em *hardware* como em *software*. Por outro lado, a extração de conhecimento é beneficiada também, uma vez que regras mais simples podem ser extraídas. Estas são algumas das razões que motivam a busca de redes que possuam o menor tamanho possível, mas que conservem sua capacidade de generalizar.

No decorrer deste capítulo foram apresentados alguns dos métodos de *Pruning* mais importantes encontrados na literatura. Estes métodos podem ser divididos em métodos por cálculo de sensibilidade e métodos penalizadores. Os do primeiro grupo estimam a sensibilidade do erro da rede à remoção de um dos seus elementos (conexão/neurônio). Assim, elementos que produzam as menores sensibilidades podem ser removidos. Já os do segundo grupo incluem um termo de penalidade à função de erro, desta forma o algoritmo de treinamento penaliza as conexões consideradas redundantes fazendo seus pesos cada vez mais próximos de 0. Conexões com pesos abaixo de um certo valor podem então ser removidas. Por último, métodos de *Pruning* para RNCs utilizam abordagens de cálculo de sensibilidade uma vez que não é possível adicionar termos de penalidade à função que computa o erro.

No total, foram detalhados dois métodos para RNCs, sete métodos de cálculo de sensibilidade e apenas um do grupo que utiliza termos de penalidade. Métodos deste último grupo não são considerados eficientes. Thimm, em [Thimm and Fiesler, 1997], afirma que redes que utilizam métodos penalizadores podem não convergir e mostra um exemplo claro desta situação.

O estudo apresentado nesta dissertação procura estimar os prós e os contras da aplicação de cada uma das diferentes abordagens adotadas para eliminar elementos considerados redundantes em RNAs. No próximo capítulo serão apresentados os objetivos perseguidos com este trabalho, a descrição dos experimentos realizados, a análise dos resultados obtidos e as conclusões extraídas durante o decorrer do mestrado.

Capítulo 5

Descrição dos Experimentos

Este capítulo apresenta uma descrição geral dos experimentos realizados com os métodos de *Pruning*. Ao mesmo tempo, é feita uma análise dos resultados obtidos, considerando os desempenhos apresentados pelos métodos avaliados.

5.1 Introdução

Visto que um dos maiores problemas na aplicação de RNAs é a escolha de sua topologia, a busca pela construção de redes “ótimas” tem recebido crescente atenção por parte da comunidade de RNAs. Entre as abordagens citadas como solução para este problema estão as técnicas de *Pruning*, que são objeto de estudo do presente trabalho.

Uma vez que nenhum dos métodos de *Pruning* existentes na literatura garante encontrar a melhor rede, torna-se necessário comparar as diferentes heurísticas existentes. Além disso o candidato desconhece a existência de trabalhos que apontem as vantagens de cada técnica e como o desempenho por elas apresentado é afetado pela aplicação. Antes de efetuar esta comparação, é importante definir como determinar qual é o melhor método, ou seja, como

medir sua eficiência.

O critério do **menor tamanho final** pode ser adotado, mas, este varia dependendo da aplicação. Muitas vezes, como em algumas implementações em *hardware*, pode ser mais interessante minimizar o número máximo de conexões que entram ou saem de um neurônio do que minimizar o número total de conexões. Outro critério que pode ser adotado é a **capacidade de generalização**. Mais uma vez, depende fortemente da aplicação. Por último, pode-se considerar o **tempo de treinamento** mas este critério é conflitante com o critério de menor tamanho porque se tem mostrado que redes grandes aprendem com maior “rapidez”.

O presente trabalho consiste em realizar um estudo comparativo de algumas das técnicas de *Pruning* descritas no capítulo 4. Devido à dificuldade para estipular um critério bem definido e único que indique a eficiência de cada método optou-se por utilizar a combinação dos três critérios citados anteriormente: menor tamanho final, tempo de treinamento e capacidade de generalização. Considerando estas limitações, as seguintes questões foram levantadas para poder comparar efetivamente os métodos:

- Quais algoritmos produzem as redes de menor tamanho?
- Qual a relação entre o tamanho inicial e final da rede?
- As redes de menor tamanho são as que melhor generalizam?
- A generalização varia conforme o tamanho da rede?
- Quais algoritmos degradam mais rapidamente a generalização?

5.2 Ferramentas Utilizadas

Para poder comparar os métodos, é necessário dispor de ferramentas de simulação de Redes Neurais Artificiais. Atualmente, no LABIC ¹ podem ser encontrados vários simuladores de RNAs instalados nas diferentes estações de trabalho. Entre eles destacam: o *SNNS (Stuttgart Neural Network Simulator)* [Zell et al., 1995] e o *Kipu ONNS (Ontogenic Neural Networks Simulator)* [Cuadros et al., 1998], sendo este último desenvolvido por alunos de pós graduação e graduação do LABIC.

Para a realização dos experimentos foram escolhidos o *SNNS* e o *Constructive Neural Network Learning Algorithm Simulator* [Parekh et al., 1996], desenvolvido pelos próprios autores do algoritmo *MTiling* e lida também com outros algoritmos construtivos. A escolha destas duas ferramentas deve-se à quantidade de algoritmos implementados (tanto de treinamento como de *Pruning*), às facilidades oferecidas e ao fato de serem distribuídas gratuitamente.

5.2.1 Stuttgart Neural Network Simulator

Este simulador, conhecido como *SNNS*, foi desenvolvido no Instituto de Sistemas Paralelos e Distribuídos de Alto Desempenho da Universidade de Stuttgart na Alemanha desde 1989 [Zell et al., 1995]. O objetivo do projeto era de oferecer um ambiente de simulação flexível e eficiente para a pesquisa e aplicação de Redes Neurais Artificiais. O *SNNS* é distribuído pela Universidade de Stuttgart como *Free Software* e pode ser obtido gratuitamente via *ftp* a partir do servidor: `ftp://ftp.informatik.uni-stuttgart.de/pub/SNNS`.

