UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO

ANÁLISE E SÍNTESE DE ESTRATÉGIAS DE APRENDIZADO PARA REDES NEURAIS ARTIFICIAIS

Leandro Nunes de Castro Silva
Orientador: Fernando José Von Zuben

UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO
DEPARTAMENTO DE ENGENHARIA DE COMPUTAÇÃO E AUTOMAÇÃO INDUSTRIAL

ANÁLISE E SÍNTESE DE ESTRATÉGIAS DE APRENDIZADO PARA REDES NEURAIS ARTIFICIAIS

Leandro Nunes de Castro Silva
Orientador: Fernando José Von Zuben

Dissertação de Mestrado apresentada à Faculdade de Engenharia Elétrica e de Computação como parte dos requisitos exigidos para obtenção do título de Mestre em Engenharia Elétrica.

Área de Concentração: Automação

Campinas – SP – Brasil
Setembro de 1998

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA - BAE - UNICAMP

Si38a

Silva, Leandro Nunes de Castro
Análise e síntese de estratégias de aprendizado para redes neurais artificiais. / Leandro Nunes de Castro Silva.--Campinas, SP: [s.n.], 1998.

Orientador: Fernando José Von Zuben
Dissertação (mestrado) - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Redes neurais (Computação). 2. Inteligência artificial 3. Programação não-linear. I. Zuben, Fernando José Von. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

Agradecimentos

Gostaria de agradecer ao Prof. Dr. Fernando José Von Zuben pela forma na qual ajudou-me a conduzir este trabalho. A escolha adequada dos créditos a serem cumpridos em cada etapa da pesquisa, juntamente com o suporte técnico necessário ao seu cumprimento foram indispensáveis na sua conclusão em prazo reduzido. Sinto-me, também, no dever de dizer que o exemplo de profissionalismo e dedicação constituíram o espelho no qual projetei-me durante o desenvolvimento de minha pesquisa.

Agradeço aos meus pais José do Carmo Castro Silva e Lásara Nunes de Castro pelo grande incentivo e apoio que sempre me deram em tudo de produtivo que fiz na vida. Sem eles nenhum dos ensinamentos que venho recebendo até este momento seriam possíveis.

À amiga e companheira Cynthia Santos Andrade pela paciência com que aceitou todos estes dias de muito trabalho passados nas salas de aula e laboratórios da FEEC/Unicamp.

Aos companheiros de grupo Eurípedes dos Santos e Eduardo Masato Yioda por terem, gentilmente, cedido algumas de suas ferramentas computacionais para que eu pudesse fazer experimentos importantes ao desenvolvimento e conclusão deste trabalho.

À Faculdade de Engenharia Elétrica e de Computação da Unicamp, pela oportunidade de realizar o curso de Mestrado em Engenharia Elétrica, e aos responsáveis pela administração e chefia do Departamento de Engenharia de Computação e Automação Industrial, por permitirem e auxiliarem na utilização de suas instalações e recursos.

À CAPES pela concessão de uma bolsa de estudos e a FAEP por fornecer auxílio para a participação no WCCI'98 realizado no Alaska/USA em maio de 1998.

À todos aqueles que contribuíram de forma direta ou indireta para a conclusão deste trabalho os meus mais sinceros agradecimentos.

Dedico mais este trabalho a meus pais, pois durante minha concepção ofereceram-me o dom da vida, e hoje oferecem-me as melhores condições e exemplos que um homem pode ter.

Índice Analítico

Resumo

CASTRO, L.N., **Análise e Síntese de Estratégias de Aprendizado para Redes Neurais Artificiais**. Campinas: FEEC, UNICAMP, Setembro de 1998. Dissertação de Mestrado – Faculdade de Engenharia Elétrica e de Computação, Universidade Estadual de Campinas, pp. 250.

Esta dissertação trata do desenvolvimento, análise e implementação de métodos de treinamento para redes neurais artificiais, envolvendo principalmente métodos que recorrem à informação de segunda ordem vinculadas a redes do tipo perceptron de múltiplas camadas. Como complemento ao estudo das arquiteturas com múltiplas camadas que utilizam treinamento supervisionado, é feita uma análise detalhada de um modelo auto-organizado e são propostas alternativas para alguns problemas críticos das redes neurais artificiais, como a determinação da arquitetura ótima a ser utilizada na solução de cada problema e métodos eficientes para a determinação de condições iniciais.

Os resultados deste trabalho têm o propósito de contribuir no sentido de anunciar à comunidade científica que a área de redes neurais artificiais está concluindo um longo e necessário processo de transição entre uma fase inicial, caracterizada pela predominância de aspectos empíricos, e uma fase de maturidade científica, caracterizada pela formalização teórica dos resultados e maior proximidade com outras áreas de atuação científica já estabelecidas.

Palavras-chave: redes neurais artificiais, métodos de otimização não-linear irrestrita.

Abstract

CASTRO, L.N., **Analysis and Synthesis of Learning Strategies for Artificial Neural Networks**. Campinas: FEEC, UNICAMP, Brazil, September 1998. Masters Dissertation – Computer and Electrical Engineering School, State University of Campinas, pp.

This dissertation aims at developing, analysing and implementing training strategies for neural networks, involving mainly methods that take into account the second order information of feedforward multilayer perceptron networks. As a complement to the study of multilayer architectures, a detailed analysis of a self-organized model is performed, and some alternatives for determining optimal dimensionality of the architecture and efficient initial conditions are also proposed.

The results of this work are supposed to contribute in the sense of annoucing to the scientific community that the artificial neural networks field is concluding a long and necessary transition between an initial phase, characterized by a great deal of heuristic content, and a phase of scientific maturity, characterized by the theoretical formalization of the results and strong interaction with other well stablished scientific fields.

Keywords: artificial neural networks, unconstrained non-linear optimization techniques.

Capítulo 1

Introdução

Esta dissertação trata do desenvolvimento, análise e implementação de métodos de treinamento para redes neurais artificiais, envolvendo principalmente métodos que recorrem à informação de segunda ordem vinculadas a redes do tipo perceptron de múltiplas camadas. Neste capítulo, o enfoque do trabalho será posicionado dentro do contexto científico em que se insere, serão apresentadas as motivações que levaram ao seu desenvolvimento, descritos os principais objetivos e, por último, apresentada a organização do texto.

1.1 Introdução

O estímulo inicial que conduziu ao desenvolvimento de modelos matemáticos de redes neurais, denominados redes neurais artificiais, foi um esforço para entender mais detalhadamente o funcionamento do cérebro humano. O objetivo era construir mecanismos que operassem de modo similar, ou seja, que tomassem decisões, processassem informação, aprendessem, lembrassem e otimizassem da mesma forma e, se possível, até de forma mais eficiente que o cérebro humano. Tomando por base os protótipos até aqui desenvolvidos, é de consenso geral que este objetivo ainda está longe de ser atingido.

No entanto, continua elevado e em forte expansão o interesse na formalização e aplicação de modelos de redes neurais artificiais. A razão para o estabelecimento desta aparente contradição é o fato do referido estímulo inicial ter dado lugar a evidências concretas e convincentes acerca do enorme potencial destas estruturas quando aplicadas na análise e síntese de sistemas não-lineares e na generalização de resultados expressivos já obtidos em outras áreas de atuação científica. Dentre estas outras áreas, destacam-se estatística, teoria de informação, teoria de aproximação de funções, teoria de processamento de sinais, teoria de controle de processos e otimização de sistemas.

A maturidade recentemente atingida pelas redes neurais artificiais como área de atuação científica tem levado invariavelmente ao desenvolvimento de ferramentas de engenharia mais eficazes e à utilização mais eficiente dos recursos computacionais hoje disponíveis, o que implica uma ampliação sem precedentes na capacidade de manipular informação em suas mais diversas formas. Estes dois fatores têm viabilizado a solução dos

mais variados tipos de problemas, contribuindo de forma significativa para a extensão das fronteiras do conhecimento humano, particularmente vinculadas à área de inteligência computacional (ZURADA *et. al*, 1994).

Embora estas afirmações, em uma primeira análise, se apresentem demasiadamente pretensiosas, elas representam apenas os resultados iniciais de um processo de consolidação da sinergia entre teoria de sistemas não-lineares e teoria de computação.

1.2 Motivação para a Utilização de Redes Neurais Artificiais

Redes Neurais Artificiais (RNA's) de processamento numérico, arquitetura em camadas e fluxo de informação com e sem realimentação, produzem estruturas de processamento de sinais com grande poder de adaptação e capacidade de representação não-linear. A presença de realimentação cria a possibilidade de armazenar informações na forma de representações internas, além de introduzir dinâmica no processo.

Sempre que métodos tradicionais, derivados da teoria de sistemas lineares, não apresentarem bom desempenho no tratamento de problemas envolvendo, por exemplo, aproximação de funções multivariáveis ou identificação e controle de sistemas dinâmicos, redes neurais artificiais do tipo descrito acima despontam como alternativas bastante competitivas, devido principalmente à sua capacidade de representação de comportamentos não-lineares arbitrários.

No entanto, para explorar adequadamente esta capacidade de representação, é necessário desenvolver algoritmos de treinamento que permitam tratar eficientemente o poder de adaptação presente nessas estruturas. Para tanto, os mais eficientes métodos de otimização não-linear, bem como resultados derivados da teoria de sistemas não-lineares, são comumente empregados.

Algumas características importantes presentes nas RNA's são:

- capacidade de processamento paralelo;
- representação distribuída de conhecimento;
- tolerância a falhas;
- possibilidade de incorporar redundâncias;
- grande capacidade de adaptação;
- flexibilidade no atendimento a critérios de generalização; e
- possibilidade de incorporação de poderosas ferramentas algébricas, facilitando a análise e interpretação dos resultados.

Certamente, essas características não são exclusivas das redes neurais artificiais. Estão presentes, ao menos em parte, em inúmeros outros modelos aplicados aos mesmos propósitos, incluindo modelos lineares.

Apesar disso, quando comparado aos tradicionais modelos lineares, qualquer modelo não-linear adequadamente projetado é potencialmente capaz de produzir melhor desempenho. Esta afirmação, apesar de verdadeira, não contempla nenhum critério de ordem prática, justificando o predomínio histórico de abordagens lineares no tratamento de problemas inerentemente não-lineares.

Conclui-se, então, que qualquer iniciativa de reverter este quadro, ou seja, de explorar a maior capacidade potencial das abordagens não-lineares, incluindo aquelas baseadas em redes neurais artificiais, deve invariavelmente estar associada a motivações de ordem prática. É por esta razão que o atual crescimento de interesse no estudo e implementação de abordagens conexionistas é justificado com base no aumento proporcional de poder de processamento a baixo custo.

A eficácia das ferramentas de engenharia, baseadas nos conceitos mais avançados de redes neurais artificiais, se manifesta na forma de dois aspectos básicos: redução drástica da participação do usuário (geralmente não-especializado) em seu desenvolvimento e capacidade de determinação automática da complexidade dos problemas a serem tratados.

1.3 Motivação para o Estudo de Redes Neurais Artificiais

Os métodos tradicionais, seqüenciais e baseados em lógica binária, são bem sucedidos em diversas áreas, mas falham na resolução de determinados tipos de problemas. O desenvolvimento de redes neurais artificiais iniciou-se aproximadamente há 55 anos. Os primeiros sucessos foram ofuscados pelo rápido desenvolvimento da computação digital. Dúvidas levantadas acerca do potencial dos primeiros modelos também fizeram com que a área sofresse um certo atraso de desenvolvimento.

Conforme os computadores modernos tornaram-se mais poderosos, os cientistas foram desafiados a utilizar estas máquinas para realizar tarefas desempenhadas pelos processos cognitivos humanos. Baseados em exemplos, juntamente com algum "professor", podemos aprender facilmente a reconhecer a letra *A*. Uma maior experiência permite-nos refinar nossas respostas e melhorar nossa representação da informação. Somos capazes de agrupar padrões similares até mesmo sem um professor.

Embora sendo capazes de descrever regras através das quais tomamos certas decisões, estas não refletem necessariamente os processos que estão sendo usados. A representação de conhecimentos através de regras possui a desvantagem de ser implementada de forma seqüencial e geralmente muito simplificada, onde cada regra representa uma parte do conhecimento como uma sentença condicional, e cada chamada às regras uma seqüência de ações (PATTERSON, 1990).

O interesse recente pelas redes neurais artificiais pode ser atribuído a vários fatores:

- técnicas de treinamento foram desenvolvidas para as arquiteturas mais sofisticadas, capazes de resolverem os problemas que causaram o descrédito dos primeiros modelos;
- contribuições provenientes de outras áreas têm guiado o processo de formalização dos resultados, substituindo procedimentos empíricos de validação;
- a tecnologia atualmente disponível pode ser utilizada para construir *hardware* específico para redes neurais artificiais;
- ao mesmo tempo que o progresso da computação clássica tornou o estudo das RNA's mais promissor, limitações encontradas no processamento inerentemente seqüencial destas máquinas, motivaram novos rumos na pesquisa em redes neurais;
- as mais recentes abordagens sobre computação paralela podem se beneficiar do estudo dos sistemas neurais biológicos, que são inerentemente paralelos.

1.4 Objetivos do Trabalho

Os resultados deste trabalho têm o propósito claro de contribuir no sentido de anunciar à comunidade científica que a área de redes neurais artificiais está concluindo um longo e necessário processo de transição entre uma fase inicial, caracterizada pela predominância de aspectos empíricos, e uma fase de maturidade científica, caracterizada pela formalização teórica dos resultados e maior proximidade com outras áreas de atuação científica já estabelecidas.

A necessidade de aplicação das redes MLP a problemas do mundo real, ou problemas que necessitam de respostas em um período de tempo curto (tempo real), é apenas um dos fatores que inviabilizam a utilização do algoritmo de treinamento clássico para a aprendizagem deste tipo de rede. O principal objetivo deste trabalho é apresentado nos Capítulos 3, 4, 5 e 6, em que são abordados vários métodos para o treinamento de redes do tipo *perceptron de múltiplas camadas* (MLP, do inglês *MultiLayer Perceptron*). Particular

ênfase será dada aos métodos cujas propriedades de convergência são comprovadamente superiores às do algoritmo clássico quando aplicados a uma grande variedade de problemas de treinamento. Este trabalho também aborda os problemas de *mínimos locais*, *capacidade de generalização* e *critérios de parada no treinamento*, *determinação do número ótimo de unidades intermediárias* (*algoritmos construtivos com etapas de poda*) e *implementação em paralelo*. Apresenta, ainda, estratégias a serem utilizadas em conjunto com os métodos de aceleração da aprendizagem para reduzir os efeitos destas limitações.

Especial atenção será dada à formalização matemática, abrangendo diversas áreas de atuação científica, como teoria de otimização clássica, teoria de aproximação de funções e teoria de análise numérica. Ao contrário de muitas variações do algoritmo original baseadas em técnicas empíricas, os métodos apresentados não requerem a definição de parâmetros pelos usuários e possuem propriedades de estabilidade e convergência bem definidas.

Os tópicos relativos à rede MLP visam apresentar:

- os princípios envolvidos no projeto de algoritmos de treinamento eficientes para redes do tipo MLP (Capítulo 3);
- a implementação de uma grande variedade de métodos de segunda ordem (incluindo métodos de gradiente conjugado, quase-Newton e Newton), (Capítulos 3 e 4);
- a compreensão dos méritos dos algoritmos de segunda ordem, derivados de teorias avançadas de otimização e comparados com os métodos convencionais de treinamento (Capítulo 5);
- o estudo da capacidade de generalização dos modelos de redes neurais resultantes (Capítulo 6);
- a comparação de desempenho de diferentes algoritmos de treinamento quando aplicados a vários problemas clássicos e de natureza distinta (Capítulos 3 e 9);
- estratégias de redução dos efeitos de mínimos locais (Capítulo 7);
- estratégias de determinação da dimensão ótima das arquiteturas empregadas (Capítulo 7) e
- estudos preliminares sobre a implementação em paralelo destas arquiteturas (Capítulo 7).

Outra contribuição relevante apresentada no Capítulo 7 é a proposta de um método de inicialização do conjunto de pesos das redes MLP que visa explorar correlações presentes no conjunto de dados e proporcionar aos neurônios a capacidade de utilizar informações contidas

nas próprias amostras de treinamento para gerar um vetor inicial de parâmetros mais adequado à tarefa a ser desempenhada. Grande parte dos diferentes métodos de inicialização do processo de treinamento que se encontram na literatura desprezam as informações existentes nos dados e consideram apenas a arquitetura de rede que está sendo utilizada para definir os pesos iniciais. A abordagem é baseada em procedimentos estatísticos, validados pelos resultados experimentais obtidos.

Para finalizar este trabalho será feito um estudo sobre os *mapas auto-organizáveis de Kohonen* (SOM, do inglês *Self-Organizing Maps*) no Capítulo 8, cujo treinamento é do tipo não-supervisionado, e será proposto um método de determinação da arquitetura mínima, ou quase-mínima, para este tipo de rede. Ainda utilizando os SOM's apresentaremos uma nova estratégia de inicialização para o método de aprendizagem baseado em modelos de projeção que será estudado no Capítulo 7. Este método visa explorar ao máximo o espaço inicial de soluções, gerando vetores aproximadamente uniformemente distribuídos em um espaço multidimensional, e será apresentado no Capítulo 8.

As redes auto-organizáveis de Kohonen são muito utilizadas para mapear um conjunto de dados de entrada em regiões definidas em um mapa de saída. Em outras palavras, desejamos agrupar os dados que possuem características semelhantes em regiões próximas do espaço de saída. Neste contexto, a determinação da dimensão do espaço de mapeamento de saída permanece uma incógnita, e uma escolha inadequada desta dimensão pode levar a redundâncias nas soluções, assim como a um desperdício do potencial computacional utilizado na modelagem da estrutura. A estratégia proposta desempenha uma análise estatística do mapeamento feito pela rede durante o treinamento e verifica se existe ou não a necessidade de eliminar unidades redundantes, ou seja, que representam o mesmo grupo de dados.

1.5 Organização do Texto

Esta dissertação está estruturada em dez capítulos, um apêndice e um relatório técnico da seguinte maneira:

Capítulo 1

Uma introdução concisa dos propósitos do trabalho, juntamente com as motivações para o seu desenvolvimento são apresentadas. O conteúdo da dissertação é apresentado e detalhado.

Capítulo 2

É feita uma discussão introdutória sobre redes neurais artificiais, objetivando fornecer a formalização matemática necessária ao resto do trabalho desenvolvido. Um breve histórico, assim como as características principais dos modelos de redes neurais artificiais e considerações sobre a representação do conhecimento também serão apresentados. Este capítulo aborda alternativas para o critério de parada "clássico" que é o erro quadrático médio, cobrindo assim um tópico que é omitido de alguns textos introdutórios sobre redes neurais artificiais.

Capítulo 3

Neste capítulo são descritos e analisados vários métodos de otimização não-linear irrestrita para o treinamento supervisionado de redes do tipo MLP. São considerados os casos em que estes métodos podem ser implementados utilizando procedimentos de busca unidimensional do passo de ajuste. Não objetivamos indicar diretamente a eficiência destes algoritmos em uma dada aplicação, mas analisar suas principais propriedades e definir o escopo de aplicação.

Capítulo 4

Neste capítulo serão feitos estudos sobre a determinação adequada do passo de ajuste utilizado no algoritmo de retro-propagação e nos métodos de otimização estudados no Capítulo 3. Várias técnicas podem ser sugeridas para o ajuste de parâmetros durante o treinamento, de forma que o passo de ajuste e a direção de busca sejam alterados.

Capítulo 5

Este capítulo analisa o treinamento de redes MLP sob o ponto de vista numérico, em particular os problemas de condicionamento, armazenagem eficiente da informação de segunda ordem, escalonamento e esforço computacional. Uma interpretação geométrica para os problemas mal-condicionados é fornecida. Por último, são apresentadas técnicas de análise dos dados utilizados no processo de treinamento da rede.

Capítulo 6

Neste capítulo serão vistos alguns critérios utilizados pelos procedimentos de validação cruzada e feitos comentários gerais sobre a teoria de regularização, regras para apresentação de resultados e problemas clássicos de teste (*benchmarks*) utilizados pela comunidade científica voltada para o estudo de RNA's.

Capítulo 7

O objetivo deste capítulo é fazer uma análise geral sobre as principais limitações que dificultam ou impedem a utilização das redes do tipo MLP. São analisados diferentes métodos de inicialização do conjunto de pesos da rede e um novo método é proposto. São estruturados também algoritmos para a determinação ótima da arquitetura e a capacidade de implementação paralela dos algoritmos.

Capítulo 8

Neste capítulo é apresentada uma descrição dos mapas auto-organizáveis de Kohonen, assim como é proposto um método de poda para esta rede, cujo objetivo principal é a determinação de uma arquitetura mínima capaz de resolver o problema abordado. Ainda utilizando os SOM's, apresentaremos uma nova estratégia de inicialização para o método de aprendizagem baseado em modelos de projeção (PPL), estudado no Capítulo 7.

Capítulo 9

Este capítulo contém todos os resultados experimentais obtidos durante os processos de simulação (síntese) de algoritmos e arquiteturas estudados e propostos.

Capítulo 10

É apresentada uma análise geral dos resultados obtidos durante todo o desenvolvimento do trabalho e são discutidas propostas de generalização das técnicas apresentadas e desenvolvidas. Perspectivas e propostas de trabalhos futuros também são sugeridas.

Apêndice

As definições e axiomas apresentados neste apêndice representam uma coletânea de conceitos básicos de álgebra linear, fundamentais para a formulação de boa parte dos resultados apresentados neste estudo. A inclusão destes conceitos na forma de apêndice se justifica pela sua utilização generalizada na apresentação dos resultados principais deste trabalho.

Relatório Técnico

O relatório técnico contém o código dos principais algoritmos e funções implementadas neste estudo. Todos os algoritmos e funções apresentados estão codificados sob a forma de programas para o software Matlab®, não sendo necessário recorrer a nenhuma função específica de pacotes suplementares (*toolboxes*).

Capítulo 2

Redes Neurais Artificiais

Este capítulo apresenta uma breve introdução sobre os conceitos básicos da teoria de redes neurais artificiais e inicia o estudo do assunto principal deste trabalho – o treinamento de redes do tipo perceptron de múltiplas camadas. O perceptron de múltiplas camadas é a arquitetura de redes neurais artificiais mais utilizada. Sua popularidade é atribuída ao fato de que tem sido aplicada com sucesso a uma grande variedade de problemas de processamento de informação, incluindo classificação de padrões, aproximação de funções e previsão de séries temporais. A derivação do algoritmo de retro-propagação e considerações sobre as virtudes e limitações das redes do tipo perceptron de múltiplas camadas serão brevemente comentadas.

2.1 Introdução

Uma *rede neural artificial* (RNA) é um sistema de processamento de informação que possui algumas características de desempenho em comum com as redes neurais biológicas. Os modelos neurais artificiais têm como principal fonte de inspiração as redes neurais biológicas. A Figura 2.1 apresenta um modelo de um neurônio biológico com a seqüência de propagação dos sinais pela célula.

A natureza das RNA's faz com que seu estudo seja multidisciplinar, envolvendo pesquisadores de diversas áreas, como neurofisiologia, psicologia, física, computação e engenharia.

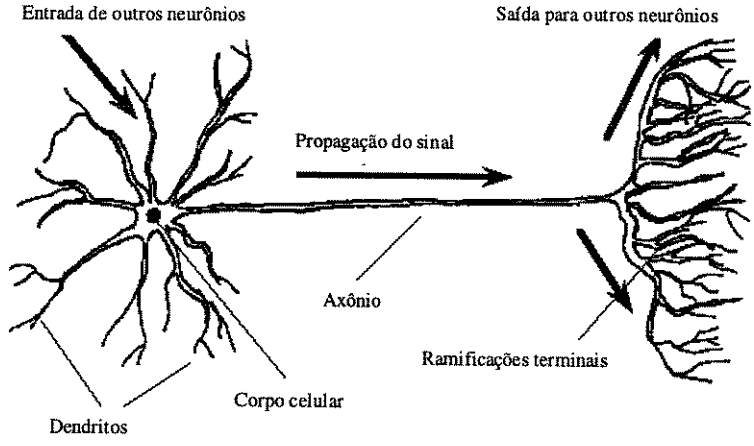


Figura 2.1: Célula neural biológica com a seqüência de propagação do sinal.

Neurofisiologistas e psicólogos estão particularmente interessados em compreender o funcionamento do sistema neural humano. As características de resposta a estímulos apresentada por neurônios individuais, bem como redes de neurônios, são alvo de estudo dos neurofisiologistas, enquanto os psicólogos estudam funções do cérebro no nível cognitivo e estão interessados na utilização de técnicas baseadas em redes neurais para criar modelos detalhados do comportamento humano.

Cientistas da área de computação têm em vista a construção de computadores dotados de processamento paralelo, buscando superar as limitações impostas pelos computadores atuais, que realizam processamento serial em nível simbólico.

Inspirados na habilidade apresentada pelos seres humanos e outros animais no desempenho de funções como o processamento de informações sensoriais e a capacidade de interação com ambientes pouco definidos, os engenheiros estão preocupados em desenvolver sistemas artificiais capazes de desempenhar tarefas semelhantes. Habilidades como capacidade de processamento de informações incompletas ou imprecisas e generalização são propriedades desejadas em tais sistemas.

2.2 Um Breve Histórico

As redes neurais artificiais passaram por um interessante processo de evolução, marcado por um período de grande atividade seguido por anos de estagnação nas pesquisas e pelo ressurgimento do interesse científico como conseqüência do desenvolvimento de novas tecnologias e fundamentos teóricos. A seguir, é apresentado um breve histórico da pesquisa em redes neurais, sendo enfatizados alguns resultados e conceitos considerados relevantes no desenvolvimento deste trabalho (VON ZUBEN, 1993).

Alguns dos mais destacados pesquisadores envolvidos no estudo e aplicação de redes neurais nas últimas três décadas estão relacionados na Tabela 2.1. Esta tabela é dividida de forma a ressaltar o período cronológico mais significativo na atividade científica de cada pesquisador, e é tomada como base no processo de seqüenciamento histórico.

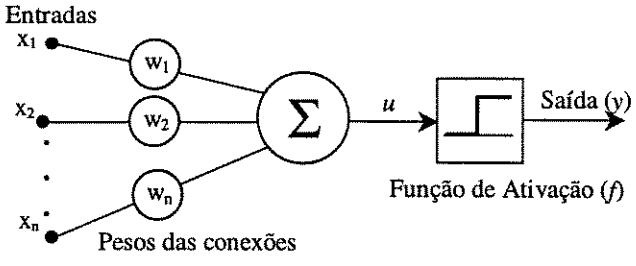


Figura 2.2: Representação funcional de um neurônio.

McCULLOCH & PITTS (1943) projetaram a estrutura que é conhecida como a primeira rede neural. Estes pesquisadores propuseram um modelo de neurônio como uma unidade de processamento binária (veja Figura 2.2) e provaram que estas unidades são capazes de executar muitas das operações lógicas. Este modelo, apesar de muito simples, trouxe uma grande contribuição para as discussões sobre a construção dos primeiros computadores digitais, permitindo a criação dos primeiros modelos matemáticos de dispositivos artificiais que buscavam analogias biológicas.

Em 1948, N. WIENER (1948) criou a palavra cibernética para descrever, de forma unificada, controle e comunicação nos organismos vivos e nas máquinas.

Em 1949, D. O. HEBB (1949) apresentou uma hipótese a respeito da maneira com que a força das sinapses no cérebro se alteram em resposta à experiência. Em particular ele sugeriu que as conexões entre células que são ativadas ao mesmo tempo tendem a se fortalecer, enquanto que as outras conexões tendem a se enfraquecer. Esta hipótese passou a influir decisivamente na evolução da teoria de aprendizagem em redes neurais artificiais.

Tabela 2.1: História da pesquisa em redes neurais

1943	McCulloch e Pitts
1948	Wiener
1949	Hebb
1957	Rosenblatt
1958	Widrow e Hoff
...	...
1969	Minsky e Papert
...	...
1960-1980	Kohonen, Grossberg, Widrow, Anderson, Caianiello, Fukushima, Ígor Aleksander
...	...
1974	Werbos
...	...
1982	Hopfield
1986	Rumelhart e McClelland

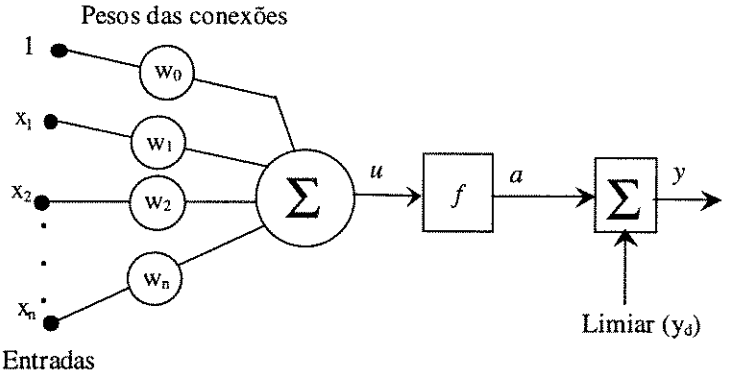


Figura 2.3: O perceptron.

Em 1957, ROSENBLATT (1957) introduziu uma nova abordagem para o problema de reconhecimento de padrões com o desenvolvimento do *perceptron*, cuja representação diagramática é apresentada na Figura 2.3. Rosenblatt também propôs um algoritmo para o ajuste dos pesos do perceptron e provou sua convergência quando os padrões são linearmente separáveis. Por volta do mesmo período, B. WIDROW (1962) e seus colaboradores desenvolveram o *adaline* (***Adaptive Linear Element***).

Apesar do sucesso do perceptron e do adaline, a pesquisa em redes neurais passou gradualmente a conviver com dois problemas fundamentais. Devido ao fato de a maior parte das pesquisas até então desenvolvidas ser de natureza heurística, o primeiro problema estava vinculado à carência de resultados teóricos que justificassem a manutenção do interesse científico pela área, o que ocasionou a redução na produção de novas idéias. O segundo problema, e talvez o de maior significado histórico, foi a expectativa exagerada criada pelos próprios pesquisadores desta área, não acompanhada de resultados à altura, o que acelerou a queda de financiamentos para pesquisa. Mas foi após a publicação, em 1969, do livro "Perceptrons" de autoria de M. L. MINSKY & S. A. PAPERT (1969), que as pesquisas na área de redes neurais sofreram uma retração significativa. Neste livro, conceitos de matemática moderna como topologia e teoria de grupo são aplicados com o objetivo de analisar as capacidades adaptativas e computacionais de neurônios do tipo apresentado na Figura 2.3. Como resultado, os autores demonstraram que o perceptron, apesar de ser capaz de executar as operações booleanas AND e OR, não é capaz de executar outras operações elementares, como XOR (OU-exclusivo). Além disso, esses autores não acreditavam que uma arquitetura adequada, juntamente com um algoritmo de ajuste de pesos, pudessem ser desenvolvidos de forma a superar esta limitação. Após a publicação destes resultados, a maior parte dos

pesquisadores da área de redes neurais passou a buscar alternativas dentro do campo da engenharia e, principalmente, da lógica matemática, que, na época, encontrava-se em franca expansão devido às grandes conquistas realizadas na área de computação.

Apesar deste êxodo generalizado, um número de pesquisadores continuou a trabalhar com redes neurais nos anos 70. Os nomes de T. Kohonen (Finlândia), S. Grossberg, B. Widrow e J. Anderson (Estados Unidos), E. Caianiello (Itália), K. Fukushima (Japão) e Ígor Aleksander (Inglaterra) estão associados a este período.

Nos anos 80, muitos fatores contribuíram para o ressurgimento definitivo das pesquisas em redes neurais:

- neurofisiologistas foram adquirindo um maior conhecimento sobre o processamento de informações nos organismos vivos;
- avanços tecnológicos tornaram disponível um maior potencial computacional a baixo custo, viabilizando ou facilitando simulações e testes com modelos neurais e
- novas teorias para a implementação de algoritmos adaptativos foram desenvolvidas, permitindo a aplicação em sistemas reais.

O ganhador do Prêmio Nobel J. HOPFIELD (1982), do Instituto de Tecnologia da Califórnia, juntamente com D. TANK, um pesquisador da AT&T, desenvolveram grande quantidade de modelos de redes neurais baseadas em pesos fixos e ativações adaptativas. Estas redes podem servir como memórias autoassociativas e serem usadas para resolver problemas de otimização restrita como o caso do "Caixeiro Viajante" (URL 3). A rede de Hopfield pode ser considerada como um sistema dinâmico com um número finito de estados de equilíbrio, de forma que o sistema invariavelmente irá evoluir para um destes estados ou para uma seqüência periódica de estados a partir de uma condição inicial. É também natural que a localização destes estados de equilíbrio possa ser controlada pela intensidade das conexões (pesos) da rede neural.

A conclusão interessante adotada por Hopfield foi que tais estados de equilíbrio podem ser utilizados como dispositivos de memória. De forma distinta daquela utilizada pelos computadores convencionais, em que o acesso à informação armazenada se dá por meio de um endereço, o acesso ao conteúdo da memória de uma rede de Hopfield se dá permitindo que a rede evolua com o tempo para um de seus estados de equilíbrio. Tais modelos de memória são denominados memórias endereçáveis por conteúdo.

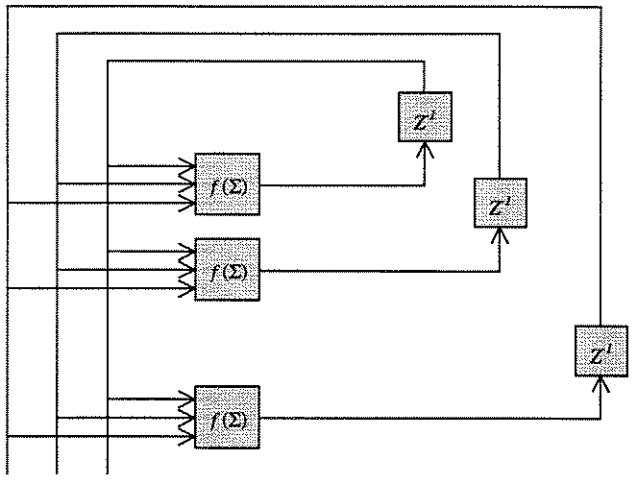


Figura 2.4: Rede Neural de Hopfield.

O trabalho de Hopfield com este tipo de rede simétrica recorrente atraiu, principalmente, matemáticos e engenheiros para a pesquisa nesta área, e as redes de Hopfield foram estudadas como memórias distribuídas e também utilizadas como ferramentas na solução de problemas de otimização restrita.

No entanto, o fato que efetivamente colocou a área de redes neurais como uma das prioritárias na obtenção de recursos foi o desenvolvimento de um método para ajuste de parâmetros de redes não-recorrentes de múltiplas camadas. Este método, baseado em um algoritmo denominado retro-propagação (*backpropagation*), tornou-se largamente conhecido após a publicação, em 1986, do livro *Parallel Distributed Processing*, editado por J. L. MCCLELLAND & D. E. RUMELHART (1986a-1986b), fazendo com que pesquisadores das mais diferentes áreas passassem a visualizar interessantes aplicações para redes neurais artificiais. A importância deste método justifica um tratamento mais aprofundado, a ser desenvolvido nas próximas seções.

2.3 Características Principais

As redes neurais artificiais têm sido desenvolvidas como generalizações de modelos matemáticos da cognição humana ou biologia neural, assumindo que:

- o processamento da informação ocorre em vários elementos chamados *neurônios*;
- os sinais são propagados de um elemento a outro através de *conexões*;
- cada conexão possui um *peso* associado, que, em uma rede neural típica, pondera o sinal transmitido; e

- cada neurônio (ou unidade) aplica uma *função de ativação* (geralmente não-linear) à sua entrada de rede (soma ponderada dos sinais de entrada) para determinar sua saída.

Uma rede neural pode ser caracterizada por três aspectos principais: (1) o padrão de conexões entre as unidades (*arquitetura*), (2) o método de determinação dos pesos das conexões (*algoritmo de treinamento ou aprendizado*) e (3) sua *função de ativação*.

Os modelos neurais artificiais oferecem um paradigma atrativo, pois "aprendem" a resolver problemas através de exemplos.

O treinamento de RNA's pode ser dividido em:

- supervisionado: necessita de um "professor" durante a fase de aprendizagem, que antecede a utilização (execução) da rede; e
- não-supervisionado: direcionado por correlações existentes nos dados de entrada e, portanto, não necessita de um "professor".

Existem vários tipos de modelos de RNA's atualmente. Novos modelos (ou pelo menos variações de alguns já existentes) são propostos constantemente. A seguir apresentamos uma lista com algumas das arquiteturas mais conhecidas até hoje. A distinção principal entre os modelos citados refere-se ao tipo de treinamento (URL 1).

Treinamento não-supervisionado:

1) Redes recorrentes:

- Grossberg Aditivo (AG)
- *Adaptive Resonance Theory* (ART1)
- Hopfield Simétrico e Assimétrico (DH/CH)
- Memória Associativa Bidirecional (BAM)
- Memória Associativa Temporal (TAM)
- Mapa Auto-organizável de Kohonen (SOM)
- Aprendizado Competitivo

2) Redes somente com propagação positiva (*feedforward*):

- *Learning Matrix* (LM)
- *Driver-Reinforcement Learning* (DR)
- Memória Associativa Linear (LAM)
- *Counterprogation* (CPN)

Treinamento Supervisionado:

1) Redes Recorrentes:

- Máquina de Boltzmann (BM)
- *Mean Field Annealing* (MFA)
- *Cascade Correlation* Recorrente (RCC)
- Aprendizado Recorrente em Tempo Real (RTRL)
- Filtro de Kalman Recorrente (EKF)

2) Redes somente com propagação positiva (*feedforward*):

- *Perceptron*
- *Adaline, Madaline*
- Retro-propagação – *Backpropagation* (BP)
- Máquina de Cauchy (CM)
- *Artmap*
- Rede Lógica Adaptativa (ALN)
- *Cascade Correlation* (CasCor)
- Filtro de Kalman (EKF)
- *Learning Vector Quantization* (LVQ)
- Rede Neural Probabilística (PNN)

2.4 Redes do Tipo Perceptron de Múltiplas Camadas (*MLP*)

As arquiteturas do tipo perceptron de múltiplas camadas (MLP) constituem os modelos neurais artificiais mais utilizados e conhecidos.

Tipicamente, esta arquitetura consiste de um conjunto de unidades sensoriais que formam uma camada de entrada, uma ou mais camadas intermediárias (ou escondidas) de unidades computacionais e uma camada de saída. Os sinais de entrada são propagados camada a camada pela rede em uma direção positiva, ou seja, da entrada para a saída. Esta arquitetura representa uma generalização do perceptron apresentado anteriormente.

As redes do tipo MLP tem sido utilizadas com sucesso para a solução de vários problemas envolvendo altos graus de não-linearidade. Seu treinamento é do tipo supervisionado e utiliza um algoritmo muito popular chamado retro-propagação do erro (*error backpropagation*). Este algoritmo é baseado numa regra de aprendizagem que "corrige" o erro durante o treinamento (HAYKIN, 1994).

Basicamente, o processo de retro-propagação do erro é constituído de duas fases: uma fase de propagação do sinal funcional (*feedforward*) e uma de retro-propagação do erro (*backpropagation*). Na fase positiva, os vetores de dados são aplicados às unidades de entrada, e seu efeito se propaga pela rede, camada a camada. Finalmente, um conjunto de saídas é produzido como resposta da rede. Durante a fase positiva, os pesos das conexões são mantidos fixos. Na retro-propagação do erro, por outro lado, os pesos são ajustados de acordo com uma regra de correção do erro. Especificamente, a resposta da rede em um instante de tempo é subtraída da saída desejada (*target*) para produzir um *sinal de erro*. Este sinal de erro é propagado da saída para a entrada, camada a camada, originando o nome "retro-propagação do erro". Os pesos são ajustados de forma que a distância entre a resposta da rede e a resposta desejada seja reduzida.

Uma rede do tipo MLP possui três características distintas:

- função de ativação;
- número de camadas e unidades intermediárias e
- forma das conexões.

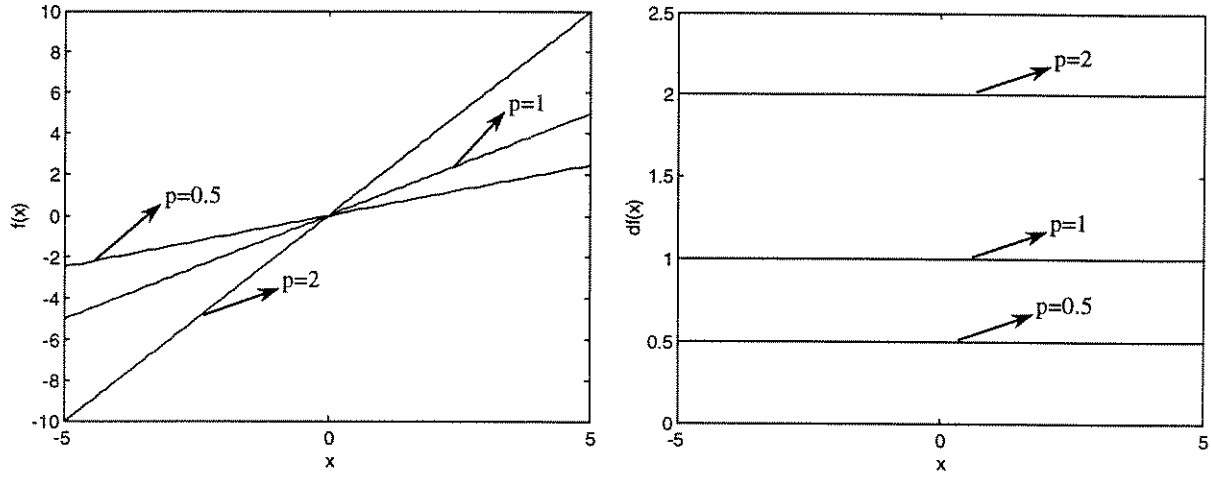
2.4.1 Função de Ativação

O modelo de cada unidade da rede pode incluir uma *não-linearidade* na sua saída. É importante enfatizar que a não-linearidade deve ser suave, ou seja, diferenciável, diferentemente da função sinal utilizada pelo perceptron original.

A função de ativação representa o efeito que a entrada interna e o estado atual de ativação exercem na definição do próximo estado de ativação da unidade. Quando propriedades dinâmicas estão envolvidas na definição do estado de ativação, equações diferenciais (caso contínuo) ou a diferenças (caso discreto) são empregadas. Tendo em vista a simplicidade desejada para as unidades processadoras, geralmente define-se seu estado de ativação como uma função algébrica da entrada interna atual, independente de valores passados do estado de ativação ou mesmo da entrada interna. Geralmente, esta função é monotonicamente não-decrescente e apresenta um tipo de não-linearidade associada ao efeito da saturação. A seguir são descritos alguns tipos de função de ativação empregados na literatura para as arquiteturas do tipo MLP (KOSKO, 1992; HAYKIN, 1994).

Função Linear

Este tipo de função de ativação é muito utilizado nas unidades que compõem a camada de saída das arquiteturas MLP.



(a) (b)

Figura 2.5: (a) Função linear. (b) Sua derivada em relação a entrada interna.

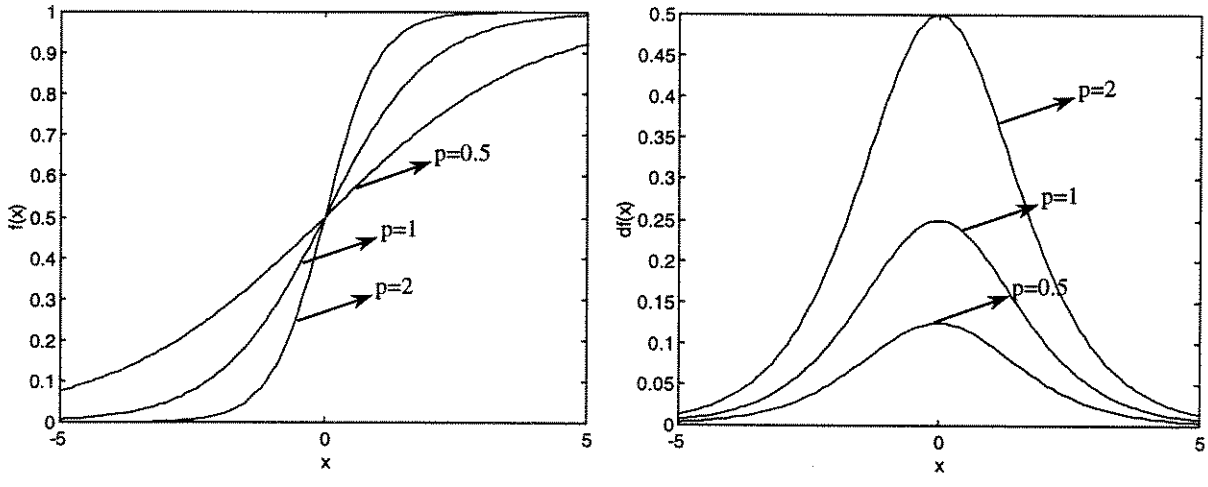
A saída linear com $p = 1$ simplesmente repete o sinal que entra no neurônio na sua saída. A Figura 2.5 apresenta saídas lineares e suas derivadas.

A expressão para esta função de ativação e sua derivada é:

$$f(x) = p.x, \qquad f'(x) = p$$

Função Logística

A Figura 2.6 (a) mostra que a função logística possui intervalo de variação entre 0 e 1.



(a) (b)

Figura 2.6: (a) Função logística. (b) Sua derivada em relação a entrada interna.

A origem deste tipo de função está vinculada à preocupação em limitar o intervalo de variação da derivada da função, pela inclusão de um efeito de saturação. Sua derivada também é uma função contínua (ver Figura 2.6).

$$f(x) = \frac{e^{px}}{1+e^{px}} = \frac{1}{1+e^{-px}}, \qquad f'(x) = p\,f(x).\left(1-f(x)\right)$$

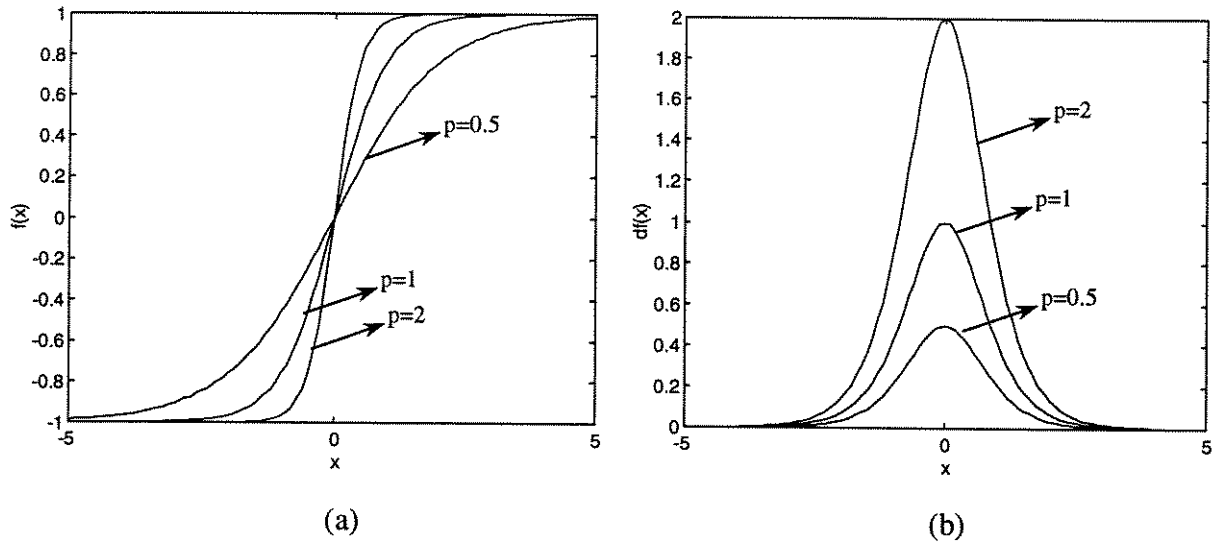
Função Tangente Hiperbólica

Pelo fato da função logística apresentar valores de ativação apenas no intervalo (0, 1), em muitos casos ela é substituída pela função tangente hiperbólica, que preserva a forma sigmoidal da função logística, mas assume valores positivos e negativos ($f(x) \in (-1, 1)$). A função tangente hiperbólica e sua derivada (ver Figura 2.7) são dadas pelas expressões:

$$f(x) = \frac{e^{px} - e^{-px}}{e^{px} + e^{-px}} = \tanh(px), \qquad f'(x) = p\left(1-f(x)^2\right)$$

Em todas as redes implementadas e testadas neste trabalho utilizou-se função de ativação linear na saída da rede. A função tangente hiperbólica foi a escolhida para as unidades intermediárias das redes do tipo MLP fixas. No Capítulo 7, são feitos estudos sobre modelos construtivos de redes, onde alguns destes modelos utilizam funções de ativação diferentes da tangente hiperbólica na camada intermediária. Estes modelos serão especificados nas seções correspondentes à sua descrição.

A função tangente hiperbólica pode ser transladada para o intervalo (0, 1), assim como a função logística pode ser transladada para o intervalo (-1, 1).



(a) (b)

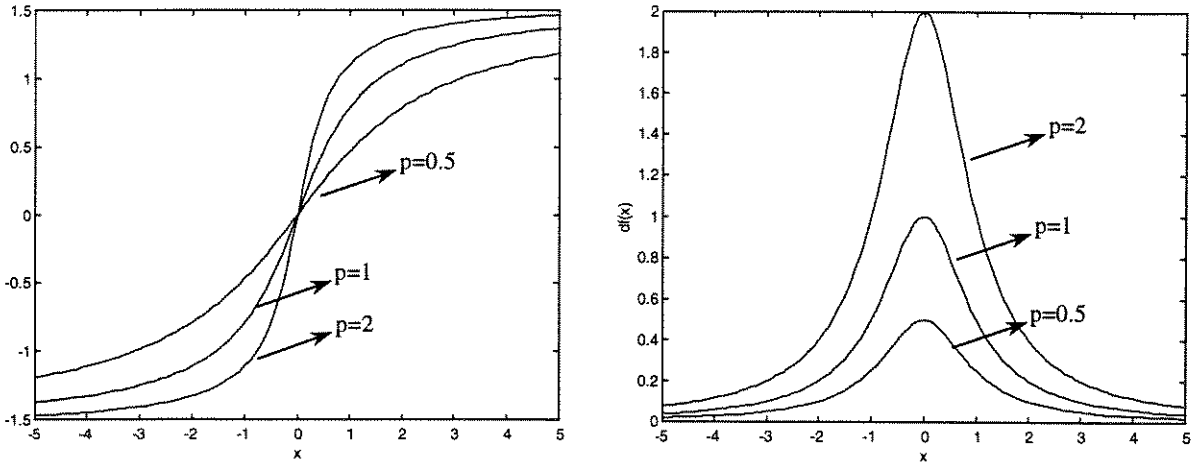
Figura 2.7: (a) Função tangente hiperbólica. (b) Sua derivada em relação a entrada interna.

Função Arco-Tangente

Esta função possui valores de ativação no intervalo $(-\pi/2, \pi/2)$, e pode ser apresentada como uma alternativa à função tangente hiperbólica para a implementação computacional, pois requer menos cálculos para sua elaboração. Comparações sucintas entre o tempo de processamento das funções tanh(.), atan(.), e logística mostram que a função atan(.) possui o menor tempo de processamento, seguida da função logística e por último a função tangente hiperbólica.

A função arco tangente é dada pela expressão abaixo:

$$f(x) = \text{atan}(px) \qquad f'(x) = p.\frac{1}{1+x^2}$$



(a) (b)

Figura 2.8: (a) Função arco-tangente (atan). (b) Sua derivada em relação a entrada interna.

Os limites dos intervalos da função apresentada acima podem ser transladados para o intervalo (-1,1) comumente utilizado.

2.4.2 Noções Gerais

A Figura 2.9 apresenta uma arquitetura do tipo MLP com duas camadas intermediárias. A rede apresentada aqui possui todas as conexões, o que significa que um neurônio em qualquer camada da rede está conectado a todas as outras unidades (neurônios) na camada anterior. O fluxo de sinais através da rede é feito positivamente, da esquerda para a direita, camada a camada.

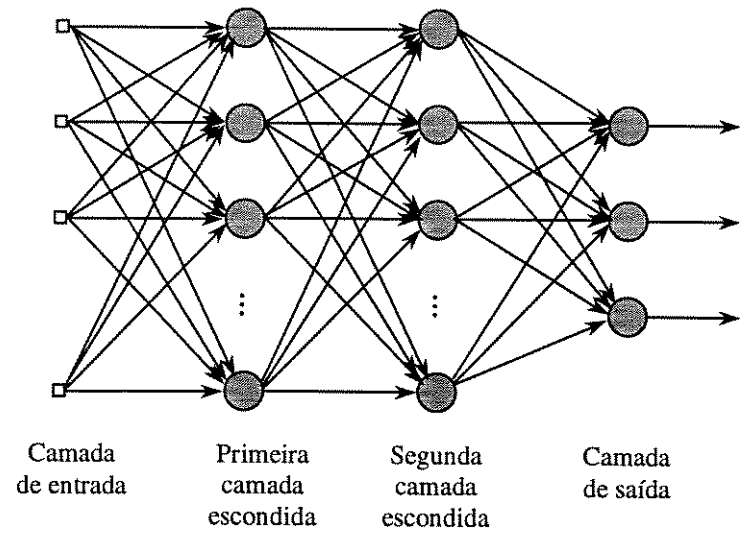


Figure 2.9: Arquitetura MLP com duas camadas intermediárias.

A Figura 2.10 mostra apenas uma parte da rede. Nesta rede, dois tipos de sinais podem ser identificados:

- *sinal funcional*: um sinal funcional é um sinal de entrada (estímulo) que chega na entrada e é propagado positivamente (neurônio a neurônio) através da rede, e aparece na saída como um sinal de saída.
- *sinal de erro*: os sinais de erro originam-se nas saídas e são retro-propagados (neurônio a neurônio) através da rede.

A camada de entrada geralmente é composta por neurônios sensoriais, ou seja, unidades que não modificam os sinais externos, apenas os distribuem para a primeira camada intermediária. As unidades de saída constituem a camada de saída da rede, e as demais unidades constituem as camadas intermediárias. As camadas intermediárias são todas aquelas que não fazem parte da entrada e nem da saída.

Cada unidade intermediária ou de saída é responsável por duas tarefas:

- calcular o sinal na saída da unidade, que geralmente é expresso como uma função não-linear do sinal de entrada e pesos sinápticos associados e
- calcular uma estimativa instantânea do vetor gradiente, que é necessário para a retro-propagação do erro através da rede.

Para facilitar a derivação da regra de retro-propagação, primeiramente apresentamos um resumo da notação utilizada.

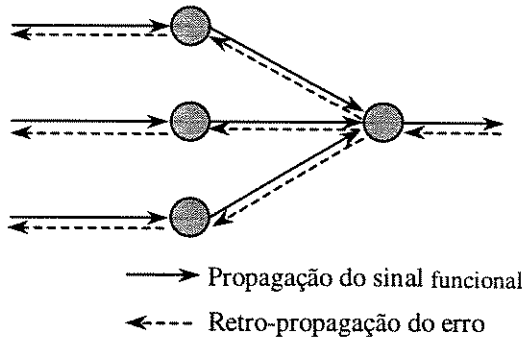


Figura 2.10: Ilustração das direções de propagação do sinal funcional e do erro.

Notação

i, j	índices referentes a diferentes neurônios da rede
n	n-ésimo vetor de entrada (iteração)
N	número de amostras (padrões de treinamento)
M	número de camadas
$y_j(n)$	sinal de saída da unidade j na iteração n
$e_j(n)$	sinal de erro da unidade de saída j na iteração n
$w_{i,j}(n)$	peso sináptico conectando a saída da unidade i à entrada da unidade j na iteração n
$u_j(n)$	ativação da unidade j na iteração n; sinal a ser aplicado à não-linearidade
$f_j(.)$	função de ativação associada à unidade j
$\mathbf{X}$	matriz de dados de entrada (amostras de treinamento)
$\mathbf{S}$	matriz de dados de saída (saídas desejadas)
$x_i(n)$	i-ésimo elemento do vetor de entradas
$s_j(n)$	j-ésimo elemento do vetor de saídas
α	taxa de aprendizagem

Todas as letras minúsculas em **negrito** (**a**, **b**, **c**) representam vetores, as letras maiúsculas em **negrito** (**A**, **B**, **C**) denotam matrizes e as letras em *itálico* (*a*, *b*, *c*) representam escalares.

As matrizes $\mathbf{W}^m$ (para $m = 0, 1, \ldots, M - 1$; onde M é o número de camadas da rede) possuem dimensão $S^{m+1} \times S^m$, onde S^0 = número de entradas da rede; e os vetores $\mathbf{b}^m$ possuem dimensão $S^{m+1} \times 1$.

2.4.3 Derivação do Algoritmo de Retro-Propagação (*Backpropagation*)

O algoritmo a ser apresentado representa uma forma eficiente e elegante de implementação computacional da regra da cadeia, principalmente quando a ferramenta utilizada possui capacidade de processamento matricial, como os pacotes Matlab® e Mathematica®.

Para simplificar o desenvolvimento do algoritmo de retro-propagação, utilizaremos a notação abreviada para uma arquitetura genérica com três camadas (ver Figura 2.11) (DEMUTH & BEALE, 1993; HAGAN *et. al.*, 1997).

Para redes de múltiplas camadas, a saída de uma camada torna-se a entrada da camada seguinte. A equação que descreve esta operação é:

$$\mathbf{u}^{m+1} = \mathbf{f}^{m+1}(\mathbf{W}^{m+1}\mathbf{y}^m + \mathbf{b}^{m+1}), \text{ para } m = 0, 1, \ldots, M-1, \tag{2.1}$$

onde M é o número de camadas da rede. As unidades na primeira camada recebem entradas externas agrupadas em um vetor na forma:

$$\mathbf{y}^0 = \mathbf{x}, \tag{2.2}$$

que representa a condição inicial para a equação (2.1). As saídas das unidades localizadas na última camada são consideradas as saídas da rede:

$$\mathbf{y} = \mathbf{y}^M. \tag{2.3}$$

Pela Figura 2.11 percebemos que a saída da rede pode ser expressa apenas em função do vetor de entradas $\mathbf{x}$, das matrizes de pesos $\mathbf{W}^m$ e dos vetores de limiares $\mathbf{b}^m$, cuja expressão é:

$$\mathbf{y}^3 = \mathbf{f}^3(\mathbf{W}^3\mathbf{f}^2(\mathbf{W}^2\mathbf{f}^1(\mathbf{W}^1\mathbf{x} + \mathbf{b}^1)+\mathbf{b}^2)+\mathbf{b}^3). \tag{2.4}$$

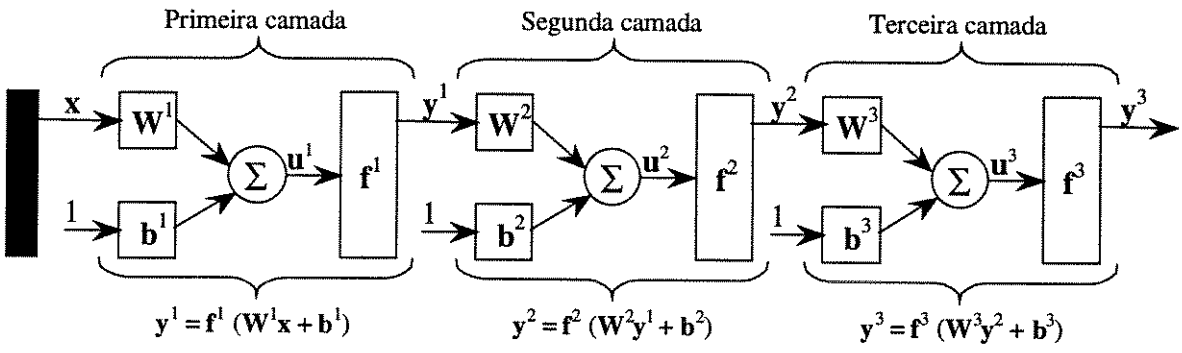


Figura 2.11: Rede com três camadas. Notação abreviada.

2.4.3.1 Índice de Desempenho

O algoritmo de retro-propagação para as redes de múltiplas camadas é uma generalização do *método dos quadrados mínimos* (LS – do inglês *Least Squares*) e utiliza como medida de desempenho o *erro quadrático médio* (MSE – do inglês *mean squared error*). Inicialmente, é apresentado um conjunto de exemplos:

$$\{(\mathbf{x}_1, \mathbf{s}_1), (\mathbf{x}_2, \mathbf{s}_2), \ldots, (\mathbf{x}_N, \mathbf{s}_N)\}, \tag{2.5}$$

onde $\mathbf{x}_n$ é a n-ésima entrada para a rede e $\mathbf{s}_n$ a saída desejada correspondente (n = 1, ..., N). Das equações (2.3) e (2.4) vemos que, se θ é o vetor de parâmetros da rede (pesos e limiares), então o vetor de saídas da rede pode ser dado na forma:

$$y^M = y(\theta, x)$$

Após cada entrada ser aplicada à rede, a saída produzida pela rede é comparada com a saída desejada. O algoritmo deve ajustar os parâmetros da rede (pesos e limiares), com o objetivo de minimizar a esperança matemática do erro quadrático médio:

$$J(\theta) = E(e(\theta)^2) = E((s - y(\theta))^2), \tag{2.6}$$

Se a rede possui múltiplas saídas, a equação (2.6) pode ser generalizada para

$$J(\theta) = E(\mathbf{e}(\theta)^T\mathbf{e}(\theta)) = E((\mathbf{s} - \mathbf{y}(\theta))^T (\mathbf{s} - \mathbf{y}(\theta))). \tag{2.7}$$

Como no algoritmo LS, o erro quadrático médio será aproximado por

$$\hat{J}(\theta) = \mathbf{e}(n)^T\mathbf{e}(n) = (\mathbf{s}(n) - \mathbf{y}(n))^T (\mathbf{s}(n) - \mathbf{y}(n)), \tag{2.8}$$

onde a esperança do erro quadrático foi substituída pelo erro na iteração n. Para não sobrecarregar a notação empregada, consideraremos $\hat{J}(\theta) = J(\theta)$.

A lei de ajuste conhecida como *steepest descent* para minimizar o erro quadrático é dada por:

$$w_{i,j}^m(n+1) = w_{i,j}^m(n) - \alpha\frac{\partial J(n)}{\partial w_{i,j}^m}, \tag{2.9}$$

$$b_i^m(n+1) = b_i^m(n) - \alpha\frac{\partial J(n)}{\partial b_i^m}, \tag{2.10}$$

onde α é a taxa de aprendizagem.

A parte mais elaborada deste cálculo está na determinação das derivadas parciais que irão gerar os componentes do vetor gradiente.

2.4.3.2 Regra da Cadeia

Para uma rede com múltiplas camadas o erro não é função direta dos pesos nas camadas intermediárias, por isso o cálculo destas derivadas não é imediato.

Como o erro é função indireta dos pesos nas camadas intermediárias, a regra da cadeia deverá ser usada para o cálculo das derivadas. O conceito da regra da cadeia será utilizado na determinação das derivadas das equações (2.9) e (2.10):

$$\frac{\partial J}{\partial w_{i,j}^m} = \frac{\partial J}{\partial u_i^m} \times \frac{\partial u_i^m}{\partial w_{i,j}^m}, \tag{2.11}$$

$$\frac{\partial J}{\partial b_i^m} = \frac{\partial J}{\partial u_i^m} \times \frac{\partial u_i^m}{\partial b_i^m}. \tag{2.12}$$

O segundo termo de cada uma das equações acima pode ser facilmente calculado, uma vez que a ativação da camada m é uma função explícita dos pesos e limiares nesta camada:

$$u_i^m = \sum_{j=1}^{S^{m-1}} w_{i,j}^m y_j^{m-1} + b_i^m. \tag{2.13}$$

Entretanto

$$\frac{\partial u_i^m}{\partial w_{i,j}^m} = y_j^{m-1}, \frac{\partial u_i^m}{\partial b_i^m} = 1. \tag{2.14}$$

Definindo agora a *sensibilidade* de J a mudanças no i-ésimo elemento da ativação da rede na camada m como:

$$\delta_i^m \equiv \frac{\partial J}{\partial u_i^m}, \tag{2.15}$$

então as equações (2.11) e (2.12) podem ser simplificadas por

$$\frac{\partial J}{\partial w_{i,j}^m} = \delta_i^m y_j^{m-1}, \tag{2.16}$$

$$\frac{\partial J}{\partial b_i^m} = \delta_i^m. \tag{2.17}$$

Agora é possível aproximar as equações (2.9) e (2.10) por

$$w_{i,j}^m(n+1) = w_{i,j}^m(n) - \alpha \delta_i^m y_j^{m-1}, \tag{2.18}$$

$$b_i^m(n+1) = b_i^m(n) - \alpha \delta_i^m. \tag{2.19}$$

Em notação matricial, as equações acima tornam-se:

$$\mathbf{W}^m(n+1) = \mathbf{W}^m(n) - \alpha \boldsymbol{\delta}^m (\mathbf{y}^{m-1})^T, \tag{2.20}$$

$$\mathbf{b}^m(n+1) = \mathbf{b}^m(n) - \alpha\boldsymbol{\delta}^m, \tag{2.21}$$

onde

$$\boldsymbol{\delta}^m \equiv \frac{\partial J}{\partial \mathbf{u}^m} = \begin{bmatrix} \dfrac{\partial J}{\partial u_1^m} \\ \dfrac{\partial J}{\partial u_2^m} \\ \vdots \\ \dfrac{\partial J}{\partial u_{S^m}^m} \end{bmatrix}. \tag{2.22}$$

2.4.3.3 Retro-propagação das Sensibilidades

Ainda é necessário calcular as sensibilidades δ^m, que requer outra aplicação da regra da cadeia. É este processo que dá origem ao termo *retro-propagação*, pois descreve a relação de recorrência na qual a sensibilidade na camada m é calculada a partir da sensibilidade na camada $m + 1$.