Entre as vantagens oferecidas por este simulador, que motivaram seu estudo e adoção para a realização dos experimentos, podem ser citadas:

- O grande número de algoritmos e modelos de RNAs implementados;

¹Laboratório de Inteligência Computacional do ICMC-USP

- A disponibilização do código fonte, permitindo uma maior flexibilidade para alterar os algoritmos implementados e/ou compilá-los em diversas plataformas da “família” *UNIX*;
- A facilidade para trabalhar tanto interativamente (por meio de uma interface gráfica bastante amigável), quanto em modo *batch*, permitindo a execução de experimentos em *background*;
- A capacidade de gerar código em linguagem C;
- O fato de ser mundialmente conhecido e bastante utilizado no meio acadêmico, facilitando o desenvolvimento deste trabalho e a consequente interação com outros pesquisadores da área.

Quanto aos métodos de *Pruning*, os seguintes algoritmos estão implementados no simulador:

- *Magnitude Pruning*;
- *Optimal Brain Damage*;
- *Optimal Brain Surgeon*;
- *Skeletonization*;
- *Non-Contributing Units*.

5.2.2 Constructive NN Learning Simulator

O projeto do *Constructive Neural Network Learning Simulator* tem por objetivo implementar um simulador extensível e que ao mesmo tempo facilite a realização de experimentos utilizando Redes Neurais Construtivas. O simulador foi desenvolvido utilizando o paradigma da Programação Orientada a Objetos, tornando fácil a incorporação de novos algoritmos e

técnicas. A distribuição ainda não é pública e para obtê-lo é necessário contatar diretamente os autores ² Os algoritmos construtivos implementados são os seguintes:

- *Tower Algorithm*;
- *Pyramid Algorithm*;
- *Upstart Algorithm*;
- *Perceptron Cascade Algorithm*;
- *Tiling Algorithm*;
- *Sequential Learning Algorithm*.

Ainda, está disponível um método de *Pruning* para redes *MTiling*.

5.3 Experimentos com Redes MLP

O principal objetivo dos experimentos descritos a continuação é estudar o comportamento dos métodos de *Pruning*, considerando a capacidade de generalização e o tamanho final das redes após a poda.

Os experimentos foram realizados utilizando redes MLP treinadas com o algoritmo *backpropagation* padrão e Redes Construtivas utilizando o algoritmo *MTiling*. Os conjuntos de dados utilizados estão listados a seguir:

- Conjunto de dados de cartões de crédito [Prechelt, 1994];
- Conjunto de dados relacionados a avaliação de crédito (Dados do Banco Banestado cedidos pela FEA-USP);

²Rajesh Parekh (parekh@cs.iastate.edu) e Vasant Honavar (honavar@cs.iastate.edu).

- Conjunto de dados de paridade (extensão do XOR para mais de duas variáveis);

Todos os experimentos foram realizados utilizando os simuladores *SNNS* e o *Constructive NN Learning Simulator* (para redes *MTiling*). Quanto ao primeiro simulador, foi necessário modificar a forma como este executa os algoritmos de *Pruning*. Para tal, a escrita de novos módulos (em linguagem C) foi necessária, sendo que ainda assim foi possível aproveitar os algoritmos nativos do simulador através de bibliotecas.

Os critérios de parada durante o treinamento e a fase de *Pruning* utilizados pelo *SNNS*, não são aconselháveis, uma vez que sempre é utilizado como critério de parada o erro obtido com o conjunto de treinamento, impossibilitando o controle da generalização e do *overfitting* da rede. Os módulos implementados, consideram como critério de parada, o aumento do erro com o conjunto de validação. Na Tabela 5.1 pode ser observada parte do código escrito para treinar e podar as redes.

Cada conjunto de dados inicial deu lugar a 3 partições diferentes, as quais foram divididas em 3 sub-conjuntos distintos cada uma: treinamento (50%), validação (25%) e teste (25%). Esta distribuição dos dados pode ser observada na Figura 5.1.

Figura 5.1: Distribuição dos dados.

Em cada partição criada, foram realizados experimentos seguindo os seguintes passos para cada simulação:

1. A rede inicial é criada, com um tamanho relativamente grande e capaz de aprender o

```

void main ( int argc, char **argv ) {
    // Load Net and Patterns
    errChk ( krui_loadNet ( NetFile, &netname ) );
    errChk ( krui_loadNewPatterns(ValidateFile, &val_set) );
    errChk ( krui_loadNewPatterns(TrainingFile, &tr_set) );

    // Initializing weights
    srand ( time ( 0 ) );
    for ( i = 1+(int) (50.0*rand()/(RAND_MAX+1.0)) ; i > 0 ; i -- )
        errChk ( krui_initializeNet ( initValues, numOfInitParams ) );

    // Train Net
    TrainNet ( TrainCycles, val_set, tr_set, StepSize, FALSE );
    errChk ( krui_loadNet("best_val.net", &netname) );
    minValError = TrainNet ( 0, val_set, tr_set, StepSize, FALSE );

    // Initialize Pruning Parameters
    errChk ( krui_setPrunFunc ( PrunFunction ) );
    if ( ! strcmp ( PrunFunction, "OptimalBrainSurgeon" ) )
        pr_obs_setInitParameter ( 1e-6 );

    // Pruning ...
    minFactor = minValError
    errChk ( krui_saveNet ( BestPruned, "best_pruned" ) );
    do {
        errChk ( pr_callPrunFunc ( -1 ) );
        if ( (minFactor * (1 + (float)MaxIncrease/100)) > currentFactor )
        {
            currentFactor = TrainNet(RetrainCycles, val_set, tr_set, StepSize, TRUE);
            if ( currentFactor < minFactor )
            {
                minFactor = currentFactor;
                errChk ( krui_saveNet ( BestPruned, "best_pruned" ) );
                printf ( "\t*" );
            }
        } else break;
    } while ( 1 );
}

```

Tabela 5.1: Parte do código utilizado nos experimentos.

conjunto de dados;

2. Os pesos são inicializados aleatoriamente com valores entre 1.0 e -1.0;
3. Começa o treinamento, que irá demorar 3.000 ciclos, onde a cada 20 ciclos é calculado o erro da rede com o conjunto de validação. Ao final, aquela rede com o menor erro na validação (considerada como a melhor generalização) é escolhida para ser submetida aos métodos de *Pruning*;
4. Começa a etapa de *Pruning*, retirando sempre um elemento da rede por vez e retreinando-a imediatamente durante 30 ciclos;
5. O passo anterior é executado até o erro de validação da rede aumentar em uma taxa maior que um valor determinado pelo usuário denominado *MaxIncrease*. Nos experimentos o valor adotado para esta variável foi de 200%.

5.3.1 Experimentos com Dados de Crédito do Proben1

Este conjunto de dados é formado por padrões referentes a dados sobre cartões de crédito disponibilizados pelo Proben1 [Prechelt, 1994]. Originalmente, estes dados foram disponibilizados para experimentos com aprendizado de máquina pelo repositório de dados da UCI (*University of California, Irvine*) [Blake and Merz, 1998]. O Proben1 é um conjunto de procedimentos e bases de dados para padronizar a realização de experimentos com RNAs. Estes procedimentos compreendem um conjunto de regras e convenções para a realização de experimentos estatisticamente mais confiáveis e que possam ser comparados com resultados obtidos em diferentes centros para os mesmos dados. Cada conjunto de dados representa problemas reais e tem por objetivo facilitar o trabalho dos pesquisadores interessados em treinar novos algoritmos de aprendizado, assim como realizar experimentos com os algoritmos já existentes.

Estes dados apresentam informações cadastrais de usuários de cartão de crédito junto com a decisão tomada por analistas quanto a concessão ou não do cartão. O problema de classificação da aprovação de crédito a pessoas físicas consiste em prever a aprovação ou não de

uma aplicação de crédito por parte de analistas de crédito observando variáveis referentes à pessoa solicitante. Existe um forte caráter subjetivo nas decisões tomadas por analistas de crédito. Algumas técnicas de aprendizado de máquina (árvores de decisão principalmente) são utilizadas com frequência para fazer avaliação de crédito, devido a seu desempenho e eficiência.

Uma vez que os dados referentes a este tipo de aplicações podem ser divididos entre positivos (crédito aprovado) e negativos (não aprovado), este problema pode ser visto como um processo de classificação de padrões. A utilização de RNAs para este problema justifica-se porque em aplicações que envolvem classificação e reconhecimento de padrões as mesmas obtêm bom desempenho.

Cada padrão deste conjunto de dados, consiste em 51 entradas e 2 saídas. Os valores de entrada são compostos por atributos discretos e também por atributos contínuos. O significado dos atributos não é descrito por motivos confidenciais. Os valores de saída determinam a aprovação do crédito (44.5%) ou a rejeição (55.5%). Os padrões formam um total de 690 exemplos onde 5% dos mesmos estão incompletos.

Os dados iniciais foram normalizados e convertidos ao formato de padrões do *SNNS*. Em seguida, foram embaralhados formando 3 partições diferentes e cada uma delas, por sua vez, foi sub-dividida em 3 sub-conjuntos: treinamento (50%), teste (25%) e validação (25%).

Durante a realização dos experimentos, em cada partição foram treinadas 10 redes diferentes para cada método. A topologia inicial de cada rede é: 51 neurônios na camada de entrada, 60 na camada intermediária e 2 na camada de saída, somando um total de 113 unidades e 3.180 conexões. É importante observar que durante a fase de treinamento e a fase de poda de cada rede, sempre foi preservada a de melhor generalização, independentemente do número final de conexões que venha a adotar.

Inicialmente, devem ser identificados aqueles métodos que conseguiram retirar mais elemen-

tos da rede.³ A Figura 5.2 mostra a média de conexões remanescentes após a aplicação dos métodos de *Pruning*. Na mesma, pode-se notar que os métodos que retiraram mais elementos são *Skeletonization* e *Non-Contributing Units*. Por outro lado, o *Optimal Brain Surgeon* foi incapaz de minimizar a complexidade da rede. Já o *Magnitude Pruning* teve um bom desempenho na partição 2 mas não foi tão bem com as outras partições. Esta última observação indica que a distribuição dos dados tem uma forte influência no desempenho obtido pela rede.

Figura 5.2: Média do número de conexões após a poda.

Na Figura 5.3 observa-se a porcentagem de padrões classificados corretamente utilizando os métodos de *Pruning*. O equilíbrio evidenciado no desempenho com os conjuntos de treinamento, teste e validação mostra que a generalização não está sendo degradada, embora sejam retirados alguns elementos da rede. As redes também estão livres da presença de *overfitting*.

Na Tabela 5.2 são apresentadas as taxas médias e os respectivos desvios padrões das classificações corretas obtidas para os conjuntos de teste, onde é interessante notar que todos os métodos estudados apresentam resultados similares, embora alguns tenham retirado mais elementos da rede que outros.

³Vale lembrar que são consideradas, apenas as redes com a melhor generalização.

Figura 5.3: Taxa média de classificação correta.

Padrões	MAG	NCU	OBD	OBS	SKE
Corretos	78,64 ± 2,62	79,40 ± 2,07	79,25 ± 2,01	79,65 ± 1,51	78,90 ± 3,81
Errados	15,87 ± 2,75	15,11 ± 1,79	16,43 ± 1,51	16,24 ± 1,55	15,21 ± 3,25
Desconhecidos	5,49 ± 4,90	5,49 ± 3,36	4,32 ± 2,40	4,11 ± 2,31	5,89 ± 6,21

Tabela 5.2: Resultado médio para os conjuntos de teste.