Para derivar a relação de recorrência das sensibilidades, utilizaremos a seguinte matriz jacobiana:

$$\frac{\partial \mathbf{u}^{m+1}}{\partial \mathbf{u}^m} = \begin{bmatrix} \dfrac{\partial u_1^{m+1}}{\partial u_1^m} & \dfrac{\partial u_1^{m+1}}{\partial u_2^m} & \cdots & \dfrac{\partial u_1^{m+1}}{\partial u_{S^m}^m} \\ \dfrac{\partial u_2^{m+1}}{\partial u_1^m} & \dfrac{\partial u_2^{m+1}}{\partial u_2^m} & \cdots & \dfrac{\partial u_2^{m+1}}{\partial u_{S^m}^m} \\ \vdots & \vdots & \ddots & \vdots \\ \dfrac{\partial u_{S^{m+1}}^{m+1}}{\partial u_1^m} & \dfrac{\partial u_{S^{m+1}}^{m+1}}{\partial u_2^m} & \cdots & \dfrac{\partial u_{S^{m+1}}^{m+1}}{\partial u_{S^m}^m} \end{bmatrix}. \tag{2.23}$$

Em seguida desejamos encontrar uma expressão para esta matriz. Considere o elemento i, j da matriz:

$$\frac{\partial u_i^{m+1}}{\partial u_j^m} = w_{i,j}^{m+1} \frac{\partial y_j^m}{\partial u_j^m} = w_{i,j}^{m+1} \frac{\partial f^m(u_j^m)}{\partial u_j^m} = w_{i,j}^{m+1} \dot{f}^m(u_j^m), \tag{2.24}$$

onde

$$\dot{f}^m(u_j^m) = \frac{\partial f^m(u_j^m)}{\partial u_j^m}. \tag{2.25}$$

Entretanto a matriz jacobiana pode ser escrita

$$\frac{\partial \mathbf{u}^{m+1}}{\partial \mathbf{u}^m} = \mathbf{W}^{m+1}\dot{\mathbf{F}}^m(\mathbf{u}^m), \tag{2.26}$$

onde

$$\dot{\mathbf{F}}^m(\mathbf{u}^m) = \begin{bmatrix} \dot{f}^m(u_1^m) & 0 & \cdots & 0 \\ 0 & \dot{f}^m(u_2^m) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \dot{f}^m(u_{S^m}^m) \end{bmatrix}. \tag{2.27}$$

Agora podemos escrever a relação de recorrência para a sensibilidade utilizando a regra da cadeia em forma matricial:

$$\begin{aligned} \boldsymbol{\delta}^m &= \frac{\partial J}{\partial \mathbf{u}^m} = \left(\frac{\partial \mathbf{u}^{m+1}}{\partial \mathbf{u}^m}\right)^T \frac{\partial J}{\partial \mathbf{u}^{m+1}} = \dot{\mathbf{F}}^m(\mathbf{u}^m)(\mathbf{W}^{m+1})^T \frac{\partial J}{\partial \mathbf{u}^{m+1}} \\ &= \dot{\mathbf{F}}^m(\mathbf{u}^m)(\mathbf{W}^{m+1})^T \boldsymbol{\delta}^{m+1}. \end{aligned} \tag{2.28}$$

Observe que as sensibilidades são propagadas da última para a primeira camada através da rede:

$$\delta^M \rightarrow \delta^{M-1} \rightarrow \ldots \rightarrow \delta^2 \rightarrow \delta^1 \tag{2.29}$$

Ainda existe um último passo a ser executado para que o algoritmo de retro-propagação fique completo. Precisamos do ponto de partida, δ^M, para a relação de recorrência da equação (2.28). Este ponto é obtido na última camada:

$$\delta_i^M = \frac{\partial J}{\partial u_i^M} = \frac{\partial (\mathbf{s}-\mathbf{y})^T(\mathbf{s}-\mathbf{y})}{\partial u_i^M} = \frac{\partial \sum_{j=1}^{S^M}(s_j - y_j)^2}{\partial u_i^M} = -2(s_i - y_i)\frac{\partial y_i}{\partial u_i^M}. \tag{2.30}$$

Como

$$\frac{\partial y_i}{\partial u_i^M} = \frac{\partial y_i^M}{\partial u_i^M} = \frac{\partial f^M(u_j^M)}{\partial u_i^M} = \dot{f}^M(u_j^M), \tag{2.31}$$

podemos escrever

$$\delta_i^M = -2(s_i - y_i)\dot{f}^M(u_j^M). \tag{2.32}$$

A expressão acima pode ser colocada em forma matricial, resultando

$$\boldsymbol{\delta}^M = -2\dot{\mathbf{F}}^M(\mathbf{u}^M)(\mathbf{s}-\mathbf{y}). \tag{2.33}$$

A Figura 2.12 mostra uma adaptação (utilizando notação matricial abreviada) do resultado de NARENDRA & PARTHASARATHY (1990) e descreve um esquema gráfico de fluxo de sinais para o algoritmo de treinamento de retro-propagação, sendo bastante útil para descrever todo o equacionamento do algoritmo de retro-propagação.

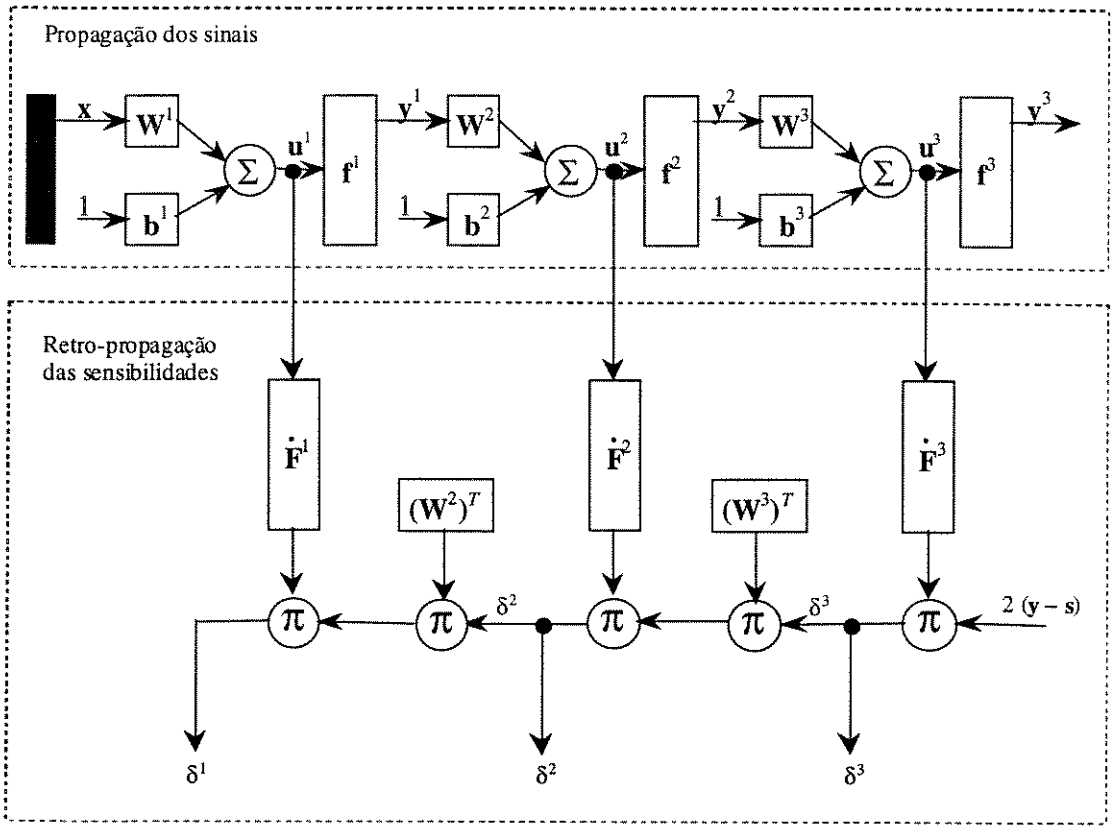


Figura 2.12: Gráfico arquitetural de uma rede com três camadas, representando a fase de propagação dos sinais e retro-propagação das sensibilidades.

2.4.4 Treinamento Local ou em Lote

Em aplicações práticas do algoritmo de retro-propagação, o aprendizado é resultado de apresentações repetidas de todas as amostras do conjunto de treinamento ao MLP. Cada apresentação de todo o conjunto de treinamento durante o processo de aprendizagem é chamada de *época*. O processo de aprendizagem é repetido época após época, até que o conjunto de pesos e limiares estabilize e o erro quadrático médio do conjunto de treinamento convirja para um valor mínimo. É uma boa prática fazer com que a ordem de apresentação das amostras seja feita aleatoriamente de uma época para a outra. Esta aleatoriedade tende a fazer com que a busca no espaço de pesos tenha um caráter estocástico ao longo dos ciclos de treinamento.

Para um dado conjunto de treinamento, o aprendizado por retro-propagação pode ser feito de duas maneiras básicas:

- **atualização local**: neste método, a atualização dos pesos é feita imediatamente após a apresentação de cada amostra de treinamento. Especificamente, considere uma época consistindo de N padrões de treinamento arbitrariamente ordenados

$\{(\mathbf{x}_1,\mathbf{s}_1),...,(\mathbf{x}_N,\mathbf{s}_N)\}$. A primeira amostra $(\mathbf{x}_1,\mathbf{s}_1)$ da época é apresentada à rede, e são efetuados o passo positivo e a retro-propagação do erro, resultando em um ajuste dos pesos das conexões. Em seguida, a próxima amostra $(\mathbf{x}_2,\mathbf{s}_2)$ é apresentada à rede e o passo positivo e a retro-propagação do erro são efetuados, resultando em mais um ajuste do conjunto de pesos. Este processo é repetido até que a última amostra seja apresentada à rede. Este método também é conhecido como método de atualização *on-line* ou *padrão a padrão.*

- **atualização em lote**: no método em lote, a atualização dos pesos só é feita após a apresentação de todas as amostras de treinamento que constituem uma época. O ajuste relativo a cada apresentação de um par de entrada é acumulado. Este método também é conhecido como método de atualização *off-line* ou *batch.*

Do ponto de vista operacional, o método local de treinamento é preferível, pois requer um menor armazenamento local para cada conexão. Como as amostras são apresentadas aleatoriamente, o uso da atualização padrão por padrão torna a busca no espaço de conexões estocástica por natureza, reduzindo a possibilidade do algoritmo ficar preso em um mínimo local. Por outro lado, a utilização do método em lote fornece uma estimativa mais precisa do vetor gradiente. Numa análise final, entretanto, a eficiência dos dois métodos depende do problema a ser tratado (HAYKIN, 1994). Neste trabalho, todas as redes implementadas utilizam o método de atualização em lote, salvo especificações em contrário.

2.4.5 Critérios de Parada

O processo de minimização do MSE (ou da função custo), em geral, não tem convergência garantida e não possui um critério de parada bem definido. Um critério de parada não recomendável, por não levar em conta o estado do processo iterativo de treinamento, é interromper o treinamento após um número fixo (definido previamente) de iterações. Serão discutidos aqui critérios de parada que levam em conta alguma informação a respeito do estado do processo iterativo, cada qual com seu mérito prático. Para formular tal critério, deve-se considerar a possibilidade de existência de mínimos locais. Seja θ^* o vetor de parâmetros (pesos) que denota um ponto de mínimo, seja ele local ou global. Uma condição necessária para que θ^* seja um mínimo, é que o vetor gradiente $\nabla J(\theta)$ (ou seja, a derivada parcial de primeira ordem) da superfície de erro em relação ao vetor de pesos θ seja zero em $\theta = \theta^*$. Sendo assim, é possível formular critérios de convergência (ou parada) como segue:

- *é considerado que o algoritmo de retro-propagação convergiu quando a norma Euclideana da estimativa do vetor gradiente* ($\|\nabla J(\theta)\|$) *atingiu um valor suficientemente pequeno.*

O problema deste critério de parada é que, para simulações bem sucedidas, o tempo de treinamento pode ser muito longo. Este critério também requer o cálculo da norma do vetor gradiente. O Capítulo 3 fará um estudo sobre técnicas de aceleração da convergência deste algoritmo.

Outra propriedade única de um mínimo, e que pode ser utilizada como critério de parada, é o fato de que a função custo, ou medida de erro, $J_{med}(\theta)$ é estacionária no ponto $\theta = \theta^*$. Assim, outro critério de parada pode ser sugerido:

- *é considerado que o algoritmo de retro-propagação convergiu quando a variação do erro quadrático de uma época para a outra atingir um valor suficientemente pequeno.*

Uma variação deste critério de parada é fazer com que o valor do erro quadrático médio $J_{med}(\theta)$ seja igual ou menor do que um limiar pré-especificado:

- *é considerado que o algoritmo de retro-propagação convergiu quando o erro quadrático médio atingir um valor suficientemente pequeno, ou seja,* $J_{med}(\theta) \leq \varepsilon$, *onde* ε *é um valor suficientemente pequeno.*

Se o critério de parada, por exemplo, é um valor mínimo para o MSE então não podemos garantir que o algoritmo será capaz de atingir o valor desejado. Por outro lado, ao tomarmos como critério de parada um valor mínimo para a norma do vetor gradiente, então devemos estar conscientes de que o algoritmo, provavelmente, irá produzir como resultado o mínimo local mais próximo da condição inicial. A Figura 2.13 ilustra uma superfície de erro, e apresenta o possível comportamento do algoritmo. Neste exemplo, se o critério de parada fosse $J_{med}(\theta) \leq \varepsilon$, então o método não seria capaz de encontrá-lo, pois o valor mínimo da superfície de erro é maior do que o valor de ε desejado (linha tracejada).

Atualmente existe uma grande quantidade de pesquisadores procurando funções de erro (custo) alternativas, com o objetivo de melhorar as características de convergência dos perceptrons de múltiplas camadas. Uma abordagem possível é adotar um funcional de erro que aumente a capacidade da rede escapar de mínimos locais (comentários sobre a superfície de erro e mínimos locais serão feitos posteriormente).

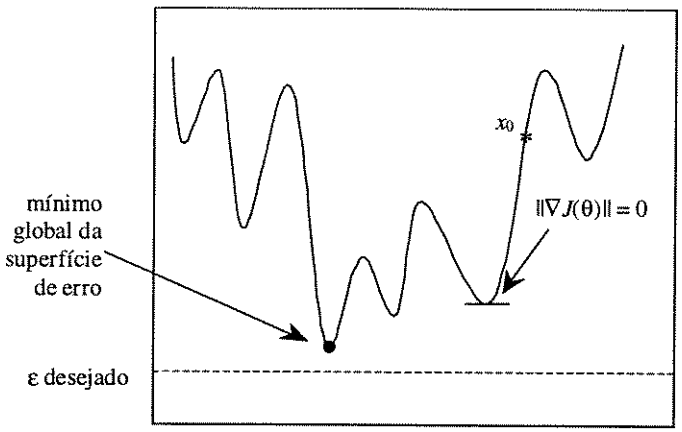


Figura 2.13: Superfície de erro com vários mínimos locais. Todos os vales (joelhos) da curva constituem mínimos locais com $\|\nabla J(\theta)\| = 0$. Partindo do ponto x_0, e utilizando como critério de parada $J_{med}(\theta) \leq \varepsilon$, o algoritmo não será capaz de determinar um conjunto de pesos capaz de satisfazê-lo independente dos mínimos locais.

Um exemplo disso é a função de custo chamada de *entropia cruzada* (*cross-entropy*) (SHEPHERD, 1997), dada por:

$$J = -\sum_{n=1}^{N}\sum_{i=1}^{S^m} \ln\left[\left(y_{i,n}^{M}\right)^{s_{i,n}}\left(1-y_{i,n}^{M}\right)^{1-s_{i,n}}\right], \tag{2.34}$$

onde a simbologia utilizada aqui é a mesma adotada na Seção 2.4, ou seja, N é o número de amsotras, M é o número de camadas, S^m é a dimensão de cada camada e y são as saídas da rede.

Métodos híbridos que combinam diferentes critérios de parada podem ser utilizados.

Outro critério de parada bastante útil, e geralmente utilizado em conjunto com algum dos critérios anteriores, é a avaliação da capacidade de generalização da rede após cada época de treinamento. O processo de treinamento é interrompido antes que a capacidade de generalização da rede seja deteriorada. Maiores detalhes sobre generalização e procedimentos de parada utilizando critérios que consideram o desempenho da rede após o treinamento serão estudados no Capítulo 6.

Neste trabalho, utilizou-se como critério de parada o último apresentado, ou seja, $J_{med}(\theta) \leq \varepsilon$, salvo especificação em contrário.

2.4.6 A Capacidade de Aproximação Universal

Uma arquitetura do tipo MLP pode ser vista como uma ferramenta prática geral para fazer um *mapeamento não-linear de entrada-saída*. Especificamente, seja k o número de entradas da rede e m o número de saídas. A relação entrada-saída da rede define um mapeamento de um espaço euclidiano de entrada k-dimensional para um espaço euclidiano de saída m-dimensional, que é infinitamente continuamente diferenciável.

CYBENKO (1989) foi o primeiro pesquisador a demonstrar rigorosamente que uma rede MLP com uma única camada intermediária é suficiente para aproximar uniformemente qualquer função contínua que encaixe em um hipercubo unitário.

O *teorema da aproximação universal* aplicável a redes MLP é descrito abaixo:

Teorema: *Seja $f(.)$ uma função contínua não-constante, limitada, e monotonicamente crescente. Seja $\boldsymbol{I}_p$ um hipercubo unitário p-dimensional $(0,1)^p$. O espaço das funções contínuas em $\boldsymbol{I}_p$ é denominado $\boldsymbol{C}(\boldsymbol{I}_p)$. Então, dada qualquer função $g \in \boldsymbol{C}(\boldsymbol{I}_p)$ e $\varepsilon > 0$, existe um inteiro M e conjuntos de constantes reais α_i e w_{ij}, onde $i = 1, \ldots, M$ e $j = 1, \ldots, p$, tais que pode-se definir*

$$F(x_1, x_2, \ldots, x_p) = \sum_{i=1}^{M} \alpha_i f\left(\sum_{j=1}^{p} w_{ij} x_j - w_{0i} \right), \tag{2.35}$$

como uma aproximação da função $g(.)$ tal que,

$$\left| F(x_1, x_2, \ldots, x_p) - g(x_1, x_2, \ldots, x_p) \right| < \varepsilon$$

Para todo $\{x_1, \ldots, x_p\} \in \boldsymbol{I}_p$.

Prova: veja CYBENKO (1989).

Este teorema é diretamente aplicável aos perceptrons de múltiplas camadas. Primeiramente percebemos que a função logística, ou tangente hiperbólica, utilizada como a não-linearidade de um neurônio é contínua, não-constante, limitada, e monotonicamente crescente; satisfazendo as condições impostas à função $f(.)$. Em seguida, verificamos que a equação (2.35) representa as saídas de uma rede MLP descrita como segue:

- a rede possui p nós de entrada e uma única camada intermediária consistindo de M unidades; o conjunto de entradas é $\{x_1, \ldots, x_p\}$
- o neurônio intermediário i possui pesos $w_{i1}, \ldots, w_{ip}$ e limiar w_{01}
- a saída da rede é uma combinação linear das saídas das unidades intermediárias, com $\alpha_1, \ldots, \alpha_M$ definindo os coeficientes dessa combinação.

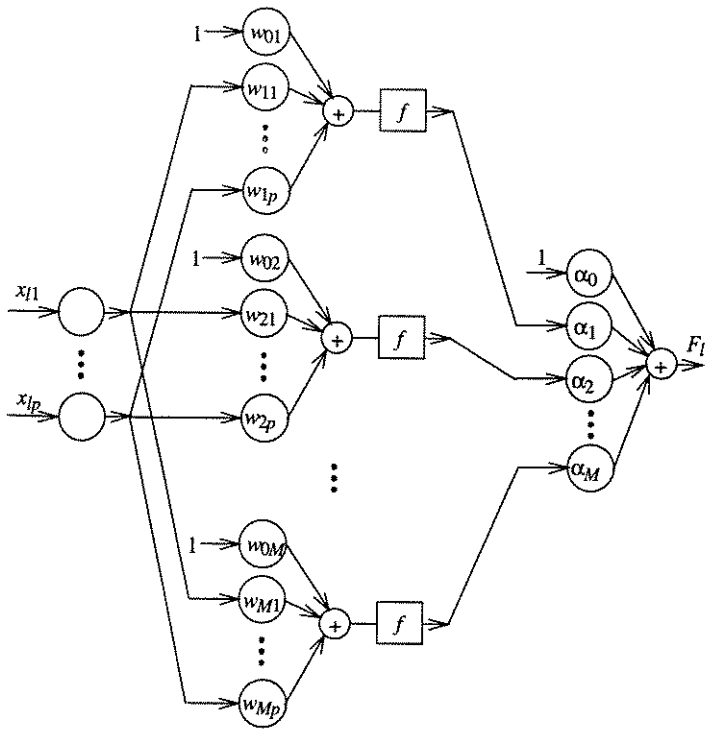
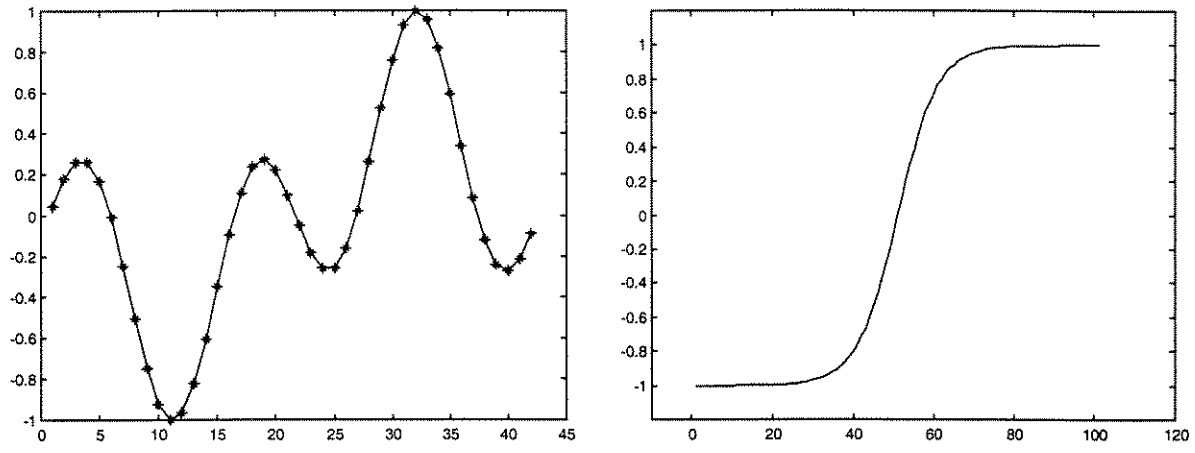


Figura 2.14: Rede MLP como aproximador universal. Os parâmetros da rede compõem a equação (2.35).

O teorema de aproximação universal é um *teorema de existência* no sentido de que fornece uma justificativa matemática para a aproximação de uma função contínua arbitrária, em oposição à representação exata. A equação (2.35) simplesmente generaliza aproximações por uma série de Fourier finita. O teorema afirma que *um perceptron de múltiplas camadas com uma única camada intermediária é capaz de realizar uma aproximação uniforme, dado um conjunto de treinamento suficientemente significativo para representar a função.* Por outro lado, o teorema não afirma que um MLP com uma única camada é ótimo no sentido de tempo de processamento, facilidade de implementação e eficiência na representação.

Exemplo 2.1:

Para ilustrar a capacidade de aproximação universal das redes do tipo MLP, considere o seguinte exemplo: desejamos aproximar um período da função sen(x)×cos($2x$) utilizando uma composição aditiva de funções básicas (f será tomado como a tangente hiperbólica) sob a forma da equação (2.35). A Figura 2.15 ilustra o problema.



(a) (b)

Figura 2.15: Aproximação universal. (a) Função a ser aproximada (informação disponível: 42 amostras do valor da função em pontos igualmente espaçados do domínio da função). (b) Função básica utilizada na aproximação. Tangentes hiperbólicas, satisfazendo as condições estabelecidas pelo teorema.

A rede utilizada para aproximar esta função é uma rede do tipo MLP com uma entrada, cinco unidades intermediárias e uma saída, descrita na Figura 2.16. O algoritmo de treinamento é o de retro-propagação, apresentado anteriormente. Analisando esta figura, e considerando que a rede apresentada possui saída linear, verificamos que y (saída da rede após a apresentação de cada amostra de treinamento) será dada por:

$$y = w_{0,1} + w_{1,1}\, z_1 + w_{2,1}\, z_2 + w_{3,1}\, z_3 + w_{4,1}\, z_4 + w_{5,1}\, z_5, \tag{2.36}$$

onde

$$z_l = f\left(\sum_{j=1}^{k} w_{jl} x_j - w_{0l}\right), \quad \text{para } l = 1,...,5, \tag{2.37}$$

onde $f(.)$ é a tangente hiperbólica e k é o número de entradas da rede. Esta expressão está de acordo com a equação (2.35) do teorema da aproximação universal das redes MLP.

Após treinar esta rede e definir um conjunto de pesos, vamos fazer uma análise gráfica dos componentes da equação (2.36) que determinam a saída da rede.

A Figura 2.17 apresenta as funções de ativação de cada unidade intermediária ao final do processo de treinamento. Percebe-se que as curvas apresentadas são as funções básicas (ver Figura 2.15(b)) deslocadas em relação à abcissa e devidamente escalonadas.

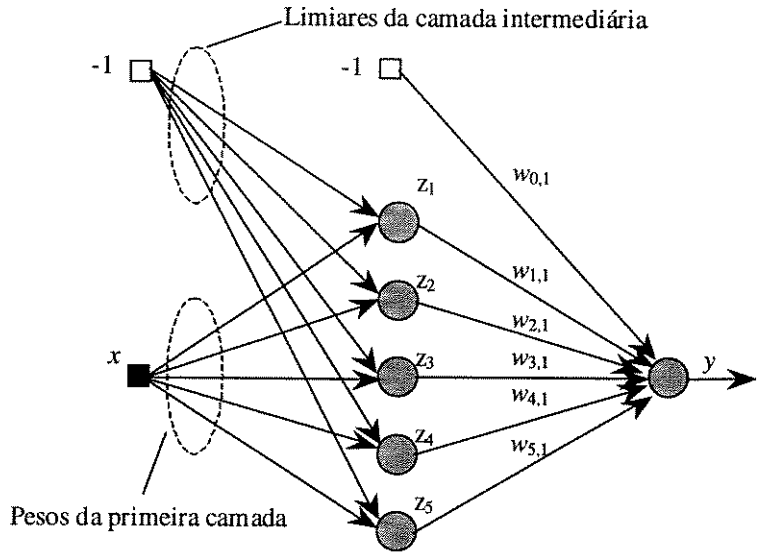


Figura 2.16: Rede MLP com treinamento via retro-propagação utilizada para aproximar a função sen(x)×cos($2x$). Funções de ativação do tipo tangente hiperbólica.

Na Figura 2.18 temos, em cada caso, a representação da diferença entre a saída desejada e a saída de cada unidade intermediária ponderada pelo respectivo peso da camada de saída. Cada figura apresenta a saída desejada menos a soma das saídas ponderadas das demais unidades.

A Figura 2.19 apresenta a combinação linear das saídas das unidades intermediárias, onde os coeficientes da combinação são os pesos da segunda camada da rede.

Finalmente, a Figura 2.20 apresenta a aproximação obtida pela rede MLP após o processo de treinamento.

Verifica-se que a arquitetura do tipo perceptron com uma única camada intermediária, e cinco neurônios nesta camada, foi capaz de aproximar uma função arbitrária, no caso sen(x)×cos($2x$), partindo de um conjunto de 42 amostras de treinamento.

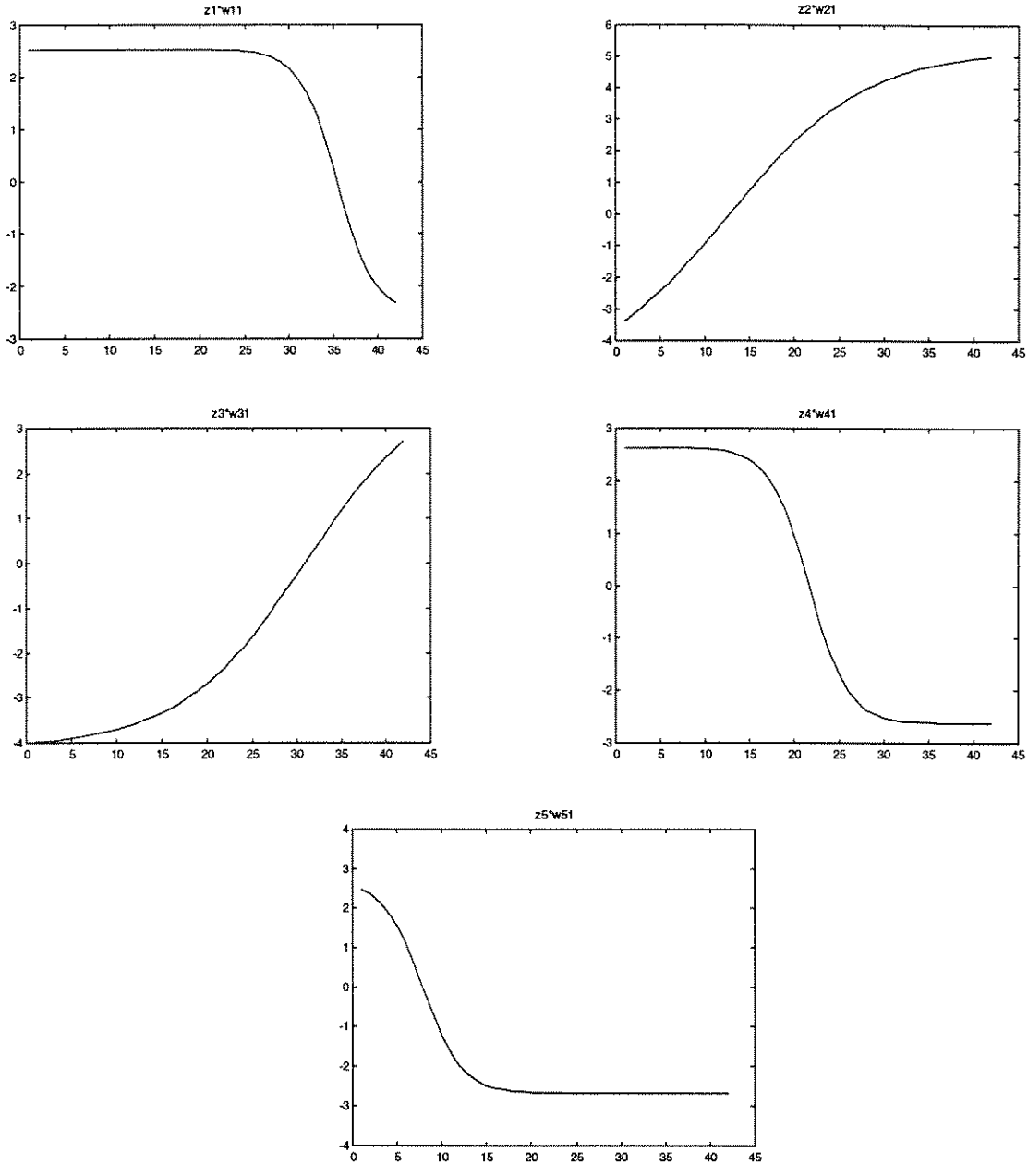


Figura 2.17: Funções de ativação dos neurônios intermediários após o treinamento da rede multiplicadas pelos pesos correspondentes da camada de saída. É perceptível que todas as funções apresentadas são tangentes hiperbólicas (funções básicas), mas estão deslocadas em relação à abcissa.

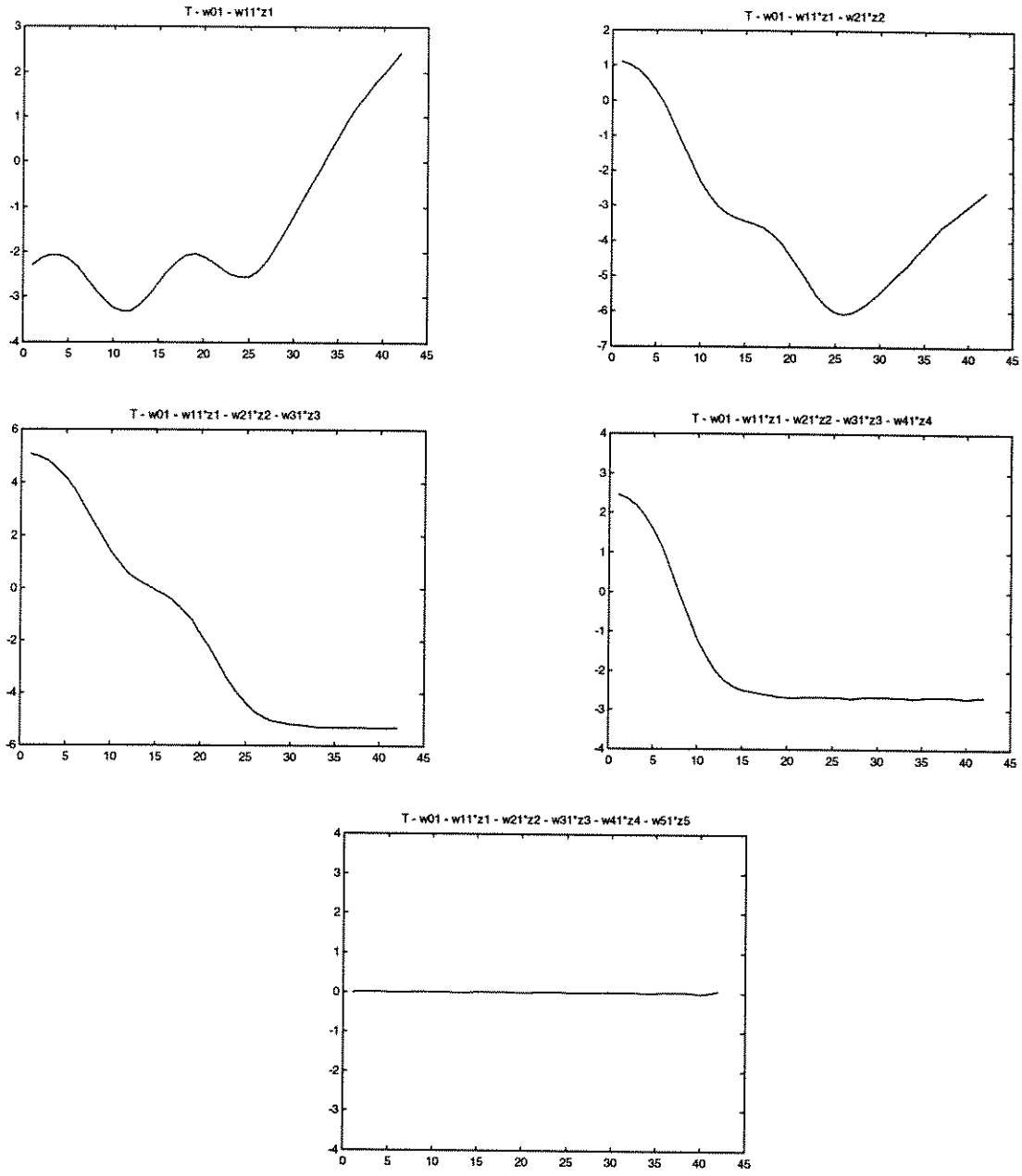


Figura 2.18: Saída desejada menos a soma da contribuição de cada neurônio da camada intermediária. O resultado final (última curva) é a diferença entre a saída desejada e a saída da rede, ou seja, a combinação linear das saídas das unidades intermediárias. Como era de se esperar, ao final do treinamento, a diferença entre a saída fornecida pela rede e a saída desejada (função a ser aproximada) é aproximadamente zero.

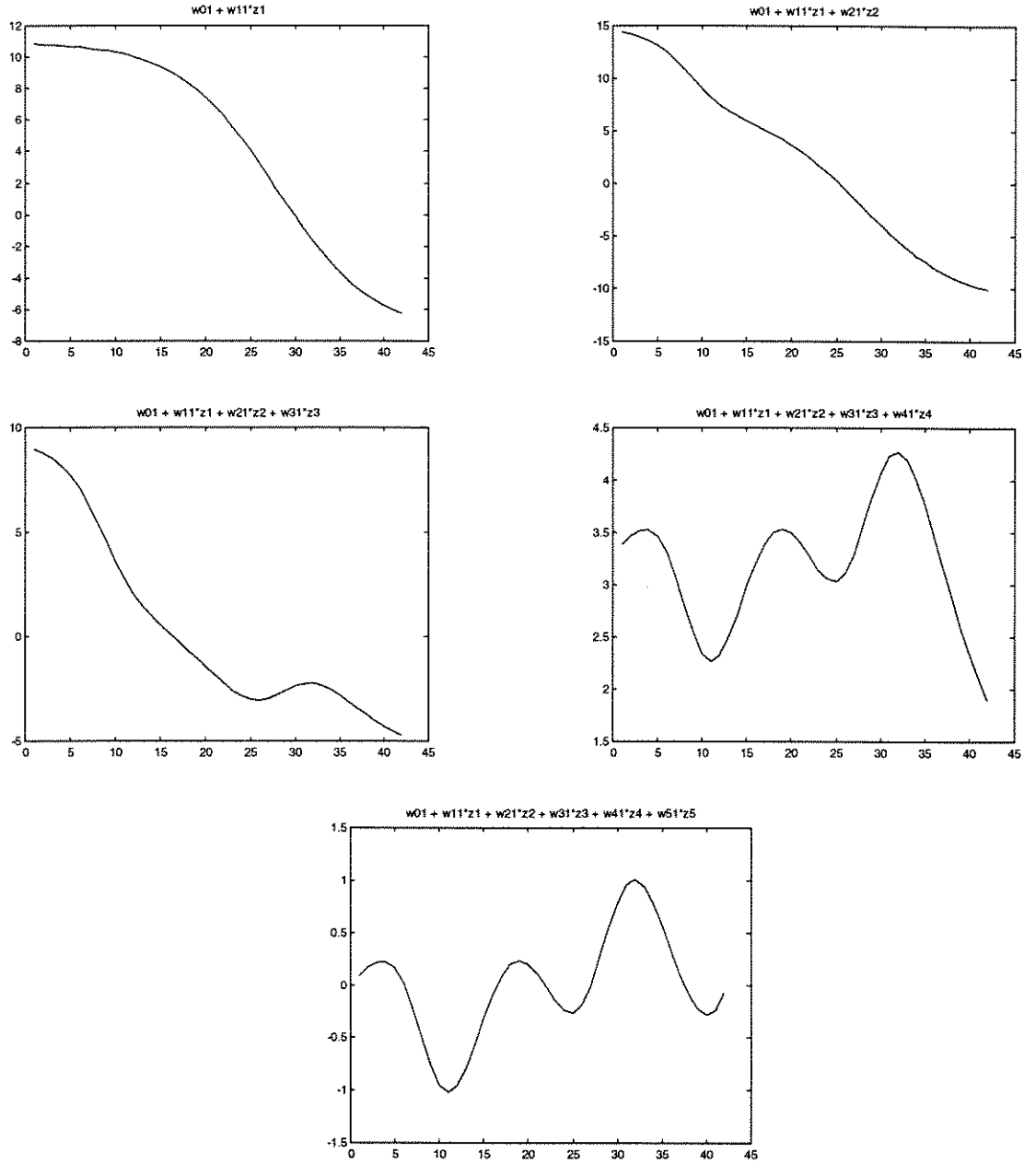


Figura 2.19: Combinação linear das saídas das unidades intermediárias. Os coeficientes da combinação são os pesos da segunda camada. Esta figura apresenta a soma das curvas que compõem a Figura 2.17, implicando na composição aditiva (combinação linear) das funções básicas dos neurônios da rede.

A saída da rede também pode ser vista na equação (2.36).

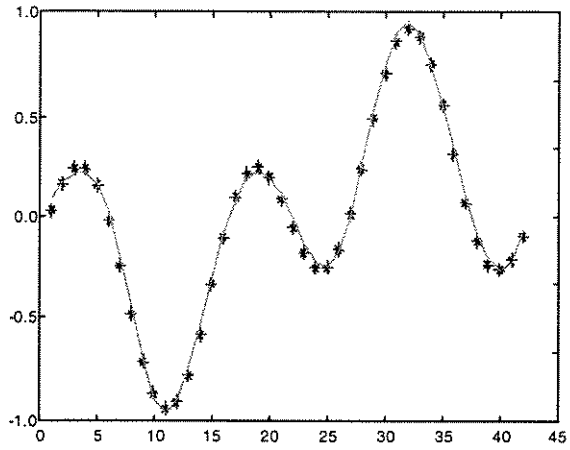


Figura 2.20: Aproximação obtida pela rede MLP para a função sen(x)×cos($2x$). Os *'s são as amostras de treinamento, e o traço contínuo (—) a aproximação da rede.

2.4.7 Virtudes e Limitações das Redes MLP

O algoritmo de retro-propagação tornou-se o algoritmo mais popular para o treinamento supervisionado do perceptron de múltiplas camadas. Basicamente, é uma composição de um método de obtenção do vetor gradiente e de uma técnica de otimização (*steepest descent*). O algoritmo de retro-propagação possui duas propriedades distintas:

- é simples de calcular localmente e
- realiza um gradiente descendente no espaço de conexões (pesos).

Estas duas propriedades do método de retro-propagação são responsáveis pelas suas vantagens e desvantagens descritas a seguir.

2.4.7.1 Conexionismo

O algoritmo de retro-propagação é um exemplo de paradigma conexionista que utiliza cálculos locais para determinar a capacidade de processamento de informações das redes neurais artificiais. A utilização de cálculos locais no projeto de arquiteturas neurais artificiais, é geralmente desejada por três razões:

- as redes neurais artificiais são vistas como metáforas para as redes biológicas;
- os cálculos locais favorecem a utilização de arquiteturas paralelas como um método eficiente para a implementação das RNA's.

Comentários sobre a implementação em paralelo de redes do tipo MLP serão feitos no Capítulo 7.

2.4.7.2 Unidades Intermediárias

As unidades intermediárias exercem um papel crítico na operação das redes MLP pois funcionam como *detetores de características* (HAYKIN, 1994). Uma nova forma de exploração deste importante atributo é no reconhecimento de características significativas que identificam os padrões de entrada de interesse.

Uma rede MLP pode operar de forma auto-organizada, e quando isso ocorre é denominado de sistema *autocodificador*. Basicamente, essa estrutura faz uma *análise de componentes principais* dos dados de entrada, fornecendo uma ferramenta ótima para a redução da dimensionalidade (compressão dos dados) do conjunto de entrada.

2.4.7.3 Superfícies de Erro e Mínimos Locais

Para o desenvolvimento de algoritmos de treinamento eficientes, é essencial a compreensão das características principais dos aspectos envolvidos no treinamento das redes MLP e suas implicações. Uma maneira extremamente útil de descrever um problema de treinamento é sob a forma de *aproximação de funções*, ou *otimização de funções* – que implica na minimização do funcional de erro J. Sob este ponto de vista, o treinamento de redes MLP é um exemplo de procedimento de *minimização do erro* ou *otimização*, e cada problema de treinamento define uma *superfície de erro* (ou de *energia*) multi-dimensional não negativa, que é formada plotando-se o valor do erro J para todos os parâmetros do vetor de pesos (θ) da rede.

Esta abordagem para o treinamento de redes MLP traz vários benefícios importantes:

- muitas estratégias eficientes de otimização de funções têm sido desenvolvidas fora da área de redes neurais artificiais. Como veremos no próximo capítulo, muitas dessas estratégias são diretamente aplicáveis ao treinamento de redes MLP;
- o conceito de superfície de erro da rede MLP permite uma visualização do processo de treinamento, pelo menos quando o número de parâmetros é reduzido.

Para cada combinação possível de arquitetura MLP, problema de aplicação e função de erro, existe uma superfície de erro correspondente de dimensão $P + 1$, para um MLP com P pesos (SHEPHERD, 1997), sendo impraticável produzir um mapeamento de superfície de erro que seja detalhado e abrangente (mesmo para pequenos problemas e redes de dimensão reduzidas). Não é surpresa, portanto, que as propriedades das superfícies de erro das redes MLP sejam assunto de grandes debates.

É importante mencionar que a prática comum de inferência de características presentes na superfície de erro é bastante questionável. Por exemplo, há uma tendência de afirmar que uma rede MLP está presa em um mínimo local sempre que a curva de decaimento do erro (gerada plotando-se o valor do erro ao longo de várias épocas de treinamento) possui um grande platô em um valor de erro elevado. Na verdade, existem várias causas igualmente plausíveis para este comportamento que não são relacionadas com a presença de mínimos locais; por exemplo, a rede está convergindo para um *ponto de sela*; o problema possui um erro final elevado na solução e a rede está convergindo para o mínimo global; a rede está atravessando um platô lentamente; o algoritmo está fazendo um zig-zag no fundo de um vale; um ou mais neurônios da rede estão saturados ou a rede está tomando passos muito pequenos a cada iteração.

Algumas evidências (como gráficos de contorno de pequenas regiões de superfícies de erro) sugerem que a maior parte das superfícies de erro das redes MLP compartilham várias características (HAGAN *et. al*, 1997):

- alto grau de suavidade;
- grandes platôs;
- pequenos vales;
- vários mínimos locais e
- um certo grau de simetria em relação a origem do sistema de coordenadas utilizado para plotar a superfície de erro.

Estas características das superfícies de erro fornecem boas indicações de quais estratégias devem ser adotadas no desenvolvimento de métodos práticos e eficientes de treinamento:

- a suavidade das superfícies de erro sugerem que métodos de otimização clássica (ver Capítulo 3) que utilizam derivadas serão eficientes;
- o efeito da precisão de ponto-flutuante torna-se um aspecto importante em regiões de platôs (ver Capítulo 5) e
- se existem mínimos locais, estratégias de busca do mínimo global, ou mínimo local satisfatório, devem ser empregados (ver Capítulo 7).

A seguir apresentaremos algumas figuras com o objetivo de ilustrar as características das superfícies de erro e seus mínimos.

Exemplo 2.2:

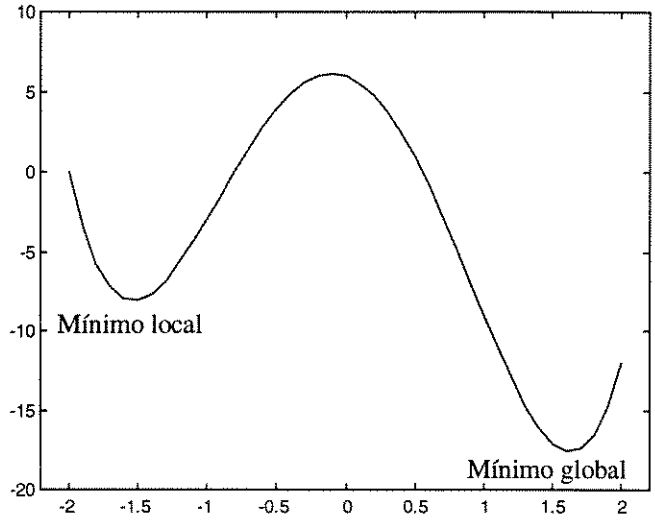


Figura 2.21: Exemplo escalar de uma função com um mínimo local e seu mínimo global.

A Figura 2.21 apresenta a função escalar dada pela expressão $F(x) = 3x^4 - 15x^2 - 3x + 6$, que possui um mínimo local e seu mínimo global para $x \in [-2,2]$.

Uma forma simples de reduzir as chances de se ficar preso em um mínimo local é escolhendo um conjunto de pesos iniciais ótimo. No Capítulo 7 serão feitas considerações sobre diferentes procedimentos de inicialização do processo de treinamento que permitem a determinação de mínimos locais mais adequados, ou até mesmo do mínimo global.

Exemplo 2.3:

Para investigar o comportamento da superfície do erro quadrático médio para redes com múltiplas camadas adotaremos um exemplo simples de aproximação de funções. A rede utilizada neste problema será apresentada na Figura 2.22.

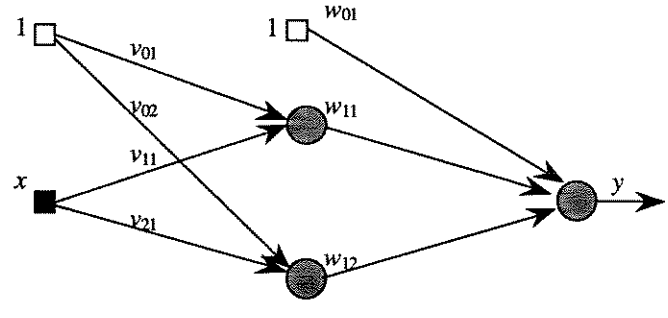


Figura 2.22: Rede para aproximação de funções.

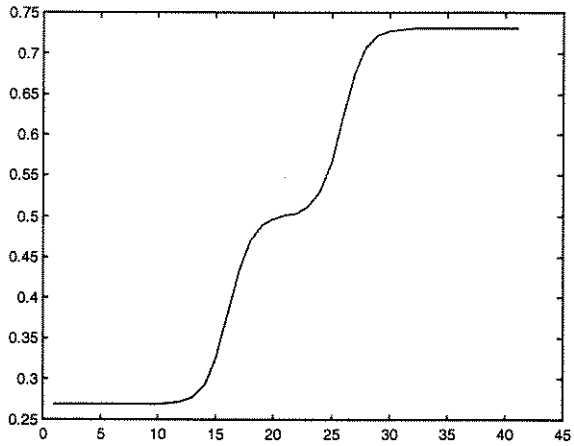


Figura 2.23: Função a ser aproximada.

A Figura 2.23 apresenta o gráfico da função a ser aproximada (HAGAN *et. al*, 1997). O objetivo é treinar a rede da Figura 2.22 para aproximar a função da Figura 2.23.

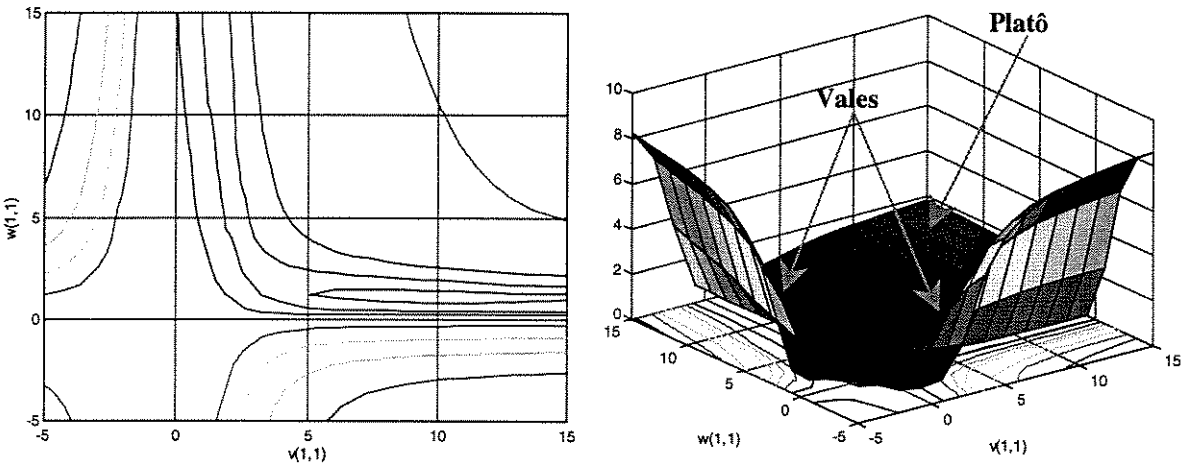


Figura 2.24: Superfície do erro quadrático e seu contorno em relação aos pesos v_{11} e w_{11}.

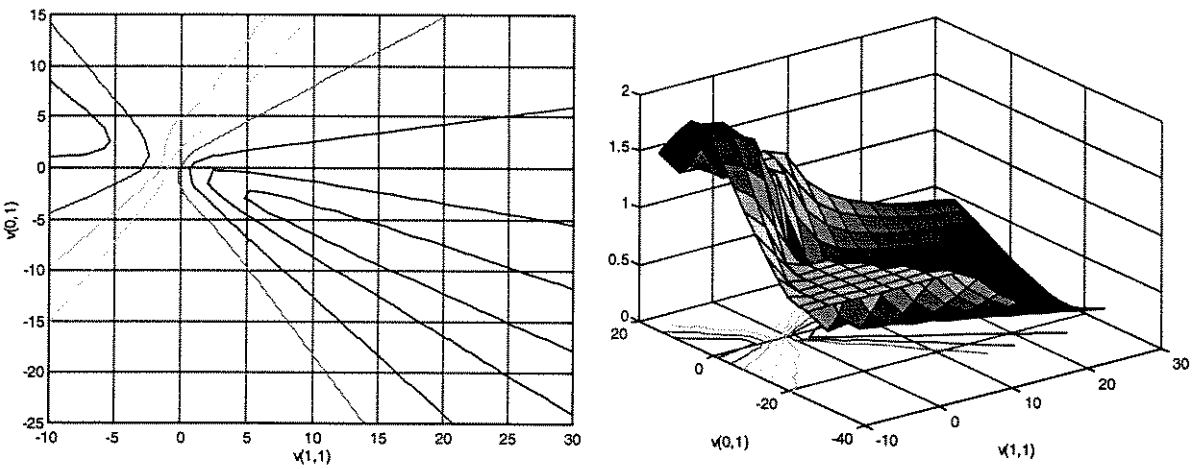


Figura 2.25: Superfície do erro e seu contorno em relação ao peso v_{11} e ao limiar v_{01}.

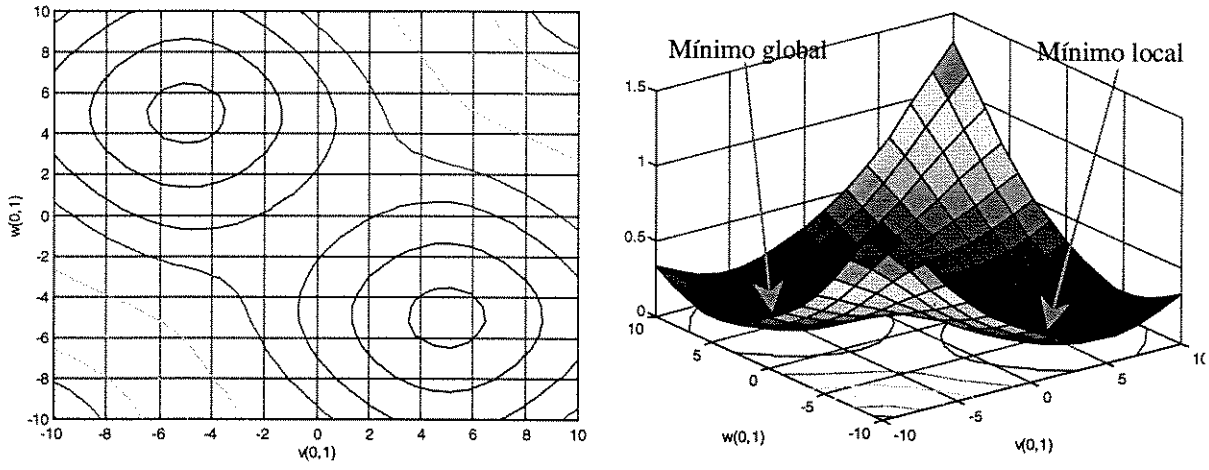


Figura 2.26: Superfície do erro quadrático e seu contorno em relação aos limiares v_{01} e w_{01}.

Para que seja possível plotar o gráfico do comportamento do funcional de erro, iremos variar somente dois parâmetros da rede a cada instante. As Figuras de 2.24 a 2.26 mostram a superfície de erro e seu respectivo contorno quando consideramos a sua variação em relação a alguns pares de pesos da rede.

As partes mais escuras das superfícies representam vales, ou regiões de mínimos. As curvas apresentadas permitem verificar algumas características importantes da superfícies de erro, como:

- suavidade;
- existência de platôs e vales; e
- presença de mínimos locais.

Fica claro na Figura 2.26 que a escolha inadequada de uma condição inicial para o algoritmo de treinamento pode fazer com que o algoritmo convirja para um ponto sub-ótimo. Como, geralmente, não conhecemos a superfície de erro, torna-se necessária a utilização de várias condições iniciais diferentes para aumentar as chances do algoritmo convergir para o mínimo global. Maiores comentários sobre condições iniciais ótimas do processo de treinamento serão feitas no Capítulo 7.

2.4.7.4 Escalonamento

Em princípio, redes neurais como as do tipo MLP treinadas com o algoritmo de retro-propagação oferecem um grande potencial. Entretanto, para que este potencial possa ser plenamente atingido, é preciso superar o *problema de escalonamento*, que refere-se à qualidade do comportamento da rede (como o tempo de processamento e capacidade de generalização) quando a tarefa a ser realizada aumenta em dimensão e complexidade. Existem

várias formas possíveis de medir a dimensão e complexidade de um problema. O Capítulo 5 apresenta algumas ferramentas que permitem fazer este tipo de medida baseado no conjunto de dados disponível para o treinamento.

Uma forma de amenizar problemas de escalonamento é fazer com que a arquitetura da rede e as restrições impostas ao conjunto de pesos incorporem informações sobre o problema que deverá ser resolvido. Outra forma de tratar o problema de escalonamento é reformular o processo de aprendizagem inserindo *modularidade* na arquitetura da rede. Um sistema computacional é dito ser *modular* se a arquitetura puder ser separada em dois ou mais pedaços responsáveis por partes distintas e possivelmente mais simples do conjunto de entradas.

2.5 Representação do Conhecimento em Redes Neurais Artificiais

As redes neurais artificiais utilizam a estrutura de interconexões para representar o conhecimento. Isso significa que, com base na configuração resultante da rede, a intensidade das conexões é ajustada com o objetivo de armazenar informação. Como resultado natural deste processo de ajuste, o conhecimento armazenado fica distribuído pela rede neural. Esta é uma das mais importantes diferenças entre processadores de informação baseados em redes neurais e aqueles baseados nos princípios da máquina de VON NEUMANN (1958), como no caso dos computadores digitais, nos quais o conhecimento é particionado em células de informação (unidades mais simples de conhecimento) que possam ser armazenadas em posições de memória bem definidas. É importante observar que todo conhecimento passível de representação em máquinas de von Neumann pode também ser adequadamente representado em redes neurais artificiais, por exemplo, tomando-se um conjunto de pesos onde as células de informação podem ser armazenadas. Portanto, o tipo de representação de conhecimento empregado no caso das redes neurais artificiais fornece a este sistemas potencialidades de representação ausentes em outros sistemas tradicionais de representação do conhecimento.

Considerando agora que todo conhecimento pode ser representado como parâmetros de um problema, a capacidade de representação das redes neurais artificiais geralmente excede a dimensão paramétrica do problema. Com isso, a relação existente entre os pesos da rede neural e as células de informação a serem armazenadas não é direta. Diz-se então que o conhecimento está distribuído pelas conexões da rede, sendo que cada conexão pode participar na representação de mais de uma célula de informação. Ao contrário do caso de máquinas de von Neumann, onde a perda ou falha de uma única posição de memória pode

acarretar a falha de todo o sistema, em representações baseadas em redes neurais, a sensibilidade à perda ou falha de uma conexão pode ser muito baixa.

Existe grande expectativa de que as redes neurais artificiais exercerão um importante papel na automação do processo de aquisição de conhecimento e codificação, entretanto, o problema de conhecimento em RNA's é representado em nível sub-simbólico tornando-se difícil para a compreensão humana. Uma forma de entender o comportamento das RNA's é extraindo seu conhecimento em função de regras (HUANG & ENDSLEY, 1997).

Em um trabalho recente, GUDWIN (1996) propõe a elaboração de uma taxonomia dos tipos de conhecimento, baseada na teoria da semiótica. Os tipos elementares de conhecimento podem ser divididos em:

- remático;
- dicente e
- argumentativo.

No conhecimento argumentativo, tem-se a idéia de argumento, que corresponde a um agente de transformação de conhecimento. Um argumento tipicamente transforma um conjunto de conhecimentos, chamados de premissas do argumento, em um novo conhecimento, chamado de sua conclusão. Esta transformação é realizada por meio de uma função de transformação, chamada de função argumentativa, que caracteriza o tipo de argumento. O conhecimento argumentativo, por sua vez, é sub-dividido em conhecimento argumentativo *indutivo*, *dedutivo* ou *abdutivo*. Os conhecimentos do tipo argumentativo são considerados os mais complexos. Em linhas gerais, um argumento indutivo é aquele que modifica um conhecimento pré-existente, o argumento dedutivo permite selecionar um conhecimento dentre vários e um argumento abdutivo gera conhecimentos de uma forma qualquer.

O conhecimento remático é o tipo de conhecimento gerado pela interpretação de remas, ou termos. Esses termos são utilizados para referenciar fenômenos do ambiente, tais como experiências sensoriais, objetos, e ocorrências. Existem três tipos de conhecimentos remáticos. O conhecimento remático *simbólico*, *indicial* e *icônico*.

No caso do conhecimento dicente, um termo ou uma sequência de termos é utilizada para representar uma expressão, que codifica uma proposição. O que caracteriza um termo ou sequência de termos como sendo uma proposição é o fato de existir um valor-verdade associado a ele. Esse valor verdade é uma medida da crença que o sistema cognitivo tem de

que uma proposição é verdadeira. Usualmente, o valor verdade é representado por um valor entre 0 e 1. Um valor-verdade igual a 0 significa que o sistema acredita que a proposição é falsa. Um valor-verdade igual a 1 representa que o sistema acredita que a proposição é verdadeira.

Nesse contexto, é perceptível que o treinamento de redes MLP realiza um argumento do tipo indutivo, onde os pesos das conexões são modificados no intuito de realizar alguma tarefa.

Capítulo 3

Algoritmos de Otimização para Treinamento Supervisionado

Neste capítulo são descritos e analisados vários métodos de otimização não-linear irrestrita para treinamento de redes multicamadas. O treinamento de redes neurais com várias camadas pode ser entendido como um caso especial de aproximação de funções, onde não é levado em consideração nenhum modelo explícito dos dados (SHEPHERD, 1997). Serão revistos os seguintes algoritmos:

- algoritmo padrão (BP);
- método do gradiente (GRAD);
- gradiente conjugado (GC);
- Fletcher-Reeves (FR);
- Polak-Ribière (PR);
- gradiente conjugado escalonado de MOLLER (1993) (SCG);
- Newton (MN);
- Levenberg-Marquardt (LM);
- Davidon-Fletcher-Powell (DFP);
- Broyden-Fletcher-Goldfarb-Shanno (BFGS);
- BATTITI (1992) One-step Secant (OSS) e
- Quickprop de FAHLMAN (1988) (QUICK).

Não objetivamos indicar diretamente a eficiência relativa destes algoritmos em uma dada aplicação, mas analisar suas principais propriedades e definir o escopo de aplicação.

3.1 Introdução

A retro-propagação (*backpropagation*) do gradiente do erro provou sua utilidade no treinamento supervisionado de redes multicamadas para aplicação a muitos problemas de classificação e mapeamento estático de funções não-lineares. A Figura 3.1 ilustra a retro-propagação do erro em uma rede MLP. Há casos em que a velocidade de aprendizagem é um fator limitante para possibilitar a implementação prática deste tipo de ferramenta computacional no processo de solução de problemas que requerem otimalidade, robustez e rapidez na convergência do processo de ajuste de parâmetros.

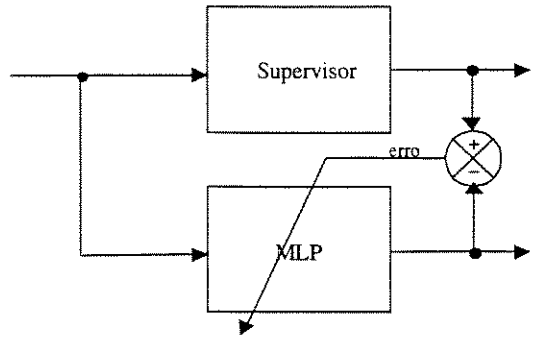


Figura 3.1: Comparação entre a saída da rede e a saída desejada do sistema feita por um supervisor (treinamento supervisionado).

Mesmo em aplicações onde uma resposta em tempo real não é necessária, a complexidade temporal do algoritmo pode resultar na intratabilidade computacional do problema. Como exemplo, o aumento da complexidade intrínseca dos problemas atuais da área de engenharia tem produzido uma explosão combinatória dos possíveis candidatos a solução, mesmo quando existem direcionamentos efetivos para a exploração do espaço de solução. Além disso, dentre os métodos de busca no espaço de solução, é consenso a idéia de que não existe um método que seja o melhor para todos os casos. Sendo assim, muitas soluções encontradas por métodos específicos podem não atender aos requisitos mínimos do problema. Uma forma eficiente de lidar com esta situação é recorrer ao potencial de processamento computacional hoje disponível e passar a operar com métodos que forneçam simultaneamente múltiplos candidatos a solução, dentre os quais se possa escolher o melhor segundo algum critério pré-estabelecido. No caso da solução ser produzida utilizando-se redes neurais artificiais, este procedimento é tão mais viável quanto maior for a velocidade de aprendizagem da rede neural. Por exemplo, um aumento de dez vezes na velocidade de busca de uma solução permite que se encontre dez vezes mais candidatos à solução com o mesmo custo computacional. Nesta classe encontram-se aplicações relacionadas com a modelagem e identificação de sistemas não-lineares, previsão de séries temporais e controle de processos adaptativos (BATTITI, 1992).

O processo de treinamento supervisionado de redes neurais artificiais multicamadas é equivalente a um problema de otimização não-linear irrestrito, onde uma função de erro global é minimizada a partir do ajuste de parâmetros da rede neural (pesos). Esta perspectiva do processo de treinamento supervisionado permite desenvolver algoritmos de treinamento baseados em resultados bem fundamentados da teoria de análise numérica convencional. Os principais procedimentos de análise numérica passíveis de implementação computacional

empregam métodos que utilizam somente o gradiente local da função ou então métodos que utilizam também as derivadas de segunda ordem. No primeiro caso, a função é aproximada pelo primeiro (constante) e segundo (linear) termos da expansão de Taylor; no segundo caso, o terceiro termo (quadrático) também é considerado.

Este capítulo pretende descrever alguns métodos que mostram acelerar a convergência média da fase de treinamento. Como estamos interessados na velocidade de convergência dos algoritmos, considerações sobre a capacidade de generalização adquirida pela rede ao final do processo de treinamento serão desenvolvidas posteriormente. Alguns destes métodos, ao passo que requerem poucas modificações no algoritmo de retropropagação padrão, resultam em elevados graus de aceleração além de não mais requererem a escolha de parâmetros críticos da rede neural como a taxa de aprendizagem e o coeficiente de momento.

Como o treinamento de redes multicamadas pode sempre ser entendido como um problema geral de aproximação de funções, far-se-á uma breve introdução sobre a teoria de aproximação de funções e avaliação do nível de aproximação. Em seguida serão apresentados métodos de otimização do processo de aproximação resultante.