5.3.2 Experimentos com Dados de Crédito do Banestado

Este conjunto de dados representa mais uma aplicação real da área financeira, desta vez referente a avaliação do risco de liberação de crédito a pessoas físicas. Cada exemplo deste conjunto contém dados de um usuário descrevendo características como estado civil, se tem casa própria, escolaridade, etc. O último atributo informa a adimplência ou inadimplência do usuário no pagamento da fatura do seu cartão de crédito. Estes dados foram disponibilizados pela FEA-USP utilizando um cadastro de pessoas físicas do Banco Banestado.

O conjunto de dados cedido pela FEA-USP é composto por um total de 1.480 padrões com 38 entradas e 2 saídas. Os dados foram inicialmente normalizados e convertidos ao formato de arquivos de padrões do simulador *SNNS*. Após este primeiro passo, foram embaralhados para formar 3 partições diferentes, onde cada uma delas é composta por 3 sub-conjuntos (treina-

mento, teste e validação) distribuídos de acordo com a metodologia adotada nos experimentos descritos anteriormente.

A topologia inicial de rede escolhida foi a seguinte: 38 neurônios na camada de entrada, 60 na camada intermediária e 2 na camada de saída, totalizando 100 unidades (neurônios) e 2.000 conexões entre elas. Assim como em todos os experimentos, em cada partição foram treinadas 10 redes diferentes (com pesos iniciais diferentes) para cada método de *Pruning*. Ao mesmo tempo, durante o treinamento e a poda, sempre foram preservadas as de melhor generalização, sem importar o número final de conexões ou unidades da rede.

Após realizar os experimentos foi possível constatar que em todas as partições os diferentes métodos tiveram um desempenho parecido em termos do número final de conexões. Na Figura 5.4, pode ser observado um gráfico de barras onde nota-se este comportamento. O *Magnitude Pruning* é o que mais conexões conseguiu extrair, embora a diferença com os outros não seja significativa.

Figura 5.4: Média do número de conexões após a poda.

Ao realizar uma comparação com os resultados obtidos nos experimentos com os dados de crédito do Proben1, percebe-se que os métodos de *Pruning* conseguiram extrair mais elementos da rede com esse conjunto de dados. Esta discrepância pode ser explicada pelo fato de haver (no caso dos dados do Banestado) um maior conjunto de dados (1.480 padrões),

se comparados com os 690 do Proben1. Ao mesmo tempo, o número de conexões inicial também difere, sendo que no caso do Banestado há apenas 2.000 conexões contra 3.180 do Proben1. Pode-se concluir que no caso do Banestado há mais conhecimento relevante a ser armazenado por menos elementos na rede, consequentemente diminui o número de elementos a serem retirados por serem considerados irrelevantes.

Observando a Figura 5.5 reforça-se a idéia de que a informação armazenada pela rede é relevante, uma vez que antes de efetuar a poda foi possível chegar a redes com um bom nível de generalização. Este contexto aparece na aplicação de cada método em todas as partições.

Figura 5.5: Taxa média de classificação correta.

Na Tabela 5.3, observa-se o resultado médio para os conjuntos de teste. É importante observar que não há presença de *overfitting*, e as redes, durante o treinamento conseguiram chegar a níveis aceitáveis de desempenho. Ao analisar o método que apresentou as menores redes após o *Pruning* (*Magnitude Pruning*), percebe-se que o mesmo tem os níveis mais altos de generalização. É importante destacar que este método possui a heurística mais simples visto que a cada passo remove a conexão com o menor peso.

<i>Padrões</i>	MAG	NCU	OBD	OBS	SKE
Corretos	97,17 $\pm$ 0,89	97,00 $\pm$ 0,76	96,98 $\pm$ 0,55	97,12 $\pm$ 0,86	97,15 $\pm$ 0,92
Errados	1,40 $\pm$ 0,78	1,44 $\pm$ 0,64	1,50 $\pm$ 0,64	1,55 $\pm$ 0,77	1,39 $\pm$ 0,75
Desconhecidos	1,43 $\pm$ 0,86	1,56 $\pm$ 0,68	1,52 $\pm$ 0,64	1,32 $\pm$ 0,55	1,46 $\pm$ 0,65

Tabela 5.3: Resultado médio para os conjuntos de teste.

5.3.3 Experimentos com Dados de Paridade

O terceiro e último conjunto de dados utilizado durante os experimentos é formado por dados referentes ao problema da paridade. Este conjunto de dados foi gerado a partir do programa mostrado na Tabela 5.4, onde é mostrado o código fonte utilizado. Um total de 4.096 padrões foram distribuídos em conjuntos de treinamento (50%), teste (25%) e validação (25%) para cada partição (segue a mesma abordagem dos conjuntos anteriores).

```

void main ( int args, char **argv )
{
    register int i, j, k;
    unsigned char number[INPUTS];

    INPUTS = atoi(argv[1]);
    for ( i = 0 ; i < pow(2,INPUTS) ; i ++ )
    {
        k = ToBinary ( i, number );

        // print input pattern
        for ( j = 0 ; j < INPUTS ; j ++ )
            printf ( "%d ", number[j] );

        // print output pattern
        printf ( "\n%d\n", k % 2 );
    }
}

```

Tabela 5.4: Código utilizado para gerar os dados de paridade.

Cada padrão deste conjunto é formado por 12 entradas e 1 saída. Tanto as entradas quanto

as saídas são valores binários. A saída tem o valor 0 (50% dos casos) quando o número de dígitos 1 no padrão for par e 1 (50% também) no caso contrário. Na Tabela 5.5 pode-se observar alguns exemplos de pares entrada-saída deste conjunto de dados.

Entrada	Saída
1 1 1 1 1 1 1 1 1 0 0	0
1 0 0 1 1 0 1 1 0 1 0 1	1
1 1 1 1 1 1 1 1 1 1 1 0	1
1 1 1 1 1 1 1 1 0 1 0 1	0
1 1 1 1 1 1 1 1 0 1 1 0	0

Tabela 5.5: Exemplos de pares entrada-saída.

A escolha destes dados para realizar os testes, deve-se ao fato de ser um conjunto fácil de aprender, além de ser frequentemente utilizado na literatura para a comparação de modelos e algoritmos de treinamento para RNAs. Assume-se que a rede facilmente aprende a função de paridade e a partir desse ponto pode-se observar se os métodos de *Pruning* conseguem reduzir a complexidade da rede sem degradar a generalização.

Da mesma forma que os casos anteriores, durante os experimentos, em cada partição foram treinadas 10 redes diferentes para cada método. A topologia escolhida para inicializar as redes antes do treinamento foi a seguinte: 12 neurônios na camada de entrada, 60 na camada intermediária e 1 na camada de saída, totalizando um total de 73 unidades e 780 conexões.

Ao analisar a Figura 5.6, onde é mostrado o número de conexões final das redes que melhor generalizaram, observa-se que há uma pequena semelhança no desempenho do *Magnitude Pruning* e do *Non-Contributing Units*. Estes métodos tiveram maior dificuldade para podar as redes da Partição 1. Ao mesmo tempo, foram os que obtiveram os melhores resultados. Já o *Optimal Brain Surgeon* por sua vez, não conseguiu podar e ao mesmo tempo manter a generalização da rede. É necessário notar também que a distribuição dos dados implica fortemente no processo de aprendizado de RNAs, principalmente quando espera-se reduzir seu tamanho aplicando técnicas de *Pruning*.

Figura 5.6: Média do número de conexões após a poda.

O número final de neurônios das redes pode ser observado na Figura 5.7, mostrada a seguir. Está claro que o número total de neurônios diminuiu na mesma proporção que o número de conexões. Os métodos: *Magnitude Pruning*, *Non-Contributing Units* e *Optimal Brain Damage*, destacam por conseguirem eliminar o maior número de unidades.

Figura 5.7: Média do número de neurônios após a poda.

Ao verificar a generalização, na Figura 5.8, conclui-se que todos os métodos mantiveram o bom desempenho das redes. Vale lembrar, que este fato foi provocado pelo controle de validação feito durante o treinamento e a poda. É por este motivo que alguns métodos parecem não ter reduzido o tamanho das redes. Na realidade, todos conseguiram extrair

elementos redundantes, mas aqui apenas são mostrados aqueles que conseguiram extrair sem alterar a generalização da rede.

Figura 5.8: Taxa média de classificação correta.

Finalmente na Tabela 5.6 é apresentada a forma como os métodos se comportaram com os conjuntos de teste, reforçando a observação de que a generalização não é alterada pelos mesmos. O *Optimal Brain Damage*, apesar de reduzir em mais de 50% o tamanho inicial da rede, conseguiu manter a mais alta taxa de acerto com o conjunto de teste.

Padrões	MAG	NCU	OBD	OBS	SKE
Corretos	96,79 ± 1,69	96,59 ± 2,30	97,29 ± 1,44	96,37 ± 1,60	96,18 ± 1,89
Errados	3,21 ± 1,69	3,41 ± 2,30	2,71 ± 1,44	3,63 ± 1,60	3,82 ± 1,89
Desconhecidos	0,00 ± 0,00	0,00 ± 0,00	0,00 ± 0,00	0,00 ± 0,00	0,00 ± 0,00

Tabela 5.6: Resultado médio para os conjuntos de teste.

5.4 Experimentos com o Algoritmo MTiling

O conjunto de dados utilizado para treinar Redes Construtivas com o algoritmo *MTiling* foi o de paridade. Da mesma forma que nos outros experimentos, foram criadas 3 partições diferentes. Dentro de cada uma delas 10 redes foram treinadas com o algoritmo *MTiling*.

Uma vez treinadas, foram submetidas a um processo de poda que utilizou os métodos de *Pruning* descritos no capítulo anterior. Considerando que os métodos utilizados com redes *MTiling* procuram remover neurônios ao invés de conexões, foi gerado o gráfico mostrado na Figura 5.9, onde pode-se observar o tamanho final médio (em número de neurônios) das redes treinadas e podadas em cada partição.

Figura 5.9: Média do número de neurônios após a poda.

Igualmente, um gráfico com as taxas médias de classificação correta pode ser construído. Na Figura 5.10 observa-se claramente que a taxa média de acerto (porcentagem) das redes finais para os conjuntos de teste não é tão alta quanto o esperado, ou no mínimo, não correspondem ao desempenho alcançado por redes MLP.

5.5 Análise dos Resultados

Ao analisar os resultados obtidos após a realização dos experimentos, algumas observações podem ser destacadas. Inicialmente, chama a atenção o fato de ser um algoritmo baseado em uma heurística não muito sofisticada que obtém os melhores resultados. O *Magnitude Pruning* na média é o que melhor se comportou com todos os conjuntos de dados.

Outros métodos de destaque são o *Skeletonization* e o *Non-contributing Units*, os quais tive-

Figura 5.10: Resultado médio para os conjuntos de teste.

ram um bom desempenho com os dados do Proben1 e de paridade. Nos testes com os dados do Banestado, as redes finais preservaram seu tamanho inicial, e apenas em alguns casos os métodos de *Pruning* conseguiram retirar alguns elementos.

O *Optimal Brain Damage* obteve bons resultados apenas com os dados de paridade. Este conjunto de dados caracteriza-se por ser de fácil aprendizado e tudo indica que grande parte da informação armazenada pela rede (após o treinamento) não seja tão relevante assim.