3.2 Aproximação de Funções

Considere o problema de aproximar uma função $g(.)$: $X \subset \Re^m \rightarrow \Re^r$ por um modelo de aproximação representado pela função $\hat{g}(.,\theta) : X \times \Re^P \rightarrow \Re^r$, onde $\theta \in \Re^P$ (P finito) é um vetor de parâmetros.

O problema geral de aproximação pode ser formalmente apresentado como segue (VON ZUBEN, 1996): Considere a função $g(.)$: $X \subset \Re^m \rightarrow \Re^r$, que mapeia pontos de um subespaço compacto $X \subset \Re^m$ em pontos de um subespaço compacto $g[X] \subset \Re^r$. Com base nos pares de vetores de entrada saída $\{(\mathbf{x}_l, \mathbf{s}_l)\}_{l=1}^N$ amostrados a partir do mapeamento determinístico definido pela função g na forma: $\mathbf{s}_l = g(\mathbf{x}_l) + \varepsilon_l$, $l = 1,\ldots,N$, e dado o modelo de aproximação $\hat{g}(.,\theta) : X \times \Re^P \rightarrow \Re^r$, determine o vetor de parâmetros $\theta^* \in \Re^P$ tal que $\text{dist}(g(.), \hat{g}(.,\theta^*)) \leq \text{dist}(g(.), \hat{g}(.,\theta))$, para todo $\theta \in \Re^P$, onde o operador dist(.,.) mede a distância entre duas funções definidas no espaço X. O vetor ε_l expressa o erro no processo de amostragem, sendo assumido ser de média zero e variância fixa. A solução deste problema, se existir, é denominada a melhor aproximação e depende diretamente da classe de funções a qual $\hat{g}$ pertence.

3.2.1 Avaliação do Nível de Aproximação

Em problemas de aproximação utilizando um número finito de dados amostrados e definido um modelo de aproximação $\hat{g}(.,\theta)$, a distância entre a função a ser aproximada e sua aproximação dist($g(.)$, $\hat{g}(.,\theta)$) é uma função apenas do vetor de parâmetros $\theta \in \Re^P$. Tomando a norma euclidiana como a medida de distância, produz-se a seguinte expressão:

$$J(\theta) = \frac{1}{N}\sum_{l=1}^{N}\left(g(\mathbf{x}) - \hat{g}(\mathbf{x},\theta)\right)^2 . \tag{3.1}$$

O funcional J: $\Re^P \rightarrow \Re$ é denominado superfície de erro do problema de aproximação, pois pode ser interpretado como uma hipersuperfície localizada "acima" do espaço de parâmetros $\Re^P$, sendo que para cada ponto $\theta \in \Re^P$ corresponde uma "altura" $J(\theta)$. O termo *funcional* corresponde a toda função $f: X \subset \Re^n \rightarrow \Re$, e por isso o problema de minimizar $J(\theta)$ torna-se um problema de minimização funcional.

Dada a superfície de erro, o problema de aproximação passa a ser um problema de otimização cuja solução é o vetor $\theta^* \in \Re^P$ que minimiza $J(\theta)$, ou seja,

$$\theta^* = \arg\min_{\theta \in \Re^P} J(\theta) . \tag{3.2}$$

Durante o processo de aproximação da função $g(.)$ pela função $\hat{g}(.,\theta)$ fornecida pela rede neural, devem ser considerados três tipos de erros (VAN DER SMAGT, 1994): o erro de representação, o erro de generalização e o erro de otimização.

Erro de Representação: primeiro consideremos o caso em que todo o conjunto amostral está disponível $\{(\mathbf{x}_l, \mathbf{s}_l)\}_{l=1}^{\infty}$. Assuma, também, que dado $\{(\mathbf{x}_l, \mathbf{s}_l)\}_{l=1}^{\infty}$, é possível encontrar um conjunto de parâmetros ótimo θ^*. Neste caso, o erro vai depender da adequação e do nível de flexibilidade do modelo de aproximação $\hat{g}(.,\theta)$. Este erro é também conhecido como erro de aproximação, ou *efeito bias.*

Erro de Generalização: em aplicações de mundo real, somente um número finito de amostras está disponível ou pode ser usado simultaneamente. Além disso, os dados podem conter ruído. Os valores de g para os quais nenhuma amostra está disponível devem ser interpolados. Devido a estes fatores pode ocorrer um erro de generalização, também conhecido como erro de estimação, ou *variância.*

Erro de Otimização: como o conjunto de dados é limitado, o erro é avaliado somente nos pontos que pertencem ao conjunto amostral.

Dado o conjunto amostral $\{(\mathbf{x}_l, \mathbf{s}_l)\}_{l=1}^{N}$, o vetor de parâmetros $\theta = \theta^*$ deve fornecer a melhor função de aproximação possível com base na representação paramétrica $\hat{g}(.,\theta)$ e na medida de distância dada pela equação (3.1). Se a superfície de erro for contínua e diferenciável em relação ao vetor de parâmetros (os parâmetros podem assumir qualquer valor real), então os mais eficientes métodos de otimização não-linear irrestrita podem ser aplicados para minimizar $J(\theta)$.

3.3 Técnicas de Otimização não-linear Irrestrita

Para a maioria dos modelos de aproximação $\hat{g}(.,\theta)$, o problema de otimização apresentado na equação (3.2) tem a desvantagem de ser não-linear e não-convexo, mas as vantagens de ser irrestrito e permitir a aplicação de conceitos de cálculo variacional na obtenção da solução θ^*. Estas características impedem a existência de uma solução analítica, mas permitem obter processos iterativos de solução, a partir de uma condição inicial θ_0, na forma:

$$\theta_{i+1} = \theta_i + \alpha_i \mathbf{d}_i, \qquad i \geq 0, \tag{3.3}$$

onde $\theta_i \in \Re^P$ é o vetor de parâmetros, $\alpha_i \in \Re^+$ é um escalar que define o passo de ajuste e $\mathbf{d}_i \in \Re^P$ é a direção de ajuste, todos definidos na iteração i. Os algoritmos de otimização revisados neste capítulo são aplicados na obtenção da direção de ajuste do processo iterativo dado pela equação (3.3). A forma pela qual cada um dos algoritmos procede no cálculo da direção de ajuste dos parâmetros e do passo a cada iteração permite estabelecer distinções entre eles (GROOT & WÜRTZ, 1994).

Quando a direção de minimização está disponível, é preciso definir o tamanho do passo $\alpha_i \in \Re^+$ para se determinar o ajuste de parâmetros naquela direção. Podem ser utilizados inúmeros procedimentos de busca unidimensional na determinação do passo (ver Capítulo 4). Nosso trabalho vai se concentrar então, na determinação da direção ótima. Geralmente são feitas avaliações da função e suas derivadas são utilizadas para determinar um mínimo, global ou local, e então encerrar-se o aprendizado. Existem métodos disponíveis (BROMBERG & CHANG, 1992) que aumentam a probabilidade de localizar mínimos globais, mas estes métodos exigem de dezenas a milhares de avaliações da função, tornando-se portanto, muito custosos computacionalmente.

Uma forma comum de classificação dos algoritmos de otimização é a 'ordem' da informação que eles devem calcular. Por ordem queremos dizer ordem das derivadas da função objetivo (no nosso caso a equação (3.1)). A primeira classe de algoritmos não requer mais do que a simples avaliação da função em diferentes pontos do espaço. Nenhuma derivada está envolvida. São os chamados *métodos sem diferenciação*. A segunda classe de algoritmos faz uso da derivada primeira da função a ser minimizada. São chamados de *métodos de primeira ordem*. Outra classe de algoritmos que será intensamente estuda neste capítulo são os chamados *métodos de segunda ordem*, e que utilizam informações sobre a derivada segunda da função de custo. Uma última divisão inclui os algoritmos cujos parâmetros são ajustados de maneira empírica, ou seja, através de procedimentos de tentativa e erro. Classificam-se como *métodos empíricos*.

A Figura 3.2 apresenta um organograma das diferentes estratégias de treinamento de redes neurais artificiais que serão revistas.

Os métodos discutidos neste capítulo têm como objetivo determinar mínimos locais, que são pontos na vizinhança dos quais o funcional de erro possui o menor valor. Teoricamente, os métodos de segunda ordem não são mais capazes de encontrar um mínimo global do que os métodos de primeira ordem.

O problema de determinação de mínimos globais, mesmo dentro de um conjunto de mínimos locais bem definidos, é difícil devido a impossibilidade fundamental de se reconhecer um mínimo global utilizando-se apenas informações locais. O ponto chave da otimização global é saber quando parar. Muitos esforços têm sido empenhados diretamente no problema de determinação de mínimos globais. Recentemente, métodos heurísticos tornaram-se populares, em particular os métodos evolutivos como algoritmos genéticos e recozimento simulado (*simulated annealing*). Entretanto, nenhuma dessas abordagens, analítica ou heurística, garante a localização de um mínimo global de uma função suave e contínua em tempo finito e com recursos computacionais limitados.

Mínimos locais não são únicos por uma de duas razões:

- a função é multimodal;
- se a Hessiana é singular em um mínimo local, este mínimo constitui um conjunto compacto ao invés de um ponto isolado. Isto é, o valor da função deve ser constante ao longo de uma direção, de um plano ou de um subespaço maior (McKeon *et. al.*, 1997).

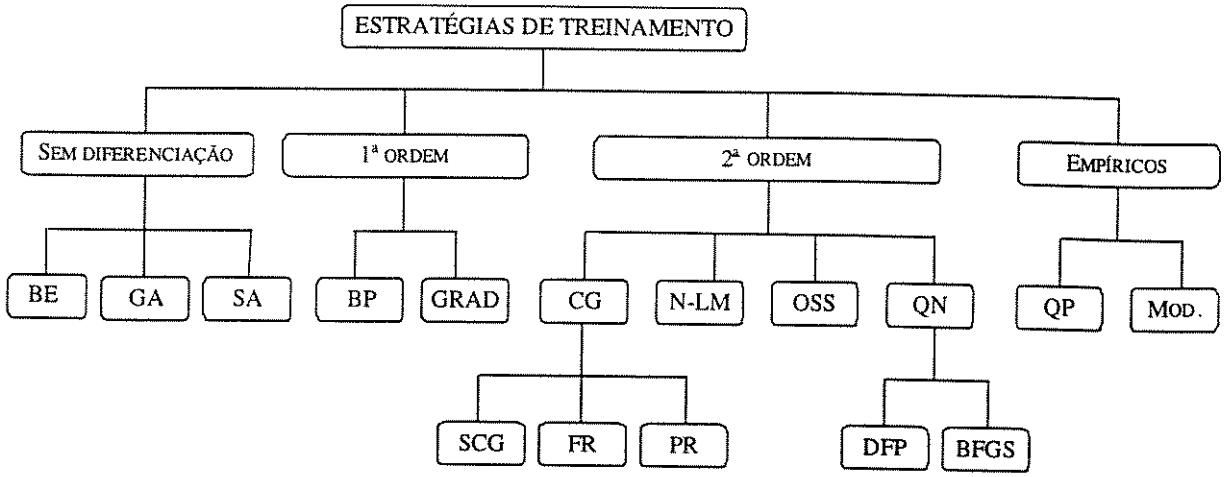


Figura 3.2: Estratégias de treinamento de redes neurais do tipo MLP.

Dentre as quatro classes de algoritmos citadas acima, serão apresentados vários métodos de otimização, sendo que alguns destes foram implementados e testados. Os resultados experimentais serão detalhados no Capítulo 9.

3.4 Métodos Sem Diferenciação

Dentre esses métodos podemos citar *simulated annealing* (SA), *algoritmos genéticos* (GA) e *busca exaustiva* (BE). Não entraremos em detalhes sobre as características do método *simulated annealing* e de *busca exaustiva*. No Capítulo 7 falaremos um pouco sobre os *algoritmos genéticos* aplicados à construção e determinação dos pesos de uma rede neural do tipo MLP. É possível adiantar que seus desempenhos são inferiores aos métodos de primeira e segunda ordem quando os problemas são diferenciáveis.

3.5 Métodos de Primeira Ordem

O erro quadrático médio a ser minimizado pode ser apresentado considerando-se seus termos até segunda ordem pela equação (3.4):

$$J_{quad}(\theta) = J(\theta_i) + \nabla J(\theta_i)^T(\theta - \theta_i) + (\theta - \theta_i)^T \nabla^2 J(\theta_i)(\theta - \theta_i), \qquad (3.4)$$

onde $\nabla J(\theta_i)$ é o vetor gradiente e $\nabla^2 J(\theta_i)$ é a matriz hessiana de $J(\theta)$, ambos calculados no ponto $\theta = \theta_i$, e $J_{quad}(\theta)$ representa a aproximação até segunda ordem de $J(\theta)$.

Nos métodos de primeira ordem apenas os termos constante e linear em θ da expansão em série de Taylor são considerados. Estes métodos, onde unicamente o gradiente local determina a direção de minimização **d** (eq. (3.3)), são conhecidos como métodos da direção de maior decrescimento (*steepest descent* ou *gradient descent*).

3.5.1 Algoritmo Padrão de primeira ordem (BP)

Este método funciona como segue. Quando a rede está em um estado θ_i, o gradiente $\nabla J(\theta_i)$ é calculado e um passo de minimização na direção oposta ao gradiente $\mathbf{d} = -\nabla J(\theta)$ é efetuado. A regra de aprendizado é dada pela equação (3.3).

No algoritmo padrão, a minimização é geralmente feita com um passo α fixo. A determinação do passo α é fundamental, pois para valores muito baixos, o tempo de treinamento pode tornar-se exageradamente alto, e para valores muito altos os parâmetros podem divergir (HAYKIN, 1994). A velocidade de convergência é geralmente melhorada adicionando-se um termo de momento (RUMELHART *et. al.*, 1986).

$$\theta_{i+1} = \theta_i + \alpha_i \mathbf{d}_i + \beta_i \Delta\theta_{i-1}, \quad i \geq 0. \tag{3.5}$$

Este termo adicional geralmente evita oscilações no comportamento do erro, pois pode ser interpretado como a inclusão de uma aproximação da informação de segunda ordem (HAYKIN, 1994).

3.5.1.1 Algoritmo

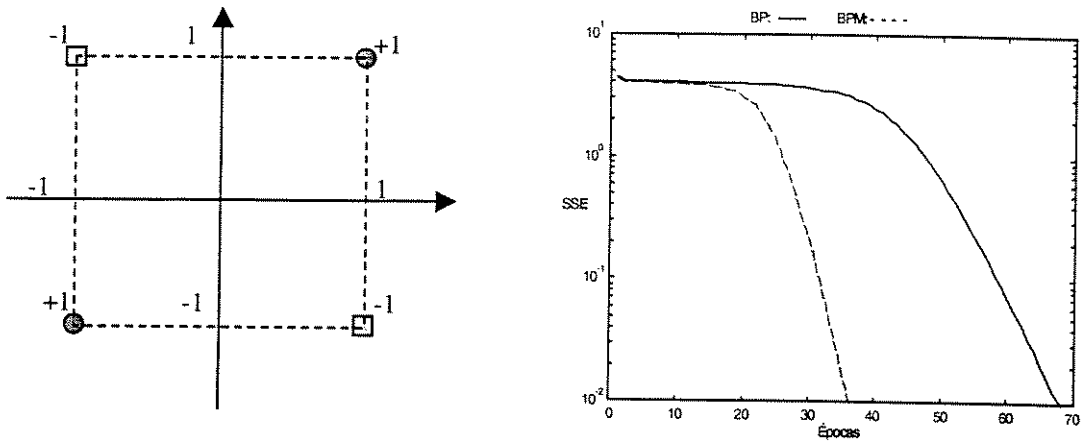
O algoritmo utilizado neste método está resumido abaixo:

1. Atribua um valor inicial $\theta_0 \in \Re^p$ para o vetor de parâmetros e um valor arbitrariamente pequeno para uma constante $\varepsilon > 0$
2. Atribua valores para os parâmetros α e β
3. Enquanto a condição de parada não for satisfeita, faça:
   3.1. $\theta_{i+1} = \theta_i - \alpha \nabla J(\theta_i) + \beta \Delta\theta_{i-1}$

Exemplo 3.1:

Para ilustrar as características de alguns dos algoritmos de treinamento apresentados neste capítulo, considere o problema de aproximar a função XOR utilizando uma rede do tipo perceptron com uma única camada intermediária e dez unidades nesta camada. A Figura 3.3(a) apresenta o problema XOR e a Figura 3.3(b) o comportamento da curva de erro quando é feita a inclusão do termo de momento no algoritmo padrão. Verifica-se que a inclusão deste termo diminui o platô da curva de erro e reduz significativamente o tempo de convergência.

A ineficiência e ausência de robustez do algoritmo padrão é devida, principalmente, ao fato de que é feita uma aproximação linear da função de erro e o passo de ajuste é arbitrário.



(a) (b)

Figura 3.3: (a) Problema a ser abordado (XOR). (b) Comparação da velocidade de convergência do algoritmo padrão com e sem momento para uma rede com 5 unidades intermediárias (SSE = 0.01). Traço contínuo: algoritmo sem o termo de momento. Curva tracejada: algoritmo com o momento.

3.5.2 Método do Gradiente (GRAD)

Dentre os métodos que utilizam diferenciação e busca, o método do gradiente é o mais simples para obtenção da direção $\mathbf{d}_i$, pois utiliza apenas informações de primeira ordem. Na i-ésima iteração, a direção $\mathbf{d}_i$ é definida como a direção de módulo unitário de maior decrescimento da função J.

$$\mathbf{d} = -\frac{\nabla J(\theta)}{\|\nabla J(\theta)\|}. \tag{3.6}$$

A lei de ajuste do método do gradiente é, então, dada por:

$$\theta_{i+1} = \theta_i - \alpha_i \frac{\nabla J(\theta_i)}{\|\nabla J(\theta_i)\|}. \tag{3.7}$$

3.5.2.1 Algoritmo

O algoritmo utilizado neste método está resumido abaixo (VON ZUBEN, 1996):

1. Atribua um valor inicial $\theta_0 \in \Re^P$ para o vetor de parâmetros e um valor arbitrariamente pequeno para uma constante $\varepsilon > 0$, faça $\alpha = 1$ e $i = 0$ e calcule $\nabla J(\theta_i)$
2. Enquanto $\|\nabla J(\theta_i)\| > \varepsilon$, faça:
   2.1. $\theta_{prov} = \theta_i - \alpha_i \frac{\nabla J(\theta_i)}{\|\nabla J(\theta_i)\|}$

2.2. Enquanto $J(\theta_{prov}) \geq J(\theta_i)$ faça:

2.2.1. $\alpha_i = t_r \alpha_i$

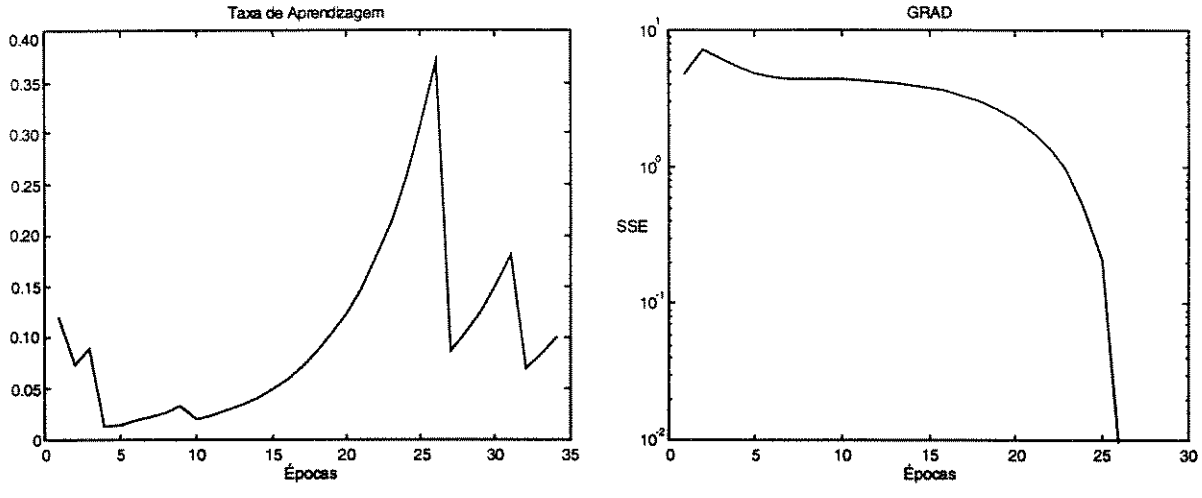
2.2.2. $\theta_{prov} = \theta_i - \alpha_i \dfrac{\nabla J(\theta_i)}{\|\nabla J(\theta_i)\|}$

2.3. $\begin{cases} \alpha_{i+1} = t_a \alpha_i \\ \theta_{i+1} = \theta_{prov} \\ i = i+1 \end{cases}$

2.4. Calcule $\|\nabla J(\theta_i)\|$.

Onde t_a e t_r são taxas de aumento e redução do passo α, respectivamente.

A Figura 3.4 ilustra o comportamento do método do gradiente quando aplicado ao problema do Exemplo 3.1. Verifica-se um comportamento monotonicamente decrescente da função de custo (erro), percebe-se que a taxa de aprendizagem oscila (aumenta e diminui) e o número de épocas para convergência é menor do que o do algoritmo BP, quando o mesmo critério de parada é adotado.



(a) (b)

Figura 3.4: Comportamento do método do gradiente durante a minimização do funcional de erro do problema XOR. (a) Evolução típica da taxa de aprendizagem. (b) Decaimento do erro, note o comportamento monotonicamente decrescente e o tempo reduzido de convergência.

Outro critério de parada que pode ser adotado, é fazer com que o erro dado pela expressão (3.1) seja menor ou igual a um valor de ε pré-estabelecido no passo 1, ao invés de escolhermos a norma do vetor gradiente. Este critério de parada não garante que o algoritmo pare de iteragir caso um mínimo local seja atingido, com $J(\theta^*_{loc}) > \varepsilon$. A parada ocorrerá garantidamente caso o critério de parada adotado seja $\|\nabla J(\theta_i)\| < \varepsilon$.

Sempre que $J(\theta)$ tiver pelo menos um mínimo, o método do gradiente associado a este procedimento de busca unidirecional vai seguramente fornecer uma solução θ^*, mínimo local do problema (3.2). A inclusão de um termo de momento junto a equação (3.3), na forma:

$$\theta_{i+1} = \theta_i - \alpha_i \frac{\nabla J(\theta_i)}{\|\nabla J(\theta_i)\|} + \beta_i(\theta_i - \theta_{i-1}), \tag{3.8}$$

não é recomendada neste caso por tornar mais complexo o atendimento da condição de garantia de convergência para mínimos locais.

3.6 Métodos de Segunda Ordem

Atualmente estes métodos são considerados a maneira mais eficiente de se fazer o treinamento de redes neurais do tipo MLP (SHEPHERD, 1997). Estes algoritmos recorrem a um rigor matemático baseado em modelos de otimização não-linear irrestrita bem definidos, não apresentando assim um vínculo natural com a inspiração biológica inicialmente proposta para as RNA's.

3.6.1 Método de Newton Modificado (NM)

O método de Newton pode ser considerado como o método local básico que utiliza informações de segunda ordem. Deve ser enfatizado que a sua aplicação prática aos perceptrons multicamadas é pouco recomendada uma vez que o cálculo da matriz Hessiana representa um elevado custo computacional pois exige a inversão, análise espectral e armazenagem de uma matriz quadrada que é da ordem do número de parâmetros P a serem ajustados (BATTITI, 1992; LUENBERGER, 1989; BAZARAA *et. al.*, 1993).

O vetor θ_{i+1}, é a solução que minimiza exatamente $J(\theta)$ dado pela equação (3.4), atendendo portanto a condição de otimalidade

$$\frac{\partial J_{quad}(\theta_{i+1})}{\partial \theta_{i+1}} = 0. \tag{3.9}$$

Aplicando a equação (3.9) na equação (3.4) resulta

$$\theta_{i+1} = \theta_i - \left[\nabla^2 J(\theta_i)\right]^{-1} \nabla J(\theta_i), \tag{3.10}$$

onde $\nabla^2 J(.)$ é a matriz hessiana e $\nabla J(.)$ o vetor gradiente.

Utilizando o mesmo raciocínio empregado no caso do método do gradiente, como a função $J(\theta)$ não é necessariamente quadrática, a minimização de sua aproximação quadrática $J_{quad}(\theta)$ dada pela equação (3.4) pode não fornecer uma solução θ_{i+1} tal que $J(\theta_{i+1}) < J(\theta_i)$. A lei de ajuste (3.10) torna-se então:

$$\theta_{i+1} = \theta_i - \alpha_i \left[\nabla^2 J(\theta_i)\right]^{-1} \nabla J(\theta_i). \tag{3.11}$$

Maiores detalhes sobre a determinação do passo α_i serão apresentados no Capítulo 4.

Na forma como foi apresentado acima, o método de Newton ainda não apresenta garantia de convergência, pois nada pode ser afirmado sobre o sinal da matriz hessiana, que deve ser definida positiva por duas razões: para garantir que a aproximação quadrática tenha um mínimo e que a inversa da matriz hessiana exista, que é a condição necessária para resolver as equações (3.10) ou (3.11) a cada iteração. Portanto, é necessário testar a positividade de $\nabla^2 J(\theta)$ a cada iteração, e na eventualidade de se obter $\nabla^2 J(\theta) \leq 0$, deve-se aplicar um procedimento de positivação desta matriz.

O Capítulo 5 fará comentários gerais sobre os problemas numéricos associados ao treinamento de arquiteturas do tipo MLP.

3.6.1.1 Algoritmo

O algoritmo utilizado neste método está resumido abaixo:

1. Atribua um valor inicial $\theta_0 \in \Re^p$ para o vetor de parâmetros e um valor arbitrariamente pequeno para uma constante $\varepsilon > 0$
2. Enquanto a condição de parada não for satisfeita, faça:
   - 2.1. Calcule $\nabla J(\theta_i)$, $\nabla^2 J(\theta_i)$ e $\mathbf{d}_i = \left[\nabla^2 J(\theta_i)\right]^{-1} \nabla J(\theta_i)$
   - 2.2. Utilize um procedimento de busca unidimensional para encontrar um α_i que seja solução ótima do problema $\min_{\alpha_i \in (0,1]} J(\theta_i + \alpha_i \mathbf{d}_i)$.
   - 2.3. Lei de ajuste: $\theta_{i+1} = \theta_i - \alpha_i \mathbf{d}_i$

3.6.1.2 Cálculo exato da Hessiana

A determinação exata da matriz hessiana não é uma tarefa fácil. CHRIS BISHOP (1992) apresenta expressões que permitem o cálculo exato da matriz hessiana para arquiteturas MLP com uma ou mais camadas intermediárias.

Apresentaremos aqui, apenas as expressões que permitem a determinação da matriz hessiana de uma rede MLP com uma única camada escondida.

Na determinação da matriz hessiana usaremos os índices k e k' para as unidades de entrada, l e l' para as unidades intermediárias e m e m' para as saídas. Sendo a_i a entrada da unidade i e z_i a ativação desta unidade, então tem-se que $z_i = f(a_i)$. A matriz hessiana para uma rede com uma única camada intermediária pode ser calculada utilizando-se três blocos de expressões:

1. Ambos os pesos pertencentes à segunda camada:

$$\frac{\partial^2 J}{\partial w_{ml} \partial w_{m'l'}} = z_l z_{l'} \delta_{mm'} H_{m'}.$$

2. Ambos os pesos na primeira camada:

$$\frac{\partial^2 J}{\partial w_{lk} \partial w_{l'k'}} = z_k z_{k'} \times \left\{ f''(a_{l'}) \delta_{ll'} \sum_m w_{ml'} \sigma_m + f'(a_{l'}) f'(a_l) \sum_m w_{ml'} w_{ml} H_m \right\}.$$

3. Um peso em cada camada:

$$\frac{\partial^2 J}{\partial w_{lk} \partial w_{ml'}} = z_k f'(a_l) \{ \sigma_m \delta_{ll'} + z_{l'} w_{ml} H_m \},$$

onde:

$$\sigma_m = g_m(.) - \hat{g}_m(.), \; H_m = -z_m(1 - z_m); \quad H_m = \frac{\partial^2 J}{\partial a_m^2} = f''(a_m) \frac{\partial J}{\partial z_m} + [f'(a_m)]^2 \frac{\partial^2 J}{\partial z_m^2},$$

e

$$\frac{\partial J}{\partial z_m} = (g - \hat{g}) \qquad \frac{\partial^2 J}{\partial z_m^2} = 1.$$

3.6.1.3 Positivando a Hessiana

Dada uma matriz **A** simétrica, conhecendo-se o menor autovalor λ_{min} desta matriz, é possível obter uma matriz **M** definida positiva a partir de **A** na forma:

- se $\lambda_{min} > 0$, então $\mathbf{M} = \mathbf{A}$
- se $\lambda_{min} \leq 0$, então $\mathbf{M} = \mathbf{A} + (\varepsilon - \lambda_{min})\mathbf{I}$, com $\varepsilon > \lambda_{min}$.

A lei de ajuste do método de Newton torna-se:

$$\theta_{i+1} = \theta_i - \alpha_i \mathbf{M}_i^{-1} \nabla J(\theta_i), \tag{3.12}$$

onde $\mathbf{M}_i$ é dada por:

$$\begin{cases} \mathbf{M}_i = \nabla^2 J(\theta_i) & \text{se } \lambda_{\min}^{[i]} > 0 \\ \mathbf{M}_i = \nabla^2 J(\theta_i) + (\varepsilon - \lambda_{\min}^{[i]})\mathbf{I} & \text{se } \lambda_{\min}^{[i]} \leq 0 \end{cases}, \tag{3.13}$$

com $\lambda_{\min}^{[i]}$ o autovalor mínimo de $\mathbf{M}_i$. Ainda não existem resultados que conduzam à determinação automática de um valor ótimo para ε (VON ZUBEN, 1996).

A Figura 3.5 apresenta alguns exemplos da aplicação do método de Newton modificado na minimização de uma função monovariável $J(\theta)$, demonstrando que o ajuste do método de Newton é sempre no sentido de atender o critério. Os pontos '•' correspondem a $(\theta_i, J(\theta_i))$, enquanto que os pontos 'o' correspondem a $(\theta_{i+1}, J(\theta_{i+1}))$. A função $J(\theta)$ é representada pela curva de traço cheio, enquanto que $J_{quad}(\theta)$ (ou sua versão modificada) é representada pela curva tracejada.

Observe que nos gráficos da coluna da esquerda foi necessário positivar a hessiana, enquanto que nos gráficos da última coluna houve a necessidade de se utilizar $\alpha_i < 1$. Situações em que estes dois procedimentos sejam necessários simultaneamente também podem ocorrer.

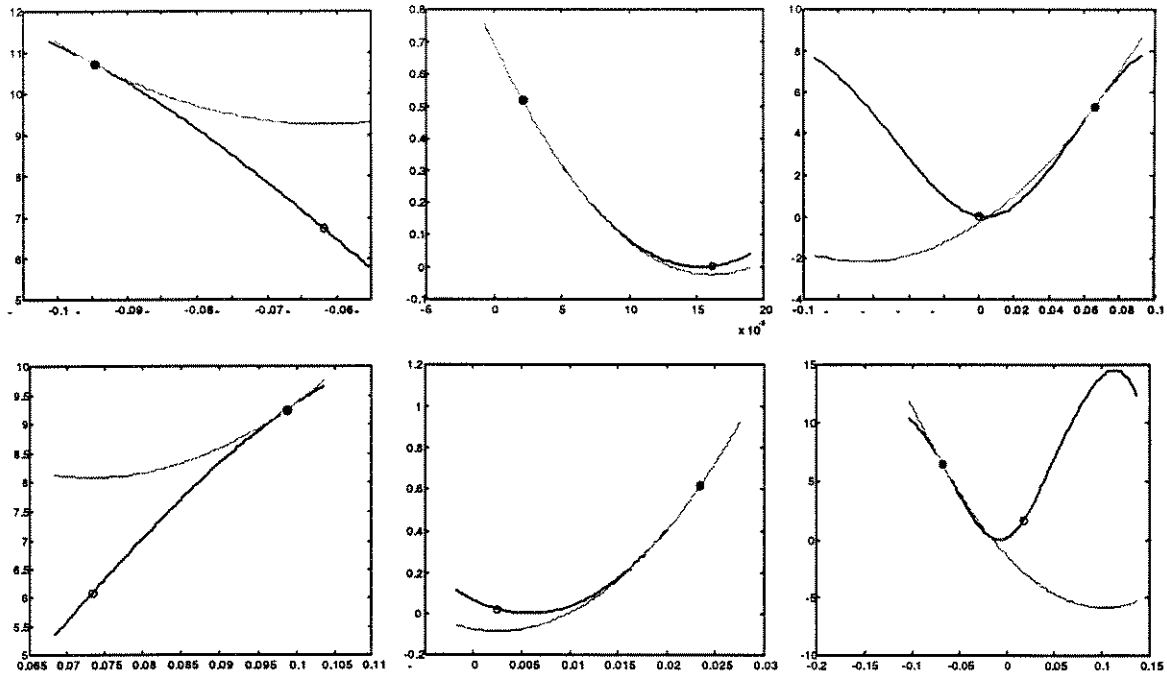


Figura 3.5: Exemplos da aplicação do método de Newton modificado.

Se ao invés de minimizar a função objetivo quiséssemos maximizá-la, bastaria apenas inverter a direção de busca, e negativar a matriz hessiana ($\nabla^2 J(\theta) < 0$) no lugar de positivá-la.

É importante mencionar ainda que em lugar de positivar $\mathbf{M}_i$ na equação (3.13), poder-se-ia, em princípio, utilizar qualquer outra matriz definida positiva de dimensões apropriadas. A razão para optar pelo processo de positivação da hessiana é a analogia com o tradicional método de Levenberg-Marquardt (que será visto a seguir) que pode ser interpretado como uma combinação entre a lei de ajuste do método do gradiente e a lei de ajuste do método de Newton.

3.6.2 Método de Levenberg-Marquardt (LM)

Este método, assim como o método de Newton, é bastante eficiente quando estamos tratando de redes que não possuem mais do que algumas centenas de conexões a serem ajustadas (HAGAN, 1994). Isto deve-se, principalmente, ao fato de que estes algoritmos necessitam armazenar uma matriz quadrada cuja dimensão é da ordem do número de conexões da rede (ver Capítulo 5).

O funcional de erro apresentado na equação (3.1) representa o erro quadrático médio (MSE). Se considerarmos como funcional a soma dos erros quadráticos (SSE), e ainda levarmos em conta que o problema pode ter múltiplas saídas, obtemos a seguinte expressão para o funcional de erro:

$$J(\theta) = \sum_{i=1}^{N} \sum_{j=1}^{m} \left(g_{ij}(\mathbf{x}) - \hat{g}_{ij}(\mathbf{x},\theta) \right)^2 = \sum_{k=1}^{q} r_l^2 \,, \tag{3.14}$$

onde N é o número de amostras, l o número de unidades intermediárias, r o erro residual, m o número de saídas, e q o produto $N \times m$.

Seja $\mathbf{J}$ o *Jacobiano* (matriz das derivadas primeiras) do funcional J dado pela equação (3.14). Esta matriz pode ser escrita da seguinte forma:

$$\mathbf{J} \equiv \begin{bmatrix} \nabla r_1^T \\ \vdots \\ \nabla r_q^T \end{bmatrix}, \tag{3.15}$$

onde r é denominado erro residual.

Diferenciando a equação (3.14) obtemos:

$$\nabla J = 2\mathbf{J}^T \mathbf{r} = 2 \sum_{k=1}^{q} r_k \nabla \mathbf{r}_k \,, \tag{3.16}$$

$$\nabla^2 J = 2\left(\mathbf{J}^T\mathbf{J} + \sum_{k=1}^{q} r_k \nabla^2 \mathbf{r}_k\right) \tag{3.17}$$

A matriz de derivadas segundas do funcional de erro é chamada de *matriz hessiana*, como foi visto anteriormente. Quando os erros residuais são suficientemente pequenos, a matriz hessiana pode ser aproximada pelo primeiro termo da equação (3.17), resultando em:

$$\nabla^2 J \approx 2\mathbf{J}^T\mathbf{J}. \tag{3.18}$$

Esta aproximação geralmente é válida em um mínimo de J para a maioria dos propósitos, e é a base para o *método de Gauss-Newton* (HAGAN, 1994). A lei de atualização torna-se então:

$$\Delta\theta = \left[\mathbf{J}^T\mathbf{J}\right]^{-1}\mathbf{J}^T\mathbf{r}. \tag{3.19}$$

A modificação de Levenberg-Marquardt para o método de Gauss-Newton é:

$$\Delta\theta = \left[\mathbf{J}^T\mathbf{J} + \mu\mathbf{I}\right]^{-1}\mathbf{J}^T\mathbf{r}. \tag{3.20}$$

O efeito da matriz adicional $\mu\mathbf{I}$ é adicionar μ a cada autovalor de $\mathbf{J}^T\mathbf{J}$. Uma vez que a matriz $\mathbf{J}^T\mathbf{J}$ é semi-definida positiva e portanto o autovalor mínimo possível é zero, qualquer valor positivo, pequeno, mas numericamente significativo, de μ será suficiente para restaurar a matriz aumentada e produzir uma direção descendente de busca.

Os valores de μ podem ser escolhidos de várias maneiras; a mais simples é escolhê-lo zero ao menos que a matriz hessiana encontrada na iteração i seja singular. Quando isso ocorrer, um valor pequeno como $\mu = 10^{-4}\sum_1^N \left(\mathbf{J}^T\mathbf{J}\right)_{ii}$ pode ser usado. Outras formas de determinação do parâmetro μ são sugeridas por HAGAN (1994), MCKEON *et. al.* (1997) e FINSCHI (1996).

É importante observar que, quanto maior for o valor de μ, menor é a influência da informação de segunda ordem e mais este algoritmo se aproxima de um método de primeira ordem.

A Figura 3.6 apresenta a curva de decaimento do erro quando o método LM é aplicado ao problema do Exemplo 3.1.

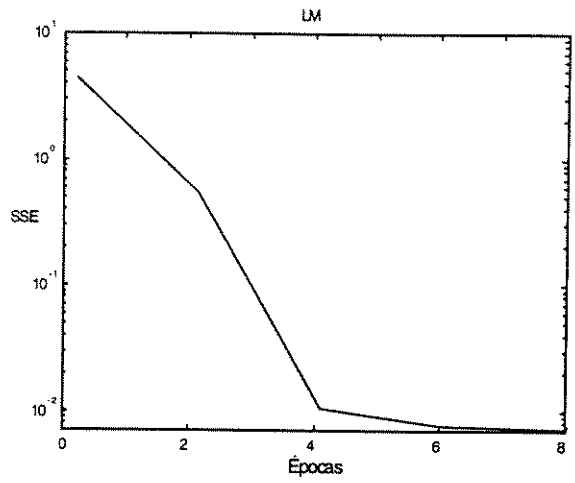


Figura 3.6: Comportamento do método Levenberg-Marquardt durante a minimização do funcional de erro do problema XOR. Este método também apresenta um comportamento monotonicamente decrescente da curva de erro, mas a aceleração do processo de convergência é ainda maior.

3.6.2.1 Algoritmo

O algoritmo utilizado neste método está resumido abaixo:

1. Atribua um valor inicial $\theta_0 \in \Re^P$ para o vetor de parâmetros e um valor arbitrariamente pequeno para uma constante $\varepsilon > 0$
2. Enquanto a condição de parada não for satisfeita, faça:
   2.1. Construa a matriz $\mathbf{J}$ utilizando as expressões (3.14) e (3.15)
   2.2. Calcule o ajuste utilizando a equação (3.20)
   2.3. Utilize um procedimento de busca unidimensional para encontrar um α_i que seja solução ótima do problema $\min_{\alpha_i \in (0,1]} J(\theta_i + \alpha_i \mathbf{d}_i)$.
   2.4. Lei de ajuste: $\theta_{i+1} = \theta_i - \alpha_i \Delta(\theta_i)$

3.6.3 Método de Davidon-Fletcher-Powell (DFP)

Este método, assim como o método BFGS que será apresentado em seguida, é classificado como método quase-Newton. A idéia por trás dos métodos quase-Newton é fazer uma aproximação iterativa da inversa da matriz hessiana, de forma que:

$$\lim_{i\to\infty} \mathbf{H}_i = \nabla^2 J(\boldsymbol{\theta})^{-1} \tag{3.21}$$

São considerados os métodos teoricamente mais sofisticados na solução de problemas de otimização não-linear irrestrita e representam o ápice do desenvolvimento de algoritmos através de uma análise detalhada de problemas quadráticos.

Para problemas quadráticos, gera as direções do método do gradiente conjugado (que será visto posteriormente) ao mesmo tempo que constrói a inversa da Hessiana. A cada passo a inversa da Hessiana é aproximada pela soma de duas matrizes simétricas de posto 1, procedimento que é geralmente chamado de correção de posto 2 (*rank 2 correction procedure*).

3.6.3.1 Construção da Inversa

A idéia é construir uma aproximação da inversa da matriz hessiana, utilizando informações de primeira ordem obtidas durante o processo iterativo de aprendizagem. A aproximação atual $\mathbf{H}_i$ é utilizada a cada iteração para definir a próxima direção descendente do método. Idealmente, as aproximações convergem para a inversa da matriz hessiana.

Suponha que o funcional de erro $J(\theta)$ tenha derivada parcial contínua até segunda ordem. Tomando dois pontos θ_i e θ_{i+1}, defina $\mathbf{g}_i = -\nabla J(\theta_i)^T$ e $\mathbf{g}_{i+1} = -\nabla J(\theta_{i+1})^T$. Se a hessiana, $\nabla^2 J(\boldsymbol{\theta})$, é constante, então temos:

$$\mathbf{q}_i \equiv \mathbf{g}_{i+1} - \mathbf{g}_i = \nabla^2 J(\theta)\mathbf{p}_i, \tag{3.22}$$

$$\mathbf{p}_i = \alpha_i \mathbf{d}_i. \tag{3.23}$$

Vemos então que a avaliação do gradiente em dois pontos fornece informações sobre a matriz hessiana $\nabla^2 J(\boldsymbol{\theta})$. Como $\theta \in \Re^P$, tomando-se P direções linearmente independentes $\{\mathbf{p}_0, \mathbf{p}_1, \ldots, \mathbf{p}_{P-1}\}$, é possível determinar unicamente $\nabla^2 J(\boldsymbol{\theta})$ caso se conheça $\mathbf{q}_i$, $i = 0, 1, \ldots, P-1$. Para tanto, basta aplicar iterativamente a equação (3.24) a seguir, com $\mathbf{H}_0 = \mathbf{I}_P$ (matriz identidade de dimensão P).

$$\mathbf{H}_{i+1} = \mathbf{H}_i + \frac{\mathbf{p}_i \mathbf{p}_i^T}{\mathbf{p}_i^T \mathbf{q}_i} - \frac{\mathbf{H}_i \mathbf{q}_i \mathbf{q}_i^T \mathbf{H}_i}{\mathbf{q}_i^T \mathbf{H}_i \mathbf{q}_i}, \; i = 0, 1, \ldots, P-1. \tag{3.24}$$

A Figura 3.7 apresenta a curva de decaimento do erro quando o método DFP é aplicado ao problema do Exemplo 3.1.

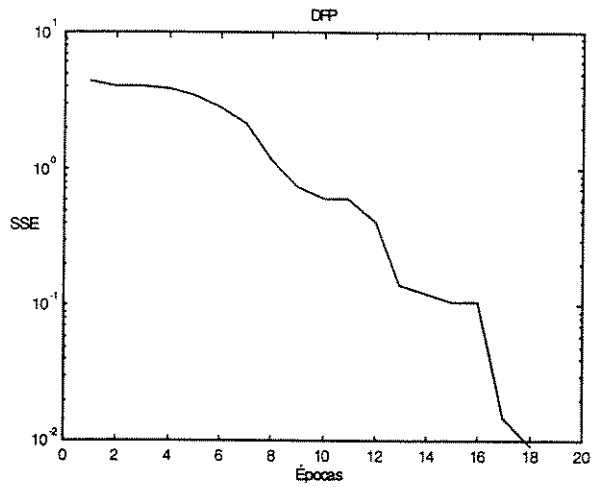


Figura 3.7: Comportamento do método DFP durante a minimização do funcional de erro do problema XOR. Verifica-se um comportamento monotonicamente decrescente da curva de erro, com grande velocidade de convergência.

Após P iterações sucessivas, se $J(\theta)$ for uma função quadrática, então $\mathbf{H}_P = \nabla^2 J(\boldsymbol{\theta})^{-1}$. Como geralmente não estamos tratando problemas quadráticos, a cada P iterações é recomendável que se faça uma reinicialização do algoritmo, ou seja, toma-se a direção de minimização como a direção oposta àquela dada pelo vetor gradiente e $\mathbf{H} = \mathbf{I}_P$.

3.6.3.2 Algoritmo

O algoritmo utilizado neste método está resumido abaixo:

1. Atribua um valor inicial $\theta_0 \in \Re^P$ para o vetor de parâmetros e um valor arbitrariamente pequeno para uma constante $\varepsilon > 0$
2. Defina $\mathbf{d}_0 = \mathbf{g}_0$, $\mathbf{H}_0 = \mathbf{I}$ e faça $i = 0$ ($\mathbf{g}_0 = -\nabla J(\theta_0)$)
3. Enquanto a condição de parada não for satisfeita, faça:
   3.1. Determine a direção $\mathbf{d}_i = \mathbf{H}_i \mathbf{g}_i$
   3.2. Se $(i \bmod P = 0)$, faça: $\mathbf{d}_i = \mathbf{g}_i$ e $\mathbf{H}_i = \mathbf{I}$
   3.3. Utilize um procedimento de busca unidimensional para encontrar um passo α_i que seja solução ótima do problema
   $$\min_{\alpha_i \in (0,1]} J(\theta_i + \alpha_i \mathbf{d}_i)$$
   3.4. Faça $\theta_{i+1} = \theta_i + \alpha_i \mathbf{d}_i$
   3.5. Calcule $\mathbf{p}_i = \alpha_i \mathbf{d}_i$, $\mathbf{g}_{i+1}$

3.6. Faça $\mathbf{q}_i = \mathbf{g}_{i+1} - \mathbf{g}_i$ de acordo com a expressão (3.22)

3.7. Calcule $\mathbf{H}_{i+1}$ pela expressão (3.24)

3.8. Faça $i = i + 1$

3.6.4 Método de Broyden-Fletcher-Goldfarb-Shanno (BFGS)

A diferença básica deste método para o proposto na seção anterior (DFP) é a forma da aproximação da inversa da matriz hessiana. A expressão que permite determinar a aproximação da inversa da hessiana é apresentada na equação (3.25).

$$\mathbf{H}_{i+1} = \mathbf{H}_i + \frac{\mathbf{p}_i \mathbf{p}_i^T}{\mathbf{p}_i^T \mathbf{q}_i}\left[1 + \frac{\mathbf{q}_i^T \mathbf{H}_i \mathbf{q}_i}{\mathbf{p}_i^T \mathbf{q}_i}\right] - \frac{\mathbf{H}_i \mathbf{q}_i \mathbf{p}_i^T + \mathbf{p}_i \mathbf{q}_i^T \mathbf{H}_i}{\mathbf{p}_i^T \mathbf{q}_i} \qquad (3.25)$$

Os vetores $\mathbf{q}_i$ e $\mathbf{p}_i$ são determinados como nas expressões (3.22) e (3.23), respectivamente.

3.6.4.1 Algoritmo

O algoritmo utilizado neste método está resumido abaixo:

1. Atribua um valor inicial $\theta_0 \in \Re^P$ para o vetor de parâmetros e um valor arbitrariamente pequeno para a constante $\varepsilon > 0$
2. Defina $\mathbf{d}_0 = \mathbf{g}_0$, $\mathbf{H}_0 = \mathbf{I}$ e faça $i = 0$ ($\mathbf{g}_0 = -\nabla J(\theta_0)$)
3. Enquanto a condição de parada não for satisfeita, faça:
   3.1. Determine a direção $\mathbf{d}_i = \mathbf{H}_i \mathbf{g}_i$
   3.2. Se $(i \bmod P = 0)$, faça: $\mathbf{d}_i = \mathbf{g}_i$ e $\mathbf{H}_i = \mathbf{I}$
   3.3. Utilize um procedimento de busca unidimensional para encontrar um passo α_i que seja solução ótima do problema $\min_{\alpha_i \in (0,1]} J(\theta_i + \alpha_i \mathbf{d}_i)$.
   3.4. Faça $\theta_{i+1} = \theta_i + \alpha_i \mathbf{d}_i$
   3.5. Calcule $\mathbf{p}_i = \alpha_i \mathbf{d}_i$, $\mathbf{g}_{i+1}$
   3.6. Faça $\mathbf{q}_i = \mathbf{g}_{i+1} - \mathbf{g}_i$ de acordo com a expressão (3.22)
   3.7. Calcule $\mathbf{H}_{i+1}$ pela expressão (3.25)
   3.8. Faça $i = i + 1$

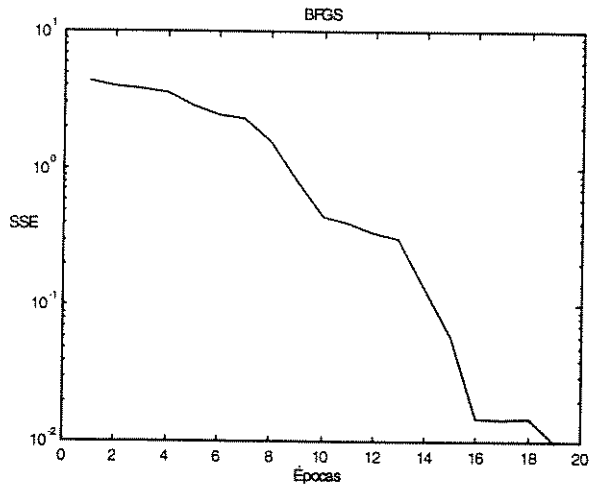


Figura 3.8: Comportamento do método BFGS durante a minimização do funcional de erro do problema XOR. Verifica-se um comportamento semelhante ao do método DFP.

A Figura 3.8 apresenta a curva de decaimento do erro quando o método BFGS é aplicado ao problema do Exemplo 3.1. Essa curva permite-nos verificar que o comportamento deste algoritmo, em relação à velocidade de convergência é similar ao do método DFP, visto anteriormente.

3.6.5 Método das Secantes de um Passo (OSS – *One Step Secant*)

O termo método de secante provém do fato de que as derivadas são aproximadas por secantes avaliadas em dois pontos da função (neste caso a função é o gradiente). Uma vantagem deste método apresentado por BATTITI (1992; 1994) é que sua complexidade é de ordem $O(P)$, ou seja, é linear em relação ao número P de parâmetros, enquanto a complexidade dos métodos DFP e BFGS é de ordem $O(P^2)$. Maiores detalhes sobre a complexidade computacional dos algoritmos citados aqui serão apresentados no Capítulo 5.

A principal razão da redução do esforço computacional deste método em relação aos anteriores (DFP e BFGS), é que agora a direção de atualização (eq. (3.3)) é calculada somente a partir de vetores determinados pelos gradientes, e não há mais a armazenagem da aproximação da inversa da hessiana.

A nova direção de busca $\mathbf{d}_{i+1}$ é obtida como segue:

$$\mathbf{d}_{i+1} = -\mathbf{g}_i + A_i\mathbf{s}_i + B_i\mathbf{q}_i, \tag{3.26}$$

onde:

$$\mathbf{s}_i = \theta_{i+1} - \theta_i = \mathbf{p}_i, \tag{3.27}$$

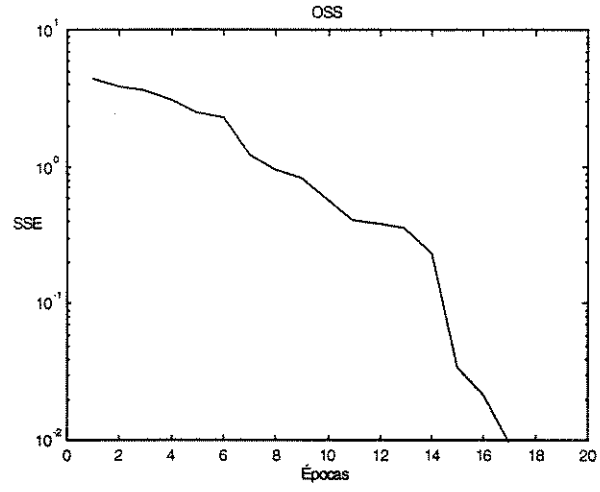


Figura 3.9: Comportamento do método OSS durante a minimização do funcional de erro do problema XOR.

$$A_i = -\left[1 + \frac{\mathbf{q}_i^T \mathbf{q}_i}{\mathbf{s}_i^T \mathbf{q}_i}\right] \frac{\mathbf{s}_i^T \mathbf{g}_i}{\mathbf{s}_i^T \mathbf{q}_i} + \frac{\mathbf{q}_i^T \mathbf{g}_i}{\mathbf{s}_i^T \mathbf{q}_i}; \qquad B_i = \frac{\mathbf{s}_i^T \mathbf{g}_i}{\mathbf{s}_i^T \mathbf{q}_i}. \tag{3.28}$$

Os vetores $\mathbf{q}_i$ e $\mathbf{p}_i$ são determinados como nas expressões (3.22) e (3.23), respectivamente.

A Figura 3.9 apresenta a curva de decaimento do erro quando o método OSS é aplicado ao problema do Exemplo 3.1.

3.6.5.1 Algoritmo

O algoritmo utilizado neste método está resumido abaixo:

1. Atribua um valor inicial $\theta_0 \in \Re^P$ para o vetor de parâmetros e um valor arbitrariamente pequeno para uma constante $\varepsilon > 0$
2. Defina $\mathbf{d}_0 = \mathbf{g}_0$, $\mathbf{H}_0 = \mathbf{I}$ e faça $i = 0$ ($\mathbf{g}_0 = -\nabla J(\theta_0)$)
3. Enquanto a condição de parada não for satisfeita, faça:
   3.1. Determine a direção $\mathbf{d}_i$ pelas expressões (3.22), (3.26), (3.27) e (3.28)
   3.2. Se $(i \bmod P = 0)$, faça: $\mathbf{d}_i = \mathbf{g}_i$
   3.3. Utilize um procedimento de busca unidimensional para encontrar um passo α_i que seja solução ótima do problema $\min_{\alpha_i \in (0,1]} J(\theta_i + \alpha_i \mathbf{d}_i)$.
   3.4. Faça $\theta_{i+1} = \theta_i + \alpha_i \mathbf{d}_i$
   3.5. Faça $i = i + 1$

3.6.6 Método do Gradiente Conjugado (GC)

Existe um consenso geral da comunidade de análise numérica que a classe de métodos de otimização chamados *métodos do gradiente conjugado*, tratam de problemas de grande escala de maneira efetiva (VAN DER SMAGT, 1994).

Os métodos do gradiente conjugado possuem sua estratégia baseada no modelo geral de otimização apresentado no algoritmo padrão e do gradiente, mas escolhem a direção de busca $\mathbf{d}_i$, o passo α_i e o coeficiente de momento β_i (equação (3.5)) mais eficientemente utilizando informações de segunda ordem. É projetado para exigir menos cálculos que o método de Newton e apresentar taxas de convergência maiores que as do método do gradiente.

Antes de apresentar o método do gradiente conjugado é necessário apresentar um resultado intermediário denominado método das direções conjugadas.

3.6.6.1 O método das Direções Conjugadas

A lei de adaptação dos processos em estudo assume a forma da equação (3.3), sendo que, havendo convergência, a solução ótima $\theta^* \in \Re^P$ pode ser expressa na forma:

$$\theta^* = \alpha_0\mathbf{d}_0 + \alpha_1\mathbf{d}_1 + \ldots = \sum_i \alpha_i\mathbf{d}_i \,.$$

Assumindo por hipótese que o conjunto $\{\mathbf{d}_0, \mathbf{d}_1, \ldots, \mathbf{d}_{P\text{-}1}\}$ forma uma base de $\Re^P$ e $\alpha = (\alpha_0 \ldots \alpha_{P\text{-}1})^T$ é a representação de θ^* nesta base, então é possível obter θ^* em P iterações da equação (3.3) na forma

$$\theta^* = \alpha_0\mathbf{d}_0 + \alpha_1\mathbf{d}_1 + \ldots + \alpha_{P-1}\mathbf{d}_{P-1} = \sum_{i=0}^{P-1} \alpha_i\mathbf{d}_i \,. \qquad (3.29)$$

Dada uma matriz **A** simétrica de dimensão $P \times P$, as direções $\mathbf{d}_i \in \Re^P$, $i = 0, \ldots, P\text{-}1$, são ditas serem **A**-conjugadas ou **A**-ortogonais se: $\mathbf{d}_j^T\mathbf{A}\mathbf{d}_i = 0$, para $i \neq j$ e $i,j = 0, \ldots, P\text{-}1$.

Se a matriz **A** for também definida positiva, o conjunto de P direções **A**-conjugadas forma uma base do $\Re^P$. Dessa forma os coeficientes $\alpha_j^*, j = 1,\ldots,P-1$, podem ser determinados pelo procedimento descrito abaixo.

Dada uma matriz **A** simétrica, definida positiva e de dimensão $P \times P$, multiplicando-se a equação (3.29) à esquerda por $\mathbf{d}_j^T\mathbf{A}$, com $0 \leq j \leq P\text{-}1$, resulta:

$$\mathbf{d}_j^T \mathbf{A}\theta^* = \sum_{i=0}^{P-1} \alpha_i^* \mathbf{d}_j^T \mathbf{A}\mathbf{d}_i, \qquad j = 1,...,P-1. \tag{3.30}$$

Escolhendo as direções $\mathbf{d}_i \in \Re^P$, como sendo **A**-conjugadas, é possível aplicar os resultados apresentados acima para obter:

$$\alpha_j^* = \frac{\mathbf{d}_j^T \mathbf{A}\theta^*}{\mathbf{d}_j^T \mathbf{A}\mathbf{d}_j}, \qquad j = 1,...,P-1. \tag{3.31}$$

É necessário eliminarmos θ^* da expressão (3.31), e para que isso seja feito são necessárias duas hipóteses adicionais:

- Suponha que o problema é quadrático, ou seja: $J(\theta) = \frac{1}{2}\theta^T \mathbf{Q}\theta - \mathbf{b}^T\theta$

  então no ponto de mínimo θ^*, é válida a expressão:

$$\nabla J(\theta^*) = 0 \Rightarrow \mathbf{Q}\theta^* - \mathbf{b} = 0 \Rightarrow \mathbf{Q}\theta^* = \mathbf{b}. \tag{3.32}$$

- Suponha que **A** = **Q**.

  Sendo assim, a equação (3.31) resulta em:

$$\alpha_j^* = \frac{\mathbf{d}_j^T \mathbf{b}}{\mathbf{d}_j^T \mathbf{A}\mathbf{d}_j}, \qquad j = 1,...,P-1, \tag{3.33}$$

  e o ponto de mínimo θ^* é dado por:

$$\theta^* = \sum_{j=0}^{P-1} \frac{\mathbf{d}_j^T \mathbf{b}}{\mathbf{d}_j^T \mathbf{A}\mathbf{d}_j} \mathbf{d}_j. \tag{3.34}$$

Assumindo solução iterativa com θ^* sendo expresso na forma da equação (3.34), os coeficientes $\alpha_j^*, j = 1,...,P-1$, são dados por:

$$\theta^* = \theta_0 + \alpha_0^* \mathbf{d}_0 + \alpha_1^* \mathbf{d}_1 + ... + \alpha_{P-1}^* \mathbf{d}_{P-1}, \tag{3.35}$$

$$\alpha_j^* = \frac{\mathbf{d}_j^T \mathbf{Q}(\theta^* - \theta_0)}{\mathbf{d}_j^T \mathbf{Q}\mathbf{d}_j}, \qquad j = 1,...,P-1. \tag{3.36}$$

Na j-ésima iteração, e levando-se em conta a equação (3.35), obtém-se:

$$\alpha_j^* = -\frac{\mathbf{d}_j^T \nabla J(\theta_j)}{\mathbf{d}_j^T \mathbf{Q}\mathbf{d}_j}, \qquad j = 1,...,P-1, \tag{3.37}$$

e a lei de ajuste do método das direções conjugadas é dada por:

$$\theta_{i+1} = \theta_i - \frac{\mathbf{d}_i^T \nabla J(\theta_i)}{\mathbf{d}_i^T \mathbf{Q} \mathbf{d}_i} \mathbf{d}_i. \tag{3.38}$$

3.6.6.2 O Método do Gradiente Conjugado

Antes de aplicarmos a lei de ajuste dada pela equação (3.38), é necessário obter as direções **Q**-conjugadas $\mathbf{d}_i \in \Re^P$, i=0,...,P-1. Uma maneira de determinar estas direções é tomá-las na forma (BAZARAA *et. al.*, 1993):

$$\begin{cases} \mathbf{d}_0 = -\nabla J(\theta_0) \\ \mathbf{d}_{i+1} = -\nabla J(\theta_{i+1}) + \beta_i \mathbf{d}_i \qquad i \geq 0 \end{cases}$$
$$\text{com} \quad \beta_i = \frac{\nabla J(\theta_{i+1})^T \mathbf{Q} \mathbf{d}_i}{\mathbf{d}_i^T \mathbf{Q} \mathbf{d}_i} \tag{3.39}$$

3.6.7 Problemas não quadráticos – Método de Polak-Ribière (PR)

As derivações das equações anteriores foram feitas supondo que estamos tratando de problemas quadráticos, o que nem sempre é verdade. Para que possamos adaptar as equações anteriores a problemas não-quadráticos, a matriz **Q** deve ser aproximada pela matriz hessiana calculada no ponto θ_i. A aplicação destes algoritmos a problemas não quadráticos envolve um procedimento de busca unidimensional do passo de ajuste (taxa de aprendizagem) e a aproximação do parâmetro β utilizando informações de primeira ordem (gradientes).

Uma destas aproximações é dada pelo método de Polak-Ribière. O algoritmo para este método torna-se então:

3.6.7.1 Algoritmo

O algoritmo utilizado neste método está resumido abaixo:

1. Atribua um valor inicial $\theta_0 \in \Re^P$ para o vetor de parâmetros e um valor arbitrariamente pequeno para a constante $\varepsilon > 0$
2. Calcule $\nabla J(\theta_0)$, faça $\mathbf{d}_0 = -\nabla J(\theta_0)$ e $i = 0$
3. Enquanto a condição de parada não for satisfeita, faça:
   3.1. Utilize um procedimento de busca unidimensional para encontrar um α_i que seja solução ótima do problema $\min_{\alpha_i \in (0,1]} J(\theta_i + \alpha_i \mathbf{d}_i)$ e faça $\theta_{i+1} = \theta_i + \alpha_i \mathbf{d}_i$
   3.2. Calcule $\mathbf{g}_i = -\nabla J(\theta_i)$;

3.3. Se $(i \bmod P \neq 0)$, faça $\mathbf{d}_{i+1} = \mathbf{g}_i + \beta_i \mathbf{d}_i$, onde $\beta_i = \dfrac{\mathbf{g}_{i+1}^T(\mathbf{g}_{i+1} - \mathbf{g}_i)}{\mathbf{g}_i^T \mathbf{g}_i}$

3.4. Senão, faça: $\mathbf{d}_i = \mathbf{g}_i$

3.5. Faça $i = i + 1$

A Figura 3.10 apresenta a curva de decaimento do erro quando o método de gradiente conjugado PR é aplicado ao problema do Exemplo 3.1.

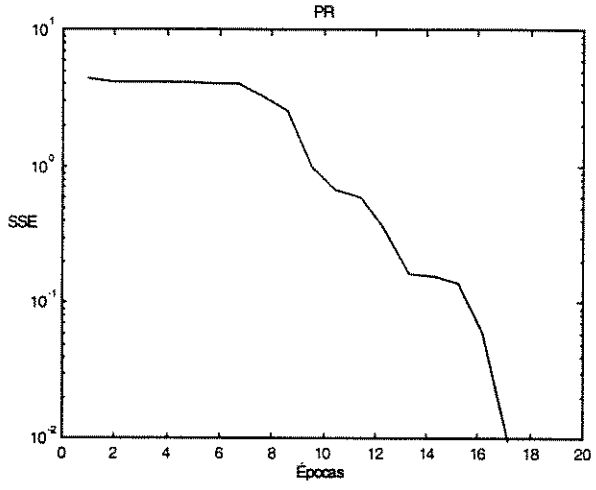


Figura 3.10: Comportamento do método PR durante a minimização do funcional de erro do problema XOR apresentado no Exemplo 3.1.

3.6.8 Problemas não quadráticos – Método de Fletcher & Reeves (FR)

É um método de direções conjugadas, assim como o de Polak-Ribière, mas diferencia-se pela forma como é determinado o parâmetro β do passo 3.3 do algoritmo anterior.

3.6.8.1 Algoritmo

O algoritmo utilizado neste método está resumido abaixo:

1. Atribua um valor inicial $\theta_0 \in \Re^P$ para o vetor de parâmetros e um valor arbitrariamente pequeno para a constante $\varepsilon > 0$
2. Calcule $\nabla J(\theta_0)$, faça $\mathbf{d}_0 = -\nabla J(\theta_0)$ e $i = 0$
3. Enquanto a condição de parada não for satisfeita, faça:

3.1. Utilize um procedimento de busca unidimensional para encontrar um α_i que seja solução ótima do problema $\min_{\alpha_i \in (0,1]} J(\theta_i + \alpha_i \mathbf{d}_i)$ e faça $\theta_{i+1} = \theta_i + \alpha_i \mathbf{d}_i$

3.2. Calcule $\mathbf{g}_i = -\nabla J(\theta_i)$;

3.3. Se $(i \bmod P \neq 0)$, faça $\mathbf{d}_{i+1} = \mathbf{g}_i + \beta_i \mathbf{d}_i$, onde $\beta_i = \dfrac{\|\mathbf{g}_{i+1}\|^2}{\|\mathbf{g}_i\|^2}$

3.4. Senão, faça: $\mathbf{d}_i = \mathbf{g}_i$

3.5. Faça $i = i + 1$

A Figura 3.11 apresenta a curva de decaimento do erro quando o método de FR é aplicado ao problema do Exemplo 3.1.