O último dos métodos testados com redes MLP é o *Optimal Brain Surgeon*, o mesmo não obteve bons resultados. Este método é derivado do OBD, e é o que demanda um tempo de processamento mais longo. Esta característica, aliada aos resultados obtidos tornam inviável sua aplicação, a não ser que alguma otimização seja feita no seu funcionamento.

A poda de Redes Neurais Construtivas, pode ser verificada ao treinar redes *MTiling* e seguidamente aplicar um algoritmo de poda. Verificou-se que a quantidade final de neurônios destas redes corresponde à quantidade final de neurônios de 3 dos métodos aplicados a redes MLP. Mas a taxa de acerto com os conjuntos de teste mostra uma média em torno de 80% apenas, contra uma média de 97% dos melhores métodos para redes MLP. Por outro lado, constatou-se que Algoritmos Construtivos trabalham com extrema rapidez durante o treinamento e poda. O tempo de processamento é mínimo quando comparado a redes MLP.

Finalmente, em relação às questões levantadas inicialmente, conclui-se que os algoritmos que produziram as redes de menor tamanho foram: *Magnitude Pruning*, *Non-contributing Units* e o *Skeletonization*. Parece existir uma relação entre o tamanho inicial e final das redes. No caso dos dados do Banestado, a topologia escolhida provavelmente estava no seu limite mínimo, impossibilitando os algoritmos de retirarem elementos da rede.

A generalização não varia conforme o tamanho da rede. Foi constatado que essa capacidade não depende diretamente do tamanho da rede e sim de uma conjugação entre o tamanho e a complexidade do conjunto de dados a ser aprendido. Por outro lado, não foi provado que as redes de menor tamanho são as que melhor generalizam, mas foi visto que é possível ter redes de tamanhos mínimos com uma taxa de generalização equivalente ou superior a redes de grandes tamanhos.

5.6 Considerações Finais

O objetivo principal deste capítulo foi o de fazer uma comparação entre um certo número de métodos de *Pruning* descritos no capítulo 4, e assim obter uma indicação de suas vantagens e desvantagens e de quais são os mais apropriados para determinados tipos de problemas. Como foi mostrado, é muito difícil estipular quando uma rede é ótima. Por exemplo, se uma rede com boa capacidade de generalização é o objetivo, então redes de diferentes tamanhos devem ser consideradas. Redes pequenas têm mostrado uma generalização melhor que redes maiores, mas isto não deve ser considerado como uma regra geral. Os métodos de *Pruning* são aplicados a redes que já mostram sinais de boa generalização e, baseados nas próprias características da rede, procuram minimizar a sua complexidade identificando elementos redundantes ou que não armazenam informações de utilidade.

Capítulo 6

Conclusões

O presente trabalho de pesquisa tem como principal objetivo apresentar um estudo comparativo de técnicas de *Pruning* para Redes Neurais Artificiais. Dentre os vários modelos de RNAs, foram estudadas as redes MLP e dois modelos de Redes Neurais Construtivas. Para executar os experimentos com redes MLP foi utilizado o algoritmo de aprendizado supervisionado *backpropagation* e para as Redes Neurais Construtivas foi utilizado o algoritmo *MTiling*.

6.1 Estrutura do Texto

No capítulo 2, foram tratados assuntos relacionados com o surgimento, história e definição de Redes Neurais Artificiais. Aspectos teóricos genéricos foram abordados dando maior ênfase àqueles modelos e paradigmas utilizados no estudo de métodos de *Pruning*. De uma forma geral, as RNAs extraem as principais características do funcionamento do cérebro humano, as quais se caracterizam principalmente pela capacidade de generalizar e aprender a partir de seu próprio ambiente. Atualmente, as RNAs estão cada vez mais presentes em diversas áreas e problemas diferentes o que motiva o seu estudo e aprimoramento.

No capítulo 3, foi dada uma maior ênfase às redes *Multilayer Perceptron* (Perceptron Multi-Camadas) e às Redes Neurais Construtivas. O primeiro modelo é um dos mais utilizados e difundidos por pesquisadores da área de RNAs. Redes deste tipo são caracterizadas por possuírem uma alta capacidade de aprendizado e facilidade de uso, embora o modelo ainda possua certas limitações quanto ao pleno entendimento do seu funcionamento interno. Alguns dos seus parâmetros foram analisados, devido à influência por eles exercida sobre o aprendizado e capacidade de generalização da rede. Todos eles devem ser cuidadosamente estimados para poder encontrar a solução desejada. As Redes Neurais Construtivas, por sua vez, possuem certas características que as tornam interessantes como alternativa para o projeto de redes.

No capítulo 4, foram descritos diversos métodos de *Pruning*. Tanto para redes MLP quanto para RNCs. Estes métodos (ou técnicas) visam minimizar a complexidade das redes (quanto ao seu tamanho) sem degradar a sua capacidade de generalização. Eles são vantajosos, uma vez que redes de menor tamanho apresentam diversas vantagens tais como: melhor generalização e facilidade para extração de regras, demanda por menos recursos na implementação, entre outras.

Os métodos de *Pruning* podem ser grosseiramente classificados em dois grandes grupos: o primeiro grupo engloba aqueles que calculam a sensibilidade do erro à remoção de algum elemento. O segundo grupo compreende os métodos que modificam a função de erro de forma a levar os pesos das conexões consideradas “redundantes” a zero durante o treinamento, para depois serem removidas. Também são apresentados métodos de *Pruning* para Redes Neurais Construtivas, os quais utilizam a abordagem do cálculo de sensibilidade já que não é possível adicionar termos de penalidade à função de erro.

O capítulo 5 foi dedicado à descrição dos experimentos e apresentação dos resultados obtidos. Estes experimentos foram realizados com 3 bases de dados diferentes, onde duas são aplicações do mundo real. As redes utilizadas nos experimentos foram redes MLP treinadas com o algoritmo *backpropagation*, e redes *MTiling* treinadas com o algoritmo do mesmo nome.