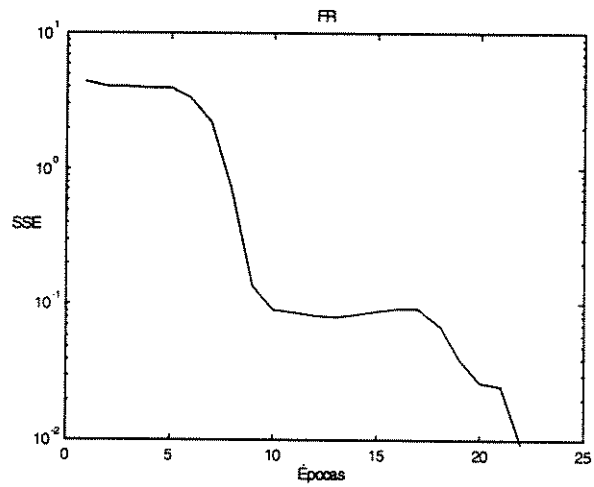


Figura 3.11: Comportamento do método FR durante a minimização do funcional de erro do problema XOR apresentado no Exemplo 3.1.

3.6.9 Método do Gradiente Conjugado Escalonado (SCG)

Os métodos de segunda ordem apresentados até agora utilizam um procedimento de busca unidimensional para a determinação da taxa de aprendizagem. A busca unidimensional envolve um grande número de avaliações da função ou de sua derivada, tornando o processo bastante custoso do ponto de vista computacional. MOLLER (1993) introduz uma nova variação no algoritmo de gradiente conjugado (Gradiente Conjugado Escalonado – SCG), que

evita a busca unidimensional a cada iteração utilizando uma abordagem de Levenberg-Marquardt cujo objetivo é fazer um escalonamento do passo de ajuste α.

Quando não estamos tratando com problemas quadráticos a matriz $\mathbf{Q}$ deve ser aproximada pela matriz hessiana calculada no ponto θ_i, e a equação (3.37) torna-se:

$$\alpha_j^* = -\frac{\mathbf{d}_j^T \nabla J(\theta_j)}{\mathbf{d}_j^T \nabla^2 J(\theta_j)\mathbf{d}_j}. \tag{3.40}$$

A idéia utilizada por Moller é estimar o termo denominado $\mathbf{s}_j = \nabla^2 J(\theta_j)\mathbf{d}_j$ do método do gradiente conjugado por uma aproximação da forma:

$$\mathbf{s}_j = \nabla^2 J(\theta_j)\mathbf{d}_j \approx \frac{\nabla J(\theta_j + \sigma_j \mathbf{d}_j) - \nabla J(\theta_j)}{\sigma_j}, \quad 0 < \sigma_j << 1. \tag{3.41}$$

A aproximação tende, no limite, ao valor de $\nabla^2 J(\theta_j)\mathbf{d}_j$. Combinando esta estratégia com a abordagem de gradiente conjugado e Levenberg-Marquardt, obtém-se um algoritmo diretamente aplicável ao treinamento de redes MLP. Isso é feito da seguinte maneira:

$$\mathbf{s}_j = \frac{\nabla J(\theta_j + \sigma_j \mathbf{d}_j) - \nabla J(\theta_j)}{\sigma_j} + \lambda_j \mathbf{d}_j. \tag{3.42}$$

Seja δ_j o denominador da equação (3.40), então utilizando a expressão (3.41), resulta:

$$\delta_j = \mathbf{d}_j^T \mathbf{s}_j \tag{3.43}$$

O ajuste do parâmetro λ_j a cada iteração e a análise do sinal de δ_j permitem determinar se a Hessiana é definida-positiva ou não.

A aproximação quadrática $J_{quad}(\theta)$, utilizada pelo algoritmo, nem sempre representa uma boa aproximação para $J(\theta)$, uma vez que λ_j escalona a matriz hessiana de maneira artificial. Um mecanismo para aumentar e diminuir λ_j é necessário para fornecer uma boa aproximação, mesmo quando a matriz for definida-positiva. Defina:

$$\begin{aligned} \Delta_j &= \frac{J(\theta_j) - J(\theta_j + \alpha_j \mathbf{d}_j)}{J(\theta_j) - J_{quad}(\alpha_j \mathbf{d}_j)} \\ &= \frac{2\delta_j \left[J(\theta_j) - J(\theta_j + \alpha_j \mathbf{d}_j)\right]}{\mu_j^2}, \end{aligned} \tag{3.44}$$

onde $\mu_j = -\mathbf{d}_j^T \nabla J(\theta_j)$.

O termo Δ_j representa uma medida da qualidade da aproximação $J_{quad}(\theta)$ em relação a $J(\theta_j + \alpha_j \mathbf{d}_j)$ no sentido de que quanto mais próximo de 1 estiver Δ_j, melhor é a aproximação. O algoritmo do gradiente conjugado escalonado resultante é apresentado a seguir.

3.6.9.1 Algoritmo

O algoritmo utilizado neste método está resumido abaixo:

1. Escolha um vetor de parâmetros inicial θ_0 e escalares $0 < \sigma << 1$ e $\lambda_0 = 0$;
2. Faça: $p_0 = r_0 = -\nabla J(\theta_0); j = 0;$ sucesso = 1
3. Se sucesso = 1, calcule a informação de 2ª ordem: $\sigma_j = \dfrac{\sigma}{\|\mathbf{d}_j\|}$

$$\mathbf{s}_j = \frac{\nabla J(\theta_j + \sigma_j \mathbf{d}_j) - \nabla J(\theta_j)}{\sigma_j} + \lambda_j \mathbf{d}_j; \qquad \delta_j = p_j^T s_j$$

4. Se $\delta_j \leq 0$, então faça a matriz hessiana definida positiva:

$$\lambda_j^N = 2\left[\lambda_j - \frac{\delta_j}{\|\mathbf{d}_j\|^2}\right]; \qquad \delta_j = \delta_j + (\lambda_j^N - \lambda_j)\|\mathbf{d}_j\|^2; \quad \lambda_j = \lambda_j^N$$

5. Calcule a taxa de ajuste:

$$\mu_j = p_j^T r_j; \quad \alpha_j = \mu_j / \delta_j$$

6. Calcule o parâmetro de comparação:

$$\Delta_j = \frac{2\delta_j [J(\theta_j) - J(\theta_j + \alpha_j \mathbf{d}_j)]}{\mu_j^2}$$

7. Se $\Delta_j \geq 0$ (o erro pode ser reduzido), então atualize o vetor de pesos:

$$\theta_{j+1} = \theta_j + \alpha_j \mathbf{d}_j; \qquad r_{j+1} = -\nabla J(\theta_{j+1});$$

8. Se (j mod P) = 0, então reinicialize o algoritmo:

$p_{j+1} = r_{j+1}$.

Senão, defina uma nova direção conjugada:

$$\beta_j = \frac{\|\mathbf{r}_{j+1}\|^2 - \mathbf{r}_{j+1}^T \mathbf{r}_j}{\mu_j}; \qquad p_{j+1} = r_{j+1} + \beta_j p_j$$

9. Se $\Delta_j \geq 0.75$, então faça $\lambda_j = 0.5\lambda_j$.

10. Se $\Delta_j \leq 0.25$, então faça $\lambda_j = 4\lambda_j$.

    Senão, uma redução no erro não é possível: sucesso = 0;

11. Se $r_{j+1} > \varepsilon$, onde $\varepsilon \rightarrow 0$, então faça: $j = j+1$; retorne ao passo 3.

12. Senão, o procedimento de ajuste chegou ao fim e θ_{j+1} é o ponto de mínimo.

3.6.9.2 Cálculo Exato da Informação de Segunda Ordem

O elevado custo computacional associado ao cálculo e armazenagem da hessiana $\nabla^2 J(\theta)$ a cada iteração pode ser reduzido drasticamente aplicando-se os resultados de PEARLMUTTER (1994). Além de permitir o cálculo exato da informação de segunda ordem, o custo computacional associado é da mesma ordem que o exigido para o cálculo da informação de primeira ordem.

Utilizando um operador diferencial é possível calcular exatamente o produto entre a matriz $\nabla^2 J(\theta)$ e qualquer vetor desejado, sem a necessidade de se calcular e armazenar a matriz $\nabla^2 J(\theta)$. Este resultado é de grande valia para os métodos de gradiente conjugado, em particular para o gradiente conjugado escalonado de Moller, onde a hessiana $\nabla^2 J(\theta)$ aparece invariavelmente multiplicada por um vetor.

Expandindo o vetor gradiente $\nabla J(\theta)$ em torno de um ponto $\theta \in \Re^P$ resulta:

$$\nabla J(\theta + \Delta\theta) = \nabla J(\theta) + \nabla^2 J(\theta)\Delta\theta + O\left(\|\Delta\theta\|^2\right), \tag{3.45}$$

onde $\Delta\theta$ representa uma pequena perturbação. Escolhendo $\Delta\theta = a\mathbf{v}$, com a uma constante positiva próxima de zero e $\mathbf{v} \in \Re^P$ um vetor arbitrário, é possível calcular $\nabla^2 J(\theta)\mathbf{v}$ como segue:

$$\nabla^2 J(\theta)\mathbf{v} = \frac{1}{a}\left[\nabla J(\theta + a\mathbf{v}) - \nabla J(\theta) + O\left(a^2\right)\right] = \frac{\nabla J(\theta + a\mathbf{v}) - \nabla J(\theta)}{a} + O(a). \tag{3.46}$$

Tomando o limite quando $a \rightarrow 0$,

$$\nabla^2 J(\theta)\mathbf{v} = \lim_{a \rightarrow 0} \frac{\nabla J(\theta + a\mathbf{v}) - \nabla J(\theta)}{a} = \frac{\partial}{\partial a} \nabla J(\theta + a\mathbf{v})\Big|_{a=0}. \tag{3.47}$$

Portanto, definindo um operador diferencial

$$\Psi_v\{f(\theta)\} = \frac{\partial}{\partial a}\nabla J(\theta + a\mathbf{v})\Big|_{a=0}. \tag{3.48}$$

É possível aplicá-lo a todas as operações realizadas para se obter o gradiente, produzindo

$$\Psi_v\{\nabla J(\theta)\} = \nabla^2 J(\theta)\mathbf{v} \quad \text{e} \quad \Psi_v\{\theta\} = \mathbf{v} \tag{3.49}$$

Por ser um operador diferencial, $\Psi_v(\theta)$ obedece as regras usuais de diferenciação. A aplicação deste operador nas equações que fazem a retropropagação do erro nos algoritmos das redes MLP torna possível o cálculo exato da informação de segunda ordem que é diretamente aplicável aos métodos de gradiente conjugado.

3.6.9.3 Gradiente Conjugado Escalonado Modificado (SCGM)

Neste algoritmo, as principais mudanças em relação ao algoritmo anterior são:

- Passo 1: não é necessário definir o parâmetro σ.
- Passo 3: s_j não é aproximado pela equação (3.42), mas calculado pela expressão (3.50) a seguir:

$$\mathbf{s}_j = \nabla^2 J(\theta_j)\mathbf{d}_j + \lambda_j \mathbf{d}_j \tag{3.50}$$

A Figura 3.12 mostra a curva de decaimento do erro quando o método do gradiente conjugado escalonado com cálculo exato da informação de segunda ordem é aplicado ao problema do Exemplo 3.1. Verifica-se que o algoritmo possui um desempenho quase comparável ao método de LM.

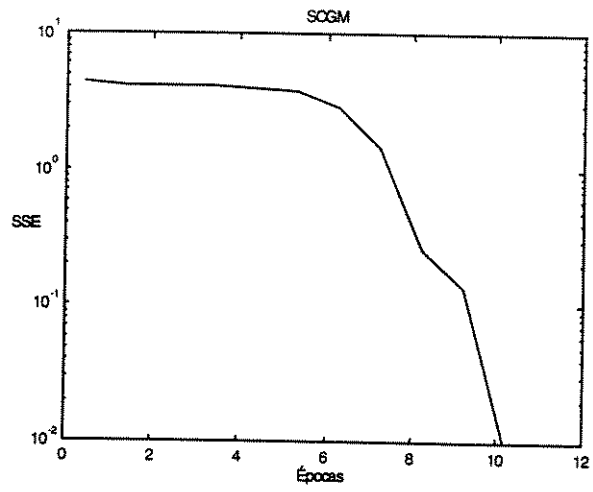


Figura 3.12: Comportamento do método SCGM durante a minimização do funcional de erro do problema XOR apresentado no Exemplo 3.1.

3.7 Métodos Empíricos

Métodos heurísticos, ou empíricos, são aqueles geralmente verificados através de experimentos, mas que não possuem formalização matemática adequada.

Neste item serão apresentadas algumas modificações empíricas que podem ser feitas no algoritmo padrão com o objetivo de melhorar seu desempenho. Dentre estas modificações serão vistos o coeficiente de momento de segunda ordem, o ganho da função de ativação, a normalização dos dados de entrada e sua importância e a reinicialização do treinamento. Por último será apresentado o algoritmo Quickprop.

3.7.1 Momento de Segunda Ordem

Como visto na equação (3.5), a inclusão de um termo de momento na expressão de atualização dos parâmetros da rede, permite um aumento na velocidade de convergência do algoritmo.

Uma maneira simples de aplicar um termo de momento de segunda ordem ao algoritmo de *backpropagation* foi apresentado por PEARLMUTTER (1992). A modificação à regra de atualização apresentada pela equação (3.5), inclui o decaimento do vetor de parâmetros considerando duas iterações anteriores, ou seja:

$$\theta_{i+1} = \theta_i + \alpha \mathbf{d}_i + \beta \Delta\theta_{i-1} + \gamma \Delta\theta_{i-2}, \qquad i \geq 0 \tag{3.51}$$

O autor recomenda um valor de $\gamma = \frac{\beta - 1}{3}$.

3.7.2 Variação do Ganho da Função de Ativação

THIMM *et. al.* (1996) provam que variar o ganho da função de ativação é equivalente a variar a taxa de aprendizagem junto ao vetor de parâmetros ajustáveis θ. Isto simplifica a regra de aprendizagem eliminando um destes parâmetros.

A função de ativação geralmente escolhida para os neurônios de uma rede do tipo MLP é a função logística ou sigmóide:

$$f(x) = \frac{\gamma}{1 + e^{-\beta x}} \tag{3.52}$$

que abrange o intervalo $(0,\gamma]$. Outras escolhas comuns para a função de ativação $f(.)$ são a tangente hiperbólica, $\gamma \tanh(\beta x)$, cujos valores de saída pertencem ao intervalo $(-\gamma, \gamma)$ e a

função Gaussiana $\gamma e^{-(\beta x)^2}$ com intervalo $(0,\gamma]$. O parâmetro β é chamado de *ganho*, e $\gamma.\beta$ de *decaimento* (*slope*) da função de ativação.

Duas redes M e N de mesma topologia, cujas funções de ativação $f(\cdot)$, ganho β, taxa de aprendizagem α, e vetor de parâmetros (pesos e limiares) θ, estão relacionados como na Tabela 3.1, são equivalentes sob o ponto de vista do algoritmo *back-propagation*, o que implica que quando os mesmos padrões de entrada forem apresentados as redes, ambas fornecerão as mesmas saídas para condições iniciais idênticas. Isso significa que um aumento do ganho por um fator β pode ser compensado dividindo-se o vetor de parâmetros inicial por β e a taxa de aprendizagem por β^2.

Tabela 3.1: Relação entre função de ativação, ganho, pesos e taxa de aprendizagem.

	RNA M	RNA N
Função de ativação	$f(x)$	$\overline{f}(x) = f(\beta x)$
Ganho	$\beta = 1$	$\overline{\beta} = \beta$
Taxa de aprendizagem	α	$\overline{\alpha} = \beta^2 \alpha$
Pesos	θ	$\overline{\theta} = \beta\theta$

3.7.3 Normalização dos Dados de Entrada

A influência dos dados de entrada no treinamento de uma RNA pode ser vista da seguinte maneira: se duas entradas, conectadas à mesma unidade, possuem intervalos de variação muito diferentes, os pesos ligados a estas unidades devem ser atualizados de maneira equalizada. Este processo de adaptação dinâmica dos intervalos pode ser muito demorado do ponto de vista computacional quando há uma variação grande nos intervalos de valores das diferentes entradas (JOOST & SCHIFFMAN, 1993).

Estes problemas podem ser eliminados utilizando-se uma simples transformação linear nos padrões de entrada, resultando em uma *normalização*:

$$\overline{x}_{i,n} = \frac{x_{i,n} - \overline{x}_i}{\sigma_i}, \tag{3.53}$$

$x_{i,n}$ denota o i-ésimo componente do n-ésimo vetor de entrada $\mathbf{x}_n$; $\bar{x}_i = \frac{1}{N}\sum_{n=1}^{N} x_{i,n}$ corresponde à média e $\sigma_i = \sqrt{\frac{1}{N-1}\sum_{n=1}^{N}(x_{i,n} - \bar{x}_{i,n})^2}$ é o desvio padrão dos dados de entrada originais.

Outro tipo de normalização possível é o escalonamento dos dados no intervalo [-1.0, 1.0] pela expressão:

$$x_{i,n} = 2 \times \frac{x_{i,n} - x^i_{[\min]}}{x^i_{[\max]} - x^i_{[\min]}} - 1, \tag{3.54}$$

onde $x^i_{[\min]}$ é o menor valor do i-ésimo vetor de entrada e $x^i_{[\max]}$ é o maior valor do i-ésimo vetor de entrada.

É importante mencionar que a normalização imposta aos dados é um processo reversível.

3.7.4 Reinicialização do Treinamento

O procedimento de reinicialização do treinamento pode ser efetuado em dois casos distintos:

- Primeiro: se uma rede não é capaz de resolver determinado problema em 100% dos casos (podendo estar presa em um mínimo local, por exemplo) então existe um ponto do treinamento depois do qual o esforço é desperdiçado (FAHLMAN, 1988). HAMEY (1992) apresenta algumas medidas de determinação da parte do treinamento na qual devemos fazer a reinicialização.
- Segundo: quando estamos utilizando métodos de segunda ordem como métodos de gradiente conjugado e métodos de quase-Newton, deve ser feita a reinicialização do treinamento a cada P iterações, pois grande parte destes métodos são deduzidos considerando-se problemas quadráticos. Testes realizados com vários algoritmos de segunda ordem mostraram que para redes com grandes dimensões um comportamento mais homogêneo da curva de erro (monotonicamente decrescente sem a existência de grandes platôs) é apresentado quando o processo de reinicialização é feito em um número menor do que P iterações. Este resultado também é comentado por SCHEWCHUK (1994) e HAGAN (1994).

3.7.5 Método Quickprop (QUICK)

Como último algoritmo de treinamento citado neste capítulo temos o algoritmo Quickprop de FAHLMAN (1988) um dos mais conhecidos de todos. Fahlman apresenta uma nova lei de ajuste para o vetor de parâmetros θ, adicionando o termo de momento à equação (3.5), mas calculando-o de maneira mais elaborada pela equação (3.55).

$$\beta_i = \frac{\nabla J(\theta_i)}{\nabla J(\theta_{i-1}) - \nabla J(\theta_i)} \tag{3.55}$$

Fahlman também introduz um *fator de crescimento máximo* (μ) que é utilizado como um limite superior para a variação do conjunto de pesos; evitando assim, grandes oscilações.

3.7.5.1 Algoritmo

O algoritmo utilizado neste método está resumido abaixo:

1. Atribua um valor inicial $\theta_0 \in \Re^P$ para o vetor de parâmetros e um valor arbitrariamente pequeno para uma constante $\varepsilon > 0$
2. Defina valores para o parâmetro α
3. Enquanto a condição de parada não for satisfeita, faça:
   3.1. Calcule β_i pela expressão (3.55)
   3.2. $\theta_{i+1} = \theta_i - \alpha \nabla J(\theta_i) + \beta_i \Delta\theta$
   3.3. Faça $i = i + 1$

Exemplo 3.2:

Para sintetizar o comportamento geral dos métodos estudados neste capítulo, considere o problema de minimizar a função $f(x_1, x_2) = (x_1 - 2)^4 + (x_1 - 2x_2)^2$, cujo mínimo ($f_{\min} = 0$) encontra-se claramente no ponto (2.0, 1.0). Serão aplicados 4 métodos para a solução deste problema, os métodos do gradiente (GRAD), Newton (MN), gradiente conjugado (GC) e quase-Newton (DFP). O enfoque dado neste exemplo será o de otimização não-linear irrestrita (BAZARAA *et. al.*, 1993).

Para verificar a evolução dos processos de otimização, listaremos os resultados dos processos de busca do mínimo na forma de uma seqüência, e em seguida apresentaremos um gráfico com o comportamento do algoritmo. A seqüência de pontos irá mostrar a evolução do vetor $\mathbf{x}_i$ ($\mathbf{x}_i = (x_{1i}, x_{2i})$), da norma do gradiente $\|\nabla f(\mathbf{x}_i)\|$ avaliada a cada iteração i e do tamanho do passo α_i. O número total de iterações, e o valor da função ao final do processo também

serão apresentados. O valor do parâmetro ε desejado é ε = 0.001. O procedimento de busca unidimensional do passo α utilizado foi o método da seção áurea que será abordado no próximo capítulo.

1. Método do Gradiente (GRAD)

A Figura 3.13 (a) e (b) apresenta os resultados deste método para as condições iniciais (a) e (b), respectivamente.

Caso (a) – $(x_1, x_2) = (3, 0)$

Tomando o ponto acima como ponto de partida do algoritmo, a seqüência de pontos gerada é apresentada na Tabela 3.2. Devido a grande quantidade de pontos gerados pelas iterações serão suprimidos alguns pontos da seqüência.

Tabela 3.2: Resumo da seqüência de pontos gerada para o método do gradiente com condição inicial $(x_1, x_2) = (3, 0)$.

Iteração i	$(x_{1i}$ $x_{2i})$	$\|\nabla f(\mathbf{x}_i)\|$	α_i
01)	(2.118755 1.057494)	15.62049	0.088125
05)	(2.106898 1.052058)	0.005116	1.117497
10)	(2.097000 1.048790)	0.011799	0.108111
15)	(2.087662 1.043047)	0.002720	1.186256
20)	(2.081826 1.041087)	0.007121	0.108111
25)	(2.075885 1.037430)	0.001730	1.216730
30)	(2.071957 1.036096)	0.004841	0.108111
35)	(2.067767 1.033516)	0.001217	1.242158
40)	(2.064903 1.032538)	0.003562	0.108111
41)	(2.063969 1.031676)	0.001018	1.247395
42)	(2.063723 1.031943)	0.003365	0.108111
43)	(2.063723 1.031943)	0.000964	0.108111

Número total de iterações: 43.

Argmin ($\mathbf{x}^*$): (2.063723 1.031943).

Mínimo da função $f(\mathbf{x}^*)$: 0.000017.

Caso (b) – $(x_1, x_2) = (0, 0)$

Tomando o ponto acima como ponto de partida do algoritmo, a seqüência de pontos gerada é apresentada na Tabela 3.3. Devido a grande quantidade de pontos gerados pelas iterações serão suprimidos alguns pontos da seqüência.

Tabela 3.3: Resumo da seqüência de pontos gerada para o método do gradiente com condição inicial $(x_1, x_2) = (0, 0)$.

Iteração i	$(x_{1i}$ $x_{2i})$	$\|\nabla f(\mathbf{x}_i)\|$	α_i
1)	(1.162006 0.000000)	32.000000	0.036313
10)	(1.714274 0.857222)	0.184362	0.125900
20)	(1.815807 0.907925)	0.049324	0.125901
30)	(1.855242 0.927633)	0.023960	0.125900
40)	(1.877256 0.938634)	0.014590	0.125901
50)	(1.891694 0.945851)	0.010028	0.125900
60)	(1.902062 0.951034)	0.007402	0.125901
70)	(1.909967 0.954987)	0.005767	0.125900
80)	(1.916239 0.958121)	0.004633	0.125901
90)	(1.921373 0.960689)	0.003837	0.125900
100)	(1.925674 0.962838)	0.003238	0.125901
110)	(1.929344 0.964673)	0.002784	0.125900
120)	(1.932523 0.966263)	0.002425	0.125900
130)	(1.935311 0.967656)	0.002134	0.125901
139)	(1.937310 0.968656)	0.000989	0.125901

Número de iterações 139.

Argmin (**x***): (1.937310 0.968656).

Mínimo da função $f(\mathbf{x}^*)$: 0.000015.

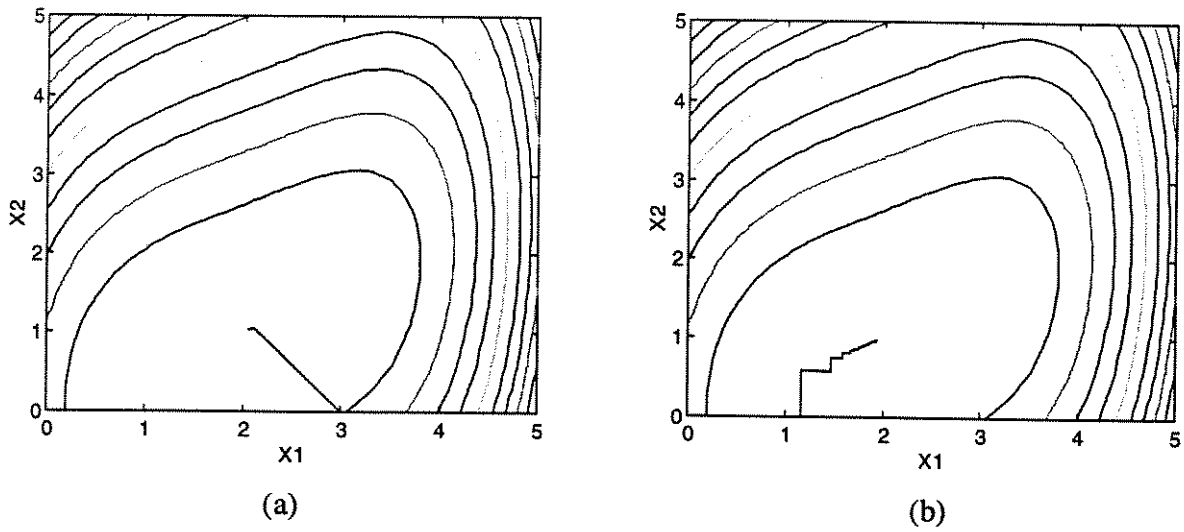


Figura 3.13: Comportamento iterativo do método do gradiente. (a) Ponto inicial $(x_1, x_2) = (3, 0)$. (b) Ponto inicial $(x_1, x_2) = (0, 0)$.

2. Método de Newton (MN)

A Figura 3.14 (a) e (b) apresenta os resultados deste método para a condição inicial (a) e (b), respectivamente.

Caso (a) – $(x_1, x_2) = (3, 0)$

Tomando o ponto acima como ponto de partida do algoritmo, a seqüência de pontos gerada é apresentada na Tabela 3.4.

Tabela 3.4: Seqüência de pontos gerada para o método de Newton com condição inicial $(x_1, x_2) = (3, 0)$.

Iteração i	$(x_{1i}$ $x_{2i})$	$\|\nabla f(\mathbf{x}_i)\|$	α_i
01)	(2.659572 1.361712)	15.620499	1.021284
02)	(2.212482 1.073243)	1.051533	2.033548
03)	(2.135875 1.070630)	0.314179	1.081591
04)	(2.060011 1.028188)	0.021551	1.675013
05)	(2.036949 1.018752)	0.016659	1.152874
06)	(2.018455 1.009088)	0.002401	1.501576
07)	(2.011006 1.005532)	0.001258	1.210975
08)	(2.011006 1.005532)	0.000261	1.210975

Número de iterações 8.

Argmin (**x***): (2.011006 1.005532).

Mínimo da função f(**x***) 0.000000.

Caso (b) – $(x_1, x_2) = (0, 0)$

Tomando o ponto acima como ponto de partida do algoritmo, a seqüência de pontos gerada é apresentada na Tabela 3.5.

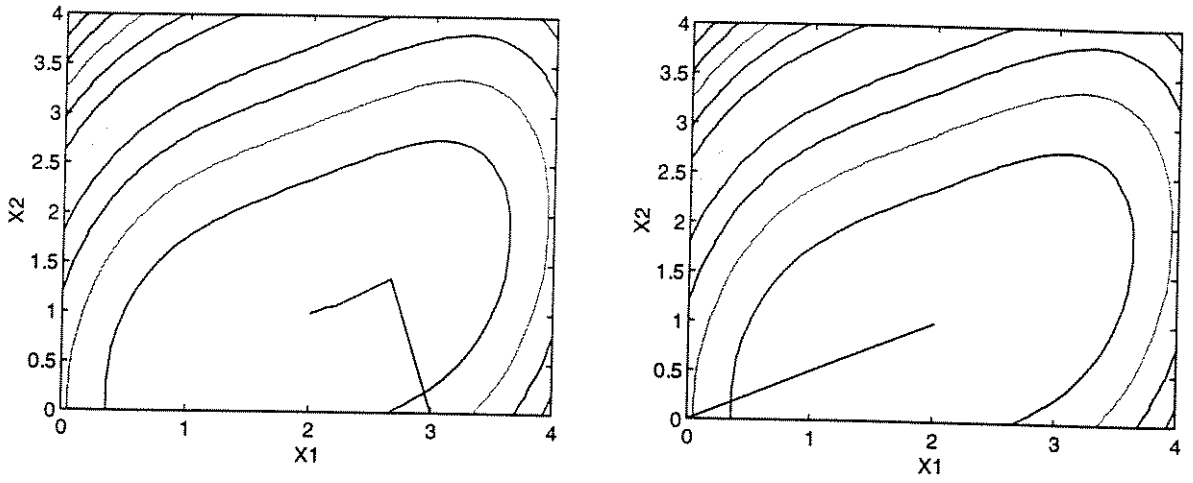
Tabela 3.5: Resumo da seqüência de pontos gerada para o método de Newton com condição inicial $(x_1, x_2) = (0, 0)$.

Iteração i	$(x_{1i}$ $x_{2i})$	$\|\nabla f(\mathbf{x}_i)\|$	α_i
01)	(1.999842 0.999921)	32.000000	2.999763
02)	(1.999842 0.999921)	0.000000	2.999763

Número de iterações 2.

Argmin (**x***): (1.999842 0.999921).

Mínimo da função f(**x***) 0.000000.



(a) (b)

Figure 3.14: Comportamento iterativo do método de Newton. (a) Ponto inicial $(x_1, x_2) = (3, 0)$. (b) Ponto inicial $(x_1, x_2) = (0, 0)$.

3. Método do Gradiente Conjugado (GC)

A Figura 3.15 (a) e (b) apresenta os resultados deste método para as condições iniciais (a) e (b), respectivamente.

Caso (a) – $(x_1, x_2) = (3, 0)$

Tomando o ponto acima como ponto de partida do algoritmo, a seqüência de pontos gerada é apresentada na Tabela 3.6.

Tabela 3.6: Seqüência de pontos gerada para o método do gradiente conjugado com condição inicial $(x_1, x_2) = (3, 0)$.

Iteração i	$(x_{1i}$ $x_{2i})$	$\|\nabla f(\mathbf{x}_i)\|$	α_i
1)	(2.118755 1.057494)	15.620499	00.088125
2)	(2.117214 1.059125)	00.020722	00.108111
3)	(2.029036 1.013706)	00.006022	15.822610
4)	(2.028702 1.014355)	00.007304	00.099835
5)	(2.028702 1.014355)	00.000085	00.099835

Número de iterações 5.

Argmin (**x***): (2.028702 1.014355).

Mínimo da função $f(\mathbf{x}^*)$ 0.000001.

Caso (b) – $(x_1, x_2) = (0, 0)$

Tomando o ponto acima como ponto de partida do algoritmo, a seqüência de pontos gerada é apresentada na Tabela 3.7. Devido a grande quantidade de pontos gerados pelas iterações serão suprimidos alguns pontos da seqüência.

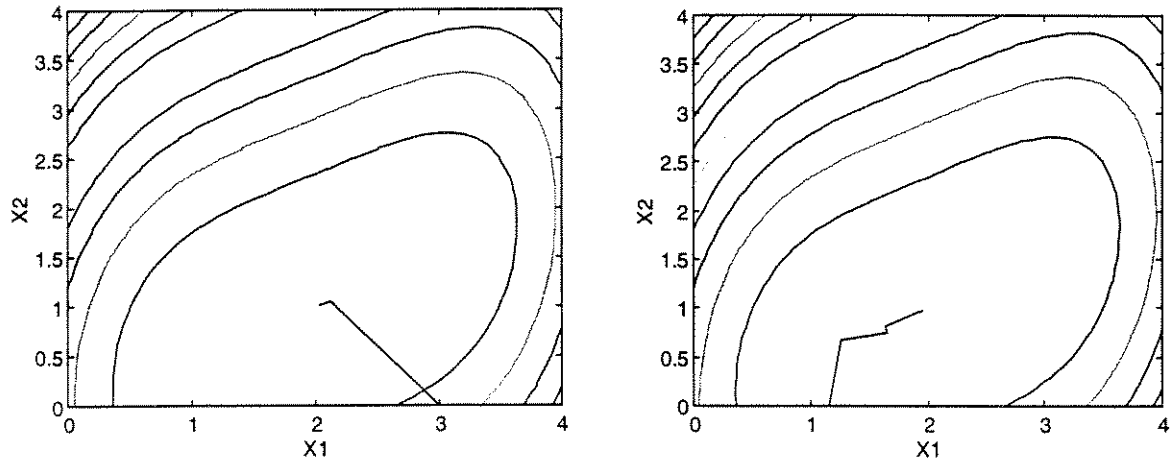
Tabela 3.7: Resumo da seqüência de pontos gerada para o método do gradiente conjugado com condição inicial $(x_1, x_2) = (0, 0)$.

Iteração i	$(x_{1i}$ $x_{2i})$	$\|\|\nabla f(\mathbf{x}_i)\|\|$	α_i
1)	(1.162006 0.000000)	32.000000	00.036313
2)	(1.262722 0.663997)	04.648119	00.142856
3)	(1.649854 0.748468)	01.753171	00.211096
4)	(1.635692 0.813055)	00.626202	00.105591
5)	(1.921681 0.967076)	00.178406	01.750728
6)	(1.924363 0.962096)	00.056659	00.099835
7)	(1.953601 0.975548)	00.001548	20.761041
8)	(1.953139 0.976552)	00.011025	00.100156
9)	(1.953139 0.976552)	00.000369	00.100156

Número de iterações 9.

Argmin (**x***): (1.953139 0.976552).

Mínimo da função $f(\mathbf{x}^*)$ 0.000005.



(a) (b)

Figura 3.15: Comportamento iterativo do método do gradiente conjugado. (a) Ponto inicial $(x_1, x_2) = (3, 0)$. (b) Ponto inicial $(x_1, x_2) = (0, 0)$.

4. Método de Davidon-Fletcher-Powell (DFP)

A Figura 3.16 (a) e (b) apresentará os resultados deste método para a condição inicial (a) e (b), respectivamente.

Caso (a) – $(x_1, x_2) = (3, 0)$

Tomando o ponto acima como ponto de partida do algoritmo, a seqüência de pontos gerada é apresentada na Tabela 3.8.

Tabela 3.8: Seqüência de pontos gerada para o método de Davidon-Fletcher-Powell com condição inicial $(x_1, x_2) = (3, 0)$.

Iteração i	$(x_{1i}$ $x_{2i})$	$\|\nabla f(\mathbf{x}_i)\|$	α_i
01)	(2.118755 1.057494)	15.620499	0.088125
02)	(2.120294 1.055865)	0.020722	0.119304
03)	(2.098496 1.050979)	0.041875	1.304161
04)	(2.053698 1.025661)	0.014191	5.011069
05)	(2.042931 1.021754)	0.010917	1.973799
06)	(2.022741 1.011141)	0.002456	4.392803
07)	(2.018472 1.009284)	0.002072	1.834248
08)	(2.018472 1.009284)	0.000423	1.834248

Número de iterações 8.

Argmin (**x***): (2.018472 1.009284).

Mínimo da função f(**x***) 0.000000.

Caso (b) – $(x_1, x_2) = (0, 0)$

Tomando o ponto acima como ponto de partida do algoritmo, a seqüência de pontos gerada é apresentada na Tabela 3.9.

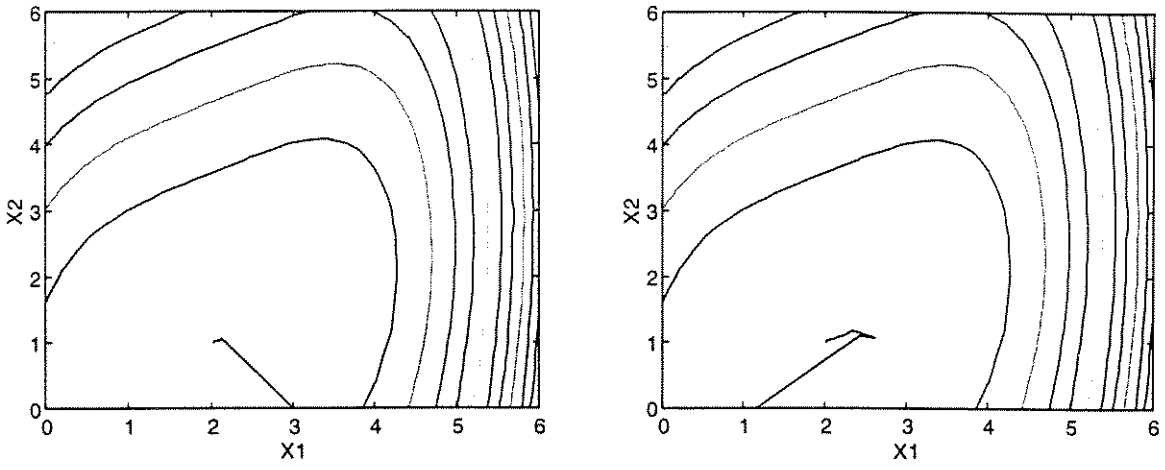
Tabela 3.9: Resumo da seqüência de pontos gerada para o método de Davidon-Fletcher-Powell com condição inicial $(x_1, x_2) = (0, 0)$.

Iteração i	$(x_{1i}$ $x_{2i})$	$\|\nabla f(\mathbf{x}_i)\|$	α_i
01)	(1.162006 0.000000)	32.000000	0.036313
02)	(2.440653 1.103978)	4.648119	0.240732
03)	(2.618004 1.063732)	1.232338	0.209218
04)	(2.337061 1.188788)	2.748911	1.053625
05)	(2.235524 1.100213)	0.177395	1.608439
06)	(2.163386 1.088226)	0.186294	2.613299
07)	(2.107974 1.050506)	0.052982	2.845322
08)	(2.076246 1.039554)	0.033693	2.549791
09)	(2.049940 1.024219)	0.012109	2.970116
10)	(2.035480 1.018045)	0.006954	2.518745
11)	(2.023083 1.011379)	0.002658	3.014445
12)	(2.016494 1.008312)	0.001479	2.487515
13)	(2.016494 1.008312)	0.000573	2.487515

Número de iterações 13.

Argmin (**x***): (2.016494 1.008312).

Mínimo da função f(**x***) 0.000000.



(a) (b)

Figura 3.16: Comportamento iterativo do método de Davidon-Fletcher-Powell. (a) Ponto inicial $(x_1, x_2) = (3, 0)$. (b) Ponto inicial $(x_1, x_2) = (0, 0)$.

Comentários sobre o Exemplo 3.2:

O método do gradiente geralmente funciona bem durante as primeiras iterações do processo de otimização, dependendo da condição inicial. Entretanto, quando um ponto estacionário (de mínimo) está próximo, o método comporta-se de maneira pobre, tomando passos pequenos e quase ortogonais. Este fenômeno, conhecido como *zigzag*, pode ser visto na Figura 3.13(b).

O problema abordado possui um único ponto de mínimo e características quadráticas, fazendo com que o desempenho do método de Newton seja surpreendente. A convergência para o ponto ótimo em uma única iteração pode ser vista na Figura 3.14(a). Mesmo com taxas de convergência elevadas, este método ainda é pouco recomendável devido ao grande esforço computacional despendido na construção e inversão da matriz hessiana.

Os métodos de gradiente conjugado possuem desempenho muito superior aos métodos de primeira ordem a um custo computacional baixo, o que faz com que sejam preferidos na solução de problemas de grande escala (problemas que envolvem um grande número de parâmetros a serem ajustados).

Os métodos de quase-Newton possuem convergência superior aos métodos de primeira ordem a um custo computacional entre o método de Newton e o de gradiente conjugado. Para problemas quadráticos geram direções conjugadas. Estes métodos também são pouco recomendáveis para problemas de grande escala.

Capítulo 4

Taxas de Aprendizagem Globais

Uma das formas mais importantes de otimizar o algoritmo de retro-propagação (*back-propagation*) é determinando automaticamente valores adequados para a taxa de aprendizagem. Várias técnicas podem ser sugeridas para o ajuste deste parâmetro durante o treinamento (SCHIFFMAN *et. al.*, 1994). Algumas destas também ajustam o coeficiente de momento, de forma que o passo de ajuste e a direção de busca sejam alterados.

4.1 Introdução

O treinamento de arquiteturas do tipo MLP pode ser feito, basicamente, de duas formas: *em lote* (*batch*, ou *off-line*), ou *local* (*on-line*). No treinamento em lote o vetor de parâmetros só é atualizado depois que todas as amostras de treinamento foram apresentadas a rede. Para o treinamento local, a atualização do vetor de parâmetros é feita imediatamente após a apresentação de cada vetor de padrões. Os dois procedimentos podem admitir um único ou múltiplos valores para o passo. Um único valor para o passo é equivalente a multiplicar a direção de ajuste por um escalar, não alterando assim a direção de ajuste. Múltiplos valores para o passo de ajuste equivale a multiplicar a direção de ajuste por uma matriz, ou seja, equivale a modificar a direção de ajuste, além de escaloná-la.

Como a determinação de um valor do passo de ajuste para cada um dos componentes do vetor de parâmetros requer o uso de informações de segunda ordem, nos restringiremos aos métodos de ajuste global da taxa de aprendizagem, pois empregaremos a informação de segunda ordem, quando for o caso, diretamente na determinação da direção de ajuste. A Figura 4.1 apresenta os tipos de taxas de aprendizagem globais que serão estudadas.

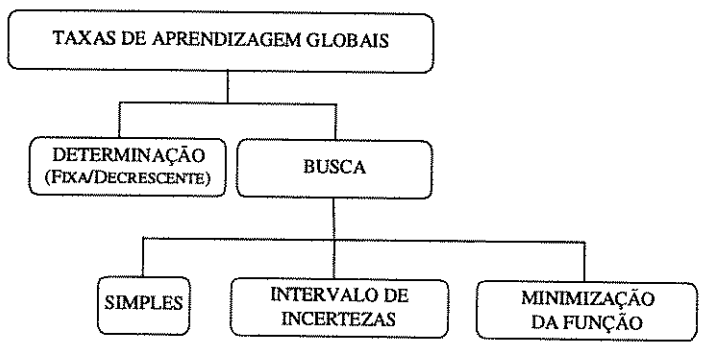


Figura 4.1: Taxas de aprendizagem globais a serem vistas.

4.2 Cálculo Fixo da Taxa

EATON & OLIVIER (1992) sugerem uma forma de determinação da taxa de aprendizagem (α) para treinamento em lote. Este cálculo assume que padrões de treinamento semelhantes resultarão em gradientes semelhantes. Assim, é desejado reduzir a taxa de aprendizagem quando houver muitas amostras similares. O conjunto de treinamento deve ser, de alguma forma, dividido em m subconjuntos de amostras similares. Sejam $N_1, N_2, \ldots, N_m$ as dimensões destes subconjuntos. A taxa de aprendizagem e o momento podem ser calculados da seguinte forma:

$$\alpha = \frac{1.5}{\sqrt{N_1^2 + N_2^2 + \cdots + N_m^2}} \quad (4.1)$$
$$\beta = 0.9$$

Assim, dado $m_1 > m_2$ tal que

$$N_1, N_2, \ldots, N_{m1} = N$$
$$N_1', N_2', \ldots, N_{m2}' = N,$$

então é possível provar que $\sqrt{N_1^2, N_2^2, \ldots, N_{m1}^2} < \sqrt{N_1'^2, N_2'^2, \ldots, N_{m2}'^2}$. Logo, para um dado $m \in (1, N)$ tal que $N_1, N_2, \ldots, N_m = N$, quanto menor m, menor será a taxa de aprendizagem α dada pela equação (4.1).

4.3 Taxa de Aprendizagem Decrescente

DARKEN & MOODY (1991) decrescem a taxa de aprendizagem durante o treinamento. Esta estratégia é sugerida quando a atualização é feita após a apresentação de cada padrão. Iniciando com um valor elevado para a taxa α_0, o valor de α é decrementado durante o treinamento para aproximadamente $\alpha_0 / (1 + i)$:

$$\alpha_i = \frac{\alpha_0}{1 + \frac{i}{r}} \quad (4.2)$$

O parâmetro constante r pode ser usado para ajustar a taxa de acordo com o período total de treinamento. Após as primeiras r iterações, a regra de atualização da taxa (eq.(4.2)) fará com que esta atinja metade de seu valor inicial.

4.4 Busca Simples da Taxa

Os métodos apresentados nas Seções 4.2 e 4.3, embora computacionalmente simples, não garantem a adoção de passos sempre minimizantes, na direção de ajuste. Para garantir passo minimizante a cada iteração, métodos de busca unidimensional devem ser empregados, exigindo um esforço computacional adicional por iteração.

Um método bastante simples de determinação do valor de ajuste da taxa a cada iteração é apresentado abaixo (θ^p, é o valor provisório do vetor de parâmetros):

4.4.1 Algoritmo

1. $\alpha_i = t_a \alpha_i$; $\theta_{i+1}^p = \theta_i - \alpha_i \dfrac{\nabla J(\theta_i)}{\|\nabla J(\theta_i)\|}$; calcule $J\left(\theta_{i+1}^p\right)$

2. Enquanto $J\left(\theta_{i+1}^p\right) \geq J(\theta_i)$ faça:

   2.1. $\alpha_i = t_r \alpha_i$

   2.2. $\theta_{i+1}^p = \theta_i - \alpha_i \dfrac{\nabla J(\theta_i)}{\|\nabla J(\theta_i)\|}$

3. $\theta_i \leftarrow \theta_i^p$

Onde t_a e t_r são a taxa de aumento e redução do passo α, respectivamente.

Em linhas gerais este algoritmo pode ser visto da seguinte forma:

- escolha um valor inicial para a taxa de aprendizagem;
- a cada iteração aumente o valor da taxa e verifique se o passo minimiza o funcional;
- caso o valor do funcional aumente, reduza o valor da taxa até encontrar um passo capaz de reduzir o valor do funcional; encontrado este valor, aceite a atualização do vetor de parâmetros θ.

Este método faz apenas uma avaliação da função caso seja determinado uma redução no erro na primeira tentativa. Caso contrário, avalia a função quantas vezes forem necessárias para garantir um comportamento monotonicamente decrescente do funcional de erro. É importante observar que, a menos que haja problemas de precisão numérica na representação computacional, é sempre possível encontrar um $\alpha_i > 0$ que garanta um passo minimizante. No entanto, este método de busca não conduz a um valor ótimo para α_i. O objetivo aqui é apenas

encontrar um passo α_i tal que $J(\theta_{i+1}) < J(\theta_i)$, e não $\min_{\alpha_i} J(\theta_{i+1})$. As Seções 4.5, 4.6 e 4.7 apresentam métodos de busca que procuram resolver o problema $\min_{\alpha_i} J(\theta_{i+1})$.

Exemplo 4.1:

O comportamento desta taxa de aprendizagem já foi apresentado no Exemplo 3.1 visto na Seção 3.5.2, e será repetido aqui. É perceptível que a taxa de aprendizagem fica oscilando, ou seja, aumentando e diminuindo de valor ao longo das iterações de forma a garantir um comportamento monotonicamente decrescente do erro.

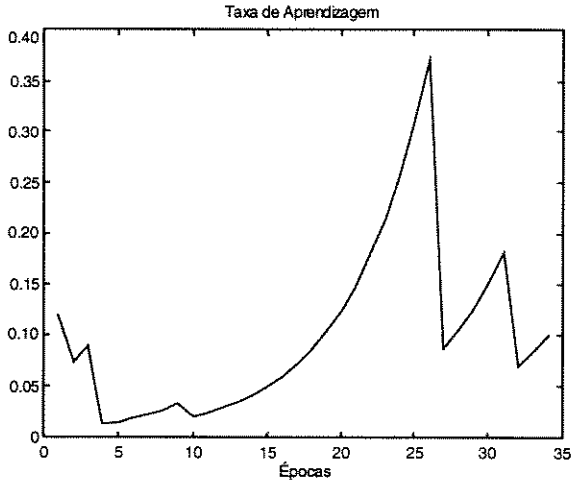


Figura 4.2: Comportamento típico do método de busca simples da taxa de aprendizagem quando aplicado ao problema do Exemplo 3.1.

4.5 Busca Ótima pelo Método de Fibonacci

O método de Fibonacci é um procedimento de busca unidimensional do passo cujo objetivo é minimizar uma função estritamente quasiconvexa em um intervalo fechado e limitado. O método de Fibonacci faz duas avaliações da função no primeiro passo e apenas uma avaliação nos passos seguintes (BAZARAA *et. al.*, 1993). A idéia por trás deste método é encontrar um valor ótimo para α_i em um intervalo $(0,\overline{\alpha}]$, denominado intervalo de incerteza. Para tanto, promove-se uma redução contínua do intervalo de incerteza, a cada iteração da função até que seja atingido um intervalo de incerteza suficientemente pequeno. O valor ótimo de α_i é tomado como o ponto central (ou um dos extremos) deste intervalo resultante.

Este método, assim como o método da Seção Áurea e da Falsa Posição, que serão apresentados em seguida, utiliza a seguinte metodologia:

• Dado um ponto θ_i, encontre um passo satisfatório $\overline{\alpha}$ que gere um novo ponto $\overline{\theta} = \theta_i + \overline{\alpha}\,\mathbf{d}_i$.

Determinar o passo α_i envolve a solução do subproblema $\min_{\alpha_i \in (0,\overline{\alpha}]} J(\theta_i + \alpha_i \mathbf{d}_i)$, que é um problema de busca unidimensional. Considere a função $J : \Re^P \rightarrow \Re$; para $\mathbf{d} \in \Re^P$ fixo, a função g: $\Re \rightarrow \Re$, tal que $g(\alpha) = J(\theta_i + \alpha_i \mathbf{d}_i)$ depende apenas do escalar $\alpha_i \in (0, \overline{\alpha}]$.

O método é baseado na seqüência de Fibonacci $\{F_v\}$:

$$F_{v+1} = F_v + F_{v-1}, \quad v = 1, 2, \ldots$$
$$F_0 = F_1 = 1 \tag{4.3}$$

A seqüência torna-se então 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, ... Na iteração i suponha que o intervalo de incerteza seja $[a_i, b_i]$. Sejam dois pontos quaisquer λ_i e μ_i, e d_N o tamanho final desejado do intervalo de incertezas. Conhecendo o valor de d_N (arbitrário) é possível saber quantas iterações (N) serão necessárias para determinar o passo α.

Exemplo 4.2:

A Figura 4.3 apresenta um comportamento da taxa de aprendizagem α, obtido por procedimentos de busca unidimensional como Fibonacci e Seção Áurea que será visto posteriormente. O problema de teste corresponde ao Exemplo 3.1.

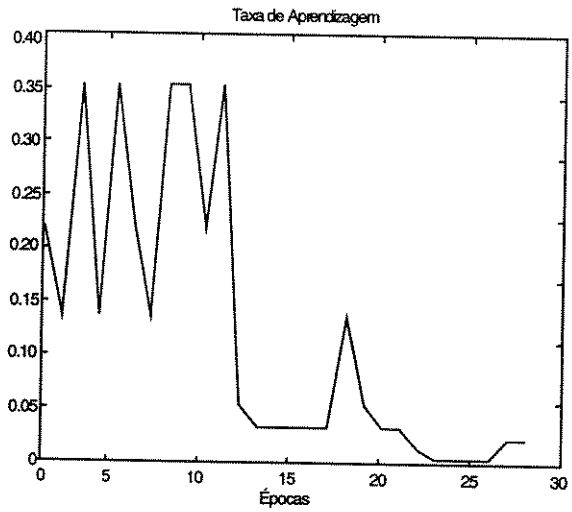


Figura 4.3: Comportamento da taxa de aprendizagem de procedimentos de busca unidimensional quando aplicados ao problema do Exemplo 3.1.

Um resumo do algoritmo de Fibonacci para solucionar o problema $\min_{\alpha_i \in (0,\overline{\alpha}]} J(\theta_i + \alpha_i \mathbf{d}_i)$ é apresentado abaixo:

4.5.1 Algoritmo

1. Geração da seqüência de Fibonacci: $\{F_v\}$
2. $[a_1, b_1]$ - intervalo inicial de incertezas
3. Escolha um valor arbitrário para d_N (critério de parada)
4. N - número de iterações: $F_N = \frac{1}{d_N}$
5. $\lambda_1 = a_1 + \frac{F_{N-2}}{F_N}(b_1 - a_1)$ e $\mu_1 = a_1 + \frac{F_{N-1}}{F_N}(b_1 - a_1)$
6. $J(\lambda_1)$ e $J(\mu_1)$
7. Enquanto $\frac{|a_1 - b_1|}{2} \geq d_N$ faça:

7.1. Se $J(\lambda_i) > J(\mu_i)$, vá para 7.1.1; e se $J(\lambda_i) \leq J(\mu_i)$, vá para 7.1.2

7.1.1. Faça:

- $a_{i+1} = \lambda_i$ e $b_{i+1} = b_i$
- $\lambda_{i+1} = \mu_i$ e $\mu_{i+1} = a_{i+1} + \frac{F_{N-i-1}}{F_N}(b_1 - a_1)$
- $J(\mu_{i+1})$

7.1.2. Faça:

- $a_{i+1} = a_i$ e $b_{i+1} = \mu_i$
- $\mu_{i+1} = \lambda_i$ e $\lambda_{i+1} = a_{i+1} + \frac{F_{N-i-2}}{F_N}(b_1 - a_1)$
- $J(\lambda_{i+1})$

4.6 Busca Ótima pelo Método da Seção Áurea

Os princípios deste método são os mesmos do método de Fibonacci, a diferença reside apenas na forma de determinação dos pontos λ_i e μ_i. A seqüência de Fibonacci não precisa mais ser gerada, reduzindo sensivelmente o custo computacional do método.

Um resumo do método da seção áurea para solucionar o problema $\min_{\alpha_i \in (0,\bar{\alpha}]} J(\theta_i + \alpha_i \mathbf{d}_i)$ é apresentado abaixo:

4.6.1 Algoritmo

1. (a_1, b_1) - intervalo inicial de incertezas
2. Escolha um valor arbitrário para d_N (critério de parada)
3. $\alpha = \frac{\sqrt{5}-1}{2} = 0.618$ - razão áurea
4. $\lambda_1 = a_1 + (1-\alpha)(b_1 - a_1)$ e $\mu_1 = a_1 + \alpha(b_1 - a_1)$
5. $J(\lambda_1)$ e $J(\mu_1)$
6. Enquanto $\frac{|a_1 - b_1|}{2} \geq d_N$ faça:

   6.1. Se $J(\lambda_i) > J(\mu_i)$, vá para 6.1.1; e se $J(\lambda_i) \leq J(\mu_i)$, vá para 6.1.2

   6.1.1. Faça:

   - $a_{i+1} = \lambda_i$ e $b_{i+1} = b_i$
   - $\lambda_{i+1} = \mu_i$ e $\mu_{i+1} = a_{i+1} + \alpha(b_{i+1} - a_{i+1})$
   - $J(\mu_i+1)$

   6.1.2. Faça:

   - $a_{i+1} = a_i$ e $b_{i+1} = \mu_i$
   - $\mu_{i+1} = \lambda_i$ e $\lambda_{i+1} = a_{i+1} + (1-\alpha)(b_{i+1} - a_{i+1})$
   - $J(\lambda_{i+1})$

4.7 Busca Ótima pelo Método da Falsa Posição

Solucionar o problema $\min_{\alpha_i \in (0,\bar{\alpha}]} J(\theta_i + \alpha_i \mathbf{d}_i)$, significa minimizar uma função unidimensional. Estratégias comuns como o método de Newton e outros podem ser utilizados. A maior desvantagem do método de Newton é a necessidade de se efetuar o cálculo da derivada segunda da função. O algoritmo da falsa posição possui três diferenças básicas em relação aos algoritmos de busca ótima vistos até agora (Fibonacci e Seção Áurea):

- necessita efetuar o cálculo da derivada da função a cada iteração;

- calcula o valor do passo ótimo minimizando a função $J(\theta_i + \alpha_i \mathbf{d}_i)$, ao invés de reduzir o intervalo de incerteza;
- possível critério de parada: $\frac{|\theta_i - \theta_{i-1}|}{|\theta_i|} \leq d_N$

4.7.1 Algoritmo

1. Escolha um valor arbitrário para d_N (critério de parada)
2. Enquanto $\frac{|\theta_i - \theta_{i-1}|}{|\theta_i|} \geq d_N$ faça:

   2.1. $\theta_{i+1} = \theta_i - \nabla J(\theta_i).\frac{\theta_{i-1} - \theta_i}{\nabla J(\theta_{i-1}) - \nabla J(\theta_i)}$

Exemplo 4.2:

Muitos algoritmos de otimização não-linear funcionam como segue. Dado um ponto $\mathbf{x}_i$, encontre uma direção $\mathbf{d}_i$ e em seguida um passo satisfatório α_i que gere um novo ponto $\mathbf{x}_{i+1} = \mathbf{x}_i + \alpha_i \mathbf{d}_i$; o processo é então repetido. Determinar o passo α_i envolve a solução do subproblema **min** $f(\mathbf{x}_i + \alpha_i \mathbf{d}_i)$, que é um problema de busca unidirecional. Considere a função $f: \Re^n \rightarrow \Re$; para $\mathbf{d} \in \Re^n$ fixo, a função $g(\alpha) = f(\mathbf{x}_i + \alpha_i \mathbf{d}_i)$ depende apenas do escalar $\alpha \in (0, \overline{\alpha}]$.

Dada a função $f(x_1, x_2) = (x_1 - 2)^4 + (x_1 - 2x_2)^2$, e duas condições iniciais e direções de busca, determine o tamanho do passo α^*.

Metodologias:

Foram implementados três algoritmos capazes de resolver o problema proposto acima, *Fibonacci*, *Seção Áurea* e *Falsa Posição*. Duas técnicas básicas serão estudadas aqui; a busca unidimensional sem utilização de derivadas (Fibonacci e Seção Áurea), e a busca utilizando derivadas (Falsa Posição). A diferença principal entre as técnicas, além da utilização, ou não, das derivadas da função, está no fato de que as duas primeiras a serem estudadas trabalham com a idéia da redução do intervalo de incertezas e a outra faz uma minimização da função $g(\alpha)$ propriamente dita.

1. Fibonacci

O objetivo é determinar o valor mínimo da função $g(\alpha)$ sobre um intervalo $(0, \overline{\alpha}]$. Serão selecionadas N avaliações sucessivas da função $g(\alpha)$ de modo a obter o menor intervalo de incertezas que contém o ponto de mínimo.

Seja o intervalo de incertezas $[a_i, b_i]$, α_{1i} e α_{2i} os dois novos valores de α calculados simetricamente pela razão de Fibonacci, e $g(\alpha_{1i})$ e $g(\alpha_{2i})$ as respectivas avaliações da função $g(\alpha) = f(\mathbf{x}_i + \alpha_i \mathbf{d}_i)$ nos pontos α_{1i} e α_{2i}.

Caso (a) – $(x_1, x_2) = (0,1)$, e $\mathbf{d} = [0 \quad 1]^T$

A Tabela 4.1 apresenta a seqüência de pontos gerada por este método para o caso (a).

Tabela 4.1: Cálculos para o método de Fibonacci – $(x_1, x_2) = (0,1)$, e $\mathbf{d} = [0 \quad 1]^T$.

Iteração i	a_i	b_i	α_{1i}	α_{2i}	$g(\alpha_{1i})$	$g(\alpha_{2i})$
1	0.000	4.000	1.528	2.472	20	20
2	1.528	4.000	2.472	3.056	0.273	0.273
3	1.528	3.056	2.112	2.472	0.273	2.357
4	1.528	2.472	1.889	2.112	0.013	0.273
5	1.889	2.472	2.112	2.249	0.013	0.013
6	1.889	2.249	2.026	2.112	0.013	0.066
7	1.889	2.112	1.974	2.026	0.001	0.013
8	1.974	2.112	2.026	2.059	0.001	0.001
9	1.974	2.059	1.974	2.026	0.001	0.004
10	1.974	2.026	2.026	2.006	0.000	0.001
11	1.994	2.026	2.006	2.014	0.000	0.000
12	1.994	2.014	1.994	2.006	0.000	0.000
13	1.994	2.006	2.006	2.002	0.000	0.000
14	1.999	2.006	2.002	2.003	0.000	0.000
15	1.999	2.003	1.999	2.002	0.000	0.000
16	1.999	2.002	2.002	2.000	0.000	0.000
17	2.000	2.002	2.000	2.001	0.000	0.000
18	2.000	2.001	2.000	2.000	0.000	0.000
19	2.000	2.001	-------	-------	0.000	0.000

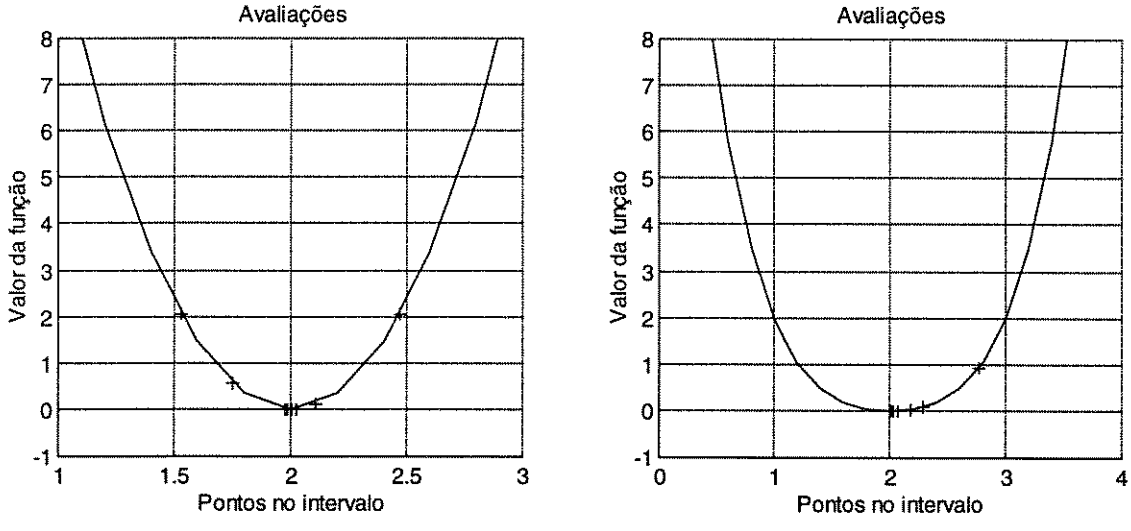
O intervalo final de incertezas é: (2.000296 2.000887) e o valor do passo é: α = 2.0006.

Caso (b) – $(x_1, x_2) = (0,3)$, e $\mathbf{d} = [1 \quad -1]^T$

Como desejamos minimizar a função $g(\alpha)$, apresentaremos a avaliação desta função nos pontos intermediários de cada novo intervalo de decisão calculado. A Figura 4.4(a) apresenta os resultados para o caso (a) e 4.4(b) para o caso (b).

Tabela 4.2: Cálculos para o método de Fibonacci – $(x_1, x_2) = (0,3)$, e $\mathbf{d} = [1 \quad -1]^T$.

Iteração i	a_i	b_i	α_{1i}	α_{2i}	$g(\alpha_{1i})$	$g(\alpha_{2i})$
1	0.000	8.000	3.0557	4.944	52	1620
2	0.000	4.944	1.889	3.056	11.3	153.2
3	0.000	3.056	1.167	1.889	00.1	11.3
4	1.167	3.056	1.889	2.334	06.7	00.1
5	1.167	2.334	1.613	1.889	00.1	01.0
6	1.613	2.334	1.889	2.059	01.4	00.0
7	1.889	2.334	2.059	2.334	00.1	00.2
8	1.889	2.164	1.994	1.889	0.00	0.00
9	1.889	2.059	1.954	2.059	0.00	0.00
10	1.954	2.059	1.994	2.164	0.00	0.00
11	1.954	2.019	1.978	2.059	0.00	0.00
12	1.978	2.019	1.994	1.994	0.00	0.00
13	1.994	2.019	2.003	2.003	0.00	0.00
14	1.994	2.009	2.000	2.009	0.00	0.00
15	1.994	2.003	1.997	2.003	0.00	0.00
16	1.997	2.003	2.000	2.000	0.00	0.00
17	1.997	2.001	1.999	2.000	0.00	0.00
18	1.999	2.001	2.000	2.000	0.00	0.00
19	2.000	2.001	2.000	2.001	0.00	0.00
20	2.000	2.001	2.000	2.000	0.00	0.00
21	2.000	2.000	-------	-------	0.00	0.00



(a) (b)

Figura 4.4: Avaliações da função nos pontos médios dos intervalos. (a) Ponto inicial $(x_1, x_2) = (0,1)$, e $\mathbf{d} = [0 \quad 1]^T$. (b) Ponto inicial $(x_1, x_2) = (0,3)$, e $\mathbf{d} = [1 \quad -1]^T$.

No caso (b) foram obtidos (1.999661 2.000113) e α = 1.9999, para o intervalo e passo respectivamente.

2. Seção Áurea

A diferença básica do algoritmo de Fibonacci para o algoritmo da Seção Áurea, é que o segundo trabalha com a redução do intervalo de incertezas na proporção áurea, e o primeiro na razão de Fibonacci dada pela equação $d_i = \left(\frac{F_{N-i+1}}{F_N} \right) d_1$, os termos F_{N-i+1} e F_N são os termos da seqüência de Fibonacci.

Caso (a) – $(x_1, x_2) = (0,1)$, e $\mathbf{d} = [0 \quad 1]^T$

Utilizando a mesma notação anterior os resultados obtidos são apresentados na Tabela 4.3 e nos comentários a seguir.

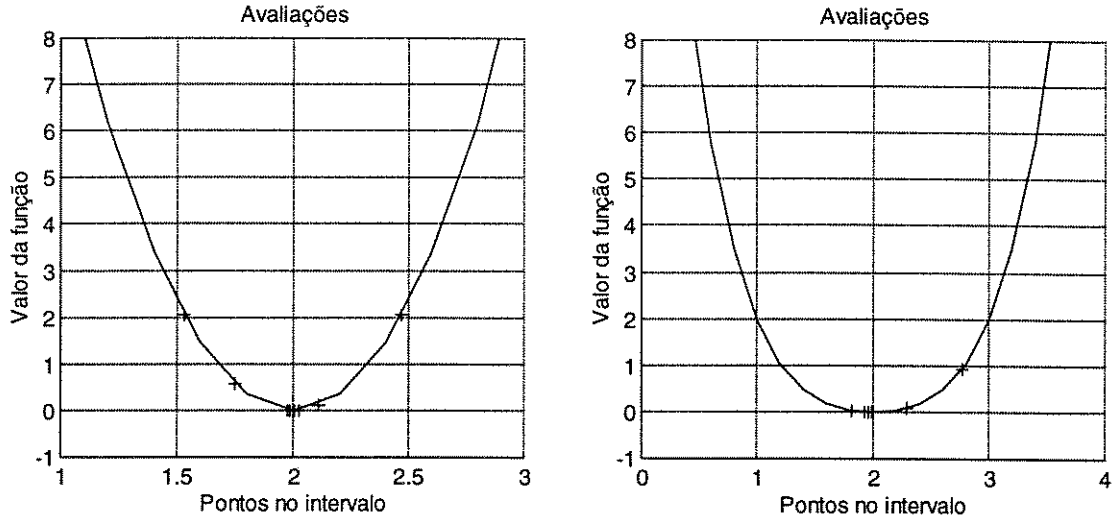
Tabela 4.3: Cálculos para o método da Seção Áurea – $(x_1, x_2) = (0,1)$, e $\mathbf{d} = [0 \quad 1]^T$.

Iteração i	a_i	b_i	α_{1i}	α_{2i}	$g(\alpha_{1i})$	$g(\alpha_{2i})$
1	0	4.000	2.472	1.528	20	20
2	1.528	4.000	3.056	2.472	0.272	0.272
3	1.528	3.056	2.472	2.112	2.357	0.273
4	1.528	2.472	2.112	1.889	0.273	0.013
5	1.528	2.112	1.889	1.751	0.013	0.013
6	1.751	2.112	1.974	1.889	0.013	0.066
7	1.889	2.112	2.026	1.974	0.001	0.013
8	1.889	2.026	1.974	1.941	0.001	0.001
9	1.941	2.026	1.994	1.974	0.001	0.004
10	1.974	2.026	2.006	1.994	0.000	0.001
11	1.974	2.006	1.994	1.986	0.000	0.000
12	1.986	2.006	1.999	1.994	0.000	0.000
13	1.994	2.006	2.002	1.999	0.000	0.000
14	1.994	2.002	1.999	1.997	0.000	0.000
15	1.997	2.002	2.000	1.999	0.000	0.000
16	1.999	2.002	2.000	2.000	0.000	0.000
17	1.999	2.000	2.000	2.000	0.000	0.000
18	1.999	2.000	2.000	2.000	0.000	0.000
19	2.000	2.000	--------	--------	0.000	0.000

O intervalo final de incertezas é: (1.999694 2.000386) e o valor do passo é: $\alpha = 2.0$.

Caso (b) – $(x_1, x_2) = (0,3)$, e $\mathbf{d} = [1 \quad -1]^T$

Os resultados para este caso estão apresentados na Tabela 4.4. A Figura 4.5 apresenta os resultados do método da seção áurea para os casos (a) e (b).



(a) (b)

Figura 4.5: Avaliações da função nos pontos médios dos intervalos. (a) Ponto inicial $(x_1, x_2) = (0,1)$, e $\mathbf{d} = [0 \quad 1]^T$. (b) Ponto inicial $(x_1, x_2) = (0,3)$, e $\mathbf{d} = [1 \quad -1]^T$.

Tabela 4.4: Cálculos para o método da Seção Áurea – $(x_1, x_2) = (0,3)$, e $\mathbf{d} = [1 \quad -1]^T$.

Iteração i	a_i	b_i	α_{1i}	α_{2i}	$g(\alpha_{1i})$	$g(\alpha_{2i})$
1	0.000	8.000	4.944	3.056	52	1620
2	0.000	4.944	3.055	1.889	153.1	11.3
3	0.000	3.055	1.888	1.167	11.3	00.1
4	1.167	3.055	2.334	1.889	00.1	06.7
5	1.167	2.334	1.888	1.613	01.0	00.1
6	1.613	2.334	2.059	1.888	00.0	01.4
7	1.888	2.334	1.994	2.059	00.2	00.1
8	1.888	2.164	2.018	1.994	0.00	0.00
9	1.888	2.059	1.994	1.953	0.00	0.00
10	1.953	2.059	2.003	1.994	0.00	0.00
11	1.953	2.018	2.009	1.978	0.00	0.00
12	1.978	2.018	2.003	1.994	0.00	0.00
13	1.994	2.018	2.009	2.003	0.00	0.00
14	1.994	2.009	2.003	2.000	0.00	0.00
15	1.994	2.003	2.000	1.997	0.00	0.00
16	1.997	2.003	2.001	2.000	0.00	0.00
17	1.997	2.001	2.000	1.999	0.00	0.00
18	1.999	2.001	2.000	2.000	0.00	0.00
19	2.000	2.001	2.000	2.000	0.00	0.00
20	2.000	2.000	-------	-------	0.00	0.00

Foram obtidos (1.999473 2.000328) e α = 1.9999, para o intervalo e passo respectivamente.

3. Falsa Posição

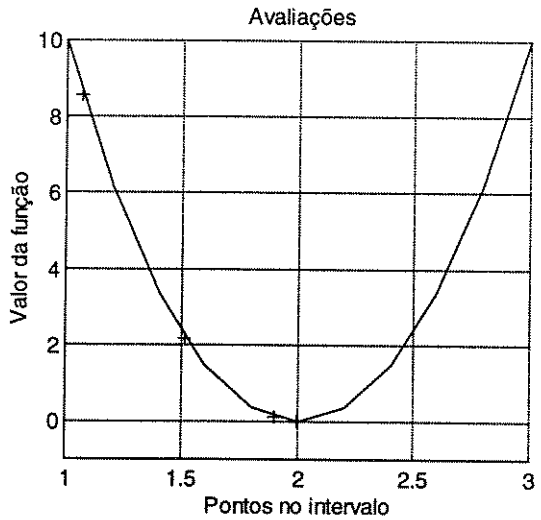
Uma estimativa para o valor α_{i+1} pode ser obtida pela expressão vista no passo 2.1 do algoritmo deste método. As Tabelas 4.5 e 4.6 trazem a seqüência de pontos gerados para os casos (a) e (b), respectivamente; e a Figura 4.6(a) e (b) apresenta o comportamento do algoritmo.

Tabela 4.5: Cálculos para o método da Falsa Posição – $(x_1, x_2) = (0,1)$, e $\mathbf{d} = [1 \quad 0]^T$.

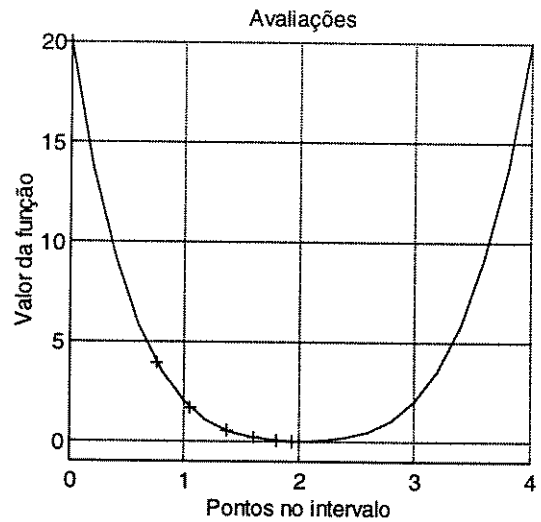
Iteração i	α_i	$g'(\alpha_i)$	α_{i+1}
1	0.000	-36.000	0.756
2	0.756	-10.195	1.054
3	1.054	-5.275	1.374
4	1.374	-2.231	1.609
5	1.609	-1.021	1.807
6	1.807	-0.415	1.942
7	1.942	-0.116	1.995
8	1.995	-0.010	2.000
9	2.000	-0.000	2.000

Tabela 4.6: Cálculos para o método da Falsa Posição – $(x_1, x_2) = (0,3)$, e $\mathbf{d} = [1 \quad -1]^T$.

Iteração i	α_i	$g'(\alpha_i)$	α_{i+1}
1	0.000	-68.000	1.069
2	1.069	-20.000	1.514
3	1.514	-9.213	1.894
4	1.894	-1.913	1.994
5	1.994	-0.115	2.000
6	2.000	-0.000	2.000



(a)



(b)

Figura 4.6: Avaliações da função nos pontos médios dos intervalos. (a) Ponto inicial $(x_1, x_2) = (0,1)$, e $\mathbf{d} = [0 \quad 1]^T$. (b) Ponto inicial $(x_1, x_2) = (0,3)$, e $\mathbf{d} = [1 \quad -1]^T$.

Comentários sobre o Exemplo 4.2:

Quanto menor o número de avaliações requeridas da função, mais eficiente é o método de Fibonacci em relação ao método da Seção Áurea. Porém, para N (número de avaliações da função) suficientemente grande, esses dois métodos são praticamente idênticos. Através dos resultados apresentados neste exemplo, vimos que os dois métodos necessitaram do mesmo número de iterações para o primeiro caso; e intervalos de incerteza equivalentes.

Também é possível verificar que o método que utiliza informações sobre a derivada da função (falsa posição) apresenta convergência mais rápida do que os métodos de redução do intervalo de incerteza. Porém, é importante ressaltar que a utilização da derivada da função de custo (erro) para determinação do passo de ajuste é pouco recomendável para aplicações em redes MLP, pois constitui um processo custoso do ponto de vista computacional.

Devido principalmente, a limitações de esforço computacional, o método utilizado nos algoritmos de treinamento implementados neste trabalho foi o da Seção Áurea, salvo especificações em contrário.

Capítulo 5

Aspectos Numéricos do Treinamento de Redes MLP

Este capítulo analisa o treinamento de redes MLP's sob o ponto de vista numérico, em particular os problemas de condicionamento e esforço computacional. O treinamento de redes do tipo MLP pode apresentar problemas de mal-condicionamento, afetando o comportamento numérico e a qualidade da solução encontrada (McKeon *et. al.*, 1997). Uma interpretação geométrica para os problemas mal-condicionados é analisada.

5.1 Introdução

As arquiteturas do tipo MLP podem ser utilizadas como aproximadores universais de funções, como visto no Capítulo 2. Este treinamento pode ser formulado como um problema de otimização não-linear irrestrita. Portanto, um grande número de técnicas de otimização está disponível para analisá-lo e resolvê-lo. Dois fatores importantes no desempenho destes métodos, assim como na confiança da resposta obtida, são o condicionamento numérico da função de erro e o esforço computacional despendido na obtenção da solução do problema. O mal-condicionamento pode ser visto, de forma genérica, como a situação na qual o valor de uma função de muitas variáveis é localmente muito mais sensível a algumas destas variáveis do que a outras.

5.2 Condicionamento Numérico

No Capítulo 3 apresentamos o funcional de erro (eq. 3.1) a ser minimizado durante o treinamento de uma rede MLP. Para a utilização dos métodos de segunda ordem foi feita uma aproximação quadrática (eq. 3.4) da função utilizando expansão em série de Taylor. Por conveniência esta expansão será repetida abaixo:

$$J_{quad}(\theta) = J(\theta_i) + \nabla J(\theta_i)^T(\theta - \theta_i) + (\theta - \theta_i)^T \nabla^2 J(\theta_i)(\theta - \theta_i). \qquad (5.1)$$

A matriz hessiana na vizinhança de um mínimo de $\nabla J(\theta)$ é importante pois, ao menos que seja uma matriz nula, ela determina a precisão com que este mínimo será localizado. Em um ponto de mínimo a matriz será pelo menos semi-definida positiva, ou seja, seus autovalores serão não-negativos. Como esta matriz é também simétrica, seus autovetores serão mutuamente ortogonais. O *autosistema* da matriz hessiana pode ser escrito como:

$$\nabla^2 J = \mathbf{Q}\mathbf{D}\mathbf{Q}^T, \tag{5.2}$$

onde $\mathbf{Q}$ é uma matriz ortonormal cujas colunas $\mathbf{q}_k$ são os autovetores da matriz hessiana e $\mathbf{D}$ é a matriz diagonal cujos elementos são os autovalores, por hipótese não-negativos. O número de condição (*condition number*) da matriz hessiana é definido como:

$$C = \frac{(d_{kk})_{\max}}{(d_{kk})_{\min}}. \tag{5.3}$$

Um valor de C maior do que, por exemplo 10^4 implica em mal-condicionamento; para uma precisão numérica de 32-bits, um valor em torno de 10^7 indica que a matriz hessiana é definitivamente singular.

Este mal-condicionamento é uma característica do funcional de erro, e não do algoritmo que está sendo utilizado para minimizá-lo.

Se J fosse linear, como nos perceptrons lineares, a hessiana seria idêntica a $\mathbf{X}\mathbf{X}^T$, onde $\mathbf{X}$ é a matriz de observação das entradas de treinamento, e portanto fixa. No caso que estamos analisando o funcional de erro é não-linear, e portanto a matriz hessiana é função do vetor de parâmetros θ. Então, quando falamos de problemas mal-condicionados nos referimos a problemas cuja matriz hessiana é mal-condicionada na vizinhança de um mínimo local ou global, e cujo Jacobiano também é mal-condicionado.

Na prática, o problema da singularidade da matriz hessiana em pontos que não sejam os pontos ótimos, pode ser aliviado utilizando estratégias como a de Levenberg-Marquardt, ou seja, adicionar um pequeno valor positivo (μ) a cada um dos autovalores da matriz $\mathbf{J}^T\mathbf{J}$ (veja a equação 3.20), ou o procedimento de positivação da Hessiana utilizado pelo método de Newton. Nos métodos de quase-Newton, como DFP e BFGS, a inversa da matriz hessiana é aproximada pela soma iterativa de matrizes simétricas definidas positivas, garantindo a não singularidade da Hessiana, e tendo como conseqüência algoritmos numericamente mais estáveis.

O mal-condicionamento da matriz hessiana pode ser interpretado geometricamente. Considere um subespaço bi-dimensional qualquer, escolhido, por exemplo selecionando duas variáveis θ_i e θ_j e mantendo as outras fixas. Variando somente estas duas variáveis, o funcional de erro $J(\theta)$ pode ser representado como uma superfície de contorno, que pode ter um ou mais mínimos locais. No Capítulo 2, Seção 2.4.7.3, são feitas várias considerações sobre a superfície de erro geradas pelo treinamento de redes MLP.

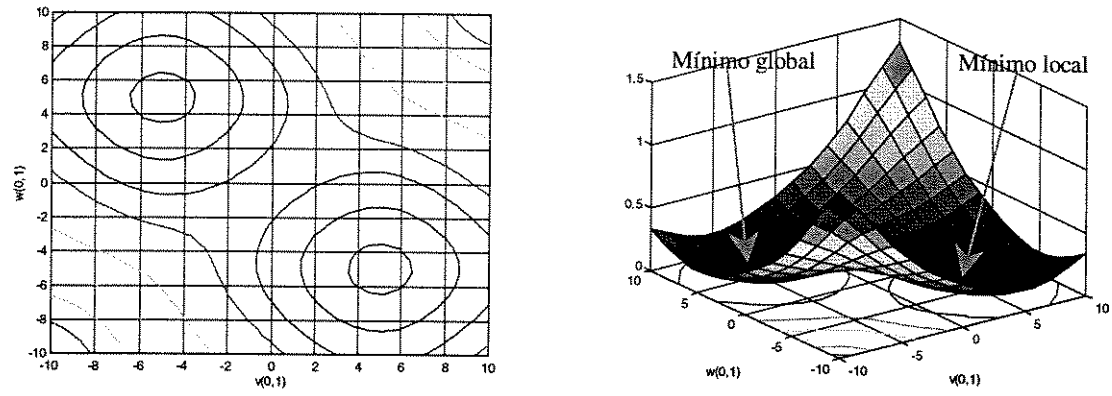


Figura 5.1: Superfície do erro quadrático e seu contorno em relação aos limiares v_{01} e w_{01} para o caso do Exemplo 2.3 (ver Capítulo 2). Forma elipsoidal na vizinhança dos mínimos.

Na vizinhança dos mínimos os contornos terão forma elipsoidais, uma vez que os termos de segunda ordem da expansão em série de Taylor serão os mais significativos (ver Figura 5.1). Quanto maior a não-linearidade do problema, menores serão as vizinhanças de cada mínimo local nas quais os contornos se aproximam de elipses. A direção paralela ao eixo mais longo da elipse representa a direção *menos sensível* no subespaço, no sentido de que, localmente, a função varia menos rapidamente nesta direção do que em qualquer outra do subespaço.

Para cada mínimo local haverá pelo menos uma direção, não necessariamente paralela a um eixo de coordenadas, que é a direção de menor sensibilidade; e outra na qual a sensibilidade é máxima. Os autovetores correspondentes aos máximos e mínimos autovalores são as direções de máxima e mínima sensibilidade respectivamente, e a razão entre o maior e o menor eixo da superfície de contorno, plotada no plano contendo estes dois autovetores, é igual a $\sqrt{C}$. A Figura 5.2(a) ilustra os contornos elipsoidais em torno de um ponto de mínimo e também as direções do menor e maior autovalores da função.

Se os contornos são plotados no plano definido por estas duas direções, suas elipses na vizinhança deste mínimo local terão excentricidade máxima. Por exemplo, se $C = 10^4$ os contornos localmente consistirão de elipses concêntricas cujo eixo maior é 100 vezes maior do que o eixo menor, e portanto, representa um grande vale plano ao longo da direção do eixo maior. No caso de singularidade da hessiana, os contornos são localmente paralelos e a posição do mínimo local não é unicamente determinada (ver Figura 5.2(b)). Em alguns casos, a Hessiana é não-singular, porém existem autovalores com sinais diferentes na parte real, caracterizando um ponto de sela (ver Figura 5.2(c)).

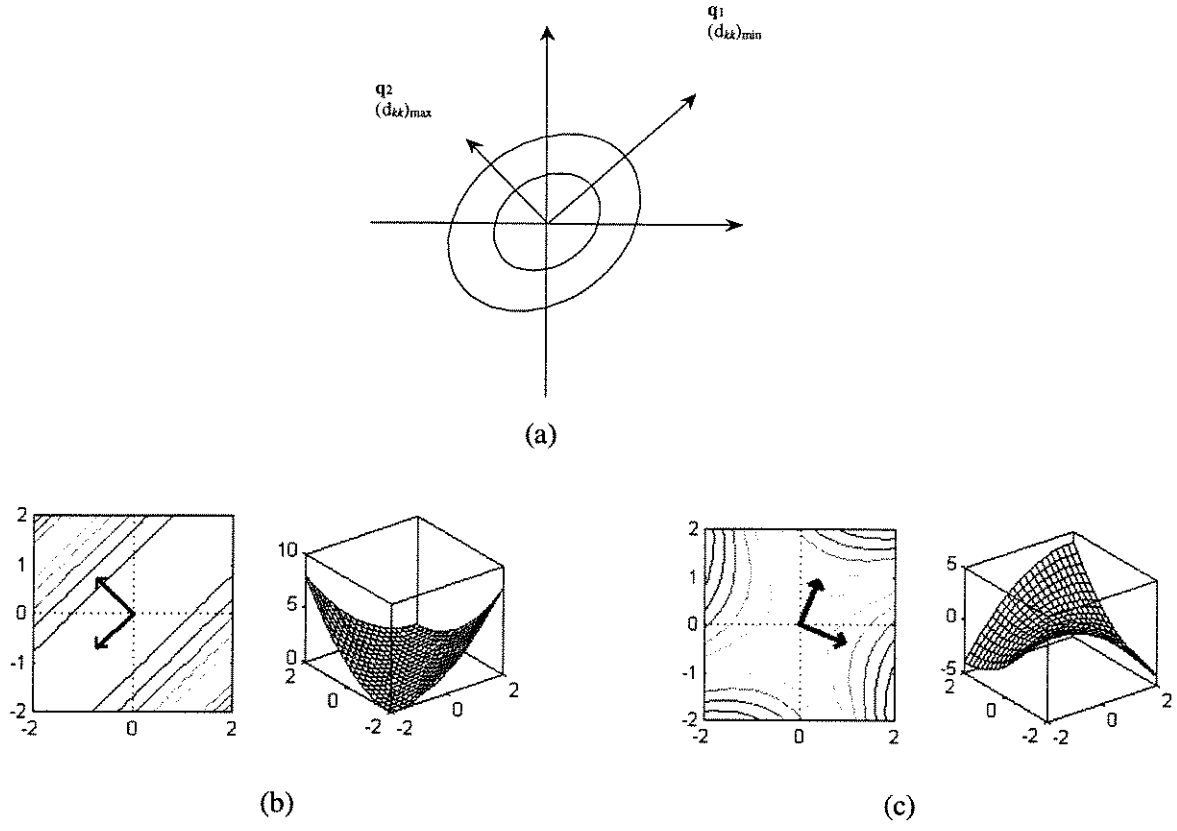


Figura 5.2: Contornos da função em torno do mínimo. (a) Forma elipsoidal, com as direções do menor e maior autovalores da função. (b) Caso em que a matriz hessiana é singular, os contornos são localmente paralelos. (c) A parte real dos autovalores da matriz hessiana possuem sinais diferentes, constituindo um ponto de sela.

Uma das principais dificuldades associadas ao mal-condicionamento é que a maior parte dos procedimentos de otimização convergem de maneira muito lenta na vizinhança de pontos mal-condicionados; isso é verdade particularmente para os métodos de primeira ordem como algoritmo padrão (BP) e método do gradiente (GRAD), mas também afeta os métodos de segunda ordem por causa dos efeitos da aproximação quadrática do funcional de erro.

5.2.1 Treinamento de Redes MLP e Mal-condicionamento

Geralmente, o treinamento de redes MLP via *backpropagation* é mal-condicionado ou singular. As três principais causas para isso são:

5.2.1.1 Conjunto Insuficiente de Dados de Treinamento

Quando o número de erros residuais ($m \times N$ – número de saídas vezes o número de amostras) é menor que o número de parâmetros a serem ajustados (P), o Jacobiano possui

menos linhas do que colunas, e o produto $\mathbf{J}^T\mathbf{J}$ é singular; este caso é semelhante a tentar construir um modelo cujo conjunto de dados é menor do que o número de parâmetros a serem determinados. Em todo mínimo no qual o erro residual é suficientemente próximo de zero, a matriz hessiana será singular e o mínimo local não será único.

Problemas desta natureza são comuns e podem ser evitados garantindo que o número de padrões independentes, multiplicado pelo número de saídas da rede, seja pelo menos igual ao número de parâmetros livres a serem ajustados.

5.2.1.2 Rede Sobre-Dimensionada

Quando a rede contém elementos redundantes, pode haver dependência linear entre as colunas do Jacobiano, causando a sua singularidade. Se os erros forem pequenos, isso causará uma quase singularidade da matriz hessiana.

5.2.1.3 Característica Assintótica da Função de Transferência

Cada coluna do Jacobiano é o conjunto de derivadas de cada erro residual em relação a um único parâmetro da rede. Se um neurônio está saturado em cada ponto do treinamento, a coluna correspondente será nula, e o Jacobiano e Hessiana serão singulares.

5.2.2 Mal-condicionamento e a Escolha do Algoritmo de Treinamento

O efeito do mal-condicionamento no processo de minimização é bem entendido. Nos métodos de primeira ordem, as iterações tendem a favorecer as direções correspondentes aos maiores autovalores da matriz hessiana. Isto resulta em uma convergência extremamente lenta ao longo de vales associados ao mal-condicionamento.

Métodos de segunda ordem, como os de Newton, quase-Newton, Levenberg-Marquardt e gradiente conjugado, não são tão afetados neste sentido pelo mal-condicionamento da matriz hessiana, sendo capazes de orientar a direção de busca para um mínimo mesmo quando os contornos da função possuem grandes platôs. Métodos como Newton modificado e Levenberg-Marquardt são capazes de restaurar o posto das matrizes com que trabalham. Entretanto, quando isso é feito, há um aumento da importância da informação de primeira ordem e conseqüente redução da importância da informação de segunda ordem. Os métodos DFP e BFGS, por sua vez, efetuam a correção de posto 2, garantindo que a cada iteração a inversa da matriz hessiana obtida seja semi-definida positiva.

5.2.3 Implicações do Mal-condicionamento no Processo de Otimização

Não é possível analisar procedimentos de otimização global do ponto de vista de sensibilidade ao mal-condicionamento; mas pode-se dizer que a presença de regiões mal condicionadas em torno de um mínimo local deve sempre ser considerada, com o objetivo de garantir que os resultados não sejam mal interpretados. Uma boa maneira de fazer isso é desenvolvendo uma análise de sensibilidade sempre que um possível mínimo global é encontrado.

5.2.4 Considerações Finais Sobre Mal-condicionamento

A presença de mal-condicionamento faz com que os métodos de otimização de primeira ordem sejam ineficientes, mas também afeta os métodos de segunda ordem. A existência de platôs associados ao mal-condicionamento numérico também faz com que a otimização global seja ainda mais difícil, criando na prática um número infinito de mínimos locais. Um mal-condicionamento excessivo significa que as soluções para o problema de otimização são indefinidas. Torna-se possível que redes com o mesmo valor de 'mínimo' da função de erro apresente pesos e limiares completamente diferentes, e portanto, *apresentarão uma generalização diferente para novos padrões de entrada.* O Capítulo 6 trata da capacidade de generalização das arquiteturas MLP.

Considerando-se os aspectos citados até aqui, é aconselhável o uso de:

- um maior número de amostras possível para treinamento;
- métodos de segunda ordem ao invés de primeira ordem; e
- uma análise de sensibilidade para avaliar a solução encontrada.

5.3 Complexidade do Método de Treinamento

Não existe um padrão estabelecido para a comparação de desempenho de métodos de treinamento de redes neurais do tipo MLP. Um critério bem definido é o número de épocas, porém é insuficiente, uma vez que não reflete as diferenças na complexidade por época de treinamento de cada método. Se esta medida de esforço computacional estiver disponível, juntamente com o número de épocas de treinamento, uma estimativa mais precisa da complexidade do treinamento pode ser feita (FIESLER, 1997).

Tabela 5.1: Propriedades dos métodos de treinamento apresentados.

	flops/iteração	*Memória*	*paralelizabilidade*
BP	$\propto (N\,l)$	$\propto (2P + N)$	Boa
GRAD	$\propto (N\,l + P)$	$\propto (2P + N)$	Boa
NM	$\propto (NP + 3P^2)$	$\propto (2P + N + P^2)$	Pobre
LM	$\propto (NP + 2P^2)$	$\propto (2P + N + P^2)$	Pobre
DFP	$\propto (NP + P^2)$	$\propto (2P + N + P^2)$	Pobre
BFGS	$\propto (NP + P^2)$	$\propto (2P + N + P^2)$	Pobre
OSS	$\propto (NP + 4P)$	$\propto (3P + N)$	Média
PR	$\propto (NP + 2P)$	$\propto (3P + N)$	Média
FR	$\propto (NP + 2P)$	$\propto (3P + N)$	Média
SCGM	$\propto (NP + P)$	$\propto (3P + N)$	Média
QUICK	$\propto (NP + 2P)$	$\propto (3P + N)$	Média

Como medida comparativa para os algoritmos apresentados no Capítulo 3, serão utilizados os seguintes critérios (STÄGER & AGARWALL, 1997; BELLO, 1992):

- quantidade de operações de ponto flutuante por época de treinamento (*flops/iteração*);
- necessidade de armazenamento (*memória*);
- facilidade de implementação paralela (*paralelizabilidade*).

Estas medidas serão apresentadas em relação aos parâmetros:

- P (número de parâmetros livres da rede);
- N (número de amostras para treinamento); e
- l (números de neurônios na última camada intermediária).

Começaremos uma análise da Tabela 5.1 pela coluna da *memória*. Neste caso, os métodos de primeira ordem necessitam armazenar apenas o vetor gradiente (∇J), o vetor de parâmetros da rede (θ) e os vetores de saídas para cada amostra de treinamento. Os vetores gradiente e de parâmetros são de dimensão proporcional ao número de parâmetros livres (P) a serem ajustados, e o vetor de saídas é proporcional ao número de saídas e à quantidade de amostras de treinamento. Os métodos de quase-Newton, LM e NM necessitam armazenar além dos vetores já citados, uma matriz (hessiana ou a aproximação de sua inversa) cuja

dimensão é da ordem $P \times P$, justificando o termo adicional (P^2) na tabela. Os métodos de gradiente conjugado não armazenam a matriz hessiana ou sua aproximação da inversa, mas por outro lado, necessitam do vetor gradiente (ordem P) na iteração anterior.

Para uma análise do esforço computacional por iteração (*flops/iteração*) é preciso considerar o procedimento de busca unidimensional do passo de atualização que está sendo utilizado, a necessidade ou não de se fazer a inversão e análise espectral das matrizes hessianas e a quantidade de unidades intermediárias da rede. A diferença básica do algoritmo padrão (BP) para o método do gradiente (GRAD) está no procedimento de busca utilizado para a taxa de aprendizagem. Este procedimento de busca simples é proporcional à avaliação da função de custo $J(.)$, que por sua vez, é proporcional ao número de parâmetros livres (P) da rede (BELLO, 1992). O método de Newton utiliza um procedimento de busca unidimensional ótima para o passo de ajuste e necessita da análise espectral e inversão da matriz hessiana a cada iteração. O método LM diferencia-se do método de Newton no sentido de que o primeiro não efetua o cálculo exato da informação de segunda ordem e também não realiza a análise espectral da aproximação da Hessiana. Os métodos de quase-Newton são menos custosos que os anteriores pois não necessitam da inversão da Hessiana efetuada pelos métodos NM e LM. Os métodos de gradiente conjugado e Quickprop são os métodos de segunda ordem menos custosos do ponto de vista computacional, principalmente para redes de dimensão elevada. O algoritmo SCGM não efetua o procedimento de busca unidimensional (ver Capítulo 3), e por isso efetua menos operações (*flops*) do que os outros métodos de gradiente conjugado.

5.4 Análise do Conjunto Amostral

Nesta seção estamos apresentando seis parâmetros que podem ser utilizados na descrição e avaliação dos conjuntos amostrais. ZHENG (1993) propõe, por exemplo, dezesseis grandezas para a descrição de problemas de classificação.

O objetivo de se definir grandezas é verificar se o conjunto de treinamento é capaz de representar o problema a ser tratado de maneira adequada.

Serão apresentadas as seguintes grandezas: *número de atributos* (NA), *número de valores nominais diferentes dos atributos* (DNAV), *existência de atributos irrelevantes* (IAtt), *tamanho do conjunto de dados* (TCD), *tamanho do espaço de descrição* (TED) e *densidade do conjunto amostral* (D). Algumas dessas grandezas são avaliadas diretamente ou medidas de forma simples, como o número de atributos (entradas) e o tamanho do conjunto de dados.

5.4.1 Número de Atributos (NA)

Corresponde à quantidade de entradas da rede. Classificação:

- *pequeno*: menor que 10;
- *médio*: entre 10 e 30;
- *grande*: maior que 30.

5.4.2 Número de Valores Nominais Diferentes dos Atributos (DNAV)

Analisa em quais intervalos estão os valores correspondentes a cada uma das entradas da rede. Classificação:

- *pequeno*: menor que 5;
- *médio*: entre 5 e 10;
- *grande*: maior que 10.

5.4.3 Atributos Irrelevantes (IAtt)

Verifica se existem, ou não, entradas que são redundantes ou pouco significativas. Para isso, podem ser utilizados métodos estatísticos, ou um treinamento não-supervisionado (CASTRO, 1997a-1998). Classificação:

- *não*: não possui atributos irrelevantes;
- *sim*: possui atributos irrelevantes.

5.4.4 Tamanho do Conjunto de Dados (TCD)

Quantidade de exemplos disponíveis para o treinamento. Classificação:

- *pequeno*: menos que 210 amostras;
- *médio*: entre 210 e 3170 amostras e
- *grande*: mais que 3170.

5.4.5 Tamanho do Espaço de Descrição (TED)

Permite verificar se o espaço que deve ser descrito é grande em relação à quantidade de amostras existentes para se fazer esta descrição. É dado pela seguinte expressão:

$$\text{Dimensão}-\text{do}-\text{espaço}-\text{de}-\text{descrição} = \prod_{i=1}^{n} N_i , \qquad (5.4)$$

onde n é o número de atributos e N_i é, para o i-ésimo atributo, o número de valores diferentes (se o atributo é binário ou nominal), ou o número de valores diferentes do conjunto (se o atributo é contínuo).

Classificação:

- *pequeno* – menor que 1.0×10^7;
- *médio* – entre 1.0×10^7 e 6.0×10^{18};
- *grande* – maior que 6.0×10^{18}.

5.4.6 Densidade do Conjunto Amostral (D)

Normalmente, o treinamento pode ser mais eficiente quando existe uma grande quantidade de dados disponível para treinamento. Entretanto, domínios diferentes possuem tamanhos diferentes para o espaço de descrição. É muito difícil dizer que um conjunto de dados contendo mais do que N exemplos é grande, e contendo menos do que N exemplos é pequeno. Além do tamanho do conjunto amostral, é necessário uma *densidade do espaço de descrição* (D) para caracterizar o conjunto de dados. A densidade D pode ser definida como:

$$D = \frac{\text{Número-de-exemplos}}{\text{Dimensão-do-espaço-de-descrição}}. \tag{5.5}$$

Classificação da densidade D:

- *baixa* – menor que 1.0×10^{-18};
- *média* – entre 1.0×10^{-18} e 6.0×10^{-7};
- *alta* – maior que 6.0×10^{-7}.

Capítulo 6

Generalização e Padronização

Neste capítulo veremos alguns critérios utilizados pelos procedimento de validação cruzada, comentários gerais sobre a teoria de regularização, regras para apresentação de resultados e por último citaremos problemas clássicos (*benchmarks*) utilizados pela comunidade científica voltada para o estudo de redes neurais artificiais.

6.1 Introdução

Quando estamos trabalhando com dados reais surgem alguns inconvenientes que geralmente, não existem nos problemas artificiais, dentre eles destacam-se:

- quantidade de amostras insuficiente;
- existência de dados redundantes e
- grande quantidade de ruído, inerente aos procedimentos de amostragem.

Procedimentos de *validação cruzada* (CV – do inglês *cross validation*) podem ser usados para detectar quando uma rede está sendo treinada de maneira excessiva (*overtraining*) e interromper o treinamento antes que isso ocorra.

6.2 Procedimentos de Validação Cruzada

Quando treinamos uma RNA, geralmente desejamos obter uma rede com a melhor capacidade de generalização possível, ou seja, a maior capacidade de responder corretamente a dados que não foram utilizados no processo de treinamento. As arquiteturas convencionais, totalmente interconectadas, como o MLP, estão sujeitas a sofrerem um sobre-treinamento (*overtraining*): quando a rede parece estar representando o problema cada vez melhor, ou seja, o erro do conjunto de treinamento continua diminuindo, em algum ponto deste processo a capacidade de responder a um novo conjunto de dados piora (PRECHELT, 1997).

6.2.1 O Efeito Bias/Variância

A capacidade de generalização pode ser tratada sob um ponto de vista predominantemente estatístico, principalmente levando-se em conta que o processo de amostragem está sujeito a ruído e que os dados são gerados a partir de uma função de densidade de probabilidade definida em X (VON ZUBEN, 1996). Tomando g: $X \subset \Re^m \rightarrow \Re^r$

como a função a ser aproximada, considere o conjunto de dados amostrados $\{(\mathbf{x}_l, \mathbf{s}_l)\}_{l=1}^{N}$ na forma $\mathbf{s}_l = g(\mathbf{x}_l) + \varepsilon_l$, $l = 1,\ldots,N$, onde ε_l expressa o erro no processo de amostragem (variável aleatória), sendo assumido ser de média zero e variância fixa (ver Capítulo 3). É assumido uma única saída.

Se $E(.)$ é o operador de esperança matemática considerando as N amostras, então $E(s / \mathbf{x})$ é a melhor aproximação de g em X, o que implica que, para qualquer função de aproximação $\hat{g}$, vale a seguinte relação (GEMAN *et. al.*, 1992):

$$E\left[(s-\hat{g})^2 / \mathbf{x}\right] \geq E\left[(s-E[s/\mathbf{x}])^2 / \mathbf{x}\right].$$

Com isso, um procedimento válido é minimizar a função

$$erro(\hat{g}) = E\left[(\hat{g}-E[s/\mathbf{x}])^2 / \mathbf{x}\right].$$

A função $erro(\hat{g})$ pode ser dividida em duas partes na forma (GEMAN *et. al.*, 1992):

$$erro(\hat{g}) = (E[\hat{g}]-E[s/\mathbf{x}])^2 + E\left[(\hat{g}-E[\hat{g}])^2\right], \tag{6.1}$$

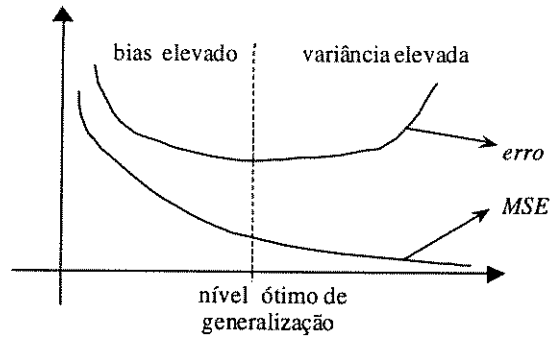
onde $E[\hat{g}]-E[s/\mathbf{x}]$ é denominado bias e $E\left[(\hat{g}-E[\hat{g}])^2\right]$ é denominado variância da função de aproximação $\hat{g}$.

Portanto a minimização de $erro(\hat{g})$ representa um compromisso entre os níveis de bias e variância associados à função de aproximação $\hat{g}$, de tal forma que uma função $\hat{g}$ muito flexível geralmente vai apresentar um bias muito baixo, mas uma variância elevada. Por outro lado, uma função $\hat{g}$ pouco flexível geralmente conduz a bias elevado e variância baixa. Desse modo, surgem as seguintes questões: como reduzir o bias sem aumentar proporcionalmente a variância e como reduzir a variância sem aumentar proporcionalmente o bias?

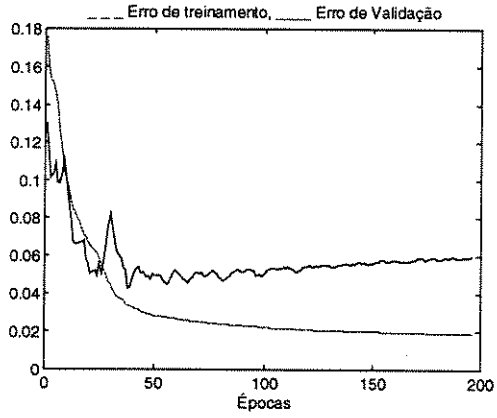
Considerando apenas a informação representada pelo conjunto de dados de aproximação $\{(\mathbf{x}_l, \mathbf{s}_l)\}_{l=1}^{N}$, é possível definir o erro quadrático médio (MSE – do inglês *mean squared error*) do problema de aproximação na forma (veja Capítulo 3):

$$MSE(\hat{g}) = \frac{1}{N}\sum_{l=1}^{N}(s_l - \hat{g}(\mathbf{x}_l))^2. \tag{6.2}$$

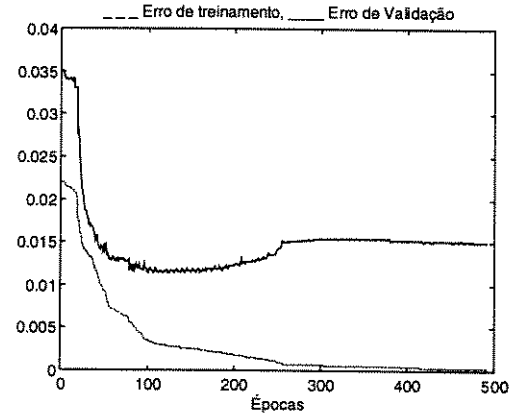
Comportamentos típicos de $erro(\hat{g})$, dado pela equação (6.1), e $MSE(\hat{g})$, dado pela equação (6.2), são apresentados na Figura 6.1(a), (b) e (c).



(a)



(b)



(c)

Figura 6.1: (a) Curvas típicas para $erro(\hat{g})$ e $MSE(\hat{g})$. (b) e (c) Comportamentos da curva de erro de treinamento e validação de uma rede MLP para problemas reais.

Existem duas formas básicas de se combater o sobre-treinamento (SARLE, 1995; PRECHELT, 1997): reduzindo a dimensão do espaço de parâmetros (dimensão da rede) ou reduzindo o tamanho efetivo da dimensão de cada parâmetro (valores dos pesos); ou seja, por seleção de modelo ou por regularização. Técnicas empregadas para produzir um espaço de parâmetros de dimensão reduzida são os *métodos construtivos*, *métodos de poda* e outros que serão analisados no Capítulo 7. Técnicas para reduzir a dimensão de cada parâmetro (elementos do vetor de pesos) são a regularização ou os procedimentos de validação cruzada que serão vistos neste capítulo.

Procedimentos de validação cruzada (CV) são largamente utilizados por serem de fácil entendimento e implementação. Também apresentam melhores resultados do que a teoria de regularização em alguns casos (PRECHELT, 1997-1998).

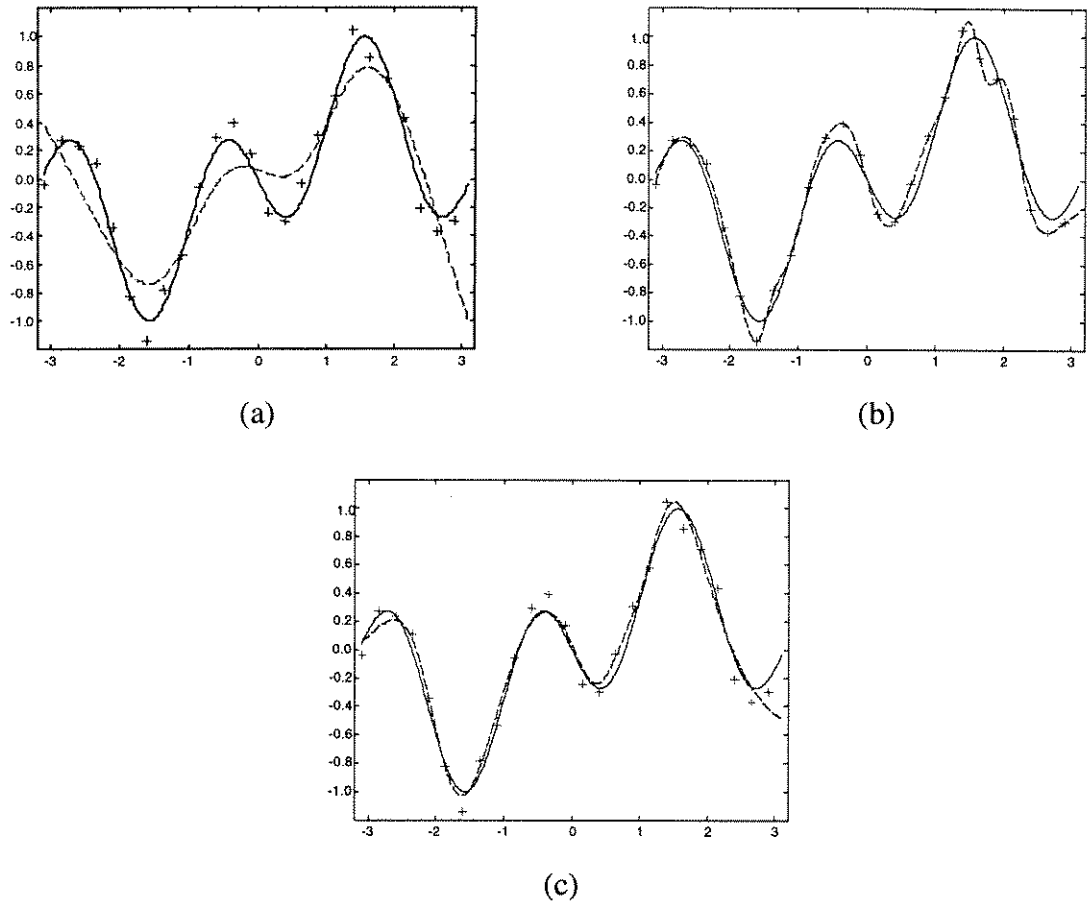


Figura 6.2: Função sen(x)×cos($2x$) com ruído uniformemente distribuído no intervalo [-0.15, 0.15]. + pontos de treinamento; — saída desejada; - - - saída da rede. (a) Treinamento interrompido muito cedo (*underfitting*). (b) Treinamento excessivo (*overfitting*). (c) Treinamento com procedimentos de validação cruzada (CV).

A Figura 6.2 mostra os três tipos de comportamentos resultantes da aproximação de funções realizada pelas redes MLP. No caso (a) a rede não treinou o suficiente para representar o conjunto de dados (*underfitting*); no caso (b) ocorreu o treinamento excessivo (*overfitting*) e no caso (c) foram utilizados procedimentos de validação cruzada para determinar um ponto ótimo de interrupção do treinamento.

6.2.2 Técnicas Básicas

A utilização de procedimentos de CV é geralmente feita da seguinte maneira:

- divida o conjunto de dados em *treinamento*, *validação* e *teste*;
- treine somente com o conjunto de treinamento e avalie o erro do conjunto de validação a cada k iterações (épocas);

- interrompa o treinamento quando o erro do conjunto de validação for maior do que era k iterações atrás e
- utilize o conjunto de pesos anteriores como o resultado do treinamento.

Esta abordagem utiliza o conjunto de validação para antecipar o comportamento em situações reais (ou em um conjunto de teste), assumindo que o erro em ambos os casos será semelhante: o erro de validação geralmente é uma estimativa do erro de generalização. A utilização de um conjunto de teste nem sempre é empregada, principalmente quando o número de amostras de treinamento é pequeno. Alguns pesquisadores preferem dividir o conjunto de dados em duas partes apenas e avaliar o comportamento do algoritmo após o treinamento utilizando outros conjuntos de dados.

A Figura 6.1(b) e (c) mostra o comportamento real das curvas de erro de treinamento e validação que ocorrem durante a evolução do processo de treinamento. Verifica-se que o erro do conjunto de validação sofre um decrescimento quase monotônico no início do processo, mas em seguida começa a aumentar continuamente conforme o treinamento prossegue. Outra característica importante destas curvas é a existência de vários mínimos locais. Não existe nenhuma regra clara para decidir quando o mínimo global do erro de generalização é obtido. As curvas de erro de validação real, como as apresentadas na Figura 6.1, quase sempre apresentam vários mínimos locais.

Infelizmente, as curvas apresentadas na Figura 6.1(b) e (c), ou quaisquer outras curvas de comportamento do erro de validação não são *típicas*, ou seja, as curvas não apresentam o mesmo comportamento qualitativo. Algumas curvas nunca alcançarão um mínimo melhor do que o primeiro, ou do que o terceiro; os vales e picos na curva podem ter tamanhos, larguras e formas diferentes. A única característica que todas as curvas têm em comum é que a diferença entre um mínimo local e os seguintes não é grande.

Como podemos ver, definir um critério de parada envolve uma relação entre tempo de treinamento e capacidade de generalização. Isso levanta a questão de qual critério utilizar, com o procedimento de validação cruzada, para decidir quando interromper o treinamento.

6.2.2.1 Particionamento do Conjunto de Dados

Não existe um consenso geral sobre como fazer o melhor particionamento do conjunto de dados, ou seja, dividi-lo de forma que possamos encontrar uma rede com a melhor capacidade de generalização em todos os casos. Por outro lado, para que haja sucesso nos

procedimentos de validação cruzada empregados, é necessário que os conjuntos sejam estatisticamente independentes (dentro do possível) e representativos do problema.

Em muitos casos, é possível gerar um conjunto de amostras para avaliar uma arquitetura desenvolvida ou aperfeiçoada. Quando isso ocorre, tem-se mais liberdade e controle sobre os dados, de forma que os pré-requisitos de independência estatística e relevância das amostras podem ser atendidos.

As abordagens mais clássicas (HAYKIN, 1994; PRECHELT, 1997) sugerem que o conjunto seja dividido da seguinte forma: 50% dos dados para treinamento, 25% para validação e 25% para teste.

AMARI *et. al.* (1996) propõem em seu artigo uma teoria estatística para o problema do sobre-treinamento, considerando procedimentos de validação cruzada e redes com dimensão muito elevada. É feita uma estimativa do valor percentual do conjunto de dados que deve ser usado para cada um dos três passos. Quando o número de parâmetros P da rede é grande, a melhor estratégia é utilizar quase todas as N amostras no conjunto de treinamento e utilizar apenas $1/\sqrt{2P}$ padrões para os outros conjuntos. Por exemplo, quando $P = 100$, somente 7% dos dados serão utilizados para validação e teste.

TETKO & VILLA (1997) investigam a possibilidade de combinar treinamento supervisionado e não-supervisionado para a partição do conjunto amostral. Esta estratégia é utilizada para fornecer um particionamento eficiente de um conjunto de dados contendo ruído. É proposto que sejam feitas várias subdivisões do conjunto amostral e que seja escolhido uma média de todos os valores obtidos como sendo o valor final. O algoritmo não-supervisionado é responsável pelas subdivisões dos dados.

6.2.3 Critérios de Parada

Existe uma grande quantidade de critérios de parada que podem ser utilizados. Para este estudo tomou-se três deles como base (PRECHELT, 1998). Para descrever um critério, façamos primeiro algumas definições. Seja *MSE* a função objetivo (erro quadrático médio – do inglês *mean squared error*). Então $MSE_{tr}(t)$ é o erro do conjunto de treinamento, $MSE_{val}(t)$ o erro do conjunto de validação utilizado pelo critério de parada e $MSE_{te}(t)$ o erro do conjunto de teste.

O valor $MSE_{opt}(t)$ é definido como sendo o menor erro do conjunto de validação obtido até o instante t:

$$MSE_{opt}(t) = \min_{t' \leq t} MSE_{val}(t). \tag{6.3}$$

Agora definimos a *perda de generalização* na iteração t como sendo o aumento relativo do erro de validação em relação ao mínimo (percentual).

$$GL(t) = 100.\left(\frac{MSE_{val}(t)}{MSE_{opt}(t)} - 1 \right). \tag{6.4}$$

Uma alta perda de generalização é um bom motivo para interromper o treinamento, pois indica diretamente que a rede está sendo sobre treinada. Isto resulta no primeiro critério de parada: pare assim que a *perda de generalização* supere um limiar ξ.

GL_α : pare após a t-ésima época caso $GL(t) > \xi$

Caso o treinamento esteja progredindo rapidamente, é desejável que ele não seja interrompido. A razão para isso é que quando o erro de treinamento está caindo muito rapidamente, sua queda sempre predomina sobre um possível aumento na variância. Para formalizar esta noção define-se um número de épocas k de treinamento, após o qual é feito o procedimento de validação cruzada. O *progresso de treinamento* (em milhares) medido após k iterações é dado por:

$$P_k(t) = 1000.\left(\frac{\sum_{t'=t-k+1}^{t} MSE_{tr}(t')}{k.\min_{t'=t-k+1}^{t} MSE_{tr}(t')} - 1 \right), \tag{6.5}$$

que significa "quanto a média do erro de treinamento durante k épocas é maior que o mínimo erro de treinamento durante k épocas?"

O critério de parada pode ser definido então como sendo o quociente entre a *perda de generalização* e o *progresso de treinamento*.

PQ_α : pare após a t-ésima época caso $\frac{GL(t)}{P_k(t)} > \xi$.

O último critério de parada estudado neste trabalho é, talvez, o mais conhecido de todos. Este critério interrompe o treinamento quando o erro de generalização aumenta s vezes a cada k épocas. Assim o critério de parada torna-se:

CV: pare após a época t se $MSE_{val}(t) > MSE_{val}(t - k)$ durante $s.k$ épocas.

Nenhum dos critérios apresentados garante uma interrupção do treinamento, portanto é importante definir um limite de épocas ou um limiar para o valor do erro ou do vetor gradiente.

6.3 Teoria de Regularização

Regularizadores são *funcionais de custo* adicionados ao funcional de erro que está sendo minimizado. Regularização é o procedimento de obtenção de um problema de aproximação bem-comportado a partir de um problema de aproximação mal-comportado pela imposição de restrições de suavidade ao modelo de aproximação (VON ZUBEN, 1996). As arquiteturas MLP minimizam a diferença entre as saídas da rede e as saídas desejadas (eqs. 3.14 e 3.2) em relação ao valor do vetor de parâmetros θ. Por conveniência as equações 3.14 e 3.2, respectivamente, serão repetidas aqui:

$$J(\theta) = \sum_{i=1}^{N} \sum_{j=1}^{m} \left(g_{ij}(\mathbf{x}) - \hat{g}_{ij}(\mathbf{x}, \theta)\right)^2 ,$$
$$\theta^* = \arg \min_{\theta \in \Re^P} J(\theta) , \tag{6.6}$$

onde g(.) é a função a ser aproximada e $\hat{g}(.)$ a aproximação fornecida pela rede.

Termos adicionais, cujo objetivo é reduzir a flexibilidade da rede neural no sentido de atender aos critérios de generalização, são chamados de *regularizadores* (JONDARR, 1996). Assim, ao mesmo tempo que minimiza o funcional de erro apresentado na equação (6.6), deve-se minimizar uma outra função, como por exemplo, a soma dos pesos da rede.

Alguns métodos de regularização comumente utilizados nas redes MLP serão discutidos brevemente a seguir.

6.3.1 Decaimento dos Pesos (*Weight Decay*)

É o regulador mais conhecido. O princípio básico é minimizar a soma dos quadrados dos pesos de cada conexão, evitando um aumento exagerado dos valores dos pesos das unidades intermediárias. Utilizando o custo com relação aos pesos (C_{decay}) e a equação (6.6), podemos definir o funcional de custo total (C_{total}):

$$C_{decay}(\theta) = \sum_{i=1}^{P} \theta_i^2 ,$$
$$C_{total} = J(\theta) + C_{decay}(\theta). \tag{6.7}$$

Tomando a derivada desta função teremos uma lei de ajuste do vetor de parâmetros que deve considerar dois termos na minimização do funcional de custo total.

6.3.2 Regularizador de Rumelhart

RUMELHART *et. al.* (1986) introduziram duas medidas diferentes de complexidade em função dos parâmetros da rede. C_{pesos} mede a complexidade de todos os parâmetros da rede:

$$C_{pesos} = \frac{\theta_i^2}{1+\theta_i^2}. \tag{6.8}$$

C_{unid} mede os pesos em cada unidade, encorajando cada nó a ser independente do vizinho.

$$C_{unid} = \frac{\theta_i^2}{1+\sum_i \theta_i^2}. \tag{6.9}$$

Também existe um fator, λ, que considera uma proporcionalidade entre o erro e a complexidade:

$$C_{total} = \lambda\, J(\theta) + (1-\lambda)\, Complexidade(\theta). \tag{6.10}$$

A equação (6.8) é utilizada para determinar a seguinte relação:

$$\frac{\partial C}{\partial \theta_i} = \frac{2\theta_i}{\left(1+\theta_i^2\right)^2}. \tag{6.11}$$

Neste caso a equação (6.1) é implementada com λ iniciando em 1 e aumentando até que o desempenho piore, ou seja, até que a complexidade domine o termo de erro.

6.3.3 Minimização do Produto dos Pesos

Ao invés de minimizar a soma dos pesos em uma dada unidade, uma alternativa é minimizar o produto dos pesos. Seja $\mathbf{w}$ a matriz de pesos correspondente ao vetor de parâmetros θ, então:

$$C_{pesos} = \sum_i \left| \prod_j \mathbf{w}_{ij} \right|, \tag{6.12}$$

fornecendo a seguinte relação:

$$\frac{\partial C_{pesos}}{\partial \mathbf{w}_{ij}} = \left| \prod_{k \neq j} \mathbf{w}_{ik} \right|. \tag{6.13}$$

6.3.4 Regularizador de Girosi

GIROSI *et. al.* (1995) introduziram uma função de regularização φ na forma:

$$\varphi(\hat{g}) = \int_X \frac{\left|\hat{G}(\mathbf{z})\right|^2}{F(\mathbf{z})} d\mathbf{z}, \tag{6.14}$$

onde $\hat{G}$ e F são as transformadas de Fourier de $\hat{g}$ e f respectivamente. F é uma função definida positiva previamente especificada que se aproxima de 0 quando $\|\mathbf{z}\| \to \infty$, de tal forma que $1/F$ corresponda a um filtro passa-alta. A função F é assumida ser simétrica (função par) e definida de forma a garantir a existência de $\hat{g}$ tal que a integral da equação (6.14) exista (VON ZUBEN, 1996).

Neste caso, para N finito, o problema de aproximação regularizado é encontrar o modelo de aproximação $\hat{g}$ que minimize o seguinte funcional:

$$J(\hat{g}) = \frac{1}{N}\sum_{l=1}^{N}\left(s_l - \hat{g}(\mathbf{x}_l)\right)^2 + \lambda\varphi(\hat{g}), \tag{6.15}$$

onde λ é um número positivo denominado parâmetro de regularização. O primeiro termo, como discutido anteriormente, representa uma medida de distância entre g e $\hat{g}$, enquanto o segundo termo representa o grau de suavidade da função $\hat{g}$. O compromisso entre suavidade do processo de aproximação e fidelidade aos dados pertencentes ao conjunto de aproximação é, portanto, controlado pelo parâmetro de regularização λ.

6.3.5 Unidades Competitivas

Este método remove a complexidade sem utilizar um regularizador propriamente dito. As unidades devem competir para aumentar os seus pesos em qualquer direção do espaço. Cada conexão possui uma direção (vetor de pesos normalizado), $\mathbf{w}^*$ e um *fator de ganho* (magnitude), g, que determina o vetor de pesos utilizado pelo neurônio.

Os neurônios terão mais facilidade para detectar características na camada intermediária.

Após cada retro-propagação do erro, a mudança no fator de ganho g, para um peso no nó i, é dada por:

$$\begin{aligned} \Delta g_i &= -\gamma \sum_{i \neq j} \left(\mathbf{w}_i^* . \mathbf{w}_j^*\right)^2 g_i \\ \mathbf{w}_i &= g_i \mathbf{w}_i^* \end{aligned} \tag{6.16}$$

6.4 Padronização

Esta seção discute porque conjuntos de treinamento padronizados (*benchmarks*) e regras de padronização para o aprendizado e apresentação de resultados de redes neurais artificiais são realmente necessárias. Comenta-se também sobre a necessidade de se utilizar problemas reais e artificiais para validar procedimentos teóricos.

6.4.1 Por que Utilizar um Benchmark?

Estudos recentes sobre algoritmos de treinamento de RNA's (PRECHELT, 1996) mostraram que a avaliação de desempenho dos algoritmos é um aspecto para o qual a comunidade científica da área não tem dado muita atenção, dificultando o "controle de qualidade" dos novos algoritmos propostos na literatura. A maior parte dos artigos encontrados em revistas e jornais apresentam resultados de desempenho considerando apenas um pequeno número de problemas – raramente mais do que três. Na maioria dos casos um ou mais destes problemas é artificial e pouco representativo, constituindo problemas de paridade, simetria ou codificação. Geralmente não são feitas comparações com algoritmos desenvolvidos por outros pesquisadores (com exceção do algoritmo padrão).

Por que isso ocorre? Várias justificativas têm sido empregadas (PRECHELT, 1994):

1. O treinamento de RNA's é demorado, tomando muito tempo de CPU.
2. Os algoritmos de outros pesquisadores geralmente não estão disponíveis sob a forma de programas; ou suas implementações não são estáveis ou são baseadas em algum ambiente exótico.
3. É difícil conseguir dados referentes a problemas reais.
4. É um processo complicado preparar dados para o treinamento de RNA's.
5. Resultados independentes obtidos para o mesmo problema não podem ser comparados diretamente, pois existem várias representações para o mesmo problema ou diferentes metodologias de experimentação.

No entanto, nenhum destes argumentos é convincente pelos seguintes motivos:

1. Não é realmente um problema. As máquinas disponíveis hoje em dia são rápidas o suficiente para que o tempo de treinamento seja, pelo menos, bem menor que o tempo de desenvolvimento do algoritmo.
2. Normalmente é verdade, mas não seria um problema se pudéssemos comparar com os resultados diretamente apresentados pelos próprios autores.

3. Parcialmente verdadeiro. Muitos pesquisadores que utilizaram dados reais em suas pesquisas estão dispostos a fornecer os conjuntos de dados que foram testados. Existem também coleções acessíveis de dados, como por exemplo o *UCI Repository of Machine Learning Databases and Domain Theories*, (URL 2).
4. Correto. Realmente é, mas nem todos precisam fazer a preparação dos dados. Como uma comunidade de pesquisadores, é necessário que haja divulgação e disponibilidade de resultados já obtidos.
5. Este é um problema real que apresenta duas faces: muitas vezes a preparação dos experimentos é feita de forma a invalidar seus resultados e algumas vezes os experimentos não são documentados de maneira coerente.

Esta discussão mostra que é necessária uma padronização dos conjuntos de dados e forma de apresentação dos resultados. Aspectos de algoritmos de aprendizagem que podem ser estudados utilizando a abordagem que é apresentada aqui são, por exemplo, velocidade de convergência, capacidade de generalização e facilidade de seleção de parâmetros definidos pelo usuário. Por outro lado, não são tratados aqui aspectos referentes à representação adequada de dados.

Várias áreas de pesquisa possuem um conjunto de benchmarks bem definidos. Atualmente não existe uma metodologia amplamente aceita para medir e comparar a velocidade de algoritmos de aprendizagem em RNA's (FAHLMAN, 1988). Alguns pesquisadores propõem novos algoritmos baseados apenas em uma análise teórica. Geralmente é difícil avaliar o desempenho prático destes modelos teóricos.

6.4.2 Por que Estabelecer Padronização?

A discussão anterior mostra que utilizar conjuntos de dados padronizados (*benchmarks*) é uma condição necessária mas não suficiente para aumentar, verdadeiramente, a qualidade das publicações científicas da área de RNA's. Um incremento na qualidade é realmente alcançado se os resultados que forem publicados puderem ser reproduzidos e comparados. Isto não é trivial, uma vez que cada aplicação de um algoritmo de treinamento a um problema particular envolve um número significativo de parâmetros que são definidos pelo usuário. Se pelo menos um destes parâmetros não for publicado com os resultados, o experimento torna-se irreprodutível e uma comparação não pode ser feita.

Assim, um conjunto de problemas padrões (*benchmark sets*) deve ser complementado com um conjunto de regras padronizadas (*benchmark rules*) que descrevem e padronizam

formas de se conduzir e documentar experimentos. Tais regras não reduzem a liberdade de escolha de várias formas possíveis de condução e apresentação de experimentos, apenas sugere um conjunto de passos a serem seguidos com o intuito de maximizar a comparabilidade dos resultados experimentais e mostrar o que deve ser documentado e de que forma. Estas regras reduzem o risco de pesquisadores produzirem resultados inválidos.

6.4.3 Regras Padronizadas (*Benchmarking Rules*)

Esta seção descreve:

- como conduzir um experimento e
- como publicar os resultados.

O propósito das regras é garantir a validação e reprodução dos resultados por outros pesquisadores. Um benefício adicional à padronização é que os resultados serão mais diretamente comparáveis.

6.4.3.1 Princípios Gerais

Os princípios gerais para formulação das regras são:

Validade: É necessário que haja um mínimo de padronização do experimento para garantir que os resultados obtidos sejam válidos, no sentido de que não constituam artefatos criados por fatores aleatórios ou por algum erro experimental.

Reprodutividade: Todos os aspectos dos experimentos necessários para sua fiel reprodução devem estar coerentemente documentados. Resultados que não podem ser reproduzidos não são resultados científicos.

Comparabilidade: É muito importante que seja possível fazer uma comparação direta dos resultados apresentados. As regras, apresentadas aqui, recomendam várias escolhas padrões que devem ser utilizadas, a menos que existam razões específicas para não fazê-lo.

6.4.3.2 Conjunto de Treinamento, Validação e Teste

Para cada problema X a ser considerado, deve-se indicar exatamente a procedência dos dados para treinamento. Se for possível, especificar quando e onde outros pesquisadores utilizaram estes dados.

Quando estamos verificando a capacidade de generalização das redes neurais artificiais utilizando procedimentos de validação cruzada é necessário dividirmos o conjunto

de dados em pelo menos duas partes: uma na qual o treinamento é efetuado (conjunto de treinamento), e outra que é utilizada na avaliação de desempenho (conjunto de validação). O conjunto de validação deve avaliar o comportamento do algoritmo quando submetido a dados de teste. Isso significa que os dados utilizados para validação não devem estar disponíveis para o treinamento; senão o benchmark é inválido.

Como visto neste capítulo, em muitos casos existe o conjunto de teste, que é utilizado para avaliar o desempenho da rede treinada, quando em operação. Diferentes partições do conjunto amostral resultarão em diferentes conclusões. É importante apresentar quantos e, se possível, quais são os dados que fazem parte de cada conjunto.

Maiores detalhes sobre procedimentos de validação cruzada podem ser vistos na Seção 6.1 deste capítulo.

6.4.3.3 Representação de Entrada e Saída

A representação dos atributos de entrada e saída de um problema de aprendizagem influencia a qualidade das soluções obtidas. Dependendo do tipo de problema, existem várias maneiras diferentes de representação dos atributos. Para cada tipo de atributo existem vários métodos plausíveis de representação. Vamos discutir alguns tipos de atributos e suas representações:

Atributos com valores reias: são geralmente normalizados por algum operador que os mapeiam no intervalo [0, 1] ou [-1, 1]. Procedimentos de normalização dos dados de entrada são apresentados no Capítulo 3.

Atributos com valores inteiros: geralmente são tratados como os atributos reais. Se o número de valores diferentes é pequeno, uma das representações utilizadas para atributos ordinais deve ser apropriada.

Atributos ordinais: com m valores distintos são mapeados em uma escala eqüidistante transformando-os em valores pseudo-reais, ou são representados por m-1 entradas das quais a k-ésima mais a esquerda tem valor 1 para representar o valor do k-ésimo atributo enquanto todos os outros são 0. Um código binário utilizando apenas $\log_2 m$ entradas também pode ser usado. Este tipo de atributo aparece muito raramente.