6.2 Contribuições e Trabalhos Futuros

O esforço envolvido e os resultados obtidos ao levar adiante o estudo de técnicas de *Pruning*, certamente contribuíram para que a escolha da arquitetura ótima para RNAs seja facilitada e assim pesquisadores e usuários comuns possam contar com ferramentas deste tipo.

Apesar dos resultados obtidos e a forma em que foram avaliados parecer bastante objetiva, é importante destacar que não é possível oferecer qualquer tipo de garantia (em termos absolutos) sobre a escolha da melhor técnica a ser utilizada. Isto se deve ao subjetivismo, inerente aos problemas citados no decorrer do texto, que são enfrentados por redes MLP. Entre os quais, a inicialização dos pesos, a distribuição estatística dos dados e o tipo de aplicação jogam um papel preponderante.

Selecionar a melhor arquitetura para uma rede é uma questão ainda em aberto, mas, de acordo com os resultados aqui apresentados, as técnicas de *Pruning* tornam-se uma alternativa viável em qualquer tipo de aplicação. Sua utilização não implica em um tempo de processamento muito longo para alcançar o resultado desejado. Além disso, estas técnicas podem ser facilmente aplicadas.

A existência de simuladores que incorporam no seu núcleo módulos para podar as redes depois de treinadas evidencia a aceitação desta abordagem para escolha da topologia ótima possui na comunidade de RNAs.

Ao finalizar este trabalho, torna-se claro que mais estudos e pesquisas ainda devem ser realizadas a fim de prover uma fundamentação teórica mais profunda e um entendimento mais claro dos diversos aspectos que envolvem a poda de RNAs. Ao mesmo tempo, métodos alternativos de *Pruning* para redes MLP treinadas com o algoritmo *backpropagation* devem ser motivos de pesquisa, visto que seu funcionamento interno ainda não é conhecido completamente (de forma exata).

Um grande número de pesquisas ainda deve ser realizado, não só na otimização de RNAs

mas também na procura de um conhecimento mais concreto da sua estrutura interna e no seu comportamento em termos de generalização e *overfitting*. Ao treinar uma rede é muito difícil prever se ela vai generalizar ou memorizar os dados. Seu comportamento depende muito dos parâmetros estipulados pelo usuário. Uma minimização da quantidade destes parâmetros aparece como outra possível matéria de investigação.

Finalmente, considerando que os tipos de redes abordados aqui foram: redes *Multilayer Perceptron* treinadas com o algoritmo *backpropagation* e Redes Neurais Construtivas, deve-se verificar o desempenho obtido ao aplicar técnicas de *Pruning* a outras arquiteturas de RNAs.

Referências Bibliográficas

- [Anderson and Rosenfeld, 1988] Anderson, J. A. and Rosenfeld, E., editors (1988). *Neuro-computing: Foundations of Research*. The MIT Press, Cambridge, MA.
- [Baum and Haussler, 1989] Baum, E. and Haussler, D. (1989). What size net gives valid generalization? *Neural Computation*, 1(1):151–160.
- [Blake and Merz, 1998] Blake, C. and Merz, C. (1998). UCI repository of machine learning databases.
- [Boers and Kuiper, 1992] Boers, E. and Kuiper, H. (1992). Biological metaphors and the design of modular artificial neural networks. Technical report.
- [Chauvin, 1989] Chauvin, Y. (1989). A back-propagation algorithm with optimal use of hidden units. In *[Touretzky, 1989]*, pages 519–526.
- [Chauvin, 1990a] Chauvin, Y. (1990a). Dynamic behavior of constrained back-propagation networks. In *[Touretzky, 1990]*, pages 642–649.
- [Chauvin, 1990b] Chauvin, Y. (1990b). Generalization performance of overtrained back propagation networks. In Almeida, L. and Wellekens, C., editors, *Proc. Neural Networks, EUROSIP Workshop 1990*, pages 46–55, Berlin, Heidelberg. Springer.
- [Cuadros et al., 1998] Cuadros, E., Alvarez, M., and Carvalho, A. (1998). Multi-thread object oriented simulator of ontogenic neural networks. In *Proc. of the ICCIMA'98, World Scientific*, pages 276–281.

- [Cun et al., 1990] Cun, Y. L., Denker, J. S., and Solla, S. A. (1990). Optimal brain damage. In *[Touretzky, 1990]*, pages 598–605.
- [Falhman and Lebiere, 1990] Falhman, S. and Lebiere, C. (1990). The cascade correlation learning architecture. In *[Touretzky, 1990]*, pages 524–532.
- [Fiesler and Beale, 1996] Fiesler, E. and Beale, R., editors (1996). *Handbook of Neural Computation*. Institute of Physics and Oxford University Press, New York, New York.
- [Finnoff et al., 1993] Finnoff, W., Hergert, F., and Zimmermann, H. G. (1993). Improving model selection by nonconvergent methods. *Neural Networks*, 6:771–783.
- [Hassibi and Stork, 1993] Hassibi, B. and Stork, D. G. (1993). Second order derivatives for network pruning: Optimal Brain Surgeon. In Giles, C. L., Hanson, S. J., and Cowan, J. D., editors, *Advances in Neural Information Processing Systems*, volume 5. Morgan Kaufmann, San Mateo, CA.
- [Haykin, 1994] Haykin, S., editor (1994). *Neural Networks, A Comprehensive Foundation*. Macmillan Publishing Company.
- [Karnin, 1990] Karnin, E. D. (1990). A simple procedure for pruning back-propagation trained neural networks. *IEEE Transactions on Neural Networks*, 1(2):239–242.
- [Lawrence et al., 1996] Lawrence, S., Giles, C. L., and Tsoi, A. C. (1996). What size neural networks gives optimal generalization? convergence properties of backpropagation. Technical Report UMIACS-TR-96-22, Institute for Advanced Computer Studies, University of Queensland, Australia.
- [Mézard and Nadal, 1989] Mézard, M. and Nadal, J. (1989). Learning in feedforward layered networks: The Tiling algorithm. *Journal of Physics A: Math. Gen.*, 22(12):2191–2203.
- [Minsky and Papert, 1969] Minsky, M. and Papert, S. (1969). *Perceptrons*. The MIT Press, Cambridge, MA.
- [Mozer and Smolensky, 1989] Mozer, M. C. and Smolensky, P. (1989). Using relevance to reduce network size automatically. *Connection Science*, 1(1):3–16.