Atributos nominais: com m valores diferentes são geralmente representados utilizando um código binário ou 1-de-m.

Atributos com valores inexistentes: podem ser substituídos por um valor fixo (por exemplo, a média dos valores existentes) ou pode ser representado explicitamente adicionando outra entrada ao atributo que é 1.

Grande parte da discussão acima também é aplicável às saídas da rede, exceto pelo fato de que os valores de saída sempre existem.

Ao utilizar benchmarks que não fazem parte de um conjunto bem definido de dados, é importante especificar exatamente a representação de entrada e saída utilizada.

6.4.3.4 Algoritmo de Treinamento

Uma especificação exata do algoritmo de treinamento utilizado é fundamental. Ao utilizar um algoritmo conhecido, especifique com referências que descrevam e utilizam o algoritmo de maneira precisa. Caso sejam feitas alterações no algoritmo original, apresente-as detalhadamente. Se está propondo um novo algoritmo, dê-lhe um nome para que outros autores possam fazer referência ao seu trabalho. Se existem muitas variantes do seu algoritmo, dê a cada variante um nome, talvez acrescentando um dígito ou letra ao nome principal.

Novo, ou não, especifique claramente para o seu algoritmo os valores de todos os parâmetros livres utilizados. Ao introduzir um novo algoritmo você deve indicar claramente um **vetor de parâmetros** protótipo (incluindo nomes de parâmetros) que deve ser apresentado na documentação de cada algoritmo. Um erro muito comum é deixar algum parâmetro não especificado.

Estes parâmetros devem incluir (dependendo do algoritmo) taxa de aprendizagem, momento, taxa de decaimento dos pesos (*weight decay*), inicialização, etc. Para cada parâmetro deve haver um único nome e talvez um símbolo. Para cada parâmetro adaptativo, a regra de adaptação deve ser explicitada. Um aspecto particularmente importante dos algoritmos de treinamento é o critério de parada.

Procure especificar porque foi escolhido um valor para determinado parâmetro, e se possível, caracterizar a sensibilidade do algoritmo a esta escolha.

6.4.3.5 Medidas de Erro

Existem várias medidas de erro (também denominadas de funcional de erro, função objetivo, função custo ou função de perda) que podem ser utilizadas para o treinamento. A mais comum é o erro quadrático apresentado na equação (6.6). Alguns pesquisadores multiplicam esta equação por ½ com o objetivo de tornar a derivada mais simples; isso não é

considerado padrão. Esta medida apresenta o valor do erro para todos os padrões. Também pode ser utilizado como medida o *erro quadrático médio* (MSE), que possui a vantagem de ser independente do tamanho do conjunto de dados.

$$J(\theta) = \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{m}\left(g_{ij}(\mathbf{x}) - \hat{g}_{ij}(\mathbf{x},\theta)\right)^2 \qquad (6.17)$$

Outras medidas de erro que podem ser citadas são: *softmax error, cross entropy,* erro linear, erro exponencial e erro de variância mínima (PRECHELT, 1994; SHEPHERD, 1997).

Para problemas de classificação, a função alvo geralmente não é a medida contínua de erro utilizada durante o treinamento, mas o desempenho de classificação. O desempenho de classificação deve ser apresentado como um percentual de padrões classificados incorretamente, o erro de classificação.

6.4.3.6 Arquitetura Utilizada

Especifique exatamente a topologia (arquitetura) da rede utilizada durante um experimento. A arquitetura de uma rede é descrita pelo grafo de nós (unidades, vértices, neurônios) e conexões (pesos, sinapses). O termo 'peso' deve ser usado para se referir ao parâmetro ligado a uma conexão, mas não à própria conexão.

Para descrever a topologia faça referência a modelos comumente utilizados. Para as redes do tipo MLP totalmente conectadas, o número de nós em cada camada pode ser dado como uma seqüência: **uma rede 5-4-6**, ou **uma rede [5, 4, 6]** que referem-se a uma rede com 5 entradas, 4 unidades intermediárias e 6 saídas. Existe confusão em como contar a quantidade de camadas de uma rede, então não chame a rede acima uma "rede com três camadas" (contando todos os grupos de nós) nem uma "rede com duas camadas" (contando somente os nós com conexões de entradas). Prefira a expressão **uma rede com uma camada intermediária** (ou **escondida**).

Especificar o número de unidades não é o suficiente, utilize expressões como **com todas as conexões de propagação direta (*feedforward*) entre camadas adjacentes** ou algo semelhante.

Grande parte das redes também apresentam um limiar (*bias* ou *threshold*) para as unidades intermediárias e/ou de saída. Especifique se a rede possui ou não os limiares. Ao calcular a quantidade de parâmetros livres da rede inclua todos os limiares.

Para redes recorrentes padrões utilize nomes usuais como, **Jordan** ou **Elman**, onde apropriado. Arquiteturas de rede que não são padrões devem ser detalhadas.

6.4.3.7 Resultados Experimentais

Se estivermos interessados na capacidade de generalização da rede, o erro de teste é geralmente o resultado principal a ser apresentado para qualquer problema tratado. Os erros de treinamento e validação são grandezas de interesse marginal, e não necessariamente precisam ser apresentados. Por outro lado, como o conjunto de dados não é dividido de maneira rigorosa, e como não existem critérios bem definidos de particionamento dos dados, é aconselhável que pelo menos o erro de validação seja apresentado.

Como o treinamento de redes neurais geralmente envolve algum processo de inicialização, os resultados de várias simulações do mesmo algoritmo com condições iniciais distintas serão diferentes, por haver uma dependência em relação à condição inicial. Para apresentar resultados que representem bem o desempenho dos algoritmos, devem ser feitas várias simulações e apresentada uma estatística da distribuição. Se possível utilizar 10 simulações, ou alguma potência de 10 para produzir a distribuição. As estatísticas geralmente utilizadas para apresentar os resultados são a média e o desvio padrão do erro do conjunto de teste, o melhor resultado e em seguida os valores mínimo e máximo da distribuição.

6.4.3.8 Tempo de Treinamento

Comentários gerais sobre tempo de treinamento e esforço computacional despendido na solução de um problema foram vistos no Capítulo 5 que trata dos aspectos numéricos do treinamento de redes MLP. Considera-se como medida de desempenho o número de épocas de treinamento necessário para cada algoritmo convergir e faz-se um estudo sobre a quantidade de operações de ponto flutuante (*flops*) consumida por iteração, para cada um dos algoritmos apresentados no Capítulo 3.

Caso seu algoritmo não requeira grande quantidade de operações adicionais além das usuais, a época constitui uma medida sensível do tempo de treinamento e pode ser vista como a quantidade de vezes que as conexões são "atravessadas" (pode-se considerar a atualização dos pesos). Esta medida é importante pois independe da máquina ou implementação. A propagação dos dados e a retro-propagação do erro pela rede podem ser analisadas separadamente para algoritmos que requerem maior quantidade de uma grandeza do que de outra.

Se seu algoritmo executa por época várias operações adicionais, além das usuais, como procedimentos de busca unidimensional, escalonamento, análise espectral e inversão de matrizes (ver Capítulos 3 e 4), então uma medida de esforço computacional deve ser

apresentada. Um dos grandes problemas dos resultados apresentados desta forma é que dependem da eficiência na implementação do algoritmo. Por isso, é recomendável que os algoritmos sejam implementados de maneira semelhante e de preferência utilizando o mesmo software computacional. Ao mencionar tempo de CPU é importante especificar a configuração da máquina utilizada para simulação; apresente comentários sobre a eficiência do algoritmo implementado.

É importante especificar critérios de parada e procedimentos de reinicialização.

6.4.3.9 Detalhes Finais

Alguns detalhes importantes que não devem ser esquecidos:

Função de ativação: especifique exatamente a função de ativação utilizada para cada unidade, ou camada. Indique se as unidades de saída possuem função de ativação diferente da linear na saída.

Método de inicialização: especifique qual o tipo de inicialização da rede. Diferentes procedimentos de inicialização serão vistos posteriormente. Assim como o critério de parada, o ponto inicial da rede possui grande impacto no resultado final apresentado.

O capítulo 9 apresenta os resultados computacionais para os diversos algoritmos implementados neste trabalho. Os resultados apresentados seguem os modelos sugeridos nesta seção, incluindo descrições detalhadas dos conjuntos amostrais, algoritmos, valores de parâmetros livres, medidas de desempenho, critérios de parada e várias referências.

Capítulo 7

Limitações

Os principais problemas das arquiteturas MLP que serão abordados são a sensibilidade a mínimos locais, o não conhecimento a priori da arquitetura ótima de rede e a implementação destes modelos em paralelo. A sensibilidade a mínimos locais será reduzida através de uma definição mais cuidadosa do conjunto inicial de pesos de treinamento; métodos construtivos e de poda serão apresentados com o objetivo de otimizar a determinação de uma arquitetura ótima de rede e comentários gerais sobre técnicas de implementação em paralelo destas arquiteturas serão revistos.

7.1 Introdução

A velocidade de treinamento das redes do tipo MLP depende principalmente:

- da condição inicial dos pesos e limiares;
- da determinação adequada das funções de ativação;
- da arquitetura da rede;
- do conjunto de dados e
- do algoritmo de treinamento.

O algoritmo de treinamento pode ser visto como um procedimento de otimização não-linear irrestrita, e como todo método iterativo de otimização necessita da definição de um ponto inicial de busca no espaço de solução. A superfície de erro caracterizada pela função a ser minimizada apresenta, em geral, uma grande quantidade de mínimos locais, e portanto a definição de uma condição inicial adequada se faz necessária.

As funções de ativação a serem utilizadas nas unidades (neurônios) que compõem a rede exercem um papel muito importante no desempenho do modelo. Alguns aspectos que devem ser considerados na escolha da função de ativação são, por exemplo, satisfazer o critério de aproximação universal e facilidade de computação (esforço computacional).

Outro aspecto importante na determinação de um modelo de aproximação é a escolha da dimensão deste modelo. Métodos de determinação da dimensão ótima da arquitetura a ser utilizada serão apresentados e comparados.

Vários algoritmos eficientes de treinamento para arquiteturas do tipo perceptron com múltiplas camadas foram apresentados nos Capítulos 3 e 4.

Por fim, podemos citar a capacidade de implementação paralela dos algoritmos como sendo uma das principais características das RNA's.

7.2 Condições Iniciais Ótimas

A qualidade e eficiência do aprendizado supervisionado em redes multicamadas depende fortemente da especificação de arquitetura da rede, função de ativação dos neurônios, regra de aprendizagem, valores iniciais do vetor de parâmetros (pesos) e dos dados de treinamento.

Valores ótimos destes itens são desconhecidos *a priori*, pois dependem principalmente do conjunto de treinamento e da natureza da solução (THIMM & FIESLER, 1997).

Assumimos aqui que a arquitetura da rede, as funções de ativação dos neurônios e a regra de aprendizado já foram determinados adequadamente, embora não necessariamente de maneira ótima. Sob essas considerações, um processo de treinamento bem sucedido passa a depender somente de uma boa definição do conjunto inicial de pesos, ou seja, um conjunto que guie o processo de treinamento para uma solução satisfatória, fora de mínimos locais pobres e problemas de instabilidade numérica.

O ajuste dos pesos da rede neural, a partir de um conjunto de dados de entrada-saída (conjunto amostral), é denominado treinamento supervisionado, e pode ser visto como um problema de otimização, cuja função de custo é função do conjunto de pesos, também conhecido como parâmetros da rede neural (FAHLMAN, 1988).

A importância de uma boa escolha do conjunto de pesos iniciais é enfatizada por KOLEN & POLLAK (1990). Eles mostraram que uma busca global pelo conjunto ótimo de pesos não é factível. Assim, por questões práticas, a regra de aprendizagem pode ser baseada em técnicas de otimização que empregam busca local (SHEPHERD, 1997). Por outro lado, a busca local implica que a solução a ser obtida está fortemente relacionada à condição inicial, pois cada condição inicial pertence à base de atração de um mínimo local particular, que atrairá para a solução (HERTZ *et. al.*, 1991).

Conseqüentemente, apenas mínimos locais podem ser produzidos, na prática, como resultados de um processo de treinamento supervisionado bem sucedido. Se este mínimo for o mínimo global, ou um bom mínimo local da função de custo, o resultado é uma rede neural

adequadamente treinada. Caso contrário, quanto pior for o mínimo local, pior o desempenho da rede treinada.

Diversas estratégias de inicialização dos pesos já foram sugeridas. Os métodos mais simples baseiam-se em uma distribuição uniforme aleatória (KOLEN & POLLAK 1990), representando a ausência total de conhecimentos sobre o conjunto amostral. Considerando abordagens melhor elaboradas, existem basicamente dois paradigmas alternativos para busca do melhor conjunto inicial de pesos via treinamento supervisionado, ou seja, melhor condição inicial para o processo de otimização resultante:

- paradigma do caminho mais fácil: não é tão comum na literatura. A idéia básica é fornecer uma condição inicial não necessariamente próxima da solução ótima, mas que seja tal que o processo de treinamento possa evoluir mais rapidamente, em média, e mais eficientemente, a partir da condição inicial. A estratégia mais simples é definir automaticamente um intervalo de inicialização para os pesos e utilizar uma distribuição uniforme neste intervalo.
- paradigma do caminho mais curto: é a abordagem geralmente empregada na literatura. A idéia básica é fornecer uma condição inicial o mais próxima possível da solução ótima, ainda desconhecida. A idéia intuitiva por trás desta abordagem é que quanto mais próxima da solução ótima estiver a condição inicial, menor a probabilidade de existência de mínimos locais no caminho até esta solução, e mais eficiente se torna o processo de treinamento. Duas estratégias podem ser consideradas: extração de conhecimento do conjunto de treinamento com o objetivo de descobrir peculiaridades da superfície de otimização (baseado em aspectos teóricos), ou exploração da superfície de otimização para aumentar as chances de se encontrar uma região promissora na busca pelo ótimo (baseada em aspectos heurísticos).

O paradigma do caminho mais fácil geralmente ignora o conjunto de treinamento na tentativa de definir um bom intervalo de valores para os pesos. Como conseqüência, o caminho entre a condição inicial e a solução ótima, embora fácil de ser atravessado, pode ser muito longo.

Por outro lado, o paradigma do caminho mais curto considera todo o conjunto de treinamento, mas geralmente ignora as conseqüências da combinação ⟨dados de entrada-saída⟩ + ⟨pesos⟩ no processamento de sinais da rede neural. Como conseqüência, o caminho entre a condição inicial e a solução ótima, embora curto, pode ser muito difícil de ser atravessado.

Na Seção 7.2.3 iremos propor um paradigma híbrido, que representa um compromisso entre o caminho mais fácil e o caminho mais curto, buscando assim definir um método de inicialização mais eficiente e robusto.

7.2.1 Exemplos do paradigma do caminho mais fácil

FAHLMAN (1988) realizou estudos sobre técnicas de inicialização aleatória para redes multicamadas. Ele propôs o uso de uma distribuição uniforme no intervalo [-1.0, 1.0], mas resultados experimentais mostraram que o melhor intervalo de inicialização para os problemas por ele abordados variaram entre [-0.5, 0.5] e [-4.0, 4.0].

Alguns pesquisadores tentaram determinar o melhor intervalo de inicialização utilizando outros parâmetros da rede. Seja d_{in} o *fan-in* do neurônio. BOERS & KUIPER (1992) inicializam os pesos utilizando uma distribuição uniforme no intervalo $\left[-3/\sqrt{d_{in}}, 3/\sqrt{d_{in}}\right]$, sem apresentar nenhuma justificativa matemática.

NGUYEN & WIDROW (1990) propõem uma simples modificação do processo aleatório de inicialização do conjunto de pesos. A abordagem é baseada em uma análise geométrica da resposta dos neurônios intermediários a uma única entrada; a análise é estendida para o caso de múltiplas entradas utilizando transformada de Fourier. Os pesos que ligam as unidades intermediárias às unidades de saída são inicializados em valores pequenos no intervalo [-0.5, 0.5]. A inicialização dos pesos das unidades de entrada para as unidades intermediárias é projetada para aumentar a habilidade dos neurônios intermediários aprenderem. O fator de escala é dado por: $\beta = 0.7(l)^{1/k} = 0.7\sqrt[k]{l}$, onde l é o número de unidades escondidas e k é o número de entradas. Os pesos são inicializados aleatoriamente no intervalo [-0.5, 0.5] e em seguida a seguinte transformação é aplicada: $\mathbf{v} = \frac{\beta \mathbf{v}}{\|\mathbf{v}\|}$, os limiares permanecem com valores no intervalo especificado acima.

KIM & RA (1991) calculam o limite inferior para o comprimento inicial do vetor de pesos como sendo $\sqrt{\alpha/d_{in}}$, onde α é a taxa de aprendizagem.

7.2.2 Um exemplo do paradigma do caminho mais curto

LEHTOKANGAS *et. al.* (1995) propõem um método baseado no algoritmo dos quadrados mínimos ortogonais (OLS). O algoritmo OLS, tem sido largamente utilizado com sucesso no treinamento de redes do tipo *função de base radial* (RBF) (CHEN *et. al.*, 1996).

7.2.2.1 Algoritmo dos Mínimos Quadráticos Ortogonais (OLS)

Uma arquitetura MLP pode ser considerada como um modelo de regressão onde os neurônios intermediários são os regressores. No processo de inicialização dos pesos o problema é determinar qual o melhor regressor disponível. Um algoritmo eficiente para determinação do regressor ótimo é o algoritmo OLS. Sejam p e q o número de unidades de entrada e intermediárias, respectivamente.

Seja **s** a saída desejada da rede. Um modelo de regressão para esta saída pode ser dado, em notação matricial, pela expressão:

$$\mathbf{s} = \mathbf{R}\theta + \varepsilon, \tag{7.1}$$

onde $\mathbf{s} = [\mathbf{s}^1, \mathbf{s}^2, ..., \mathbf{s}^n]^T$, **R** é a matriz de regressores (funções fixas das entradas), $\theta = [\theta_0, \theta_1, ..., \theta_M]^T$ são parâmetros do modelo e $\varepsilon = [\varepsilon^1, \varepsilon^2, ..., \varepsilon^n]^T$ são os ruídos absorvidos durante o processo de amostragem. M é o número de regressores.

$$\mathbf{R} = \begin{bmatrix} 1 & R_1[\mathbf{x}^1] & \cdots & R_M[\mathbf{x}^1] \\ \cdots & \cdots & \cdots & \cdots \\ 1 & R_1[\mathbf{x}^n] & \cdots & R_M[\mathbf{x}^n] \end{bmatrix}. \tag{7.2}$$

A matriz de regressão **R** pode ser decomposta em:

$$\mathbf{R} = \mathbf{HU}, \tag{7.3}$$

onde a equação (7.3) representa uma decomposição ortogonal, com $\mathbf{U}(M+1)\times(M+1)$ sendo triangular superior com 1 em toda a sua diagonal

$$\mathbf{U} = \begin{bmatrix} 1 & \alpha_{12} & \alpha_{13} & \cdots & \alpha_{1(M+1)} \\ 0 & 1 & \alpha_{23} & \cdots & \alpha_{2(M+1)} \\ 0 & 0 & 1 & \cdots & \alpha_{3(M+1)} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \tag{7.4}$$

H é uma matriz $n \times (M+1)$ com colunas ortogonais $\mathbf{h}_j$ tais que:

$$\mathbf{H}^T\mathbf{H} = \mathbf{B} \tag{7.5}$$

A matriz **B** é diagonal e seus elementos são:

$$b_{jj} = \mathbf{h}_j^T \mathbf{h}_j = \sum_{l=1}^{n} \left(h_j^l\right)^2, \qquad j = 1,2,...,M+1. \tag{7.6}$$

A equação (7.1) pode ser reescrita como

$$\mathbf{s} = \mathbf{H}\mathbf{g} + \varepsilon, \tag{7.7}$$

quando a condição $\mathbf{U}\mathbf{w} = \mathbf{g}$ é satisfeita. A estimativa dos mínimos quadráticos para o novo vetor de parâmetros $\mathbf{g}$ pode ser calculada pela expressão:

$$\hat{\mathbf{g}} = (\mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^T\mathbf{s} = \mathbf{B}^{-1}\mathbf{H}^T\mathbf{s}. \tag{7.8}$$

A decomposição ortogonal da equação (7.3) pode ser obtida utilizando-se o algoritmo de Gram-Schimidt, cujo procedimento é o seguinte:

$$\left.\begin{aligned} \mathbf{h}_1 &= \mathbf{r}_1 \\ \alpha_{ij} &= \frac{\mathbf{h}_j^T \mathbf{r}_i}{\mathbf{h}_j^T \mathbf{h}_j} \\ \mathbf{h}_j &= \mathbf{r}_j - \sum_{i=1}^{j-1} \alpha_{ij} \mathbf{h}_i \end{aligned}\right\} \begin{array}{l} i = 1,2,...,(j-1) \\ j = 2,3,...,(M-1) \end{array}. \tag{7.9}$$

Como os regressores $\mathbf{h}_i$ e $\mathbf{h}_j$ são ortogonais para $i \neq j$, a soma dos quadrados de $\mathbf{s}$ é:

$$\mathbf{s}^T\mathbf{s} = \sum_{j=1}^{M+1} g_j^2 \mathbf{h}_j^T \mathbf{h}_j + \varepsilon^T \varepsilon. \tag{7.10}$$

Se $\mathbf{s}$ é o vetor de saídas desejadas depois de ser tirada a sua média, então a estimativa de sua variância é dada por:

$$\text{var}(\mathbf{s}) = \frac{1}{n}\mathbf{s}^T\mathbf{s}. \tag{7.11}$$

O termo da soma da equação (7.10) é a parte da variância da saída desejada que pode ser explicada pelos regressores $\mathbf{h}_j$. Assim, cada regressor possui sua própria contribuição para a soma total e o problema é determinar os q regressores que possuem a maior contribuição. Pode-se definir uma razão de redução do erro pela expressão abaixo:

$$err_j = \frac{g_j^2 \mathbf{h}_j^T \mathbf{h}_j}{\mathbf{s}^T\mathbf{s}}, \qquad j = 1,2,...,(M+1). \tag{7.12}$$

O procedimento prático pode ser descrito, em linhas gerais, como segue:

1. Calcule a razão de redução do erro para cada um dos regressores originais ($\mathbf{h}_j = \mathbf{r}_j$) e selecione aquele com maior valor. O regressor selecionado torna-se $\mathbf{h}_1$ e é retirado do conjunto dos $\mathbf{r}_j$.

2. Utilize os $\mathbf{r}_j$ regressores restantes como candidatos para obter $\mathbf{h}_2$. Faça como no passo 1 até que os q melhores regressores tenham sido selecionados.

7.2.2.2 Inicialização com o algoritmo OLS

Considere a saída da rede como sendo linear, e função de ativação das unidades intermediárias a tangente hiperbólica (tanh). Seja $\mathbf{X}$ a matriz de dados de entrada e $\mathbf{w}_1$ a matriz de pesos da primeira camada da rede (incluindo os limiares). A matriz de regressão é dada por:

$$\mathbf{R} = \tanh(\mathbf{v}^T\mathbf{X}), \quad \text{ou}$$
$$R_j = \tanh\left(\sum_{i=1}^{p} v_{ij}x_i\right) . \tag{7.13}$$

Deve ser observado que durante o processo de inicialização, o número de unidades intermediárias é M, que deve ser significativamente maior do que o número desejado q. É sugerido um valor de $M = 10q$. A equação (7.13) mostra que cada uma das M unidades intermediárias corresponde a um regressor, então devemos utilizar o algoritmo OLS para selecionar os q melhores. Antes de utilizar o algoritmo OLS, os M candidatos a regressores devem ser gerados. Uma forma simples de inicializar os pesos dos candidatos a regressores é atribuindo valores uniformemente distribuídos em um intervalo $[-a, a]$. Se um regressor é escolhido pela rede, então os valores iniciais dos regressores selecionados são os valores iniciais da rede. Cada regressor possui $p + 1$ conexões. O número de entradas determina a dimensão do espaço de parâmetros formado pelos regressores. Quanto menor a dimensão do espaço de parâmetros, menor a quantidade de graus de liberdade existentes para a inicialização dos regressores.

O processo de inicialização da rede pode ser sintetizado como segue:

1. Linearize a saída da rede. Defina M regressores de forma que $M >> q$. Inicialize os regressores, ou seja, os pesos da primeira camada $\mathbf{v}$ com valores uniformemente distribuídos. Selecione os q melhores regressores utilizando o algoritmo OLS e tome os valores iniciais dos regressores como sendo os valores iniciais da rede.
2. Calcule as saídas das q unidades intermediárias para todos os padrões. Faça uma regressão linear para a parte linear da rede e utilize os coeficientes de regressão obtidos como sendo o vetor de pesos $\mathbf{w}$ da camada de saída da rede.

7.2.3 Uma Alternativa Simples e Robusta para Inicialização (INIT)

Nesta seção é proposto um paradigma simples e eficiente que pode ser interpretado como um método intermediário entre os paradigmas do caminho mais fácil e do caminho mais curto. Este paradigma híbrido explora a informação contida nos dados de treinamento, ao mesmo tempo em que tenta considerar os aspectos de processamento de sinal da rede.

É de consenso geral que, uma vez saturadas, as unidades da rede neural não conseguem sair desta condição facilmente, pois o gradiente do erro é tão pequeno que o tamanho da direção de busca para estes pesos é quase zero[1], e os pesos correspondentes não sofrerão mudanças significativas nas próximas iterações. STÄGER & AGARWAL (1997) desenvolveram um algoritmo para detectar unidades intermediárias saturadas e, de alguma forma, reativá-las enquanto transfere suas contribuições para os vetores de limiar da mesma camada.

Na abordagem a ser descrita a seguir, busca-se evitar a saturação das unidades intermediárias no momento da inicialização dos pesos, através de uma definição apropriada do conjunto inicial de pesos. Para que isso seja feito, consideramos tanto o conjunto de treinamento quanto o seu efeito no processamento, camada a camada. O objetivo é garantir que, para os dados de entrada considerados, as unidades intermediárias estejam inicialmente ativas na parte linear da função de ativação.

Esta idéia torna o algoritmo mais flexível para convergir rapidamente para a solução ótima, assim como na abordagem do caminho mais fácil, e o intervalo de definição dos pesos é obtido como sendo função do conjunto de treinamento, assim como na abordagem do caminho mais curto, e com um esforço computacional reduzido.

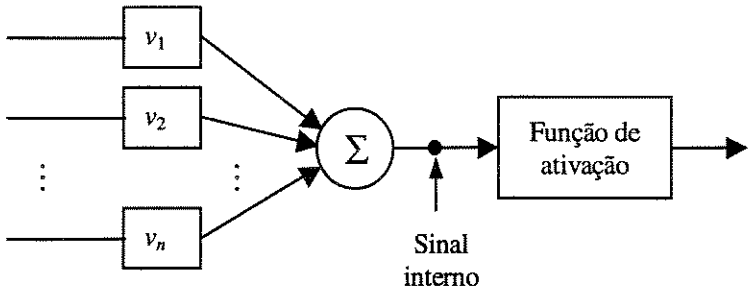


Figura 7.1: Definição do sinal interno de um neurônio.

[1] Dependendo da precisão da máquina utilizada, o gradiente pode ser avaliado em zero.

Para desenvolver nosso método, consideremos que os dados possuem distribuição uniforme ou gaussiana em torno de uma média. Outras distribuições podem ser consideradas, sem perda de generalidade.

Na Figura 7.2(a), apresentamos nosso objetivo, ou seja, posicionar a distribuição dos sinais internos dos neurônios (veja Figura 7.1) em torno da região aproximadamente linear da função de ativação. As Figuras 7.2(b) e (c) apresentam os casos que se deseja evitar. Os pesos iniciais não devem ser definidos de forma a extrapolar a região aproximadamente linear das funções de ativação, como mostrado na Figura 7.2(b), para um subconjunto dos dados de treinamento, e na Figura 7.2(c), para todos os dados de treinamento. É óbvio que se inicializarmos os pesos de forma que a resposta interna de uma ou mais unidades esteja fora da região linear da rede para um ou mais padrões de treinamento, a derivada da função de ativação será aproximadamente zero, tornando o processo de treinamento mais lento e sujeito a problemas numéricos.

Seja **v** o vetor de pesos para a camada intermediária, **X** o conjunto de dados de entrada para treinamento e **z** as saídas (ativação) das unidades intermediárias quando **X** é apresentado à rede.

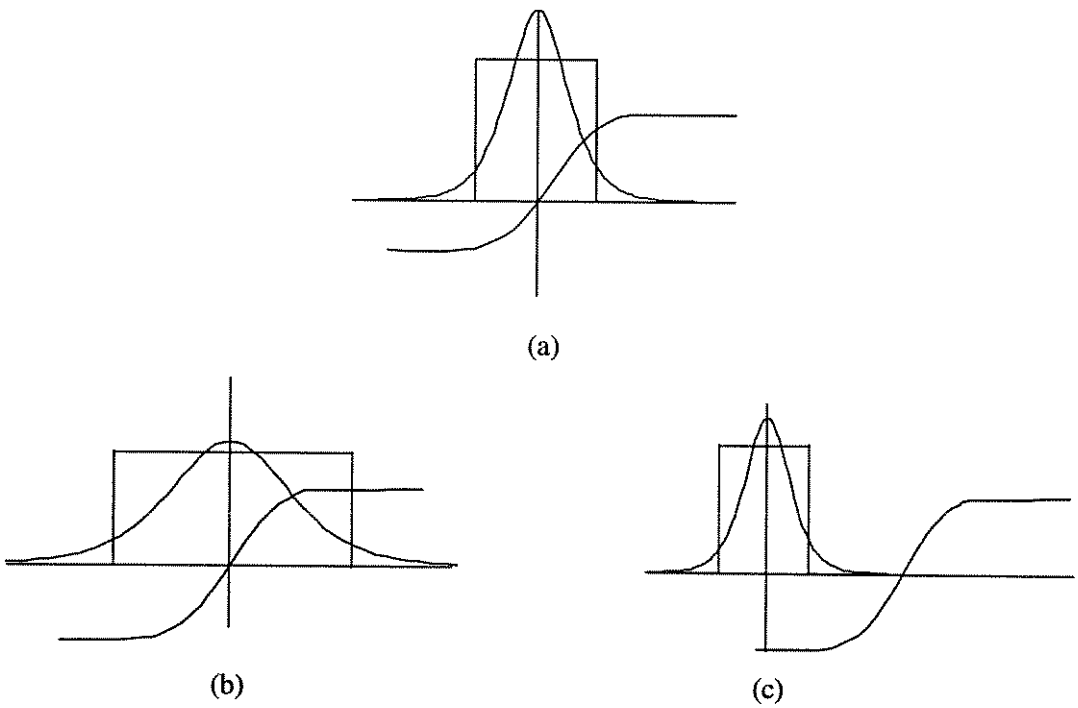


Figura 7.2: Função de ativação tangente hiperbólica, com a distribuição de sinais (uniforme ou gaussiana). (a) Distribuição de sinais em torno da região aproximadamente linear da função. (b) Distribuição de sinais parcialmente fora da região aproximadamente linear da função. (c) Distribuição de sinais completamente fora da região aproximadamente linear da função.

Em notação matricial, a saída das unidades intermediárias **z** pode ser calculada pela expressão:

$$\mathbf{z} = \tanh(\mathbf{v}^T\mathbf{X}). \tag{7.14}$$

Se as unidades intermediárias estão respondendo, supostamente, na região linear da função de ativação, então a equação (7.14) pode ser reescrita pela forma aproximada:

$$\mathbf{z} \cong \mathbf{v}^T\mathbf{X}. \tag{7.15}$$

A seguinte pergunta pode ser feita: "Dada uma distribuição de múltiplas entradas, qual a melhor combinação dessas entradas tal que a combinação de saída **z** seja uma distribuição normal de média zero e variância pequena?"

A resposta para essa questão é a solução do seguinte problema de minimização:

$$\min_{\mathbf{v}\neq\mathbf{0}} \left\| \mathbf{v}^T\mathbf{X} - \mathbf{z} \right\|. \tag{7.16}$$

A restrição imposta a este problema de minimização é devido ao fato de que não estamos interessados na solução trivial, ou seja, $\mathbf{v} = \mathbf{0}$. A solução trivial pode ser evitada tomando-se uma distribuição uniforme com média zero e variância fixa na definição de **z**.
Uma solução para o problema apresentado na equação (7.16) é obtida por meio da pseudo inversão:

$$\mathbf{v} = \left(\mathbf{X}\mathbf{X}^T\right)^{-1}\mathbf{X}.\overline{\mathbf{z}}^T. \tag{7.17}$$

Nesta abordagem é preciso garantir que **X** seja uma matriz de posto completo, o que pode ser facilmente obtido simplesmente manipulando os dados de entrada de forma a eliminar redundâncias.

Como último passo do algoritmo híbrido proposto, é necessário inicializar os pesos das camadas seguintes. Estes pesos são inicializados com o mesmo valor pequeno, ao invés de se utilizar uma distribuição normal ou uniforme, pois as saídas da primeira camada intermediária já estão caracterizadas por uma certa distribuição em torno da faixa linear da resposta dos neurônios, que deve ser preservada até as saídas da rede.

7.3 Determinação Automática da Dimensão da Rede Neural

Até recentemente, a determinação adequada da dimensão de uma rede MLP para uma dada aplicação geralmente envolvia apenas a experiência do projetista na implementação de métodos de tentativa e erro. No entanto, trabalhando com um conjunto de informações maiores sobre o conjunto de treinamento resultante, muitos pesquisadores têm proposto vários métodos para tratar o problema de determinação da dimensão da rede de forma automática

(KWOK & YEUNG, 1993). Dentre estes, a arquitetura chamada *cascade-correlation* é provavelmente a mais conhecida (FAHLMAN, 1988). Outros métodos como algoritmos que trabalham com *busca de projeção* (*projection pursuit learning*) (VON ZUBEN, 1996) e *estratégias evolutivas* (algoritmos genéticos) (BELLEW *et. al.*, 1990) estão sendo também bastante difundidos. Serão revistos alguns métodos utilizados para resolver problemas de regressão. A idéia básica é começar com uma arquitetura de rede de dimensão reduzida, e ir adicionando unidades intermediárias e conexões até que uma solução satisfatória seja apresentada.

Outra forma, também muito estudada, de determinação da dimensão da arquitetura de rede são os chamados métodos de poda (*pruning methods*), cuja idéia é iniciar com uma arquitetura de dimensão elevada e ir retirando unidades ou conexões até que se chegue a uma dimensão satisfatória.

7.3.1 Arquiteturas Pré-Fixadas, Redes Construtivas e Métodos de Poda

Como dito anteriormente, a utilização da arquitetura MLP para a solução de problemas de classificação, aproximação de funções e regressão, pode ser mal-condicionada (ver Capítulo 5) caso não seja feita uma escolha adequada da dimensão do espaço de parâmetros a ser explorado. Além disso, redes com dimensão muito reduzida podem não aprender bem determinados problemas, enquanto redes sobre-dimensionadas geralmente apresentam pouca capacidade de generalização. Portanto, algoritmos capazes de determinar a arquitetura da rede automaticamente são desejáveis.

A utilização de procedimentos de validação cruzada (CV) ou teoria de regularização, vistos no Capítulo 6, diminuem os problemas em relação ao sobre-treinamento da rede e conseqüentemente amenizam os efeitos das redes sobre-dimensionadas. Outro problema é que as redes sobre-dimensionadas exigem um esforço computacional mais intenso e desnecessário.

Os *métodos construtivos* começam com uma arquitetura mínima de rede e adicionam neurônios até que uma solução satisfatória seja encontrada, caminho oposto ao utilizado pelos métodos de poda. A determinação da arquitetura de rede inicial nos métodos construtivos é imediata, enquanto nos métodos de poda não sabemos na prática qual o tamanho inicial da rede, já que não é possível determinar a priori o que seria uma dimensão suficientemente elevada para um dado problema. Além disso, os algoritmos construtivos sempre efetuam a busca a partir de arquiteturas de dimensão reduzida, resultando em um menor esforço

7.3.2.4.2 Definindo o Novo Grafo de Conectividade

Independente de Δ ser mono ou multivariável, deve ser considerado como gerar o próximo estado a partir do estado atual; ou de outra forma, como será o próximo grafo de conectividade. Uma forma simples é fazer com que todas as unidades estejam na mesma camada intermediária, resultando numa rede com uma única camada escondida. A estrutura resultante é regular e o número de conexões ligadas a cada neurônio intermediário (*fan-in*) é constante, facilitando a implementação em *hardware*. Entretanto, se as unidades intermediárias possuem apenas formas simples, então cada uma delas só poderá aproximar funções simples.

Existem algoritmos que permitem a construção de redes com várias camadas intermediárias. As unidades intermediárias que estão mais próximas da saída são capazes de representar funções mais complexas das entradas. Outra forma de gerar funções mais complexas da entrada é permitir uma maior complexibilidade das funções de ativação das unidades intermediárias de uma rede neural com uma única camada.

7.3.2.4.3 Aproximação Universal e Convergência

A família de funções implementadas pela rede resultante deve ser abrangente o bastante para conter a saída desejada $g(.)$ ou uma boa aproximação $\hat{g}(.,\theta)$ desta. Esta característica de *aproximação universal* também é fundamental em arquiteturas fixas e métodos de poda (ver Capítulo 2).

A seqüência de redes produzidas pelo algoritmo deve convergir para a saída desejada. Uma seqüência $\{ \hat{g}_n(.,\theta) \}$ é dita convergir para $g(.)$ se $\lim_{n\to\infty} \left\| g(.)-\hat{g}_n(.,\theta) \right\| = 0$.

7.3.2.5 Generalizando a Busca

Cada estado no espaço de estados corresponde a apenas um conjunto de parâmetros de *uma* rede neural e o algoritmo construtivo mantém apenas *uma* rede neural a cada instante. Alguns algoritmos mantém um conjunto de redes e permite uma exploração mais ampla do espaço de estados simultaneamente. Esta estrutura pode ser generalizada utilizando-se um espaço de estados generalizado *GS*, onde cada estado em *GS* é um conjunto contendo vários estados como os descritos anteriormente. Exemplos de algoritmos que mantém um conjunto de redes são os *algoritmos genéticos*.

7.3.3 Características Gerais dos Algoritmos de Treinamento

Dados C e Γ a determinação de W é direta, basta apenas ajustar os parâmetros da rede neural (treinar a rede). Nas redes com arquitetura fixa, os pesos são conduzidos a seu valor ótimo apenas uma vez, enquanto nos métodos construtivos e de poda, o treinamento é repetido a cada vez que é introduzido um novo neurônio. Assim, a eficiência computacional em termos de tempo e espaço é um aspecto importante. Existem geralmente duas formas de se reduzir a complexidade computacional:

- treinando apenas as novas unidades introduzidas, ou
- combinando o processo de treinamento com memorização.

7.3.3.1 Treinando Toda a Rede

Uma abordagem simples é treinar toda a rede após a inclusão de cada nova unidade na camada intermediária. O esforço computacional envolvido no processo de treinamento depende do algoritmo de otimização não-linear utilizado. Uma discussão aprofundada sobre métodos de otimização não-linear irrestrita para o treinamento de redes MLP é apresentada no Capítulo 3.

A utilização de métodos de treinamento mais eficientes como o método de Newton é sempre recomendável caso o treinamento possa ser reduzido a problemas de pequena dimensão.

7.3.3.2 Treinar Apenas as Novas Unidades Intermediárias

Uma forma de simplificar o problema de otimização é assumir que as unidades intermediárias já introduzidas são capazes de modelar parte da função objetivo. Assim, é possível manter fixos os pesos das conexões que alimentam estas unidades, e permitir a variação apenas dos pesos que ligam as novas unidades às entradas e saídas. O número de pesos a serem otimizados e o tempo e memória consumidos por iteração serão reduzidos de maneira significativa. Esta redução é ainda maior quando a quantidade de neurônios na camada intermediária é grande.

Cada unidade intermediária é treinada para reduzir ao máximo os erros residuais e em seguida é instalada permanentemente na rede.

7.3.3.3 Retro-ajuste (*Back-fitting*)

O ajuste fino das conexões das unidades já introduzidas é feito por um procedimento chamado de *back-fitting* (FRIEDMAN & STUETZLE, 1981; VON ZUBEN & NETTO, 1997). Este processo faz um ajuste cíclico das conexões ligadas às unidades já instaladas, enquanto mantém os parâmetros (pesos e outros parâmetros que definem as funções de ativação das unidades intermediárias) das outras unidades fixos, até que não haja variação significativa no desempenho.

Lembre-se que unidades intermediárias que utilizam funções de ativação simples são capazes de modelar uma variedade limitada de funções considerando-se uma rede com uma única camada intermediária.

7.3.3.4 Reduzindo o Esforço Computacional

Outra forma de reduzir o esforço computacional do procedimento de otimização é transformando-o em um problema de otimização linear, utilizando neurônios intermediários com ativação polinomial. Outra abordagem é decompor o conjunto de pesos a serem treinados em subconjuntos disjuntos, de forma que o problema complexo de otimização original seja decomposto em subproblemas com dimensão inferior. Nos algoritmos construtivos, isso geralmente é feito treinando-se camada por camada: um grupo de pesos é otimizado de cada vez, enquanto os outros são mantidos fixos.

7.3.4 Algoritmos Construtivos

Nesta seção serão apresentados alguns algoritmos construtivos, como *projection pursuit learning* (PPL), *cascade-correlation* (CC), *A*-algorithm* (A*), e uma estrutura baseada em algoritmos genéticos (GA).

7.3.4.1 Projection Pursuit Learning (PPL)

Algoritmos nesta classe são inspirados por técnicas estatísticas baseadas nos modelos de regressão por busca de projeção (*projection pursuit regression* – PPR) propostas por FRIEDMAN & STUETZLE (1981). Geralmente a função de ativação é a soma de n funções não lineares (sem perda de generalidade, assuma uma única saída):

$$\hat{g}(\mathbf{x}) = \sum_{j=1}^{n} f_j(\mathbf{a}_j^T \mathbf{x}), \tag{7.20}$$

onde $\mathbf{a}_j$ é o vetor de projeção, $\mathbf{x}$ é o vetor de entrada e f_j é a função de ativação do j-ésimo neurônio, chamado de *smoother* na literatura referente a estatística. Existe uma similaridade evidente entre o PPR e as redes MLP com uma única camada intermediária. Considerando uma arquitetura MLP com uma única camada intermediária, a expressão para a função aproximada pela rede é dada por:

$$\hat{g}_p = w_{0p} + \sum_{j=1}^{l}\left[w_{jp} f_j\left(\sum_{i=1}^{k} v_{ij}^T x_i + v_{0j} \right)\right], \tag{7.21}$$

onde l e k são o número de unidades intermediárias e de entradas, respectivamente; v_{ij} ($i\neq 0$) é o peso conectando a i-ésima entrada à j-ésima unidade intermediária, w_{jp} ($j\neq 0$) é o peso conectando a j-ésima unidade intermediária à p-ésima saída e w_0 e v_0 são os limiares.

O mapeamento de transição de estados é monovariável. O número de unidades intermediárias é aumentado de um a cada transição de estados, e a nova unidade sempre é adicionada à mesma camada. Ao invés de utilizar função de ativação sigmoidal e efetuar o treinamento completo, estes algoritmos utilizam unidades intermediárias com função de ativação mais complexas e treinam apenas as novas unidades. É feita a utilização do procedimento de retro-ajuste (*back-fitting*) mencionado anteriormente.

7.3.4.1.1 Funções de Ativação das Unidade Intermediárias

Uma característica deste algoritmo é a utilização de funções de ativação mais complexas. No PPL a função de ativação é não-paramétrica, como, por exemplo as *splines*, ou os polinômios de Hermite. Para as redes que utilizam os polinômios de Hermite, cada unidade intermediária é representada como sendo uma combinação linear dos polinômios de Hermite:

$$f(z) = \sum_{r=1}^{R} c_r h_r(z), \tag{7.22}$$

onde $h_r(z)$ é o polinômio ortonormal de Hermite e R é um parâmetro definido pelo usuário, chamado de *ordem*. Estas funções (polinômios de Hermite) permitem a interpolação suave e o cálculo rápido e preciso das derivadas. O parâmetro R controla a flexibilidade das unidades intermediárias.

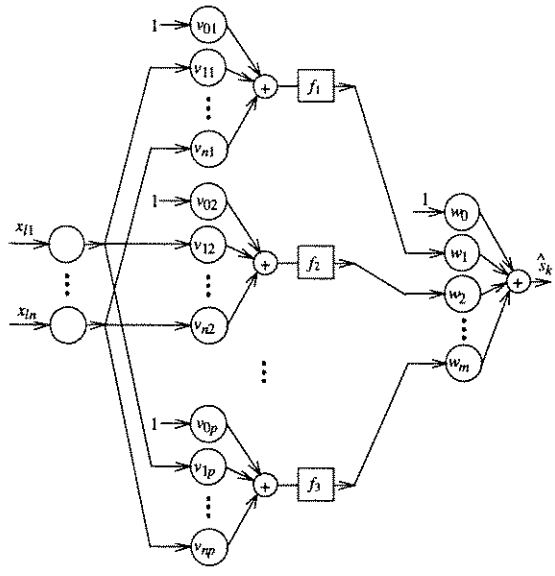


Figura 7.5: Arquitetura típica de um modelo PPL.

7.3.4.1.2 Treinamento

Outra característica desta arquitetura é que ao invés de retreinar toda a rede após inserir uma nova unidade intermediária, apenas a nova unidade é treinada. Assim, para o PPL, após adicionar a n-ésima unidade intermediária, os únicos parâmetros a serem treinados são a função de ativação f_n e o vetor de projeção $\mathbf{a}_n$ (em (7.20)).

HWANG *et. al.* (1994) faz com que os parâmetros que devem ser treinados após a adição do n-ésimo neurônio sejam divididos em três grupos:

1. conexões da entrada para a saída;
2. parâmetros associados à função de transferência em (7.22), ou seja, c_r e parâmetros associados aos polinômios de Hermite $h_r(z)$ e
3. pesos da camada intermediária para a saída.

Os pesos são otimizados em relação ao critério de erro, mantendo os grupos 2 e 3 fixos. Em seguida os parâmetros associados a função de transferência são atualizados, enquanto os grupos 1 e 3 permanecem inalterados. Por último, atualiza-se os pesos das conexões de saída. Assim, o problema original de otimização não-linear é reduzido para vários problemas de menor dimensão. Para o caso de saídas lineares, somente a otimização das direções de projeção permanecem um problema não-linear.

7.3.4.1.3 Propriedade de Convergência

Teoricamente, considerando-se que o PPL utiliza polinômios de ordem finita, as propriedades de aproximação universal e convergência para a função alvo não se garantem. É possível fazer uma modificação na equação (7.20) através da inclusão de um limiar (*bias*) em cada projeção linear das variáveis de predição

$$\hat{g}(\mathbf{x}) = \sum_{j=1}^{n} f_j(\mathbf{a}_j^T \mathbf{x} + \lambda_j), \tag{7.23}$$

de forma que o PPL recupere a capacidade de aproximação universal e convergência (KWOK & YEUNG, 1996). Experimentalmente, essa modificação também aumenta a taxa de convergência com relação ao número de neurônios intermediários e fornece um melhor desempenho de generalização (KWOK & YEUNG, 1996).

No Capítulo 8 será apresentado um método, utilizando os mapas auto-organizáveis de Kohonen, de definição de um conjunto inicial de vetores visando uma maior exploração do espaço de busca inicial dos modelos baseados em busca de projeção (arquiteturas PPL).

7.3.4.2 Cascade-Correlation

Ao contrário do algoritmo anterior, a arquitetura *cascade-correlation* proposta por FAHLMAN & LEBIERE (1990), constrói redes com múltiplas camadas intermediárias. Este arranjo estrutural permite o desenvolvimento de detetores de características poderosos mesmo com unidades intermediárias simples.

7.3.4.2.1 Esquema de Conectividade para as Novas Unidades Intermediárias

Quando uma nova unidade intermediária está para ser criada, além de estabelecer a conexão com cada uma das entradas e saídas originais da rede, também é feita uma conexão entre a nova unidade e as unidades intermediárias preexistentes. Cada nova unidade adiciona uma nova camada com um único neurônio à rede, gerando uma arquitetura em cascata (Figura 7.6). Geralmente é utilizado com unidades intermediárias simples, mas esta estrutura em cascata pode ser combinada com neurônios intermediários mais complexos. LITTMANN & RITTER (1992) apresentam exemplos onde cada unidade intermediária é um mapeamento linear local que aprende uma aproximação linear da função alvo.

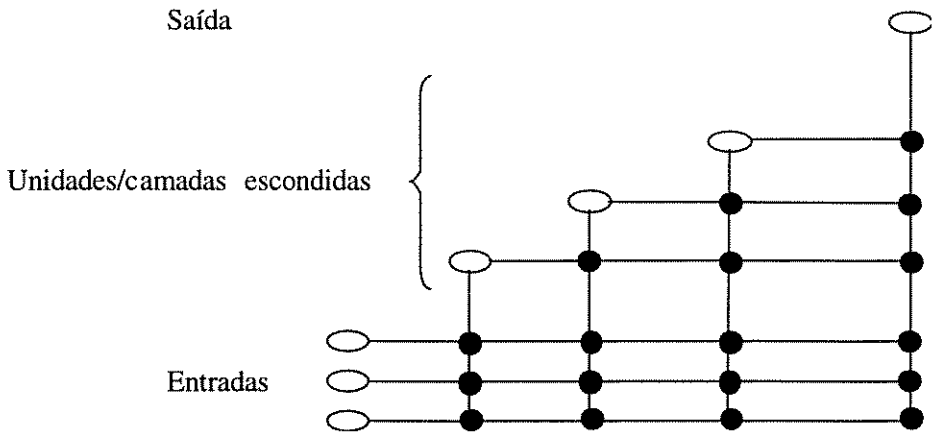


Figura 7.6: Arquitetura *cascade-correlation*. No diagrama, elipses vazias representam unidades de entrada, intermediárias e de saída, enquanto os pontos pretos referem-se a conexões entre unidades.

Embora a estrutura de rede resultante seja um detetor de características poderoso, o número de conexões para a *n*-ésima unidade aumenta muito e leva a dois problemas:

1. o desempenho em termos de generalização é reduzido quando *n* é grande e
2. haverá grandes atrasos de propagação e um aumento da quantidade de conexões ligadas a cada unidade intermediária, dificultando a implementação em hardware.

O primeiro problema pode ser aliviado utilizando procedimentos de validação cruzada e teoria de regularização, como discutido no Capítulo 6.

7.3.4.2.2 Treinamento

Somente as conexões ligadas às novas unidades intermediárias sofrem modificação, e são atualizadas camada a camada. Primeiro os pesos ligados à nova unidade são treinados (treinamento da entrada) através da otimização da função objetivo, enquanto todas as outras conexões são mantidas fixas. Geralmente existe um conjunto de candidatos a unidades intermediárias, cada um utilizando uma condição inicial aleatória diferente. Em seguida, as conexões ligadas a esta unidade são fixadas e os pesos que conectam as unidades intermediárias às saídas são atualizados (treinamento da saída). Se a saída da rede é linear, o treinamento da saída torna-se um problema linear e o conjunto de pesos da saída é convenientemente determinado utilizando-se o cálculo exato da pseudo-inversa.

As novas unidades intermediárias maximizam a covariância entre o erro residual e a ativação das unidades intermediárias:

$$S_{cascor} = \sum_{o} \left| \sum_{p} (J_{po} - \bar{J}_o)(H_p - \bar{H}) \right|, \qquad (7.24)$$

onde p faz a varredura por todos os padrões de treinamento, o trata de todas as unidades de saída, H_p é a ativação da nova unidade intermediária para o padrão p, J_{po} é o erro residual da saída o para o padrão p antes da adição da nova unidade intermediária e $\bar{H}$ e $\bar{J}_o$ são os valores médios correspondentes para todos os padrões.

7.3.4.2.3 Funções de ativação

O algoritmo original proposto por FAHLMAN & LEBIERE (1990) foi modificado para permitir a utilização de mais de uma função de ativação para os candidatos (ENSLEY & NELSON, 1992). Isso significa que unidades com funções de ativação diferentes podem competir simultaneamente. O algoritmo CASCOR seleciona automaticamente o melhor candidato, permitindo definir qual função de ativação, a partir de um conjunto fechado, é mais adequada ao problema.

As Tabelas 7.1, 7.2 e 7.3 apresentam possíveis conjuntos de funções de ativação que podem ser utilizados na implementação do CASCOR.

Tabela 7.1: Funções de ativação senoidais.

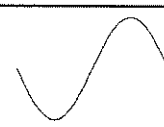
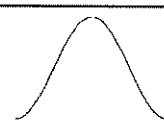
Função de ativação	Expressão	Gráfico	Intervalo
Sen	$f(x) = \text{sen}(x)$		[-1.0, 1.0]
Cos	$f(x) = \cos(x)$		[-1.0, 1.0]

Tabela 7.2: Funções de ativação gaussianas.

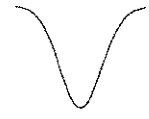
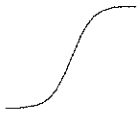
Função de ativação	Expressão	Gráfico	Intervalo
Gaussiana	$f(x) = e^{-(x^2/2)}$		(0.0, 1.0]
Inverso da Gaussiana	$f(x) = -e^{-(x^2/2)}$		[-1.0, 0.0)

Tabela 7.3: Função de ativação hiperbólica.

Função de ativação	Expressão	Gráfico	Intervalo
Tangente Hiperbólica	$f(x) = \tanh(x)$		(-1.0, 1.0)

7.3.4.2.4 Propriedades de Convergência

A capacidade de generalização universal desta arquitetura é evidente, pois estas redes podem ser reduzidas a redes com uma única camada intermediária; basta eliminar as conexões em cascata. Existem demonstrações formais para a propriedade de convergência desta arquitetura utilizando neurônios na camada intermediária com ativação dada pela tangente hiperbólica (DOERING *et. al.*, 1997).

7.3.4.3 Arquitetura A* (A*-*Algorithm*)

A construção de um grafo de estruturas de rede e a determinação de valores de avaliação para estes grafos são os problemas abordados pelo algoritmo A*. A aplicação deste algoritmo garante, teoricamente, a otimalidade da solução e da busca.

7.3.4.3.1 Algoritmo A*

O algoritmo A* trabalha no grafo cujos arcos são denominados custos. Um subconjunto do conjunto de nós é o conjunto alvo, cujos elementos satisfazem algum critério de sucesso. Os objetivos do algoritmo A* são:

1. se existe uma solução, encontrar um elemento do conjunto alvo que seja uma solução admissível; e
2. encontrar a solução com o menor esforço de busca possível em relação ao número de unidades introduzidas (otimalidade).

A idéia básica do algoritmo A* é que dado um nó n_i do grafo, o custo do caminho ótimo partindo deste nó, ao nó alvo n_G^* (em termos de custos) seja descrito por uma função $h^*(n_i)$. Nem o caminho ótimo, nem os custos associados a ele são conhecidos. Por outro lado, se o custo associado ao caminho ótimo puder ser estimado por uma função, como $h(n_i)$, que avalia alguma heurística conhecida sobre o problema de busca, então a função de avaliação pode ser dada por:

$$f(n_i) = g(n_i) + h(n_i) \tag{7.25}$$

$g(n_i)$: custos associados ao melhor caminho conhecido, partindo do nó n_0 ao nó n_i.

$h(n_i)$: estimativa dos custos do melhor caminho do nó n_i ao elemento mais próximo do conjunto alvo n_G^* (função heurística)

Esta função quantifica quão promissora é a expansão do nó n_i. O algoritmo A* simplesmente busca a estratégia de sempre expandir o nó mais promissor. Duas condições são necessárias para o sucesso do algoritmo A*:

1. $\forall n$: $h(n) \leq h^*(n)$ (condição admissível)
2. $\forall n$: $\forall n' \in \{\Gamma(n)\}$: $h(n) \leq Cost(n, n') + h(n')$ (condição de monotonicidade)

onde $Cost(n, n')$ representa o custo do caminho ótimo entre os nós n e n' e $\Gamma(n)$ representa um operador de expansão que será definido posteriormente. A condição (1) garante que o algoritmo A* encontrará o caminho do nó alvo com custo mínimo; e se a condição (2) for satisfeita, o algoritmo A* será superior a qualquer algoritmo admissível que utiliza informações heurísticas (DOERING *et. al.*, 1997).

7.3.4.3.2 Aplicação do Algoritmo A* para a Solução do Problema de Otimização da Estrutura

Considere o grafo de rede dado na seção 7.3.2.1 por C (V, E), onde V corresponde aos nós (unidades) e E corresponde às conexões (pesos). Partindo desse grafo, é possível construir um conjunto $A = \{C\}$ finito de estruturas de rede. Para aplicar um algoritmo de busca a este conjunto, é preciso definir relações entre seus membros.

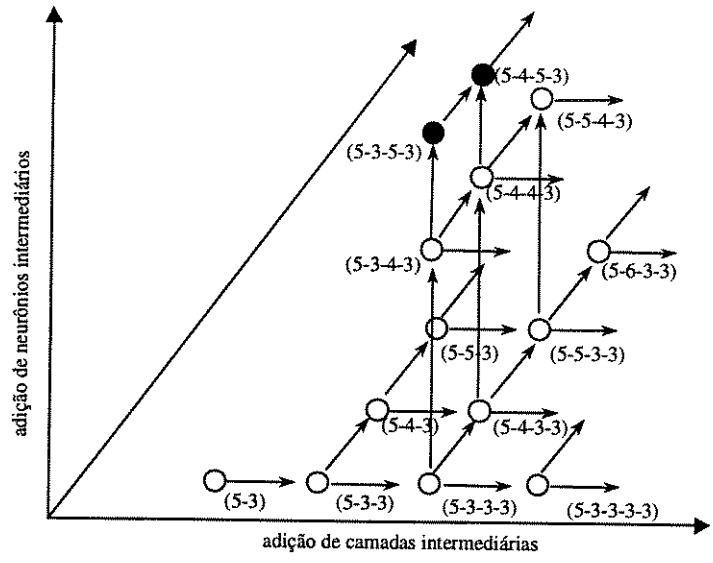


Figura 7.7: Aplicação sucessiva do operador de expansão $\Gamma(C)$ à uma estrutura inicial com cinco entradas, três saídas e nenhuma unidade intermediária.

A aplicação de um algoritmo de busca é equivalente a definir um operador de expansão $\Gamma(C)$: $\boldsymbol{A} \rightarrow 2^{\boldsymbol{A}}$ que mapeia qualquer estrutura $C \in \boldsymbol{A}$ em um conjunto de sucessores (que é subconjunto de $\boldsymbol{A}$).

Cada elemento $C \in \boldsymbol{A}$ pode ser visto como um nó do grafo cujos arcos são determinados por $\Gamma(C)$ (ver Figura 7.7). Um critério de avaliação que determina o conjunto alvo $\boldsymbol{G}$ está definido:

$$\boldsymbol{G} = \{\ C \in \boldsymbol{A} \mid \eta\ (C) \leq \eta_0\ \} \tag{7.26}$$

Para aplicar o algoritmo A* a um valor não negativo propriamente escolhido de η_0 ainda é necessário definir uma função custo e uma função heurística.

Função custo: a cada operador de expansão $C' \in \Gamma(C)$ é especificado um vetor de função de custo.

Função heurística: a cada estrutura de rede C pode ser especificado um vetor de parâmetros θ^* que minimiza a esperança de uma função de custo, $\theta^* = \text{argmin}_\theta\ \boldsymbol{E}(g_T(f_{NS}(\mathbf{X},\theta),\boldsymbol{Y}))$.

$g_T(f_{NS}(\mathbf{X},\theta),\boldsymbol{Y})$ denota o erro de generalização da rede.

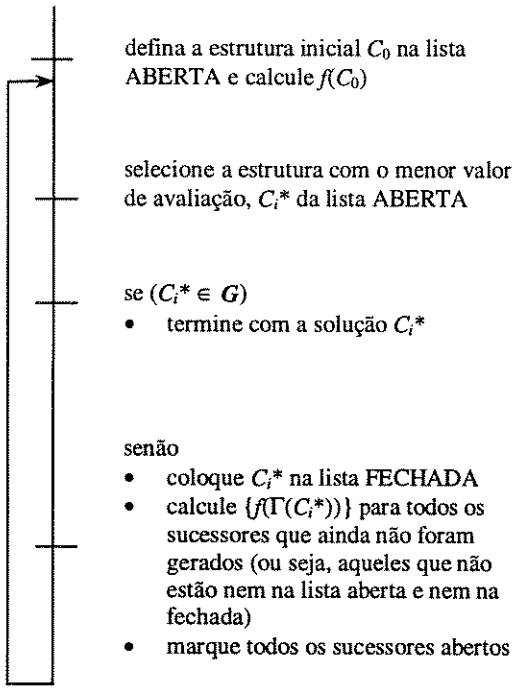


Figura 7.8: Algoritmo A* para otimização da estrutura. O cálculo da avaliação da função $f(C_i)$ inclui a fase de treinamento e avaliação.

A aplicação do algoritmo A* para a construção da estrutura da rede pode ser resumida pelo diagrama da Figura 7.8.

7.3.4.4 Algoritmo Evolutivo para a Definição da Estrutura

Existem, atualmente, vários algoritmos evolutivos utilizados para a determinação da estrutura de uma rede do tipo MLP. Estes algoritmos podem ser capazes, ou não, de definir arquiteturas multicamadas, otimizar o vetor de parâmetros θ e otimizar o número de unidades em cada camada. Nosso objetivo nesta seção é apresentar apenas uma estratégia evolutiva simplificada para a determinação da quantidade ótima de unidades a serem adicionadas à uma única camada intermediária e a otimização do vetor de parâmetros θ. Dentre algumas referências que tratam com algoritmos mais poderosos, podemos citar os trabalhos de SHIFFMANN *et. al.* (1992) e de OPITZ & SHAVLIK (1997).

7.3.4.4.1 Estratégia

O método apresentado determina a estrutura apropriada, ou seja, o número apropriado de unidades intermediárias, de forma que soluções ótimas locais sejam evitadas (SARKAR & YEGNANARAYANA, 1997). Neste método a rede é evoluída de forma a determinar um conjunto ótimo de parâmetros (θ), ou seja, pesos e limiares. A eficiência deste método é aumentada através da incorporação de conceitos de mutação adaptativa da estrutura.

Para a obtenção de uma arquitetura ótima de rede, são necessários métodos avançados de busca e otimização. O método de busca visa determinar: (a) o número ótimo de unidades intermediárias e (b) o valor ótimo para o conjunto de parâmetros. A função objetivo é então formulada de modo que sua minimização gere uma configuração ótima de rede.

A programação evolutiva (PE) minimiza uma função objetivo, como o erro quadrático médio (MSE) para um conjunto de treinamento. A minimização do erro quadrático médio permite obter uma arquitetura ótima de rede e um conjunto de parâmetros ótimo. A PE otimiza a função objetivo empregando uma busca estocástica controlada, e faz esta busca de forma paralela partindo de mais de um ponto. Em outras palavras, ao mesmo tempo que busca um mínimo local, esta técnica explora vários caminhos simultaneamente.

Embora a PE seja baseada em uma busca estocástica, ela não é totalmente aleatória – é uma busca aleatória controlada. Esta ação de controle é feita através de alguns parâmetros de mutação. A escolha dos parâmetros de mutação possuem um forte impacto no desempenho de convergência do método apresentado.

7.3.4.4.2 Programação Evolutiva (PE) da Estrutura de Rede

Quando estamos projetando uma rede MLP para um determinado problema, objetivamos encontrar um número ótimo de neurônios da camada intermediária e um conjunto ótimo de parâmetros. Formalmente, isso pode ser escrito como um problema de determinação de um mínimo global para o funcional $g(.)$: $X \subset \Re^m \rightarrow \Re^r$. A idéia geral pode ser formulada como segue:

1. gere uma população aleatória de redes (chame-a de *pais*);
2. encontre o valor de aptidão (*fitness*) para cada pai (*fitness* – medida da qualidade da aproximação da rede para um dado problema);
3. gere uma rede descendente (*offspring*) para cada um dos pais;
4. determine o valor de aptidão para cada rede descendente;
5. a competição começa entre todos os pais e descendentes baseada no valor da aptidão;
6. as redes mais aptas sobrevivem (chame-as de pais)
7. se o número de gerações é menor que um limiar pré-especificado volte ao passo 3.

A rede que apresenta os melhores resultados em um conjunto de teste é considerada a melhor. O conjunto de teste contém vetores de entrada que não existem no conjunto de treinamento.

7.3.4.4.3 Características de Implementação

Função de aptidão: o valor da aptidão de uma rede define quão boa é a rede durante a fase de competição. Redes com maiores valores de aptidão possuem maiores chances de sobreviverem. O inverso do erro quadrático médio para um conjunto de treinamento é usado como função de aptidão. $F = 1/J$, onde J é dado pela expressão (5.4).

Geração dos descendentes: para gerar os descendentes é necessário fazer a *duplicação dos pais* e a *mutação*. Nesta abordagem, a rede é representada pelo número de neurônios intermediários, e pelo vetor de parâmetros. Estes valores então são copiados dos pais para os filhos gerados.

Mutação: O operador de mutação é utilizado para evitar mínimos locais durante o processo de minimização da função objetivo. Em particular, durante a busca por um conjunto ótimo de parâmetros são encontrados mínimos locais (chamados de mínimos locais paramétricos), e durante a busca por um número ótimo de neurônios intermediários são

encontrados mínimos locais (chamados mínimos locais estruturais). Mínimos locais paramétricos e estruturais são aliviados por mutação paramétrica e estrutural, respectivamente.

Mutação paramétrica: neste procedimento, cada peso θ, é perturbado com um ruído gaussiano. Assim, $\theta = \theta + \mathcal{N}(0,T)$. Claramente, o grau de mutação $\mathcal{N}(0,T)$ é um número aleatório gaussiano com média 0 e variância fixa T. A intensidade da mutação paramétrica deve ser alta quando a aptidão dos pais é baixa e vice-versa. Se T, para uma rede particular, for considerada como sendo sua temperatura, definida na forma:

$$T = \alpha U(0,1)\left[\frac{aptidão\ \ mínima}{aptidão\ \ da\ \ rede}\right], \tag{7.27}$$

onde $U(0,1)$ é uma variável aleatória uniformemente distribuída no intervalo (0, 1) e $\alpha \leq 1$ constante. Grandes mutações são necessárias para escapar de mínimos locais paramétricos durante a busca; mas muitas vezes isso pode afetar a capacidade dos descendentes apresentarem melhor desempenho do que os pais.

Mutação estrutural: durante este processo, uma unidade intermediária pode ser adicionada ou removida na geração dos descendentes. Os momentos específicos de adição ou retirada de unidades intermediárias dependem da probabilidade de mutação estrutural, que por sua vez depende da aptidão da rede. A unidade intermediária que deve ser introduzida ou retirada, é selecionada de forma aleatória (uniformemente). A quantidade de mutação estrutural depende do valor de probabilidade da mutação estrutural (p_m). O valor de p_m é aumentado quando a população tende a ficar presa em um mínimo local e deve ser reduzido quando a diversidade da população é grande. Sejam $\overline{F}$ e $F_{\max}$ os valores médio da aptidão e a aptidão máxima da população e k_{m1} e k_{m2} duas constantes. Para medir a diversidade da população, o conceito de entropia do valor de aptidão pode ser utilizado. Para medir a entropia o intervalo (F_{min}, F_{max}) é dividido em L subintervalos ($F_{min} + jD$, $F_{min} + (j+1)D$), onde $j = 0, 1, \ldots, L\text{-}1$ e $D = (F_{max} - F_{min})/L$. Aqui $\tilde{p}_j$ é definido como $\tilde{n}_j / 2P$ onde $\tilde{n}_j$ é o número de membros da população no j-ésimo intervalo e $2P$ é o tamanho da população (P pais + P filhos). A entropia é então definida como:

$$\varepsilon = \frac{1}{\ln(2P)} \sum_{i=0}^{L-1} \tilde{p}_i \ln \tilde{p}_i \,. \tag{7.28}$$

Se os membros estão uniformemente distribuídos, então o maior valor de ε é 1. O valor de p_m é definido por:

$$p_m = \begin{cases} k_{m1}\dfrac{F_{\max} - F}{F_{\max} - \overline{F}}(1-\varepsilon^s) + k_{m2} & \text{se} \quad F \geq \overline{F} \\ = k_m(1-\varepsilon^s) & \text{se} \quad F < \overline{F} \end{cases} \tag{7.29}$$

7.3.5 Métodos de Poda

Como não conhecemos, de antemão, a melhor arquitetura de rede, podemos treinar várias redes de diversos tamanhos e escolher a menor delas capaz de aprender o conjunto de dados. Esta abordagem, embora direta, é bastante ineficiente pois muitas redes devem ser treinadas até que seja determinada a melhor. Mesmo se a dimensão ótima da rede for conhecida, as menores redes com complexidade suficiente apenas para aprender os dados, serão mais sensíveis às condições iniciais e aos parâmetros de treinamento (REED, 1993). É difícil dizer se uma rede é muito pequena para aprender um problema, se ela está aprendendo muito lentamente, ou se ficou presa em algum mínimo local devido à condições iniciais impróprias.

A abordagem adotada pelos algoritmos descritos nesta seção é treinar uma rede que seja maior do que o necessário e remover as partes que não são necessárias. A grande dimensão inicial permite que a rede aprenda rapidamente com menor sensibilidade às condições iniciais, enquanto a complexidade reduzida do sistema que sofreu algum tipo de poda favorece o aumento da capacidade de generalização.

Nenhum método de poda existente garante a determinação da melhor estrutura de rede para solucionar determinado problema (THIMM & FIESLER, 1995). Objetivamos nesta seção apresentar alguns poucos métodos de poda dentre os vários existentes.

Em contradição aos métodos construtivos, nos métodos de poda tem-se

1. $V_2 \subseteq V_1$;
2. $E_2 \subset E_1$.

7.3.5.1 Algoritmos de Poda

Um método de poda para cada peso, utilizando força bruta, é: leve cada peso do conjunto de pesos para zero e avalie o comportamento do erro; se ele aumentar muito então restaure a conexão, senão remova-a. Uma abordagem ainda mais conservativa seria avaliar a variação do erro para todos os pesos e padrões e eliminar o peso que influencia menos o

comportamento do erro. Como estes tipos de análises podem ser muito lentos para redes de grande dimensão, os métodos apresentados aqui empregam abordagens menos diretas.

A maioria dos algoritmos podem ser colocados em dois grandes grupos. Um grupo estima a sensibilidade da função de erro em relação a remoção de uma unidade; os elementos com os menores efeitos podem ser removidos. O outro grupo adiciona um termo a função objetivo que faz com que a rede escolha a solução mais eficiente. Um termo proporcional a soma da dimensão de todos os pesos, por exemplo, favorece as soluções com os menores pesos; aqueles que estão próximos de zero não exercerão muita influência sobre a saída e poderão ser eliminados.

Em geral, os métodos de sensibilidade modificam a rede treinada, ou seja, a rede é treinada, as sensibilidades são estimadas, e depois os pesos ou nós são removidos. Os métodos que utilizam um termo de penalização, por outro lado, modificam a função objetivo de forma que o algoritmo de treinamento (ver Capítulo 3) zere os pesos desnecessários durante a otimização da função objetivo. Mesmo os pesos não sendo removidos, a rede funciona como um sistema reduzido. Os dois métodos de poda apresentados são detalhados em (REED, 1993).

7.3.5.1.1 Um Método de Cálculo da Sensibilidade

Este método mede a sensibilidade da função de erro em relação a remoção de cada conexão e remove os pesos com menor sensibilidade. A sensibilidade do peso w_{ij} é dada por:

$$S_{ij} = -\frac{J(w^f) - J(0)}{w^f - 0} w^f , \tag{7.30}$$

onde w^f é o valor final do peso após o treinamento, 0 é o valor após a remoção, e $J(0)$ é o erro quando o peso é removido.

Ao invés de remover o peso e calcular o erro $J(0)$ diretamente, a sensibilidade S é aproximada pela monitoração da soma de todas as mudanças dos pesos durante o treinamento. A sensibilidade estimada é:

$$\hat{S}_{ij} = -\sum_{n=0}^{E-1} \frac{\partial J}{\partial w_{ij}} \Delta w_{ij}(n) \frac{w_{ij}^f}{w_{ij}^f - w_{ij}^i} , \tag{7.31}$$

onde E é o número de épocas de treinamento e i é o valor inicial do peso. Todos os termos da expressão (7.31) estão disponíveis durante o treinamento, tornando esta equação fácil de avaliar.

Ao final do treinamento cada peso possui uma sensibilidade estimada e os pesos com menores sensibilidades são removidos. Se todas as conexões de saída de um nó forem removidas, então o nó pode ser removido.

7.3.5.1.2 Um Método de Termo de Penalidade

Estes métodos modificam a função objetivo de forma que o algoritmo de treinamento efetue a poda da rede fazendo com que alguns pesos sejam levados para zero durante o processo de treinamento. Estes pesos podem ser removidos quando decaem abaixo de um limiar pré-definido.

Uma forma de modificar a função objetivo para incluir o termo de penalidade pode ser vista abaixo:

$$J(\theta) = \sum_{l=1}^{N} \left(g_l(\mathbf{x}) - \hat{g}_l(\mathbf{x},\theta)\right) + \lambda \sum_{i=1}^{C} \frac{w_i^2 / w_o^2}{1 + w_i^2 / w_o^2}, \tag{7.32}$$

onde N representa o conjunto de todas as amostras e C o grafo de conexões. O segundo termo representa uma medida da magnitude dos pesos em relação a constante w_o. Para $|w_i| >> w_o$ o custo de uma conexão aproxima-se de λ. Para $|w_i| << w_o$ o custo aproxima-se de zero.

Quando λ é grande, este método assemelha-se ao método de decaimento dos pesos apresentado no Capítulo 6. O valor de λ depende do problema e requer ajustes. Se for muito pequeno, não terá efeito significativo; se for muito grande, todos os pesos tenderão para zero.

7.4 Implementação em Paralelo

A velocidade de aprendizagem das redes com múltiplas camadas sempre demonstrou ser um aspecto limitante na aplicação desta arquitetura. Quanto maior a popularidade das RNA's em domínios diferentes, mais crítica torna-se a necessidade de obtenção de resultados em tempo real. Embora exista uma grande variedade de implementações de RNA's em máquinas seqüenciais, a quantidade de tempo de processamento destes algoritmos é muito elevada, especialmente quando as dimensões são muito elevadads (MISRA, 1997).

7.4.1 Introdução

A principal fonte de inspiração para as RNA's são os modelos biológicos que são massivamente paralelos, constituídos de várias unidades básicas que cooperam entre si na solução de determinados problemas. A implementação de modelos massivamente paralelos em máquinas seqüenciais é altamente ineficiente quando o processador tem que simular um

modelo unidade por unidade. Isso resulta em elevado tempo de processamento. É natural pensarmos em uma implementação paralela destes modelos neurais. Entretanto, a implementação paralela destes modelos não é trivial e requer técnicas sofisticadas de projeto dos algoritmos. Os algoritmos devem ser convertidos para um ambiente que seja de fácil manipulação para os usuários de RNA's. Neste tópico discutiremos alguns esforços que têm sido feitos no intuito de atingir tais implementações paralelas, incluindo tecnologias de suporte necessárias para estas implementações. Nosso objetivo final é verificar se existe um ambiente paralelo capaz de mapear os vários modelos neurais existentes em modelos para serem usados em máquinas paralelas.

7.4.2 O Estado da Arte

Antes de começarmos a falar sobre a área de implementação paralela de RNA's, faremos breves comentários sobre alguns trabalhos relevantes na área de simuladores seqüenciais. Isto nos permitirá identificar características que devem ser consideradas nos ambientes de simulação paralela. Um dos simuladores neurais mais populares é o PDP, que acompanha o livro PDP (RUMELHART *et. al.*, 1986). Uma nova versão (PDP++) já está disponível. Esta versão apresenta uma interface gráfica que permite ao usuário estruturar a rede, escolher parâmetros, definir parâmetros de treinamento e inspecionar os resultados da simulação. Este software é projetado utilizando técnicas de orientação a objetos e implementado utilizando C++.

Outro excelente simulador seqüencial de domínio público é o SNNS (Stuttgart Neural Network Simulator) desenvolvido pela Universidade de Stuttgart. Algumas das idéias neste simulador foram inspiradas por outro simulador seqüencial, o RCS (Rochester Connectionist Simulator). SNNS possui uma opção que permite treinar múltiplas cópias da mesma RNA, ou diferentes RNA's simultaneamente em várias máquinas conectadas em rede. Existem ambientes de simulação como o DESCARTES (Development Environment for Simulating Connectionist ARchiTEctureS) que permitem-nos simular redes heterogêneas/híbridas.

Vários outros simuladores comerciais estão disponíveis para máquinas seqüenciais. Dentre eles podemos citar PlaNet, UCLA-SFINX, Xerion, NeuroGraph, BrainMaker, Asprin-Migraines, Pigmalion e o TOOLBOX do Matlab. Detalhes sobre estes simuladores podem ser encontrados no site comp.ai.neural-nets (URL 1).

Alguns autores têm trabalhado na área de desenvolvimento de linguagens concisas na descrição de arquiteturas de RNA's. Estas linguagens podem ser usadas para criar ambientes

de simulação tanto para máquinas seqüenciais quanto para máquinas paralelas. Antes que um ambiente seja desenvolvido para implementar modelos de RNA's de maneira eficiente em máquinas paralelas, deve ser feita uma análise teórica do mapeamento dos modelos neurais para os vários modelos de arquiteturas paralelas existentes.

Alguns requisitos necessários para uma implementação eficiente em um multicomputador de um modelo neural genérico deve envolver estratégias de mapeamento e uma análise das simulações dos mapeamentos.

7.4.3 Características Desejadas

Ainda não parece haver um ambiente integrado que permita ao usuário escolher um modelo neural dentre os mais comuns a partir de um menu e implementá-lo em uma variedade de máquinas paralelas. Este ambiente facilitaria a difícil transição encontrada por grande parte dos pesquisadores quando desejam migrar de uma implementação seqüencial para uma paralela. Quais características este ambiente deveria possuir? A seguir encontram-se alguns itens que devem ser considerados quando da criação de um ambiente que permita ao usuário gerar códigos paralelos otimizados para implementar determinado modelo de RNA em uma máquina paralela:

- **Análise teórica do paralelismo inerente**: antes de criarmos um ambiente para a implementação paralela, uma análise detalhada do paralelismo inerente da arquitetura de rede a ser implementada deve ser levado em conta. Esta análise deve investigar qual modelo de computação paralelo explora melhor o paralelismo existente em cada arquitetura de RNA. Isto fará com que o pesquisador tenha uma idéia da quantidade de paralelismo que pode ser explorada quando cada modelo é implementado na máquina paralela mais selecionada. O Capítulo 5 apresenta comentários sobre a paralelizabilidade dos principais algoritmos estudados neste trabalho.
- **Portabilidade**: um dos grandes problemas no mundo da computação paralela é que a maior parte das máquinas possui uma vida útil de poucos anos. Um usuário que desenvolveu um código para uma máquina há alguns anos atrás, geralmente tem que rescrevê-lo para uma outra máquina caso a original não esteja mais disponível. No ambiente de simulação otimizado, os usuários devem ser capazes de atualizar facilmente suas implementações para uma nova máquina. Uma forma de alcançar esta portabilidade é desenvolvendo o ambiente de maneira modular,

onde a descrição de uma determinada arquitetura de rede seja mantida em um formato independente da máquina, e os módulos de transição estejam disponíveis para cada máquina paralela.

- **Facilidade de uso do ponto de vista do usuário**: o ambiente deve apresentar uma interface gráfica de usuário (GUI) que permita ao usuário entrar dados no sistema. O usuário deve ser capaz de escolher a arquitetura de rede desejada, definir parâmetros apropriados e escolher a máquina paralela a ser usada a partir da GUI. O ambiente deve então gerar um código otimizado para executar a rede escolhida na máquina especificada.
- **Acesso a descrição de modelos das RNA's em vários níveis**: o ambiente deve fornecer aos usuários acesso a descrição dos modelos de RNA's em vários níveis. Um novo usuário do sistema deve ser capaz de utilizá-lo apenas em nível da GUI, escolher a arquitetura de rede, definir os parâmetros de usuário e escolher a máquina naquele nível. Um usuário intermediário deve ser capaz de acessar a descrição do modelo neural, e modificá-lo diretamente neste nível. O usuário avançado deve ser capaz de acessar o código paralelo diretamente e modificá-lo quando desejado.

Um ambiente ideal para a implementação das RNA's em máquinas paralelas deve então ser portável, fácil de se utilizar e que permita aos usuários implementar vários modelos neurais em diferentes máquinas paralelas de maneira eficiente.

MISRA (1997) apresenta uma rica análise sobre quais autores tem trabalhado na área de implementação de modelos paralelos para redes neurais artificiais, porém afirma que grande parte dos trabalhos desenvolvidos até o momento concentram-se em uma arquitetura específica de rede e/ou de máquina. Uma arquitetura de implementação ideal deve fornecer aos usuários métodos ótimos de implementação de grande variedade de modelos neurais em diferentes tipos de máquinas paralelas, algo muito distante quando avaliamos as implementações até aqui desenvolvidas.

Capítulo 8

Mapas Auto-Organizáveis de Kohonen

Neste capítulo visamos apresentar uma descrição de um método de poda desenvolvido para as redes auto-organizáveis de Kohonen e um novo método de inicialização das direções de busca dos modelos de aprendizagem baseada em busca de projeção. A partir de uma análise estatística do conjunto amostral, desenvolveu-se um algoritmo simplificado cujo objetivo principal é a eliminação das unidades de saída excedentes de um mapa auto-organizável unidimensional, permitindo assim a determinação de uma arquitetura mínima capaz de representar o conjunto de dados. O algoritmo apresentado como resultado pode ser facilmente aplicado a estruturas de dimensões mais elevadas, como os mapas bidimensionais.

Os resultados aqui apresentados diferem daqueles descritos e desenvolvidos nos capítulos anteriores por constituírem técnicas de treinamento não-supervisionado. Duas são as motivações básicas que fundamentam sua apresentação no contexto deste trabalho:

1. a ênfase na necessidade de se ajustar a complexidade da rede neural em função da natureza do problema abordado, como também se verificou no caso de treinamento supervisionado e
2. a necessidade de aplicação destes mapas auto-organizáveis na definição de condições iniciais ótimas para as redes neurais sujeitas a treinamento supervisionado, particularmente no caso de treinamento construtivo baseado em busca de projeção (*projection pursuit*).

8.1 Introdução

O Capítulo 7 apresenta procedimentos de poda para as arquiteturas do tipo MLP. Para os mapas auto-organizáveis de Kohonen (SOM), a abordagem torna-se um pouco diferente.

As arquiteturas auto-organizáveis, como propostas por KOHONEN (1982), geram mapeamentos de um espaço de dimensão elevada em estruturas cuja dimensão topológica é inferior à original. Estes mapeamentos são capazes de preservar as relações de vizinhança dos dados de entrada. Isto os torna interessantes para aplicações em diversas áreas, como reconhecimento de voz, análise exploratória de dados (KASKI, 1995; 1997) e otimização combinatória. O fato de que mapeamentos similares podem ser encontrados em diversas áreas

do cérebro humano e de outros animais indica que a preservação da topologia é um princípio importante pelo menos em sistemas de processamento de sinais.

A redução de dimensão pode acarretar uma perda de informação (a qual se procura minimizar) ou então pode levar a uma transformação topológica da informação original (de modo a explicitar relações não evidenciadas até então). Em ambos os casos, o que se procura é preparar a informação disponível para um processamento posterior, eliminando todo tipo de redundância e apresentando a informação de forma que ela possa ser diretamente manipulada.

Em termos de aplicação, foi verificado que a utilização de estruturas do modelo de Kohonen com dimensão arbitrária mas fixa implicam em limitações nos mapeamentos resultantes (FRITZKE, 1993). Um grande número de variações do algoritmo original, com o objetivo de determinação de uma arquitetura mais adequada ao problema a ser abordado, tem sido proposto na literatura. FRITZKE (1993) apresenta um mapa auto-organizável construtivo para treinamento supervisionado e não-supervisionado. É apresentado um procedimento controlado de inclusão de unidades (crescimento), juntamente com procedimentos de poda para a remoção ocasional de neurônios. CHO (1997) introduz um método dinâmico de "divisão" de unidades que representam mais de uma classe. São feitos comentários sobre procedimentos de poda para a eliminação de unidades pouco significativas, mas nenhum método é formalmente descrito e suas propriedades não são analisadas.

8.2 Os Mapas Auto-Organizáveis de Kohonen

Os mapas auto-organizáveis de Kohonen fazem parte de um grupo de redes neurais chamado redes baseadas em *modelos de competição*, ou simplesmente *redes competitivas* (FAUSETT, 1994). Estas redes combinam competição com uma forma de aprendizagem para fazer os ajustes de seus pesos.

Outra característica importante deste tipo de rede é que elas utilizam treinamento não-supervisionado, onde a rede busca encontrar similaridades baseando-se apenas nos padrões de entrada. O principal objetivo dos mapas auto-organizáveis de Kohonen é agrupar os dados de entrada que são semelhantes entre si formando classes ou agrupamentos denominados *clusters*.

Em uma rede classificadora há uma unidade de entrada para cada componente do vetor de entrada. Cada unidade de saída representa um *cluster*, o que limita a quantidade de *clusters* ao número de saídas. Durante o treinamento a rede determina a unidade de saída que melhor

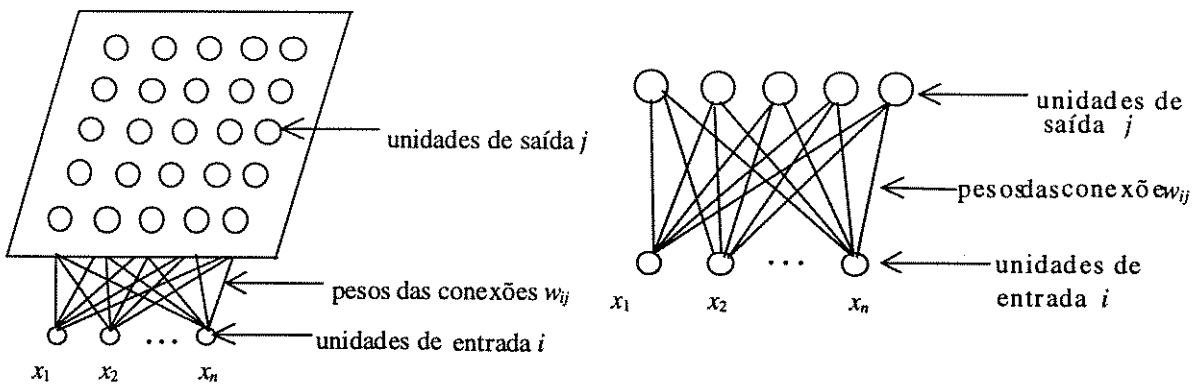
responde ao vetor de entrada; o vetor de pesos para a unidade vencedora é ajustado de acordo com o algoritmo de treinamento a ser descrito na próxima seção.

Durante o processo de auto-organização do mapa, a unidade do cluster cujo vetor de pesos mais se aproxima do vetor dos padrões de entrada é escolhida como sendo a "vencedora". A unidade vencedora e suas unidades vizinhas têm seus pesos atualizados segundo uma regra a ser descrita a seguir.

8.2.1 Arquiteturas

A Figura 8.1 apresenta arquiteturas típicas de um mapa auto-organizável de Kohonen, considerando configurações de vizinhança unidimensional e bidimensional, embora dimensões mais elevadas possam ser consideradas. Além disso, dada a dimensão, a quantidade de unidades ou neurônios de saída pode ser arbitrada e mantida fixa, ou então definida automaticamente pelo algoritmo de treinamento, como será descrito mais adiante para o caso de vizinhanças unidimensionais.

A quantidade de elementos de entrada depende do banco de dados a ser utilizado no treinamento da rede. O *grid* de saída pode ser de várias dimensões, com quantidade de elementos variável.



(a) (b)

Figura 8.1: Arquiteturas típicas de uma rede de Kohonen. (a) Arquitetura bidimensional. (b) Arquitetura unidimensional.

8.2.2 Algoritmo de Treinamento

O algoritmo padrão de treinamento é apresentado abaixo (KASKI, 1997):

1. Inicialização e definições de parâmetros
   - Inicialize os pesos w_{ij}.
   - Defina os parâmetros de vizinhança.
   - Defina os parâmetros de aprendizagem.
2. Enquanto a condição de parada é falsa, faça:
   2.1. Para cada j calcule:
      2.1.1. $D(j) = \arg\min_j \left\{ \left\| w_j - x_j \right\| \right\}$
      2.1.2. Encontre o índice J tal que $D(J)$ seja um mínimo.
      2.1.3. $\forall\ j \in Nc$ de J, e $\forall\ i$:

$$w_{ij}(novo) = w_{ij}(antigo) + \alpha\left[x_i - w_{ij}(antigo)\right]$$

   2.2. Atualize a taxa de aprendizagem
   2.3. Reduza o raio de vizinhança

A taxa de aprendizagem decresce lentamente com o tempo. A formação de um mapa ocorre em duas fases:

- formação inicial da ordem correta
- convergência final

A Figura 8.2 mostra um fluxograma representativo do algoritmo de treinamento do SOM. O algoritmo começa lendo os vetores dos padrões de entrada $x_1, x_2, \ldots, x_N$ (bloco 1). É assumido que são utilizados N padrões diferentes durante o processo de treinamento.

O próximo passo é selecionar os valores iniciais para os pesos das conexões w_{ij} (i=1, ..., n e j=1, ..., m) e o raio de vizinhança Nc. Kohonen sugeriu que os pesos das conexões sejam inicializados com valores aleatórios pequenos KOHONEN (1982).

O raio de vizinhança Nc é importante para atualizar os pesos (veja o bloco 7 da Figura 8.2). Como mostrado na Figura 8.3, os vizinhos de um nó de saída são definidos dentro de um bloco quadrado, com o nó j sendo o centro do bloco. Por exemplo, os vizinhos do nó de saída j com Nc = 4 incluem 81 nós no quadrado maior NE_j (Nc = 4) da Figura 8.3.

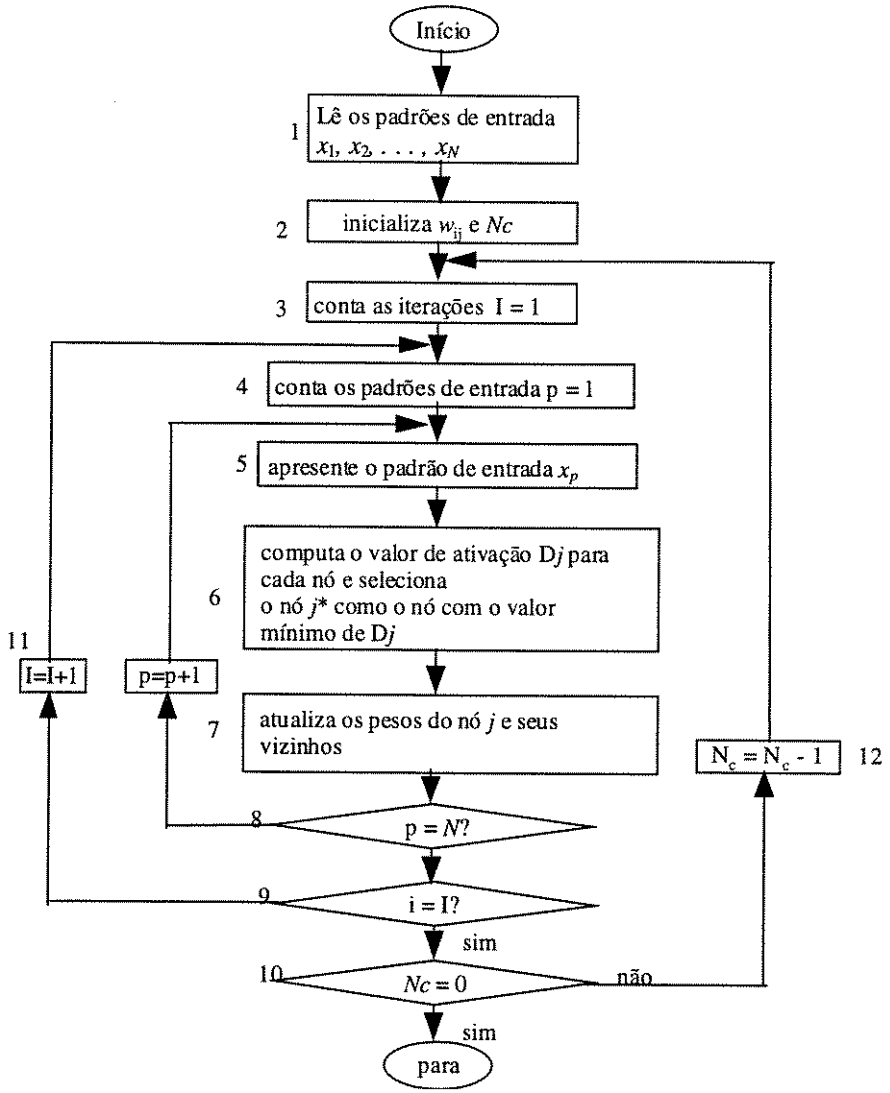


Figura 8.2: Diagrama de fluxo para o algoritmo que produz os mapas auto-organizáveis.

No processo de treinamento, *Nc* é decrementado de 1 após um determinado número de iterações (veja bloco 12 da Figura 8.2) até que *Nc* seja igual a zero. Aqui, diz-se que ocorreu uma iteração quando o vetor de padrões $x_1, x_2, \ldots, x_N$ tiver sido apresentado uma vez. O procedimento detalhado dentro de uma iteração é descrito como segue.

Depois de especificado os pesos iniciais das conexões, o valor de ativação de cada nó de saída pode ser computado. O nó *J* com o máximo valor de ativação é selecionado como a unidade vencedora. Isto é o que o algoritmo de treinamento faz no bloco 6 da Figura 8.2.

No bloco 7, os pesos das conexões para o nó *J* e todos os nós em sua vizinhança definidos por NE_j como mostrado na Figura 8.3 são atualizados.

Os procedimentos acima são repetidos para cada padrão de entrada. Quando todas as *N* amostras de entrada tiverem sido apresentadas, dizemos que uma iteração está completa.

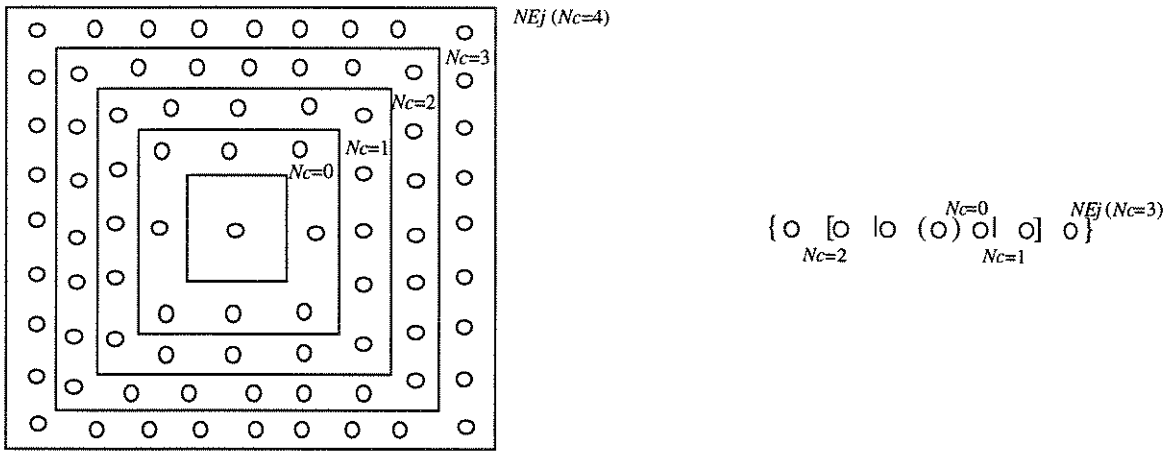


Figura 8.3: Vizinhos do nó j, NE_j (Nc). (a) Arquitetura bidimensional. (b) Arquitetura unidimensional.

8.2.3 Exemplo e Motivação para o Procedimento de Poda

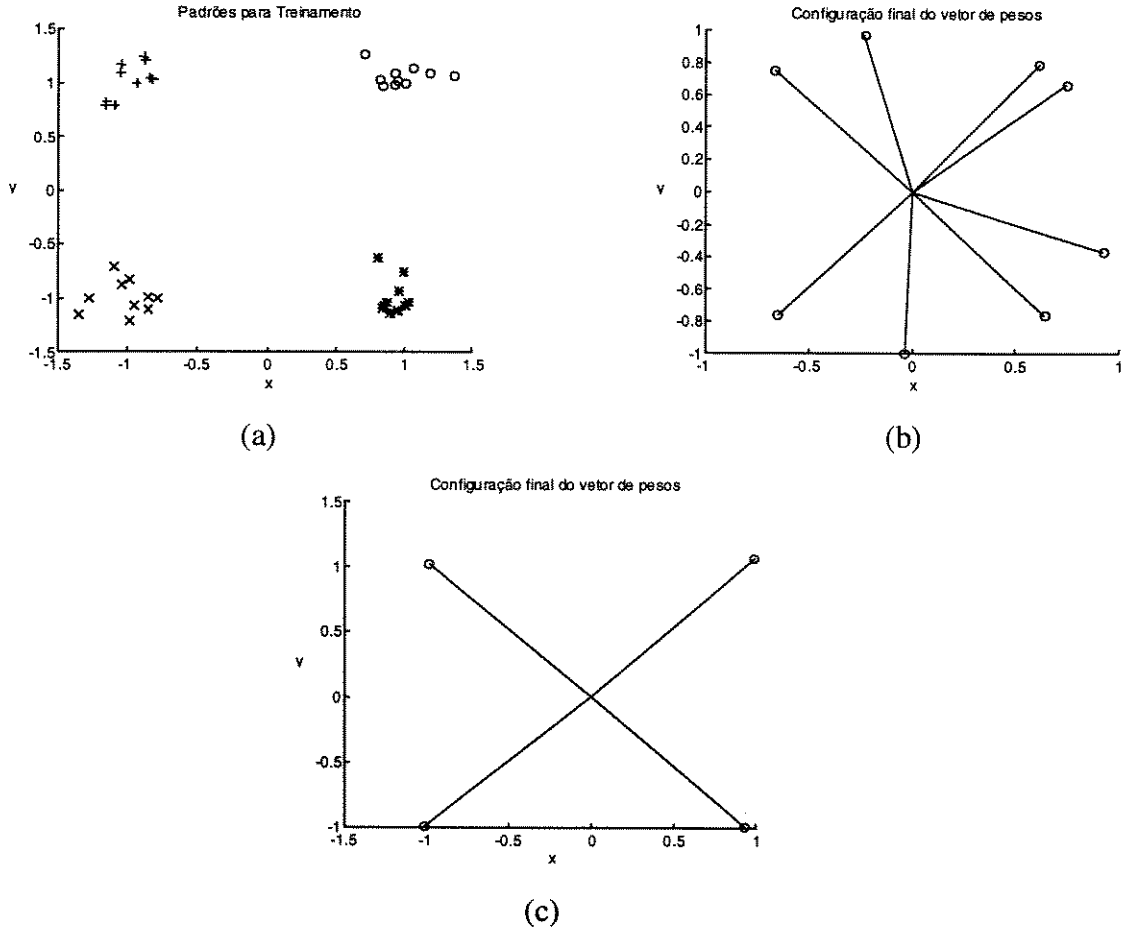


Figura 8.4: Motivação para o desenvolvimento de um método de poda para o SOM. (a) Conjunto de pontos a serem mapeados. (b) Mapeamento resultante para rede com 8 saídas. (c) Mapeamento resultante para uma rede inicialmente com oito saídas e cujas unidades redundantes foram retiradas pelo procedimento de poda proposto.

A Figura 8.4(a) apresenta um conjunto de dados composto por quatro classes disjuntas distribuídas no espaço $\Re^2$. Cada classe é composta por dez elementos. O algoritmo original, com oito unidades de saída, proposto por Kohonen e descrito nas seções precedentes apresenta, como um possível resultado, a configuração de pesos vista na Figura 8.4(b).

É intuitivo que quatro dos oito vetores de pesos podem ser eliminados de alguma forma, para que a rede possua uma arquitetura mínima e ainda seja capaz de representar satisfatoriamente os dados de treinamento. Um conjunto de vetores de pesos como vistos na Figura8.4(c), pode ser obtido através de procedimentos de poda, o qual certamente não conhece previamente o número de classes do problema.

8.3 Procedimento de Poda (PSOM) para Vizinhança Unidimensional

Quando pensamos em eliminar unidades pouco relevantes de uma rede neural artificial, três questões básicas surgem naturalmente:

- existem efetivamente unidades em excesso?
- se existirem, quais unidades devem ser eliminadas e de que forma?
- como a rede deve comportar-se após a retirada de uma unidade?

Ao longo desta seção pretendemos apresentar uma resposta para estas perguntas.

Como mencionado anteriormente, este método tem por objetivo reduzir a dimensão topológica do mapa gerado pelo algoritmo de treinamento original, e suas aplicações podem ser diversas, mas com ênfase em problemas de classificação ou agrupamento de padrões.

Os algoritmos de poda apresentados no Capítulo 7 permitem a retirada de unidades intermediárias através de uma análise de sensibilidade ou da inclusão de um termo de penalidade na função objetivo a ser minimizada.

Para as redes auto-organizadas em estudo, não existe um critério de erro a ser minimizado, mas sim um critério de distância a ser avaliado. Nosso objetivo é retirar unidades que são pouco representativas. Para isso, é preciso saber quais são estas unidades, se existirem. Definimos um vetor medida de agrupamento (**MA**) responsável por avaliar o grau de representatividade de cada unidade de saída da rede. Aquelas unidades cujas medidas de agrupamento **MA** são inferiores a um determinado limiar (ξ) são retiradas uma-a-uma, juntamente com todas as suas conexões.

Por outro lado, se uma determinada unidade de saída possui uma medida de agrupamento **MA** muito grande, um termo de penalização (τ) pode ser adicionado à medida de

agrupamento desta unidade, aumentando a probabilidade de que outras unidades possam classificar parte do conjunto amostral, evitando assim, excessos no processo de poda.

Após avaliada a medida de agrupamento de cada unidade de saída, e feita a poda de uma destas unidades (uma-a-uma), alguns fatores devem ser considerados. Geralmente os critérios de parada dos mapas auto-organizáveis são um limite de iterações ou um valor mínimo do passo de ajuste (α), nos casos em que α decresce durante o processo de aprendizagem. O procedimento de poda deve ser um procedimento retardado, ou seja, a rede é treinada da maneira usual durante algumas iterações, e assim que o mapa apresenta uma topologia pré-definida, inicia-se a eliminação de unidades cuja **MA** é pequena. Torna-se óbvio, então, que ao retirarmos uma unidade em fase adiantada do treinamento, o valor reduzido da taxa de aprendizagem permitirá pequenos ajustes no vetor de pesos. Este argumento evidencia a necessidade de uma reinicialização do valor do passo (α) e do raio de vizinhança *Nc* sempre que uma unidade for retirada.

A poda implica na retirada da unidade e reinicialização dos parâmetros de treinamento, tendo como condições iniciais, por exemplo, o estado anterior do processo. O aproveitamento do resultado obtido pela rede antes do processo de poda, torna intuitiva a idéia de que o algoritmo proposto (PSOM) seja sempre superior ao algoritmo padrão (SOM) para a mesma arquitetura final e mesmos parâmetros comuns, como α e α_{min}, caso o critério de parada seja α_{min}.

A condição de parada sugerida é um valor mínimo (α_{min}) para a taxa de aprendizagem (α).

O parâmetro ξ é um valor percentual, ou seja, se uma unidade não representa nem ξ% do conjunto de dados, estão ela pode ser retirada. Se a distribuição do conjunto de dados a ser utilizado é conhecida, então é sugerido que a metade do valor percentual da classe menos representativa do conjunto seja utilizada. Caso não saibamos nada a respeito dos dados de treinamento, geralmente são utilizados valores baixos como 0.5% ou 1% para o parâmetro ξ, de forma que a rede não perca a capacidade de representar todas as classes possíveis, inclusive as menos representativas.

Para conjuntos amostrais pequenos, com até algumas dezenas de amostras, o número de saídas inicial da rede pode ser tomado como sendo igual ao número de amostras disponíveis, ou seja, $m = N$; pois não é possível que existam mais do que N elementos distintos em um conjunto contendo N dados. Se o conjunto de dados é grande, esta estratégia é pouco

recomendável devido ao esforço computacional do início do processo de treinamento, mesmo contando etapas de poda.

Algumas das desvantagens do método proposto são a quantidade de parâmetros a serem arbitrados e um processo que inicia com uma arquitetura grande e tenta reduzi-la durante a adaptação.

Neste trabalho foi considerada a aplicação do método de poda apenas para o caso de vizinhanças unidimensionais, pela simplicidade na redefinição da vizinhança após a poda.

8.3.1 Algoritmo

O algoritmo PSOM de treinamento é apresentado abaixo:

1. Inicialização e definições de parâmetros
   - Inicialize os pesos w_{ij}
   - Defina os parâmetros de vizinhança (Nc_0) e aprendizagem (α_0)
   - Defina os parâmetros de penalização (τ) e poda (ξ)
   - Defina o número mínimo de saídas (m_{min}) e o retardo (ret)
2. Enquanto a condição de parada é falsa, faça:
   - 2.1. Para cada j calcule:
     - 2.1.1. $D(j) = \arg\min_j \left\{ \left\| w_j - x_j \right\| \right\}$
     - 2.1.2. Encontre o índice J tal que $D(J)$ seja um mínimo
     - 2.1.3. Incremente o componente **MA** correspondente a J
     - 2.1.4. $\forall\ j \in Nc$ de J, e $\forall\ i$:

$$w_{ij}(novo) = w_{ij}(antigo) + \alpha\left[x_i - w_{ij}(antigo)\right]$$

   - 2.2. Atualize a taxa de aprendizagem
   - 2.3. Se ($m > m_{min}$) & ($n_{ep} > ret$)
     - 2.3.1. Então, retire a unidade que classifica menos que ξ amostras, faça $\alpha = \alpha_0$ e $Nc = Nc_0$
     - 2.3.2. Senão, mantenha todas as unidades
   - 2.4. Se **MA**(k) é grande, penalize-o com o parâmetro τ
   - 2.5. Reduza o raio de vizinhança

8.4 Um Método de Inicialização para os Modelos de Busca por Projeção (KODIR)

Como visto no Capítulo 7, a determinação de uma condição inicial adequada do conjunto de pesos é de fundamental importância para garantir a convergência do processo de treinamento das arquiteturas do tipo MLP.

Embora apresentando uma dependência menor em relação à condição inicial, por tratar cada neurônio individualmente, os modelos baseados em busca de projeção (PPL), também estudados no Capítulo 7, devem utilizar um conjunto inicial de vetores de pesos para cada neurônio com o objetivo de explorar ao máximo o espaço de busca da solução ótima. Foi desenvolvida uma estratégia que faz uso dos mapas auto-organizáveis de Kohonen (SOM), no intuito de gerar um conjunto de vetores aproximadamente uniformemente distribuídos em um espaço multidimensional, para que a característica de máxima exploração do espaço de solução fosse atendida.

Como mencionado anteriormente, o algoritmo de treinamento do SOM tem como um de seus aspectos intrínsecos a realização de um mapeamento de um espaço de dimensão n (número de entradas) para um espaço de dimensão m (número de saídas). A questão que surge é: para que os vetores de entrada permitam um mapeamento aproximadamente uniforme nas saídas qual deve ser o conjunto de treinamento?

Como resposta para esta pergunta propomos a seguinte alternativa: se estamos buscando um conjunto de vetores que estejam o mais uniformemente distribuídos no espaço, então (pelas características do SOM) os vetores de treinamento também devem ser distribuídos da maneira mais uniforme possível.

Proposta uma forma de solução para o problema, torna-se necessário avaliar a qualidade dos resultados oferecidos por esta estratégia. A abordagem adotada neste item envolve uma análise de dados direcionais (MARDIA *et. al.*, 1979).

Seja $\mathbf{I}_i$, $i = 1, 2, \ldots, N$, um vetor unitário qualquer no espaço m-dimensional. Se $m = 2$, estes pontos podem ser representados no espaço de dimensão 2 e visualizados.

Considere:

$$\bar{\mathbf{I}} = \frac{1}{N}\sum_{i=1}^{N}\mathbf{I}_i \,, \tag{8.1}$$

como sendo o vetor médio. A direção do vetor médio $\bar{\mathbf{I}}$ é

$$\mathbf{I}_0 = \frac{\bar{\mathbf{I}}}{\bar{R}}, \quad \bar{R} = \left(\bar{\mathbf{I}}^T \bar{\mathbf{I}}\right)^{1/2}, \tag{8.2}$$

onde $\bar{R}$ é a distância do vetor médio em relação à origem e ^{T} denota a operação de transposição vetorial.

Por fim, considera-se o vetor resultante como sendo

$$\mathbf{r} = N.\bar{\mathbf{I}} = \sum_{i=1}^{N} \mathbf{I}_i. \tag{8.3}$$

Para avaliar o espalhamento dos vetores fornecidos pelo método proposto, considera-se a distância do vetor médio resultante à origem do sistema de coordenadas, ou seja, a amplitude do vetor resultante.

O método aceita como parâmetros de entrada a dimensão do espaço de saída (número de entradas da rede) e o número de direções que se deseja gerar (número de saídas da rede).

Alguns exemplos de aplicações no espaço de saída de dimensão 2 serão apresentados mais adiante no Capítulo 9.

Capítulo 9

Resultados Experimentais

Neste trabalho foram feitas simulações envolvendo grande parte dos tópicos apresentados. Primeiramente foi feita uma análise dos conjuntos amostrais (Capítulo 5). Em seguida foram comparados alguns dos métodos de otimização não linear irrestrita apresentados no Capítulo 3 e verificado as suas respectivas capacidades de generalização como no Capítulo 6. Os procedimentos de inicialização e métodos construtivos vistos no Capítulo 7 também serão comparados. Para verificar o desempenho dos algoritmos implementados, fez-se uma breve comparação com alguns dos algoritmos que constituem o TOOLBOX de Redes Neurais do MATLAB®.

Por último, serão apresentados os resultados de desempenho do método de poda e do método de inicialização das direções dos modelos baseados em busca de projeção propostos no Capítulo 8, que trata das redes auto-organizadas de Kohonen.

9.1 Os Problemas Abordados (*Benchmarks*)

Para a análise dos algoritmos estudados e validação dos métodos propostos, foram abordados oito problemas (*benchmarks*). Estes problemas podem ser divididos em três grupos:

- problemas de paridade;
- classificação; e
- aproximação de funções.

Seja k o número de entradas, N o número total de amostras e m o número de saídas. Uma descrição mais detalhada dos conjuntos amostrais é apresentada a seguir.

9.1.1 XOR

Problema de *paridade*, $k = 2$; $N = 4$ e $m = 1$. Trata-se de um problema artificial bastante conhecido na literatura. É caracterizado por um alto grau de não linearidade e por não preservar a idéia de vizinhança, ou seja, mudando apenas um *bit* de cada vetor de entradas, a saída é invertida.

9.1.2 COD/DEC

Problema de *paridade*, $k = 10$; $N = 10$ e $m = 10$. Uma classe de *benchmarks* muito utilizada em redes neurais é a família dos codificadores *N-l-N*. A rede consiste de N unidades de entrada, N unidades de saída e l neurônios na camada intermediária. O vetor de entrada é composto de N bits, um dos quais possui valor '1', e o resto '0'. O vetor de saídas desejadas é idêntico ao vetor de entradas, de forma que a tarefa da rede é realizar uma autoassociação entre os vetores de entrada e saída. O objetivo é aprender o mapeamento de N entradas para l unidades escondidas (codificação) e em seguida das l unidades intermediárias para as N saídas (decodificação). Em geral $l < N$. Se $l \leq \log_2 N$ diz-se que se trata de um 'codificador apertado' (*tight encoder*) (FAHLMAN, 1988).

9.1.2.1 Quando o Treinamento está Completo

Uma fonte de confusão no estudo de velocidade de aprendizagem reside no fato de que cada pesquisador escolhe um critério de parada diferente. Para os problemas da família COD/DEC (codificador/decodificador), este critério é estabelecido de maneira clara. Adota-se um critério de "limiar e margem" semelhante ao utilizado pelos projetistas de lógica digital: se a saída da rede é linear, deve-se escolher um valor acima do qual a saída é considerada um bit ativo, ou seja '1', e um valor abaixo do qual a saída é considerada inativa, ou seja '0'. Deve existir também uma faixa marginal entre estes valores onde a saída não pertence a nenhuma das duas classes.

Para definir qual a faixa de valores mais adequada a uma saída linear, executou-se os algoritmos algumas vezes. Os valores determinados, através de testes, para os limiares superior e inferior foram 0.45 e 0.4, respectivamente. Isso significa que uma saída com valor inferior a 0.4 é considerada um bit '0', e uma saída com valor superior a 0.45 é considerada bit '1'.

Definida a faixa de valores de decodificação da rede neural, esta é considerada treinada quando é capaz de reproduzir a matriz de saídas desejadas na saída da rede após a decodificação.

9.1.3 sen(x)×cos(2x)

Problema de *aproximação de funções*, $k = 1$; $N = 25$ e $m = 1$. Este problema é um pouco mais complicado do que o problema de aproximar a função sen(x), mas ainda é muito

fácil para os modelos de rede do tipo MLP. As 25 amostras são distribuídas uniformemente ao longo de um período (2π) da função (VAN DER SMAGT, 1994; SHEPHERD, 1997).

9.1.4 ESP (Estabilizador de Sistemas de Potência)

Problema de *aproximação de funções*, $k = 2$; $N = 75$ e $m = 5$. A função de um ESP implementado em um gerador síncrono é melhorar a estabilidade dinâmica deste tipo de sistema, fornecendo um torque de amortecimento adicional à malha de excitação do gerador. O projeto do ESP é feito utilizando-se modelos linearizados do sistema de potência em torno de um determinado ponto de operação.

Os dados utilizados para sintetizar o ESP neural são compostos pelos parâmetros do estabilizador com ganhos programados, juntamente com os pares de valores de P e Q, potência ativa e reativa, respectivamente. A rede neural possui duas entradas que recebem os valores de P e Q e cinco saídas que produzem os parâmetros do estabilizador (BARREIROS *et. al.*, 1996).

9.1.5 SOJA

Problema de *aproximação de funções*, $k = 36$; $N = 144$ e $m = 1$. Trata-se de um problema de mundo real no qual os dados amostrados são compostos de análises químicas do solo e das folhas das plantas quando estão com, aproximadamente, três meses de idade. A variável dependente é a produtividade, dada em kg/ha (CASTRO *et. al.*, 1997-1998a).

9.1.6 IRIS (Íris de Plantas da família das iridáceas)

Problema de *classificação*, $k = 4$; $N = 150$ e $m = 1$. O conjunto amostral é composto de três classes, cada uma contendo 50 amostras, referentes a um tipo de íris de planta diferente (as 4 variáveis são largura e comprimento da sépala e largura e comprimento da pétala). Uma classe é linearmente separável em relação as outras duas; as duas últimas não são linearmente separáveis entre si. A definição do número de saídas da rede com treinamento supervisionado é feita considerando-se uma saída para cada classe, ou seja, três saídas ($m = 3$). Este conjunto de dados está disponível no endereço URL 2 (URL 2), e faz parte do banco de dados do *UCI Repository of Machine Learning Database*.

9.1.6.1 Distribuição das Classes

As classes que compõem esta base de dados estão distribuídas da seguinte maneira:

- Classe 1: 50 amostras – 33,33% (Iris setosa);

- Classe 2: 50 amostras – 33,33% (Iris versicolor);
- Classe 3: 50 amostras – 33,33% (Iris virgínica);

9.1.7 GLASS (Tipos de Vidro)

Problema de *classificação*, $k = 10$; $N = 214$ e $m = 1$. O estudo de classificação de tipos de vidro foi motivado por investigações criminológicas. O conjunto de dados é composto por 214 amostras, com 10 entradas (atributos) e uma saída que caracteriza o tipo de vidro. Uma das entradas contém apenas o número de identificação, e foi suprimida nos processos de treinamento supervisionado. É definido que o número de saídas da rede com treinamento supervisionado três ($m = 3$), indicando uma codificação binária para as classes. Este conjunto de dados está disponível no endereço URL 2 (URL 2), e faz parte do banco de dados do *UCI Repository of Machine Learning Database*.

9.1.7.1 Distribuição das Classes

As classes que compõem esta base de dados estão distribuídas da seguinte maneira:

- Classe 1: 70 amostras – 32,71% (vidros utilizados em construção);
- Classe 2: 76 amostras – 35,51% (vidros utilizados em construção*);
- Classe 3: 17 amostras – 7,94% (vidros de automóveis);
- Classe 4: 13 amostras – 6,07% (*containers*);
- Classe 5: 9 amostras – 4,21% (vidros de mesa) e
- Classe 6: 29 amostras – 13,55% (bulbos de lâmpadas).

* A diferença entre as classes 1 e 2 está no processo de fabricação.

9.1.8 ECOLI (Localização de Proteínas)

Problema de *classificação*, $k = 7$; $N = 336$ e $m = 1$. Trata-se de um problema de predição de localização de proteínas em células eucarióticas. É definido que o número de saídas da rede com treinamento supervisionado três ($m = 3$), indicando uma codificação binária para as classes. Este conjunto de dados está disponível no endereço URL 2 (URL 2), e faz parte do banco de dados do *UCI Repository of Machine Learning Database*.

9.1.8.1 Distribuição das Classes

As classes que compõem esta base de dados estão distribuídas da seguinte maneira:

- Classe 1: 143 amostras – 42,56% (citoplasma);

- Classe 2: 77 amostras – 22,92% (membrana interna sem a seqüência de sinais);
- Classe 3: 52 amostras – 15,48% (periplasma);
- Classe 4: 35 amostras – 10,42% (membrana interna com a seqüência de sinais);
- Classe 5: 20 amostras – 5,95% (membrana externa);
- Classe 6: 5 amostras – 1,49% (membrana externa lipoproteica);
- Classe 7: 2 amostras – 0,60% (membrana interna lipoproteica) e
- Classe 8: 2 amostras – 0,60% (membrana interna com a seqüência de sinais).

9.2 Análise dos Conjuntos Amostrais

O Capítulo 5 apresenta seis critérios que podem ser utilizados para avaliar a relevância dos dados utilizados no processo de treinamento. Utilizaremos estes critérios para avaliar os conjuntos amostrais empregados neste trabalho. Serão analisados o número de atributos (*NA*), o número de valores nominais diferentes por atributo (*DNAV*), a quantidade de atributos irrelevantes (*IAtt*), o tamanho do conjunto amostral (*TCD*), o tamanho do espaço de descrição (*TED*) e a densidade do conjunto amostral (*D*). Os resultados estão apresentados na Tabela 9.1.

Segundo os critérios adotados, e a partir da Tabela 9.1, os problemas (1), (2), (3), (5) e (6) possuem conjuntos de dados que são relevantes o suficiente para permitir bons resultados computacionais; enquanto os problemas (7) e (8) estão em um ponto intermediário; e o problema (4) possui uma base de dados pouco significativa para descrevê-lo.

Tabela 9.1: Análise dos conjuntos amostrais utilizados nos experimentos.

ID#	PROBLEMA	*NA*	*DNAV*	*IAtt*	*TCD*	*TED*	*D*	*CLAS*
1	XOR	02	00	*Não*	4	4	1.00	*Alta*
2	Sen(x)×cos(2x)	01	00	*Não*	25	25	1.00	*Alta*
3	COD/DEC	10	02	*Não*	10	1.02e+03	97.66e-04	*Alta*
4	SOJA	36	10	*Sim*[1]	144	1.62e+58	8.89e-57	*Baixa*
5	ESP	02	00	*Não*	75	3.60e+03	0.020833	*Alta*
6	ÍRIS	04	00	*Não*	150	7.62e+05	1.97e-04	*Alta*
7	GLASS	09	09	*Não*	214	3.77e+17	5.68e-16	*Média*
8	ECOLI	07	03	*Não*	336	7.32e+09	4.59e-08	*Média*

[1] De acordo com uma classificação feita por um mapa auto-organizável de Kohonen (SOM), existem atributos irrelevantes neste conjunto amostral.

Obs.: *CLAS* refere-se à classificação da densidade do conjunto.

9.3 Velocidade de Convergência

Tomando alguns dos algoritmos de treinamento descritos no Capítulo 3, nesta seção objetivamos verificar seu desempenho em relação à velocidade de convergência para cada um dos oito problemas propostos acima. Os algoritmos a serem comparados são:

- algoritmo padrão (BP);
- método do gradiente (GRAD);
- Fletcher-Reeves (FR);
- Polak-Ribiére (PR);
- secante de um passo (OSS);
- gradiente conjugado escalonado modificado (SCGM);
- Davidon-Fletcher-Powell (DFP); e
- Broyden-Fletcher-Goldfarb-Shanno (BFGS).

Somam-se ao total oito algoritmos dos onze apresentados no Capítulo 3. Acrescentamos também que uma versão do método de Newton chegou a ser implementada, mas não foi utilizada para comparações pois a construção exata da matriz hessiana é uma tarefa excessivamente custosa do ponto de vista computacional, considerando-se a dimensionalidade dos problemas abordados.

Os resultados serão apresentados individualmente para cada problema, com uma descrição detalhada da arquitetura utilizada e parâmetros associados a cada um dos algoritmos. Para diminuir a variância nas soluções, todos os algoritmos foram empregados no treinamento de uma rede neural inicializada com o mesmo vetor de pesos (distribuídos uniformemente em um dado intervalo) em cada caso, sendo que foi utilizada uma quantidade diferente de condições iniciais (N_{ci}) para cada problema. Utilizou-se redes do tipo MLP com uma única camada intermediária, com função de ativação tangente hiperbólica ($f(x)$ = tanh(x)) para as unidades intermediárias e função linear ($f(x) = x$) para as unidades de saída. O limite de épocas de treinamento foi definido em 50.000 para os métodos de primeira ordem e 5.000 para os métodos de segunda ordem.

Para os problemas menos complexos, todos os algoritmos foram capazes de convergir, incluindo os métodos de primeira ordem (ver Capítulo 3). Nestes casos, utilizou-se como medida de erro (critério de parada) a soma dos erros quadráticos (*SSE*) pois sua medida independe do número de amostras para treinamento. Ao tratar problemas de complexidade

mais elevada, verificou-se que alguns dos algoritmos testados não foram capazes de convergir para os critérios de erro desejados. Uma forma de amenizar este problema é utilizar procedimentos de validação cruzada, onde o conjunto amostral foi dividido em três grupos (treinamento, validação e teste) e o treinamento interrompido segundo algum critério de generalização pré-definido. Ao dividir o conjunto de dados em partes de diferentes tamanhos, o número de amostras de cada parte não pode influenciar no valor absoluto do erro, e por isso utilizou-se o erro quadrático médio (*MSE*) para os casos em que não houve convergência.

O método de busca simples do passo, utilizado pelo algoritmo do gradiente possui os seguintes parâmetros: $t_a = 1.2$ e $t_r = 0.618$ (razão áurea).

O coeficiente de momento μ é utilizado apenas pelos métodos de primeira ordem (BP e GRAD). Os valores das taxas de aprendizagem (α_{bp}) foram definidos através de procedimentos de tentativa e erro de forma a adequar-se aos problemas tratados e, por isso geralmente, apresentam valores diferentes para problemas distintos. O intervalo de inicialização do conjunto de pesos da rede, em alguns problemas, foi reduzido de [-1.0; 1.0] para [-0.2; 0.2]. Isso deve-se, principalmente, ao fato de que estudos efetuados (CASTRO *et. al.* 1998b; CASTRO & VON ZUBEN, 1998c) demonstraram que ao inicializarmos os pesos da rede na região aproximadamente linear das funções de ativação (tangentes hiperbólicas) aumenta-se as chances de que o algoritmo tenha convergência mais rápida e seja menos susceptível a problemas de instabilidade numérica.

Legenda:

Net	[entradas, intermediárias, saídas]
P	número de parâmetros livres da rede
N	número de amostras
N_{ci}	quantidade de condições iniciais diferentes utilizadas no treinamento
Inic.	intervalo de inicialização dos pesos
SSE	soma dos erros quadráticos desejada
T	tempo de processamento dado em segundos*
flops	número de operações de ponto flutuante (complexidade computacional)
$\|\|\nabla J(\theta)\|\|$	norma L_2 do vetor gradiente após convergência
α_{bp}	taxa de aprendizagem do algoritmo padrão
α_{quick}	taxa de aprendizagem do quickprop
μ	coeficiente de momentum utilizado nos métodos de primeira ordem

O tempo de processamento foi medido utilizando-se estações de trabalho SPARC ULTRA 1 da *Sun*. É importante mencionar que essa medida é função da carga à qual a máquina está submetida e de sua arquitetura.

9.3.1 XOR

Parâmetros da rede:

Net:	[2, 35, 1]	*SSE*:	0.01
P:	141	α_{bp}:	0.001
N:	4	μ:	0.95
N_{ci}:	20	*Inic.*:	[-1.0; 1.0]

A Tabela 141.8 apresenta a comparação de desempenho médio ($N_{ci} = 20$) dos algoritmos para o problema XOR. Todos os métodos foram capazes de convergir.

Para este problema em particular, o número de unidades intermediárias escolhido (35) foi bastante elevado, pois foram feitos experimentos simplificados sobre o comportamento dos algoritmos em relação ao aumento da complexidade do problema. Sendo assim, para a mesma arquitetura de rede ([k, 35, 1]), variou-se a paridade $k = 2, 3, \ldots, 9$.

Tabela 9.2: Velocidade média de convergência dos métodos de otimização para o problema XOR.

	ÉPOCAS	$\|\nabla J(\theta)\|$	T(seg.)	*flops* $\times 10^6$
BP	861	0.12879	14.65	5.34
GRAD	85	0.24059	1.48	0.66
FR	151	0.51931	26.94	9.95
PR	19	0.37450	3.93	1.49
OSS	45	0.79577	10.05	3.45
SCGM	9	0.33544	1.10	0.67
DFP	30	0.52997	13.05	153.13
BFGS	23	0.52833	12.71	151.40

9.3.2 COD/DEC

Parâmetros da rede:

Net:	[10, 7, 10]	*Inic.*:	[-0.2; 0.2]	N_{ci}:	20
P:	157	α_{bp}:	0.001		
N:	10	μ:	0.95		

Tabela 9.3: Velocidade média de convergência dos métodos de otimização para o problema COD/DEC.

	SSE	ÉPOCAS	$\|\nabla J(\theta)\|$	$flops \times 10^6$	T(seg.)
BP	2.7256	404	0.595155	5.26	6.55
GRAD	2.8821	85	1.720502	1.22	1.49
FR	3.9119	22	2.596097	2.39	2.16
PR	3.0021	60	1.166772	3.53	4.36
OSS	2.8655	392	1.789399	3.86	3.20
SCGM	2.7054	22	0.672523	2.17	2.18
DFP	2.9421	125	0.735807	344.52	41.22
BFGS	2.6851	122	1.579901	537.35	44.43

Neste caso tem-se $7 > \log_2 10$, e portanto não se trata de um codificador apertado. A Tabela 9.3 apresenta a comparação de desempenho médio ($N_{ci} = 20$) dos algoritmos de otimização para este problema.

Como mencionado anteriormente, o critério de parada adotado para este problema é diferente dos demais, e portanto é feita apenas uma avaliação da soma do erro quadrático após a convergência.

9.3.3 sen(x)×cos(2x)

Parâmetros da rede:

Net:	[1, 10, 1]	*SSE*:	0.1
P:	31	α_{bp}:	0.005
N:	25	μ:	0.95
N_{ci}:	20	*Inic.*:	[-1.0; 1.0]

A Tabela 0 apresenta a comparação da velocidade média ($N_{ci} = 20$) de convergência dos algoritmos para o problema de aproximação da função sen(x)×cos(2x).

Tabela 9.4: Velocidade média de convergência dos métodos de otimização para o problema sen(x)×cos($2x$).

	ÉPOCAS	$\|\|\nabla J(\theta)\|\|$	*flops* $\times 10^6$	T(seg.)
BP	15257	0.021516	195.86	374.67
GRAD	11324	0.022548	143.74	342.45
FR	360	0.623722	32.21	60.81
PR	414	0.328516	36.41	65.00
OSS	2709	0.251328	282.89	469.95
SCGM	172	0.717829	17.19	23.63
DFP	134	0.325639	21.39	19.74
BFGS	199	0.568088	35.35	33.23

9.3.4 ESP

Parâmetros da rede:

Net:	[2, 10, 5]	*SSE*:	0.1
P:	85	α_{bp}:	0.005
N:	75	μ:	0.95
N_{ci}:	10	*Inic.*:	[-0.2; 0.2]

A Tabela 9.5 apresenta a comparação de desempenho médio ($N_{ci} = 10$) dos algoritmos de treinamento quando aplicados ao problema do estabilizador de sistemas de potência (ESP).

Tabela 9.5: Velocidade média de convergência dos métodos de otimização aplicados ao problema ESP.

	SSE	ÉPOCAS	$\|\|\nabla J(\theta)\|\|$	*flops* $\times 10^6$	T(seg.)
BP	0.234668	50000	0.006857	3377.82	2282.60
GRAD	0.226450	50000	0.006580	3786.36	2410.22
FR	0.099937	881	0.090815	474.99	256.68
PR	0.099839	2116	0.386405	1046.72	488.36
OSS	0.097950	7586	0.354955	2071.72	1097.94
SCGM	0.099199	587	0.115537	276.95	177.87
DFP	0.251334	5000	0.011506	3059.57	1341.53
BFGS	0.251657	5000	0.011414	3665.34	1437.99

9.3.5 SOJA

Parâmetros da rede:

Net:	[36, 10, 1]	*SSE*:	0.1
P:	381	α_{bp}:	0.005
N:	116	μ:	0.95
N_{ci}:	10	*Inic.*:	[-0.2; 0.2]

A Tabela 9.6 apresenta a comparação da velocidade média ($N_{ci} = 10$) de convergência dos métodos de otimização aplicados ao problema de previsão de produtividade de soja (SOJA).

Tabela 9.6: Velocidade média de convergência dos métodos de otimização para o problema SOJA.

	ÉPOCAS	$\|\nabla J(\theta)\|$	*flops* $\times 10^6$	*T*(seg.)
BP	20318	8.454046	4503.32	1289.66
GRAD	18803	0.128312	4784.54	1050.39
FR	1221	0.252503	2637.69	506.47
PR	540	0.444752	1135.71	198.31
OSS	1383	0.614701	3340.76	692.731
SCGM	257	0.715872	521.39	96.68
DFP	291	1.049450	32660.79	3694.84
BFGS	322	0.698576	30941.98	3199.89

9.3.6 IRIS

Parâmetros da rede:

Net:	[4, 10, 3]	*SSE*:	0.15
P:	83	α_{bp}:	0.001
N:	150	μ:	0.95
N_{ci}:	10	*Inic.*:	[-0.2; 0.2]

A Tabela 9.7 apresenta a comparação da velocidade média ($N_{ci} = 10$) de convergência dos métodos de otimização aplicados ao problema de classificação de tipo de íris (IRIS). Neste caso, os algoritmos de primeira ordem, os métodos de quase-Newton e o método OSS, não foram capazes de convergir em nenhuma das 10 simulações. Como os métodos de gradiente conjugado foram capazes de convergir dentro do limite de épocas pré-especificado, não foram testadas redes com dimensões superiores à apresentada ([4, 10, 3]).

Tabela 9.7: Velocidade média de convergência dos algoritmos de otimização para o problema IRIS.

	SSE	ÉPOCAS	$\|\nabla J(\theta)\|$	*flops* $\times 10^6$	T(seg.)
BP	14.69	50000	0.069797	5826.55	3203.67
GRAD	6.82	50000	0.849566	6116.95	5210.66
FR	0.15	2405	0.250552	2016.16	1093.83
PR	0.15	2111	1.155984	1716.25	966.08
OSS	4.48	5000	0.462826	5695.23	2574.17
SCGM	0.15	1019	0.326285	812.62	503.15
DFP	6.42	5000	1.730771	10845.78	2536.61
BFGS	9.11	5000	0.776759	11840.64	2972.65

9.3.7 GLASS

Parâmetros da rede:

Net:	[9, 16, 3]	*SSE*:	0.15
P:	211	α_{bp}:	0.001
N:	214	μ:	0.95
N_{ci}:	10	*Inic.*:	[-0.2; 0.2]

Nenhum algoritmo foi capaz de convergir para o critério de erro desejado. Para que ainda fosse possível fazer a comparação entre os diferentes métodos, foi feito o treinamento exaustivo de todos os métodos, ou seja, os métodos de primeira ordem foram treinados até um limite de 50.000 épocas, e os métodos de segunda ordem até 5.000 épocas. O critério de erro utilizado foi o erro quadrático médio (MSE), pois como dito anteriormente, este é independente do tamanho do conjunto de dados. MSE_{tr} corresponde ao erro do conjunto de treinamento, MSE_{val} ao erro do conjunto de validação e MSE_{te} ao erro do conjunto de teste. Maiores detalhes sobre estas medidas de erro serão dados posteriormente, nas simulações que envolvem procedimentos de validação cruzada.