- [Parekh et al., 1996] Parekh, R., Yang, J., and Honavar, V. (1996). Constructive neural network learning algorithms. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence and the Eighth Innovative Applications of Artificial Intelligence Conference*, pages 1398–1398, Menlo Park. AAAI Press/MIT Press.
- [Parekh et al., 1997] Parekh, R., Yang, J., and Honavar, V. (1997). Pruning strategies for the mtiling constructive learning algorithm. In *Proceedings of the IEEE/INNS International Conference on Neural Networks (ICNN'97)*, pages 1960–1965.
- [Prechelt, 1994] Prechelt, L. (1994). PROBEN 1 - a set of benchmarks and benchmarking rules for neural network training algorithms. Technical Report iratr-1994-21, Universität Karlsruhe, Institut für Programmstrukturen und Datenorganisation.
- [Prechelt, 1995] Prechelt, L. (1995). Adaptive parameter pruning in neural networks. Technical Report TR-95-009, International Computer Science Institute, Berkeley, CA.
- [Prechelt, 1998] Prechelt, L. (1998). Early stopping — but when? *Lecture Notes in Computer Science*, 1524:55–69.
- [Radcliffe, 1991] Radcliffe, N. J. (1991). Genetic set recombination and its application to neural network topology optimisation. Technical Report EPCC-TR-91-21, Edinburgh Parallel Computing Centre, University of Edinburgh.
- [Reed, 1993] Reed, R. (1993). Pruning algorithms — A survey. *IEEE Transactions on Neural Networks*, 4(5):740–746.
- [Rosembblatt, 1958] Rosembblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:386–408.
- [Rueckl et al., 1989] Rueckl, J., Cave, K., and Kosslyn, S. (1989). Why are ‘what’ and ‘where’ processed by separate cortical visual systems ? a computational investigation. *Journal of Cognitive Neuroscience*, 1:171–186.
- [Rumelhart et al., 1986a] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986a). Learning internal representation by error propagation. In Rumelhart, D. E. and McClelland,

- land, J. L., editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1, pages 318–362. The MIT Press. Cambridge, MA.
- [Rumelhart et al., 1986b] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986b). Learning representations by back-propagating errors. *Nature*. 323:533–536. [Reprinted in [Anderson and Rosenfeld, 1988]].
- [Schwartz, 1988] Schwartz, J. (1988). The new connectionism - developing relationships between neuroscience and artificial intelligence. In *The Artificial Intelligence Debate*. The MIT Press, Cambridge, MA.
- [Sietsma and Dow, 1991] Sietsma, J. and Dow, R. (1991). Creating artificial neural networks that generalize. *Neural Networks*, 4(1):67–79.
- [Thimm and Fiesler, 1995] Thimm, G. and Fiesler, E. (1995). Evaluating pruning methods. In *1995 International Symposium on Artificial Neural Networks (ISANN'95)*, pages A2 20–25.
- [Thimm and Fiesler, 1997] Thimm, G. and Fiesler, E. (1997). Pruning of neural networks. Technical Report RR-97-03, IDIAP Research Group.
- [Touretzky, 1989] Touretzky, D. S., editor (1989). *Advances in Neural Information Processing Systems. Proceedings of the 1988 Conference*. Morgan Kaufmann Publishers Inc., San Mateo, CA.
- [Touretzky, 1990] Touretzky, D. S., editor (1990). *Advances in Neural Information Processing Systems 2*. Morgan Kaufmann Publishers Inc., San Mateo, CA.
- [Towell et al., 1991] Towell, G. G., Craven, M. W., and Shavlik, J. W. (1991). Constructive induction in knowledge-based neural networks. In *Proc. of the Eighth International Workshop on Machine Learning*, pages 213–217.
- [Waugh and Adams, 1995] Waugh, S. and Adams, A. (1995). Pruning within cascade correlation. In *Proc. of the International Conference on Neural Networks (ICNN'95)*, pages 1206–1210.

- [Widrow and Hoff, 1960] Widrow, B. and Hoff, M. E. (1960). Adaptive switching circuits. In *1960 IRE WESCON Convention Record*, pages 96–104, New York.
- [Yang et al., 1996a] Yang, J., Parekh, R., and Honavar, V. (1996a). Empirical comparison of graceful variants of the perceptron learning algorithm on non-separable data sets. Technical report, Artificial Intelligence Research Group, Department of Computer Science, Iowa State University.
- [Yang et al., 1996b] Yang, J., Parekh, R., and Honavar, V. (1996b). Mtiling - a constructive neural network learning algorithm for multi-category pattern classification. In *Proc. of the WCNN'96*, pages 182–187.
- [Yao, 1993] Yao, X. (1993). A review of evolutionary artificial neural networks. *International Journal of Intelligent Systems*, 8(4):539–67.
- [Yu, 1992] Yu, X. (1992). Can backpropagation error surface not have local minima. *IEEE Transactions on Neural Networks*, 3(6):1019–1021.
- [Zell et al., 1995] Zell, A. et al. (1995). SNNS User Manual, Version 4.1. Technical report, Institute for Parallel and Distributed High Performance Systems, University of Stuttgart.