A Tabela 9.8 apresenta uma comparação da velocidade média ($N_{ci} = 10$) de convergência dos métodos de otimização implementados quando aplicados ao problema de classificação de tipos de vidro (GLASS).

Tabela 9.8: Velocidade de convergência dos métodos de otimização para o problema GLASS.

	MSE			*EP*	$\|\nabla J(\theta)\|$	$flops \times 10^6$
	MSE_{tr}	MSE_{val}	MSE_{te}			
BP	0.205891	2.2426	3.08130	50000	59.3554	8529.80
GRAD	0.324729	2.4465	1.79153	50000	134.632	9942.66
FR	0.156150	12.7743	11.3970	5000	0.6438	7066.38
PR	0.136103	2.3864	4.56294	5000	2.6744	7450.81
OSS	0.307864	2.4813	2.84755	5000	3.2956	9360.64
SCGM	0.081086	3.3898	4.29241	5000	104.3746	7545.26
DFP	0.688553	1.8633	2.90665	5000	2.0361	101991.95
BFGS	0.281159	4.9137	1.78893	5000	2.9778	105521.20

9.3.8 ECOLI

Parâmetros da rede:

Net: [7, 16, 3]

P: 179

N: 336

N_{ci}: 10

SSE: 0.15

α_{bp}: 0.001

μ: 0.95

Inic.: [-0.2; 0.2]

Tabela 9.9: Velocidade média de convergência dos métodos de otimização para o problema ECOLI.

	MSE			*EP*	$\|\nabla J(\theta)\|$	$Flops \times 10^6$
	MSE_{tr}	MSE_{val}	MSE_{te}			
BP	0.246722	1.001101	0.600617	50000	127.9263	14292.85
GRAD	0.216865	0.979296	0.566425	50000	0.21687	16178.95
FR	0.163703	3.889876	1.246959	5000	0.71825	11788.05
PR	0.155086	2.567767	1.480892	5000	0.43058	15116.22
OSS	0.454733	0.774083	0.740226	5000	2.41160	15116.22
SCGM	0.067541	4.562005	4.669367	5000	366.9990	12274.50
DFP	0.211168	1.671458	0.922441	5000	1.6766	71799.41
BFGS	0.188654	1.541187	1.030033	5000	1.9453	74278.86

Os algoritmos não foram capazes de resolver este problema dentro do limite de épocas especificado, por isso as mesmas considerações do problema anterior serão feitas.

A Tabela 9.9 apresenta uma comparação da velocidade média ($N_{ci} = 10$) de convergência dos métodos de otimização implementados quando aplicados ao problema de classificação de proteínas (ECOLI).

9.3.9 Estatísticas e Comentários

A Figura 9.1 apresenta as estatísticas de desempenho de todos os algoritmos testados para todos os problemas propostos. Nos últimos dois problemas (GLASS e ECOLI), não foram comparados o tempo de processamento e nem o esforço computacional envolvido no processo de aprendizagem, pois nenhum algoritmo convergiu e foi considerado o treinamento exaustivo. Nestes casos, serão comparados apenas os algoritmos que foram capazes de atingir o menor erro quadrático médio para o conjunto de treinamento.

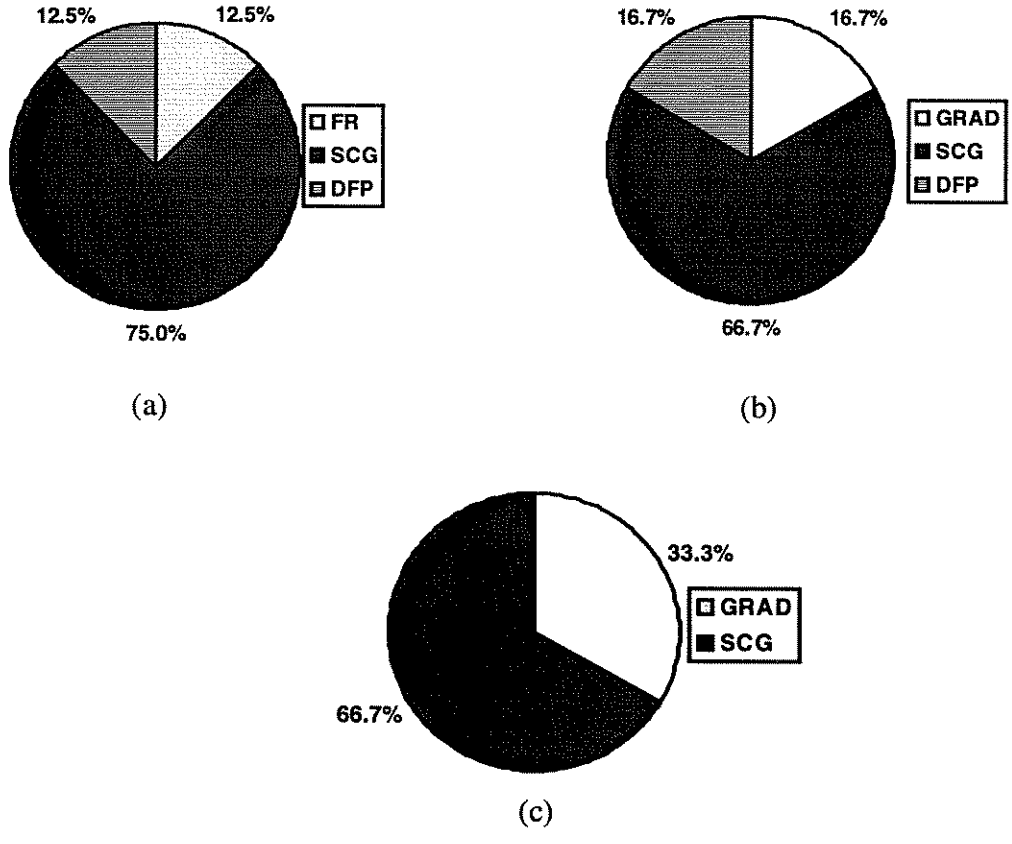


Figura 9.1: Comparação geral de desempenho dos algoritmos em relação aos oito problemas abordados. Os valores percentuais indicam o quanto cada um dos algoritmos mostrou-se superior aos outros para todos os problemas. (a) Número de épocas para convergência. (b) Menor tempo de processamento. (c) Menor esforço computacional.

O tempo de processamento não foi considerado, na simulação dos dois últimos problemas (GLASS e ECOLI), pois os algoritmos foram executados em máquinas com arquiteturas muito distintas, inviabilizando uma comparação envolvendo este parâmetro.

Os resultados apresentados nesta seção confirmam os argumentos propostos no Capítulo 5, que trata dos aspectos numéricos envolvidos no processo de aprendizagem das arquiteturas MLP. Os métodos que utilizam informações de segunda ordem tiveram convergência em um número menor de épocas para todos os problemas, e exigiram menor esforço e tempo de processamento na maioria dos casos estudados.

Estes resultados despertam o sentimento de que existe a necessidade clara de utilização de abordagens mais atuais, eficientes e bem fundamentadas teoricamente no processo de treinamento das redes do tipo perceptron de múltiplas camadas. Em contrapartida à inspiração biológica inicialmente proposta para as redes neurais artificiais, estamos propondo, formalizando e mostrando na prática que a utilização dos algoritmos de otimização não-linear irrestrita no aprendizado das RNA's constitui uma análise indispensável para o desenvolvimento de algoritmos mais eficientes e robustos.

9.4 Capacidade de Generalização

Como estudado no Capítulo 6, ao trabalhar com conjuntos de dados reais, deve sempre ser considerado o fato de que as amostras a serem utilizadas estão sujeitas a ruídos, obtidos principalmente, durante os procedimentos de amostragem. Além disso, podem haver dados redundantes e um número de amostras insuficientes para representar o problema desejado. Estes fatores tornam necessária a utilização de procedimentos de validação cruzada, ou regularização, para que se possa obter uma rede com melhor capacidade de generalização, ou seja, uma rede que responda corretamente a dados que não foram utilizados durante o processo de treinamento. Sendo assim, para os problemas reais (todos com exceção do XOR e COD/DEC) foram aplicados procedimentos de validação cruzada (CV) no treinamento. Para o problema de aproximação da função $\text{sen}(x).\cos(2x)$ introduziu-se ruído uniformemente distrubuído para que pudessem ser aplicados os procedimentos de CV.

Foram utilizados procedimentos de validação cruzada, tomando-se como base os critérios CV e PQ apresentados no Capítulo 6 (seção 6.1.2), com $s = 2, 4, 6$ e $PQ = 0.5$. Por conveniência repetiremos a seguir a descrição dos critérios de parada CV e PQ.

9.4.1 Critérios de parada dos procedimentos de validação cruzada (CV)

Existe uma grande quantidade de critérios de parada que podem ser utilizados. Para este estudo tomou-se três deles como base. Para descrever um critério, façamos primeiro algumas definições. Seja *MSE* a função objetivo (erro quadrático médio – do inglês *mean squared error*). Então $MSE_{tr}(t)$ é o erro do conjunto de treinamento, $MSE_{val}(t)$ o erro do conjunto de validação utilizado pelo critério de parada e $MSE_{te}(t)$ o erro do conjunto de teste.

O valor $MSE_{opt}(t)$ é definido como sendo o menor erro do conjunto de validação obtido até o instante *t*:

$$MSE_{opt}(t) = \min_{t' \leq t} MSE_{val}(t). \tag{9.1}$$

Agora definimos a *perda de generalização* na iteração *t* como sendo o aumento relativo do erro de validação em relação ao mínimo (percentual).

$$GL(t) = 100.\left(\frac{MSE_{val}(t)}{MSE_{opt}(t)} - 1 \right). \tag{9.2}$$

Uma alta perda de generalização é um bom motivo para interromper o treinamento, pois indica diretamente que a rede está sendo sobre treinada. Isto resulta no primeiro critério de parada: pare assim que a *perda de generalização* supere um limiar α .

GL_α : pare após a *t*-ésima época caso $GL(t) > \alpha$

Caso o treinamento esteja progredindo rapidamente, é desejável que ele não seja interrompido. A razão para isso é que quando o erro de treinamento está caindo muito rapidamente, sua queda sempre predomina sobre um possível aumento na variância. Para formalizar esta noção define-se um número de épocas *k* de treinamento, após o qual é feito o procedimento de validação cruzada. O *progresso de treinamento* (em milhares) medido após *k* iterações é dado por:

$$P_k(t) = 1000.\left(\frac{\sum_{t'=t-k+1}^{t} MSE_{tr}(t')}{k.\min_{t'=t-k+1}^{t} MSE_{tr}(t')} - 1 \right), \tag{9.3}$$

que significa "quanto a média do erro de treinamento durante *k* épocas é maior que o mínimo erro de treinamento durante *k* épocas?"

O critério de parada pode ser definido então como sendo o quociente entre a *perda de generalização* e o *progresso de treinamento*.

PQ_α : pare após a *t*-ésima época caso $\frac{GL(t)}{P_k(t)} > \alpha$.

O último critério de parada estudado neste trabalho é, talvez, o mais conhecido de todos. Este critério interrompe o treinamento quando o erro de generalização aumenta s vezes a cada k épocas. Assim o critério de parada torna-se:

CV: pare após a época t se $MSE_{val}(t) > MSE_{val}(t - k)$ durante $s.k$ épocas.

Nenhum dos critérios apresentados garante uma interrupção do treinamento, portanto é importante definir um limite de épocas ou um limiar para o valor do erro ou do vetor gradiente.

9.4.2 Medidas de desempenho

A descrição da arquitetura de rede e seus parâmetros é a mesma utilizada anteriormente. O critério de erro é o erro quadrático médio (MSE). Além do MSE foi utilizada uma outra grandeza que mede a quantidade de dados que a rede está sendo capaz de representar, chamada *fração de variância inexplicada* (*FVU*), dada por:

$$FVU = \frac{MSE}{MST} \tag{9.4}$$

onde:

$$MSE = \frac{1}{N}\sum_{i=1}^{m}\sum_{j=1}^{N}\left(T(i,j) - y(i)\right)^2 \text{ e}$$

$$MST = \frac{1}{N}\sum_{i=1}^{m}\sum_{j=1}^{N}\left(T(i,j) - \bar{y}(i)\right)^2 \text{ onde } \bar{y}(i) = \frac{1}{N}\sum_{j=1}^{N} y(i,j).$$

Legenda:

N_{tr}: número de amostras utilizadas no treinamento

N_{val}: número de amostras utilizadas nos procedimentos de validação cruzada

N_{te}: número de amostras utilizadas no teste da rede

Os dois melhores resultados em relação ao menor erro de teste e menor fração de variância inexplicada (*FVU*) do conjunto de teste estão sombreados. O tracejado em alguma linha implica que o algoritmo não atingiu o critério antes de esgotado o limite de épocas.

9.4.3 sen(x)×cos(2x)

Divisão do conjunto amostral:

N_{tr}: 25 N_{te}: 25

N_{val}: 25

Tabela 9.10: Capacidade de generalização para o critério $s = 2$, 4 e 6. Problema sen(x)×cos($2x$).

	s	MSE			FVU			EP	*flops* ($\times 10^6$)
		MSE_{tr}	MSE_{val}	MSE_{te}	FVU_{tr}	FVU_{val}	FVU_{te}		
BP	2	0.12975	0.11856	0.13974	0.49398	0.47806	0.52150	1320	21.84
	4	0.13100	0.11813	0.13915	0.49877	0.47636	0.51929	1950	31.84
	6	0.02165	0.03404	0.02536	0.08242	0.13727	0.09463	2640	39.50
GRAD	2	0.13208	0.12034	0.14192	0.50288	0.48524	0.52963	590	11.00
	4	0.12843	0.11780	0.13875	0.48899	0.47500	0.51780	1120	20.68
	6	--------	--------	--------	--------	--------	--------	------	--------
FR	2	0.12329	0.11437	0.13412	0.46941	0.46116	0.50051	110	12.14
	4	0.00522	0.01003	0.01281	0.01988	0.04044	0.04779	730	70.10
	6	0.33862	0.40668	0.38042	1.28923	1.63987	1.41968	990	82.62
PR	2	0.02180	0.03227	0.02589	0.08299	0.13013	0.09662	260	27.60
	4	0.00432	0.01088	0.01142	0.01645	0.04387	0.04262	1310	114.78
	6	0.00432	0.01088	0.01142	0.01645	0.04387	0.04262	1310	114.78
SCG	2	0.11838	0.11266	0.13063	0.45070	0.45428	0.48747	210	27.94
	4	0.00275	0.01136	0.01238	0.01047	0.04582	0.04621	1390	129.63
	6	0.00144	0.01382	0.01284	0.00547	0.05572	0.04791	4550	420.60
OSS	2	0.02178	0.03457	0.02529	0.08294	0.13941	0.09439	590	79.20
	4	0.02178	0.03457	0.02529	0.08294	0.13941	0.09439	590	79.20
	6	0.02178	0.03457	0.02529	0.08294	0.13941	0.09439	590	79.20
DFP	2	0.00771	0.01090	0.01117	0.02935	0.04395	0.04169	730	120.10
	4	0.00539	0.01149	0.01270	0.02052	0.04632	0.04740	1760	217.67
	6	0.00605	0.01171	0.01103	0.02305	0.04723	0.04115	2350	305.62
BFGS	2	0.00925	0.01136	0.01152	0.03522	0.04582	0.04230	760	741.77
	4	0.00466	0.01184	0.01251	0.01773	0.04773	0.04669	1220	199.42
	6	0.00540	0.01128	0.01161	0.02056	0.04548	0.04333	1420	228.50

Para que pudéssemos aplicar procedimentos de validação cruzada neste problema, foi inserido ruído uniformemente distribuído no intervalo [-0.15, 0.15], tanto nas amostras de treinamento, quanto nas amostras de validação e teste.

As Tabelas 9.10 e 9.11 apresentam os resultados para a capacidade de generalização dos algoritmos aplicados ao problema do sen(x)×cos($2x$).

Tabela 9.11: Capacidade de generalização para o critério PQ = 0.5. Problema sen(x)×cos($2x$).

	PQ	MSE			FVU			*EP*	*flops* (×10^6)
		MSE_{tr}	MSE_{val}	MSE_{tes}	FVU_{tr}	FVU_{val}	FVU_{tes}		
BP	0.5	0.17960	0.15870	0.17801	0.68381	0.63992	0.66431	1340	22.33
GRAD	0.5	-------	-------	-------	-------	-------	-------	-----	-------
FR	0.5	0.07492	0.07622	0.08607	0.28523	0.30733	0.32120	130	14.34
PR	0.5	0.06117	0.06272	0.07345	0.23291	0.25292	0.27409	170	17.20
SCG	0.5	0.02633	0.04172	0.03178	0.10025	0.16824	0.11861	80	10.32
OSS	0.5	-------	-------	-------	-------	-------	-------	-----	-------
DFP	0.5	0.02585	0.03988	0.03179	0.09842	0.16079	0.11864	100	17.70
BFGS	0.5	0.06966	0.06418	0.08228	0.26523	0.25877	0.30701	320	66.53

9.4.4 ESP

Divisão do conjunto amostral:

N_{tr}: 63

N_{val}: 6

N_{te}: 6

Tabela 9.12: Capacidade de generalização para o critério $s = 2$, 4 e 6. Problema ESP.

	s	MSE			FVU			EP	*flops*
		MSE_{tr}	MSE_{val}	MSE_{te}	FVU_{tr}	FVU_{val}	FVU_{te}		($\times 10^6$)
BP	2	0.30237	0.18766	0.41197	0.44816	0.28256	0.43215	230	14.04
	4	0.29884	0.19739	0.39279	0.44294	0.29680	0.41204	260	15.86
	6	0.30066	0.19515	0.39930	0.44563	0.29343	0.41886	270	16.48
GRAD	2	0.16010	0.12090	0.12814	0.23729	0.18178	0.13442	760	49.48
	4	0.12581	0.08941	0.07595	0.18647	0.13443	0.07967	2240	139.90
	6	0.08005	0.07265	0.05245	0.11865	0.10923	0.05502	5870	347.84
FR	2	0.31732	0.20193	0.41704	0.47033	0.30361	0.43747	30	13.59
	4	0.08135	0.06503	0.07027	0.12058	0.09778	0.07372	330	142.37
	6	0.05501	0.05134	0.04380	0.08153	0.07720	0.04594	420	172.16
PR	2	0.13243	0.06562	0.10275	0.19629	0.09867	0.10778	130	58.45
	4	0.06741	0.05499	0.06081	0.09991	0.08268	0.06379	950	393.98
	6	0.01727	0.05895	0.04480	0.02560	0.08863	0.04700	4800	1839.1
SCG	2	0.13031	0.08319	0.10389	0.19315	0.12508	0.10898	90	40.04
	4	0.05559	0.05057	0.05535	0.08240	0.07604	0.05806	300	123.46
	6	0.01472	0.03770	0.01908	0.02182	0.05669	0.02002	1050	395.09
OSS	2	0.12004	0.08021	0.08807	0.17792	0.12059	0.09238	350	190.61
	4	0.12004	0.08021	0.08807	0.17792	0.12059	0.09238	350	190.61
	6	0.02985	0.04227	0.03457	0.04425	0.06356	0.03626	4450	2171.2
DFP	2	0.11174	0.07277	0.10150	0.16562	0.10941	0.10647	630	1110.7
	4	0.06544	0.05911	0.04371	0.09699	0.08887	0.04585	2160	3780.2
	6	--------	--------	--------	--------	--------	--------	------	--------
BFGS	2	0.14428	0.09436	0.14054	0.21385	0.14188	0.14743	240	435.63
	4	0.04721	0.04834	0.03414	0.06997	0.07075	0.03881	3250	5905.6
	6	--------	--------	--------	--------	--------	--------	------	--------

A divisão deste conjunto amostral foi feita baseada no critério proposto por AMARI *et. al.* (1996), apresentado no Capítulo 6.

Tabela 9.13: Capacidade de generalização para o critério PQ = 0.5. Problema ESP.

	PQ	MSE			FVU			*EP*	*flops* ($\times 10^6$)
		MSE_{tr}	MSE_{val}	MSE_{tes}	FVU_{tr}	FVU_{val}	FVU_{tes}		
BP	0.5	-------	--------	-------	-------	--------	--------	----	------
GRAD	0.5	0.29857	0.19630	0.37408	0.44254	0.29514	0.39241	30	1.93
FR	0.5	0.29795	0.23408	0.35773	0.44162	0.35195	0.37525	80	35.99
PR	0.5	0.19899	0.10488	0.31194	0.29495	0.15770	0.32722	40	18.89
SCG	0.5	-------	--------	-------	-------	--------	--------	----	------
OSS	0.5	-------	--------	-------	-------	--------	--------	----	------
DFP	0.5	0.23097	0.15896	0.31266	0.34234	0.23901	0.32798	70	122.72
BFGS	0.5	-------	--------	-------	-------	--------	--------	----	------

9.4.5 SOJA

Divisão do conjunto amostral:

N_{tr}: 116

N_{val}: 14

N_{te}: 14

Tabela 9.14: Capacidade de generalização para o critério s = 2, 4 e 6. Problema SOJA.

	s	MSE			FVU			EP	*flops*
		MSE_{tr}	MSE_{val}	MSE_{te}	FVU_{tr}	FVU_{val}	FVU_{te}		($\times 10^6$)
BP	2	0.08739	0.08662	0.06488	0.63772	0.59313	0.74735	60	12.90
	4	0.08739	0.08662	0.06488	0.63772	0.59313	0.74735	60	12.90
	6	0.05476	0.06533	0.15324	0.39964	0.44733	1.76529	560	121.34
GRAD	2	0.11583	0.12159	0.07768	0.84533	0.83264	0.89484	170	42.20
	4	0.10184	0.08471	0.10656	0.74318	0.58009	1.22749	230	56.97
	6	0.05790	0.07882	0.14997	0.42254	0.53977	1.7276	1230	305.94
FR	2	0.06734	0.08206	0.09283	0.49145	0.56192	1.06941	20	41.63
	4	0.06408	0.06631	0.09478	0.46763	0.45408	1.09183	30	63.54
	6	0.00302	0.11282	0.23094	0.02204	0.77253	2.66034	450	971.73
PR	2	0.05753	0.07740	0.14399	0.41986	0.53004	1.65868	60	129.14
	4	0.05399	0.06720	0.17493	0.39402	0.46019	2.01509	70	150.96
	6	0.02431	0.11928	0.52467	0.17744	0.81677	6.04408	190	413.14
SCG	2	0.04709	0.05308	0.30947	0.34362	0.36348	3.56501	30	70.69
	4	0.02478	0.07667	0.43000	0.18084	0.52501	4.95348	60	140.82
	6	0.00810	0.16703	0.65537	0.05909	1.14374	7.54965	140	323.36
OSS	2	0.09673	0.09246	0.06824	0.70588	0.63316	0.78612	20	50.74
	4	0.07213	0.12710	0.27320	0.52642	0.87037	3.14718	40	104.43
	6	0.07009	0.06332	0.19461	0.51151	0.43360	2.24187	50	130.84
DFP	2	0.04992	0.09066	0.20595	0.36433	0.62081	2.37249	40	3917.00
	4	0.02897	0.08738	0.50772	0.21139	0.59836	5.84874	50	5178.37
	6	0.02897	0.08738	0.50772	0.21139	0.59936	5.84874	50	5178.37
BFGS	2	0.06183	0.05721	0.21518	0.45123	0.39179	2.47883	20	196.70
	4	0.04988	0.09061	0.20584	0.36440	0.62045	2.37116	40	3936.18
	6	0.01812	0.15315	0.48108	0.13226	1.04876	5.54191	110	11225.46

Tabela 9.15: Capacidade de generalização para o critério PQ = 0.5. Problema SOJA.

	PQ	MSE			FVU			*EP*	*flops* ($\times 10^6$)
		MSE_{tr}	MSE_{val}	MSE_{tes}	FVU_{tr}	FVU_{val}	FVU_{tes}		
BP	0.5	0.06368	0.11079	0.31610	0.46475	0.75867	3.64138	870	189.46
GRAD	0.5	0.03588	0.11758	0.11405	0.26183	0.80512	1.13381	2890	718.15
FR	0.5	0.05805	0.07305	0.13006	0.42360	0.50020	1.49828	30	65.92
PR	0.5	0.03184	0.16603	0.43486	0.23238	1.13693	5.00951	90	197.97
SCG	0.5	0.03076	0.08382	0.44568	0.22449	0.57400	5.13407	50	117.38
OSS	0.5	0.70562	1.56487	0.99932	5.14944	10.7157	11.5119	140	374.35
DFP	0.5	0.04875	0.09267	0.28475	0.35576	0.63454	3.28020	30	3221.07
BFGS	0.5	0.04839	0.08650	0.30971	0.35316	0.59232	3.56774	30	3235.99

9.4.6 IRIS

Divisão do conjunto amostral:

N_{tr}: 126

N_{val}: 12

N_{te}: 12

Tabela 9.16: Capacidade de generalização para o critério $s = 2$, 4 e 6. Problema IRIS.

	s	MSE			FVU			EP	*flops*
		MSE_{tr}	MSE_{val}	MSE_{te}	FVU_{tr}	FVU_{val}	FVU_{te}		($\times 10^6$)
BP	2	0.12372	0.00780	0.05362	0.04640	0.00293	0.02011	15780	1621.26
	4	0.12079	0.00738	0.04884	0.04530	0.00277	0.01832	23660	2290.13
	6	0.12047	0.00658	0.05004	0.04517	0.00247	0.01876	28100	2799.46
GRAD	2	0.13927	0.01039	0.10210	0.05223	0.00390	0.03829	660	73.71
	4	0.13723	0.01047	0.08129	0.05146	0.00393	0.03048	1420	155.82
	6	0.10841	0.02424	0.11668	0.04065	0.00909	0.04376	21690	2353.53
FR	2	0.14075	0.01931	0.05806	0.05278	0.00724	0.02177	140	107.23
	4	0.10888	0.00696	0.07773	0.04083	0.00261	0.02915	480	355.73
	6	0.00295	0.02255	0.07068	0.00111	0.00846	0.02650	830	619.25
PR	2	0.11854	0.00522	0.04771	0.04445	0.00196	0.01789	260	209.64
	4	0.11662	0.00523	0.04662	0.04373	0.00196	0.01748	450	346.73
	6	0.10562	0.01554	0.03530	0.03961	0.00583	0.01324	580	456.39
SCG	2	0.12680	0.00576	0.04799	0.04755	0.00216	0.01800	80	68.78
	4	0.11765	0.00324	0.04041	0.04412	0.00122	0.01515	220	173.30
	6	0.11652	0.00323	0.04111	0.04369	0.00121	0.01542	230	186.71
OSS	2	0.12446	0.00637	0.04946	0.04667	0.00239	0.01855	660	664.68
	4	0.11719	0.00585	0.04519	0.04394	0.00219	0.01695	1550	1485.67
	6	0.11600	0.00673	0.04199	0.04350	0.00252	0.01575	1640	1648.44
DFP	2	0.13828	0.01235	0.09607	0.05185	0.00463	0.03603	180	372.27
	4	0.11951	0.00714	0.04584	0.04482	0.00268	0.01719	1020	2090.76
	6	0.11934	0.00730	0.04475	0.04475	0.00274	0.01678	1070	2193.04
BFGS	2	0.13572	0.00964	0.06319	0.04976	0.00362	0.02370	240	542.85
	4	0.13269	0.00782	0.08072	0.05090	0.00293	0.03027	270	599.77
	6	0.11325	0.01401	0.05603	0.04247	0.00525	0.02101	1810	4022.31

Tabela 9.17: Capacidade de generalização para o critério PQ = 0.5. Problema IRIS.

	PQ	MSE			FVU			*EP*	*flops* (×10^6)
		MSE_{tr}	MSE_{val}	MSE_{tes}	FVU_{tr}	FVU_{val}	FVU_{tes}		
BP	0.5	-------	-------	-------	-------	-------	-------	------	-------
GRAD	0.5	0.65968	0.94557	1.07244	0.24738	0.35459	0.40216	40	4.41
FR	0.5	0.58239	0.32184	0.36670	0.21840	0.12069	0.13751	30	26.23
PR	0.5	0.22833	0.10365	0.27611	0.08562	0.03887	0.10354	30	26.43
SCG	0.5	0.19102	0.03893	0.19962	0.07163	0.01460	0.07486	30	27.88
OSS	0.5	0.39231	0.49700	0.68612	0.14712	0.18638	0.25729	30	31.91
DFP	0.5	0.30226	0.31778	0.52996	0.11335	0.11917	0.19873	30	58.75
BFGS	0.5	0.30336	0.32917	0.53841	0.11376	0.12344	0.20190	30	64.56

9.4.7 GLASS

Divisão do conjunto amostral:

N_{tr}: 118

N_{val}: 48

N_{te}: 48

Tabela 9.18: Capacidade de generalização para o critério $s = 2$, 4 e 6. Problema GLASS.

	s	MSE			FVU			EP	*flops*
		MSE_{tr}	MSE_{val}	MSE_{te}	FVU_{tr}	FVU_{val}	FVU_{te}		($\times 10^6$)
BP	2	1.83201	1.65293	1.57631	0.76520	0.71747	0.68422	30	7.17
	4	1.72320	1.62628	1.52358	0.71976	0.70591	0.66133	50	10.61
	6	1.72320	1.62628	1.52358	0.71976	0.70591	0.66133	50	10.61
GRAD	2	1.63608	1.65609	1.46894	0.68337	0.71885	0.63761	40	9.47
	4	1.04312	1.57713	1.25956	0.43570	0.68457	0.54673	820	188.71
	6	--------	--------	--------	--------	--------	--------	------	--------
FR	2	0.80625	1.96817	1.58372	0.33676	0.85431	0.68743	130	198.07
	4	0.39355	1.90151	2.47399	0.16438	0.82537	1.07387	430	631.46
	6	0.39355	1.90151	2.47399	0.16438	0.82537	1.07387	430	631.46
PR	2	1.32120	1.68135	1.34222	0.55185	0.72981	0.58261	40	67.57
	4	0.43120	2.00269	2.04357	0.18011	0.86929	0.88703	420	653.23
	6	0.43120	2.00269	2.04357	0.18011	0.86929	0.88703	420	653.23
SCG	2	0.17484	3.15870	2.32111	0.07303	1.37107	1.00751	1680	2561.64
	4	0.08116	3.38325	4.29195	0.03390	1.46854	1.86297	4990	7530.34
	6	0.08116	3.38325	4.29195	0.03390	1.46854	1.86297	4990	7530.34
OSS	2	1.68219	1.69053	1.52019	0.70263	0.73377	0.65986	20	41.01
	4	0.87257	1.61428	1.43276	0.36446	0.70070	0.62191	360	746.27
	6	0.77479	1.64684	1.59313	0.32362	0.71483	0.69152	440	904.81
DFP	2	1.83201	1.65293	1.57631	0.76520	0.71747	0.68422	10	171.72
	4	1.14208	1.55740	1.35144	0.47703	0.67601	0.58661	440	8669.63
	6	1.14208	1.55740	1.35144	0.47703	0.67601	0.58661	440	8669.63
BFGS	2	1.83198	1.65292	1.57630	0.76519	0.71747	0.68421	30	175.27
	4	0.60927	2.07322	1.79416	0.25448	0.89990	0.77878	280	5502.96
	6	0.41644	2.26906	1.76495	0.17394	0.98491	0.76610	1220	25408.5

Tabela 9.19: Capacidade de generalização para o critério PQ = 0.5. Problema GLASS.

	PQ	MSE			FVU			*EP*	*flops* ($\times 10^6$)
		MSE_{tr}	MSE_{val}	MSE_{tes}	FVU_{tr}	FVU_{val}	FVU_{tes}		
BP	0.5	1.58542	2.49193	1.84903	0.66221	1.08165	0.80259	1380	275.91
GRAD	0.5	0.33650	2.32994	1.78269	0.14055	1.01134	0.77380	43089	7900.51
FR	0.5	0.91067	2.31361	1.97953	0.38037	1.00425	0.85924	110	170.57
PR	0.5	0.37616	2.31726	2.14649	0.15712	1.00583	0.93171	680	1040.57
SCG	0.5	0.25189	2.40206	1.82690	0.10521	1.04264	0.79299	570	884.16
OSS	0.5	0.33095	2.34534	2.70962	0.13823	1.01802	1.17614	3890	7349.75
DFP	0.5	1.68567	2.14927	1.92107	0.70408	0.93298	0.83386	250	4962.96
BFGS	0.5	1.27689	2.03501	1.41634	0.53334	0.88332	0.61478	160	3134.70

9.4.8 ECOLI

Divisão do conjunto amostral:

N_{tr}: 224

N_{val}: 56

N_{te}: 56

Tabela 9.20: Capacidade de generalização para o critério s = 2, 4 e 6. Problema ECOLI.

	s	MSE			FVU			EP	*flops* ($\times 10^6$)
		MSE_{tr}	MSE_{val}	MSE_{te}	FVU_{tr}	FVU_{val}	FVU_{te}		
BP	2	0.53082	0.58753	0.67932	0.26778	0.28121	0.30470	1100	368.19
	4	0.50215	0.51938	0.57536	0.25329	0.24859	0.25807	1460	493.55
	6	0.48534	0.51836	0.56013	0.24481	0.24811	0.25124	1650	548.95
GRAD	2	0.81479	0.78813	0.98327	0.41098	0.37723	0.44103	140	50.07
	4	0.46389	0.50769	0.58326	0.23399	0.24300	0.26161	2140	788.01
	6	0.43770	0.49756	0.56299	0.22078	0.23815	0.25252	2550	930.40
FR	2	0.68037	0.68275	0.91642	0.34318	0.32679	0.41105	100	269.75
	4	0.42746	0.51481	0.54139	0.21561	0.24640	0.24283	270	703.13
	6	0.42746	0.51481	0.54139	0.21561	0.24640	0.24283	270	703.13
PR	2	0.45459	0.54873	0.57700	0.22930	0.26264	0.25881	150	404.63
	4	0.44778	0.47420	0.59202	0.22586	0.22697	0.26554	180	496.61
	6	0.41487	0.56777	0.59610	0.20926	0.27175	0.26737	220	580.62
SCG	2	0.47419	0.47654	0.59744	0.23919	0.22809	0.26797	90	265.13
	4	0.41998	0.60122	0.58256	0.21184	0.28777	0.26130	100	283.25
	6	0.43091	0.48196	0.56495	0.21735	0.23068	0.25340	120	344.44
OSS	2	0.46638	0.52056	0.57191	0.23524	0.24916	0.25652	450	1503.63
	4	0.47498	0.53332	0.59037	0.23958	0.25527	0.26480	460	1590.15
	6	0.47498	0.53332	0.59037	0.23958	0.25527	0.26480	460	1590.15
DFP	2	0.48037	0.59970	0.63661	0.24230	0.28703	0.28554	140	1907.66
	4	0.46529	0.51125	0.58543	0.23469	0.24470	0.26258	170	2380.95
	6	0.46529	0.51125	0.58543	0.23469	0.24470	0.26258	170	2380.95
BFGS	2	0.44210	0.52418	0.56679	0.22210	0.25089	0.25422	170	2519.09
	4	0.44210	0.52418	0.56679	0.22210	0.25089	0.25422	170	2519.09
	6	0.41548	0.60041	0.55158	0.20957	0.28737	0.24740	270	4073.86

Tabela 9.21: Capacidade de generalização para o critério PQ = 0.5. Problema ECOLI.

	PQ	MSE			FVU			*EP*	*flops* ($\times 10^6$)
		MSE_{tr}	MSE_{val}	MSE_{tes}	FVU_{tr}	FVU_{val}	FVU_{tes}		
BP	0.5	0.65761	0.70936	0.74208	0.33170	0.33952	0.33285	1750	578.17
GRAD	0.5	0.26879	0.78770	0.58293	0.13558	0.37702	0.26147	17890	6046.57
FR	0.5	0.37224	0.72822	0.52557	0.18776	0.34855	0.23573	350	936.22
PR	0.5	0.36675	0.73178	0.58880	0.18499	0.35026	0.26410	260	703.32
SCG	0.5	0.36045	0.82725	0.62972	0.18181	0.39595	0.28245	140	389.45
OSS	0.5	--------	--------	--------	--------	--------	--------	-----	--------
DFP	0.5	1.03698	1.03410	1.17037	0.52306	0.49495	0.52495	140	1916.18
BFGS	0.5	0.25812	0.80169	0.63013	0.13020	0.38372	0.28264	2650	39585.1

9.4.9 Estatísticas e Comentários

As comparações apresentadas na Figura 9.2 baseiam-se apenas no critério de parada CV. O critério PQ foi apresentado de forma ilustrativa, devido ao fato de que alguns algoritmos não foram capazes de satisfazê-lo para todos os problemas abordados, indicando que, nestes casos, o quociente entre a *perda de generalização* e o *progresso de treinamento* permanece inferior ao limiar 0.5 pré-estabelecido. Isto quer dizer que o valor da perda de generalização está pequeno em relação ao progresso de treinamento. Outros valores para o limiar *PQ* poderiam ter sido utilizados, fornecendo resultados superiores aos apresentados.

A Figura 9.2(c) demonstra que houve um certo equilíbrio entre o desempenho dos diversos métodos testados. Um dos fatores que podem ter levado a esta igualdade foi uma má escolha (divisão) dos conjuntos utilizados no treinamento, validação e teste. A utilização de estratégias melhor elaboradas na partição do conjunto amostral constitui uma boa ferramenta para evitar este tipo de "empate técnico". É importante mencionar que, embora tenha havido um empate técnico entre os métodos de primeira e segunda ordem, os últimos apresentam taxas de convergência superiores, justificando assim a sua preferência.

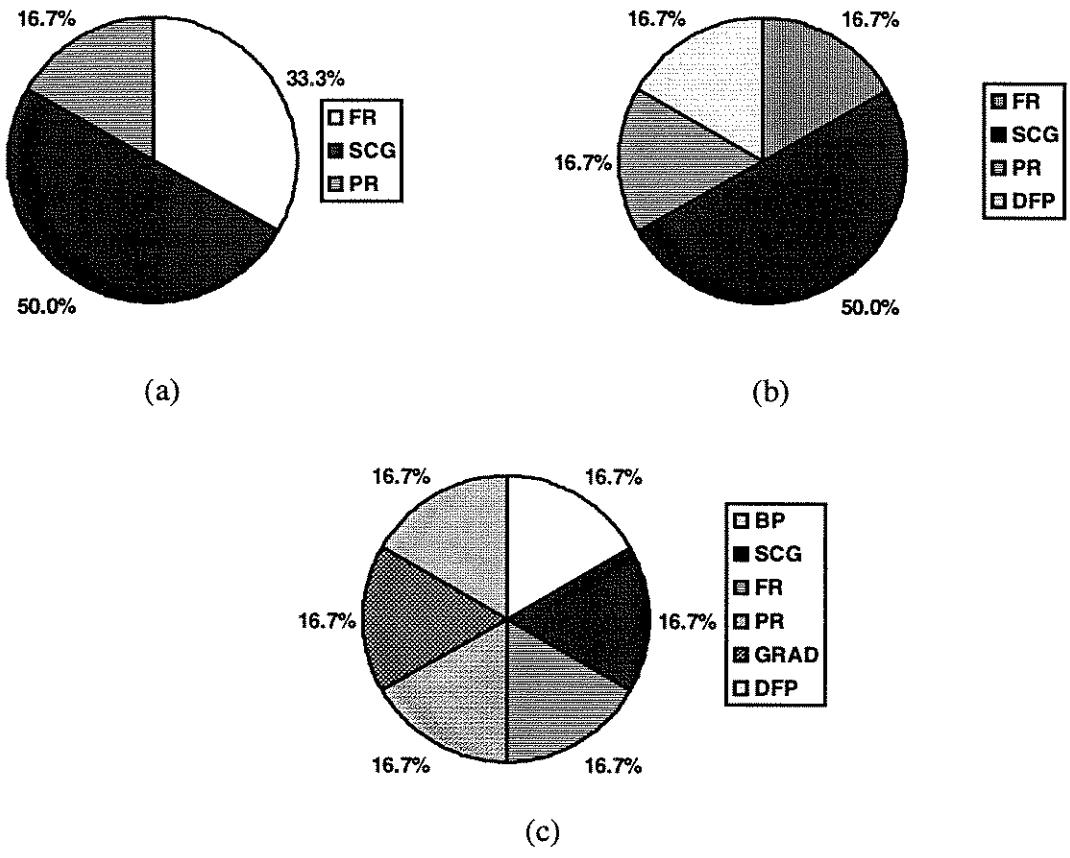


Figura 9.2: Comparação geral da capacidade de generalização dos algoritmos em relação a seis dos oito problemas abordados. Os valores percentuais indicam o quanto cada um dos algoritmos mostrou-se superior aos outros para todos os problemas. (a) Menor erro de treinamento. (b) Menor erro de validação. (c) Menor erro de teste.

9.5 Algoritmos de Inicialização

Foram testados os seguintes métodos de inicialização para redes MLP:

- método de BOERS;
- método de WIDROW;
- método de KIM;
- método OLS e
- o método proposto chamado INIT.

Estes métodos foram testados apenas para os conjuntos de dados cujos algoritmos foram capazes de convergir. Para testá-los utilizou-se apenas o algoritmo do gradiente conjugado escalonado modificado de MOLLER (1993) SCGM, devido a superioridade demonstrada pelo mesmo em relação a velocidade de convergência. Os parâmetros da rede (número de unidades intermediárias e critério de parada) foram os mesmos utilizados anteriormente. As Tabelas 9.22 e 9.23 apresentam os valores mínimo, máximo, médio e o desvio padrão δ do número de épocas necessárias para convergência. A média foi calculada considerando-se 10 condições iniciais distintas para cada algoritmo.

Devido ao critério de parada e à complexidade do problema COD/DEC, nem sempre o algoritmo é capaz de convergir para a solução desejada podendo ficar preso em um mínimo local, onde este é considerado quando a norma euclidiana do vetor gradiente é menor do que um limiar $\zeta = 1\times10^{-4}$. A Tabela 9.22 apresenta os resultados para este problema e quantas vezes o algoritmo ficou preso em um mínimo local em função do número total de simulações. Neste caso, foram utilizadas 20 condições iniciais distintas.

Tabela 9.22: Comparação de desempenho dos métodos de inicialização apresentados no Capítulo 8. Problema codificador/decodificador.

Problema	Método	Máximo	Mínimo	Média	δ	Convergências
COD/DEC	BOERS	40	12	22.21	7.94	14/20
	WIDROW	32	7	12.36	6.81	14/20
	KIM	298	10	38.17	69.07	18/20
	OLS	276	7	97.78	102.03	9/20
	INIT	16	6	9.44	2.53	16/20

Tabela 9.23: Comparação de desempenho dos métodos de inicialização apresentados no Capítulo 8.

Problema	Método	Máximo	Mínimo	Média	δ
XOR	BOERS	129	6	42.90	48.32
	WIDROW	93	8	20.50	25.92
	KIM	84	6	32.50	25.91
	OLS	166	5	46.70	46.50
	INIT	47	8	18.60	12.76
$\sin(x).\cos(2x)$	BOERS	187	79	135.10	35.39
	WIDROW	243	123	178.60	41.27
	KIM	231	95	164.50	44.89
	OLS	133	39	91.40	33.06
	INIT	449	181	254.80	80.69
ESP	BOERS	1618	340	883.40	467.52
	WIDROW	2280	236	825.40	639.74
	KIM	1763	425	715.40	397.82
	OLS	2052	35	545.42	586.62
	INIT	762	383	479.20	118.19
SOJA	BOERS	174	136	158.60	13.14
	WIDROW	464	176	266.30	79.27
	KIM	280	177	219.40	28.34
	OLS	5000	4303	4922.40	219.05
	INIT	236	136	188.00	29.30
IRIS	BOERS	1568	735	1102.40	281.33
	WIDROW	1240	676	918.60	183.72
	KIM	2063	767	1294.80	438.34
	OLS	5000	2075	4140.20	1384.74
	INIT	1407	662	869.80	216.73

9.5.1 Estatísticas e Comentários

As Tabelas 9.22 e 9.23 mostram que em 66,67% dos casos o método proposto é em média superior aos outros. A Figura 9.3(a) indica que em 42.9% dos casos o método dos quadrados mínimos ortogonais (OLS) e também o método INIT necessitaram de um menor número de épocas para convergir, o método BOERS apresentou os melhores resultados em 16,67% dos problemas tratados.

É importante ressaltar que, mesmo o algoritmo OLS apresentando bons resultados para os problemas artificiais, seus resultados são praticamente os piores quando se trata de problemas de mundo real.

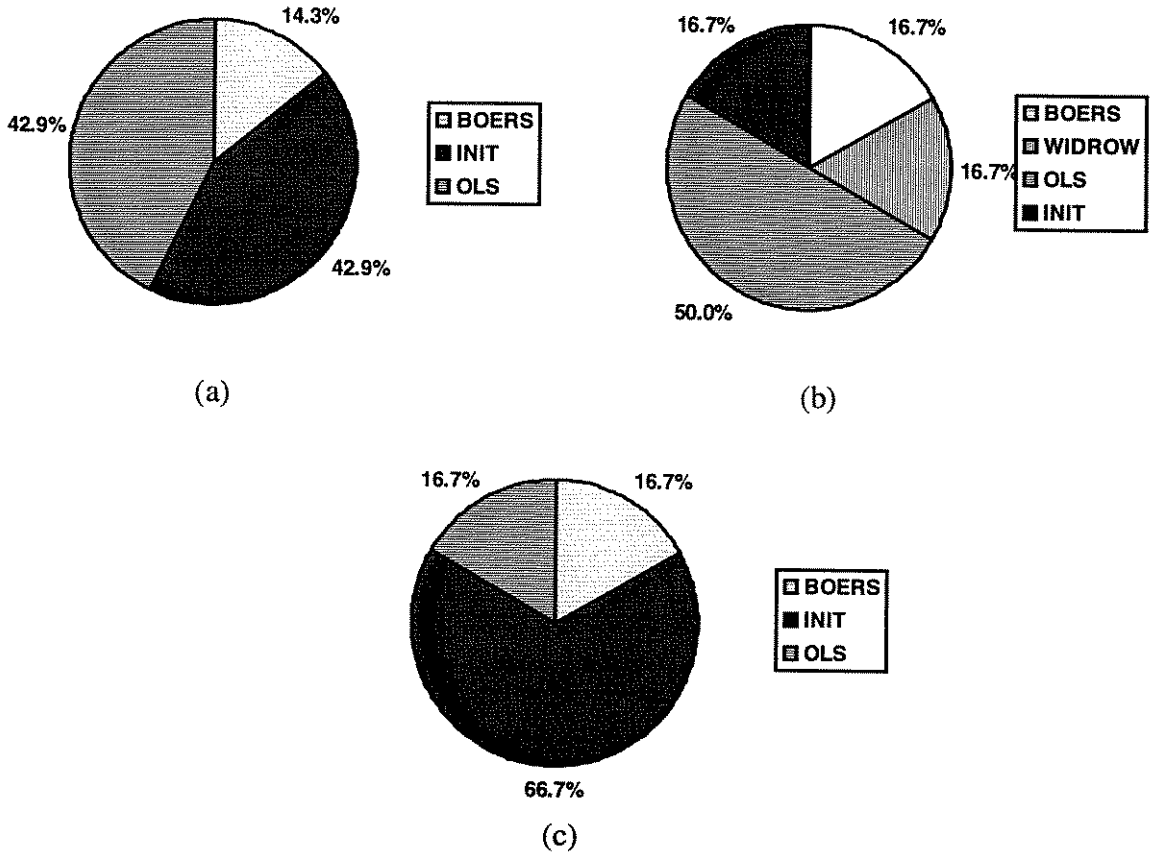


Figura 9.3: Comparação de desempenho dos diferentes métodos de inicialização em relação a seis dos oito problemas abordados. Os valores percentuais indicam o quanto cada um dos algoritmos mostrou-se superior aos outros para todos os problemas. (a) Menor número mínimo de épocas. (b) Menor número máximo de épocas. (c) Menor média.

9.6 Comparação com o TOOLBOX do MATLAB®

Para simples verificação do desempenho dos algoritmos implementados foram feitas comparações com alguns algoritmos do TOOLBOX 2.0 de Redes Neurais do MATLAB®. Estes algoritmos são muito utilizados pela comunidade acadêmica em problemas de aplicação prática e também em atividades didáticas.

Algoritmos comparados:

- Algoritmo padrão
- Método do gradiente

Problemas abordados:

- XOR com 2 entradas

  Arquitetura: [2,35,1]

  SSE = 0.01
- sen(x)×cos(2x)

  Arquitetura: [1,10,1]

  SSE = 0.1

Parâmetros:

α: 0.001 t_a: 1.2

β: 0.95 t_r: 0.618

Obs.: Os parâmetros t_a e t_r são utilizados nos procedimentos de busca simples do passo α.

Tabela 9.24: Comparação de desempenho para os problemas XOR e sen(x).cos(2x). Algoritmo padrão e método do gradiente.

	ÉPOCAS	*flops* × 10^6	Problema
BP	861	5.34	XOR
TBP	682	3.05	
GRAD	50	0.49	
TGRAD	43	0.19	
BP	15257	195.86	sen(x)×cos(2x)
TBP	50000	314.95	
GRAD	7720	96.31	
TGRAD	5983	36.42	

Sejam TBP e TGRAD os algoritmos do TOOLBOX 2.0 do MATLAB®. A Tabela 9.24 apresenta os resultados dos algoritmos implementados neste trabalho quando comparados com seus equivalentes presentes no TOOLBOX 2.0 do MATLAB®. Os valores apresentados são a média tomada a partir de 10 simulações.

Como pode ser visto pela Tabela 9.24, os algoritmos do TOOLBOX são, em geral, um pouco superiores aos algoritmos implementados quando está sendo comparada a quantidade de épocas para convergência, entretanto o algoritmo padrão implementado foi muito superior para o problema sen(x)×cos($2x$).

Os algoritmos implementados demandam maior esforço computacional pois efetuam um maior número de cálculos por iteração, como por exemplo, o vetor gradiente é construído a cada passo e sua norma também é calculada, enquanto os algoritmos contidos no TOOLBOX não se preocupam com este tipo de análise, a qual mostrou-se necessária para o sucesso na convergência do problema sen(x)×cos($2x$). Outra característica dos códigos implementados é a utilização da função de ativação do tipo tangente hiperbólica (tanh), enquanto os algoritmos do TOOLBOX trabalham com a função logística (logsig) que demanda um esforço computacional inferior para seu cálculo.

9.7 Algoritmos Construtivos

Como última análise das redes *feedforward*, serão comparados o desempenho de três métodos construtivos estudados no Capítulo 7:

- treinamento por busca de projeção – PPL;
- A*; e
- *cascade correlation* com múltipla escolha da função de ativação – CASCOR.

A ordem dos polinômios de hermite utilizados pelo algoritmo PPL foi 5 para os problemas XOR e sen(x)×cos($2x$) e 10 para os demais. O algoritmo A* possui funções de ativação do tipo tangente hiperbólica para os neurônios de saída, enquanto o PPL e o CASCOR possuem saídas lineares. As camadas intermediárias do CASCOR são treinadas com o algoritmo quickprop e a saída linear é calculada diretamente pelo método dos quadrados mínimos. O conjunto fechado de funções de ativação que podem ser utilizadas na camada intermediária do algoritmo CASCOR é composto pelas seguintes funções: tangente hiperbólica, sen(x), cos(x), gaussiana e inverso da gaussiana (ver Seção 7.3.4.2). Maiores detalhes sobre os algoritmos construtivos testados podem ser encontrados no Capítulo 7.

Para avaliar o comportamento de cada um dos algoritmos, utilizou-se sete dos oito problemas sugeridos, excluindo-se apenas o COD/DEC que é caracterizado pela definição prévia da arquitetura da rede. O resultado a ser apresentado será o número de unidades e camadas intermediárias utilizadas por cada um dos algoritmos para solucionar os problemas, com critério de parada definido como $MSE = 0.01$ para todos os casos. O número de unidades intermediárias a serem introduzidas no PPL ficou limitado a 20 devido a problemas numéricos e a 50 para os outros algoritmos (A* e CASCOR). Nestes casos, será apresentado o valor do erro quadrático médio (*MSE*) ao final do processo de treinamento.

A Tabela 9.25 apresenta os resultados obtidos pelos métodos construtivos. Pelos resultados apresentados nesta tabela percebemos que os métodos A* e CASCOR não necessitam de nenhuma unidade intermediária para solucionar o problema ESP.

Tabela 9.25: Arquitetura resultante dos métodos construtivos.

	Arquitetura resultante		
	PPL	A*	CASCOR
XOR	[2, **1**, 1]	[2, **2**, 1]	[2, **1**, 1]
$\sin(x).\cos(2x)$	[1, **3**, 1]	[1, **6**, 1]	[1, **2**, 1]
ESP	[2, **5**, 5]	[2, **0**, 5]	[2, **0**, 5]
SOJA	[36, **4**, 1]	[36, **3**, 1]	[36, **3**, 1]
IRIS	[4, **20**, 3], $MSE = 0.0233$	[4, **4**, 3]	[4, **9**, 3]
ECOLI	[7, **20**, 3], $MSE = 0.2444$	[7, **50**, 3], $MSE = 0.0484$	[7, **48**, 3]
GLASS	[9, **20**, 3], $MSE = 0.0699$	[9, **24**, 3]	[9, **28**, 3]

9.7.1 Comentários sobre os Algoritmos Construtivos

Os resultados apresentados na Tabela 9.25 mostram que os algoritmos A* e CASCOR nem sempre necessitam adicionar neurônios ou unidades intermediárias para resolverem os problemas propostos.

Verificou-se que, embora o A* permita a criação de redes com múltiplas camadas e múltiplos neurônios por camada, sempre uma rede com uma única camada intermediária foi escolhida. Este fato reforça o teorema da aproximação universal, aplicável às redes neurais artificiais, apresentado no Capítulo 2.

Outro aspecto importante, é o fato de que o CASCOR sempre escolheu (para os problemas abordados) como função de ativação as funções seno ou coseno.

O algoritmo PPL apresentou resultados muito pobres para a maior parte dos problemas cuja rede possui mais de uma saída. Esta é uma das dificuldades do método e vários estudos têm sido feitos no sentido de amenizar os seus efeitos.

9.8 Mapas Auto-Organizáveis de Kohonen

A partir de uma análise estatística do conjunto amostral, desenvolveu-se um algoritmo simplificado cujo objetivo principal é a eliminação das unidades de saída excedentes de um mapa auto-organizável unidimensional, permitindo assim a determinação de uma arquitetura mínima capaz de representar o conjunto de dados. O algoritmo apresentado como resultado pode ser facilmente aplicado a estruturas de dimensões mais elevadas, como os mapas bidimensionais. Outra aplicação dos mapas auto-organizáveis de Kohonen (SOM) foi a sua utilização na geração das direções de busca dos modelos de aprendizagem baseados em busca de projeção (ver Capítulo 8).

A partir desta seção apresentaremos os resultados referentes as variações introduzidas nas redes auto-organizadas de Kohonen (SOM). Os resultados das simulações com o método de poda para os mapas de Kohonen (PSOM) e com o novo método de inicialização das direções de busca dos modelos de aprendizagem baseados em busca de projeção serão fornecidas.

9.8.1 Método de Poda para as Redes de Kohonen

Nesta seção apresentaremos os resultados computacionais do método de poda, proposto no Capítulo 8, para os mapas auto-organizáveis de Kohonen. Será feita uma comparação de desempenho do algoritmo proposto com o algoritmo padrão desenvolvido por KOHONEN (1982). O comportamento dos algoritmos será avaliado em relação a quatro problemas, onde dois são artificiais e dois naturais. Os problemas naturais em questão são o problema de classificação de tipos de vidro (GLASS) e classificação de tipos de íris (IRIS) vistos na Seção 9.1. A legenda utilizada nesta seção pode ser vista abaixo.

Legenda:

m	quantidade inicial de saídas da rede
m_min	quantidade mínima permitida de saídas da rede
N_c	raio de vizinhança inicial
α	taxa de aprendizagem inicial
α_min	taxa de aprendizagem mínima (final)
χ	valor do decrescimento geométrico da taxa de aprendizagem

max_ep	número máximo de épocas permitido
ξ	valor percentual utilizado pelo procedimento de poda
Inic.	intervalo de inicialização do conjunto de pesos

Para os quatro problemas abordados, foram tomadas trinta condições iniciais distintas e os resultados apresentados referem-se ao valor médio obtido.

9.8.1.1 Agrupamento de Padrões Disjuntos

Como um primeiro e mais simples conjunto de dados, tomou-se quatro classes compostas de dez elementos cada uma. As classes são disjuntas e podem ser vistas na figura abaixo.

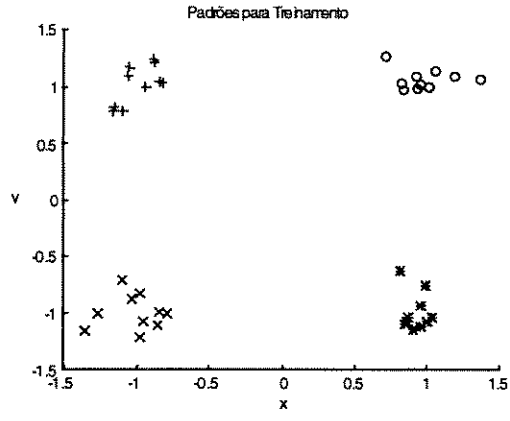


Figura 9.4: Amostras utilizadas no treinamento do mapa de Kohonen.

Para o problema apresentado na Figura 9.4, o método de poda proposto para a rede de Kohonen foi capaz de determinar a arquitetura mínima, ou seja, com quatro unidades de saída e 100% de classificação correta para os dados. O mesmo resultado foi obtido para o algoritmo original.

9.8.1.2 Agrupamento de Padrões Não Disjuntos

Este problema é muito semelhante ao problema anterior, a diferença é que agora existem cinco classes que não são disjuntas, e cada classe contém cinqüenta amostras.

A Figura 9.5 apresenta as classes a serem agrupadas. Esta figura permite observar que existem elementos de algumas classes que estão "dentro" de outras. A seguir são apresentados os parâmetros da rede utilizados para solucionar este problema.

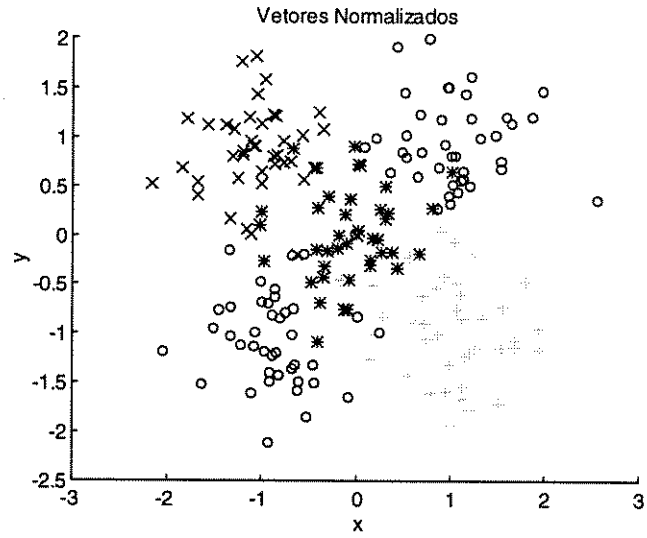


Figura 9.5: Classes a serem agrupadas.

Parâmetros da rede SOM:

m:	$\in$ [6; 10]	χ:	0.8
N_c:	*floor*(m/2)	*max_ep*:	100
α:	0.9	*Inic.* :	[-0.1; +0.1]
α_*min*:	0.001		

Parâmetros da rede PSOM:

m:	10	χ:	0.8
m_min:	3	*max_ep*:	100
N_c:	5	ξ:	2%
α:	0.9	*Inic.*:	[-0.1; +0.1]
α_*min*:	5		

Para os parâmetros apresentados acima das redes SOM e PSOM, os resultados da Tabela 9.26 foram obtidos. Esta tabela traz as seguintes informações: para os parâmetros apresentados acima, o algoritmo PSOM em 90% dos casos, atingiu uma arquitetura mínima com 7 unidades de saída e um percentual médio de classificação correta de 79.5%.

Tabela 9.26: Comparação de desempenho entre o algoritmo original (SOM) e algoritmo proposto com o método de poda (PSOM) para o problema de agrupamento de classes não disjuntas.

	Classificação Correta (%)				
m	6	7	8	9	10
SOM	79.0	74.0	80.5	76.5	82.5
PSOM	----------	79.5	----------	----------	83.0
Ocorrência (%)	0	90.0	0	0	10.0

9.8.1.3 Classificação de Tipos de Vidro (GLASS)

Para este problema tomou-se os seguintes parâmetros:

Parâmetros da rede SOM:

m:	$\in [5; 9]$	χ:	0.8
N_c:	*floor*(*m*/2)	*max_ep*:	100
α:	0.9	*Inic.*:	[-0.1; +0.1]
α_*min*:	0.001		

Parâmetros da rede PSOM:

m:	15	χ:	0.8
m_min:	2	*max_ep*:	100
N_c:	7	ξ:	0.5%
α:	0.9	*Inic.*:	[-0.1; +0.1]
α_*min*:	5		

Tabela 9.27: Comparação de desempenho entre o algoritmo original (SOM) e algoritmo proposto com o método de poda (PSOM) para o problema de classificação de tipos de vidro.

	Classificação Correta (%)				
m	5	6	7	8	9
SOM	88.31	92.99	84.58	89.72	89.72
PSOM	88.79	----------	91.59	----------	----------
Ocorrência (%)	90	0	10	0	0

9.8.1.4 Classificação de Tipos de Íris (IRIS)

Para este problema tomou-se os seguintes parâmetros:

Parâmetros da rede SOM:

m:	$\in [3; 7]$	χ:	0.8
N_c:	$Floor(m/2)$	max_ep:	100
α:	0.9	$Inic.$:	[-0.1; +0.1]
α_min:	0.001		

Parâmetros da rede PSOM:

m:	15	χ:	0.8
m_min:	2	max_ep:	100
N_c:	7	ξ:	15%
α:	0.9	$Inic.$:	[-0.1; +0.1]
α_min:	5		

Tabela 9.28: Comparação de desempenho entre o algoritmo original (SOM) e algoritmo proposto com o método de poda (PSOM) para o problema de classificação de tipos de íris.

	Classificação Correta (%)				
m	3	4	5	6	7
SOM	80.66	78.67	88.67	90.00	88.67
PSOM	82.00	----------	90.67	----------	----------
Ocorrência (%)	10	0	90	0	0

9.8.1.5 Mapeamentos

Para verificar o comportamento das arquiteturas, descreveremos o mapeamento realizado por algumas estruturas em alguns dos problemas abordados.

Agrupamento de Classes Não Disjuntas

Como pode ser visto pela Tabela 9.29, para os dois métodos, as unidades de saída mapearam a mesma quantidade de vetores de entrada, a única diferença foi a classe mapeada em cada unidade. No algoritmo original, a classe (2) composta pelos elementos que estão localizados na região central do espaço de distribuição misturaram-se aos demais, causando erros de representação maiores que o algoritmo PSOM.

Tabela 9.29: Mapeamentos efetuados pelos mapas auto organizáveis de Kohonen. Problema das classes não disjuntas. *QAM* representa a quantidade de elementos mapeados pelo neurônio indicado.

	Neur.	1	2	3	4	5	6	7
SOM	*QAM*	43	18	37	16	31	16	39
	Classe	(4)	(4)	(5)	(5)	(3)	(3)	(1)
	Neur.	1	2	3	4	5	6	7
PSOM	*QAM*	43	18	37	16	31	16	39
	Classe	(4)	(2)	(5)	(5)	(3)	(2)	(1)

Agrupamento de Tipos de Vidro (GLASS)

O agrupamento dos diversos tipos de vidro para uma arquitetura com sete unidades de saída é apresentado na Tabela 9.30.

Para este problema, e no caso de uma arquitetura com sete unidades de saída, é possível perceber algumas diferenças entre os resultados apresentados pelos dois mapas. A Seção 9.1 apresenta detalhes sobre a distribuição de cada uma das classes. No algoritmo original, a classe (4) que contém 13 elementos difundiu-se nas demais; no algoritmo PSOM, a classe (5) que contém apenas 9 elementos difundiu-se nas outras. Para este problema foi possível perceber que classes como a (1), (2) e (3) são facilmente separáveis das demais, a dificuldade reside nas classes que são pouco representativas, como as classes (4) e (5). Embora a classe (3) possua apenas 17 elementos, os algoritmos ainda são capazes de distinguí-la.

Tabela 9.30: Mapeamentos efetuados pelos mapas auto organizáveis de Kohonen. Problema GLASS. *QAM* representa a quantidade de elementos mapeados pelo neurônio indicado.

	Neur.	1	2	3	4	5	6	7
SOM	*QAM*	65	18	16	27	19	12	57
	Classe	(1)	(3)	(5)	(6)	(2)	(2)	(2)
	Neur.	1	2	3	4	5	6	7
PSOM	*QAM*	55	16	17	56	17	20	33
	Classe	(1)	(1)	(3)	(2)	(2)	(4)	(6)

Agrupamentos dos Tipos de Íris (IRIS)

Tabela 9.31: Mapeamentos efetuados pelos mapas auto organizáveis de Kohonen. Problema IRIS. *QAM* representa a quantidade de elementos mapeados pelo neurônio indicado.

SOM	*Neur.*	1	2	3	4	5
	QAM	36	26	35	3	50
	Classe	(3)	(3)	(2)	(2)	(1)
PSOM	*Neur.*	1	2	3	4	7
	QAM	50	3	35	26	36
	Classe	(1)	(2)	(2)	(3)	(3)

A Tabela 9.31 apresenta um resultado que já era conhecido, ou seja, a classe (1) é linearmente separável das demais. Ambos os algoritmos são capazes de identificá-la facilmente. A diferença entre os resultados apresentados pelos dois métodos é mínima.

9.8.1.6 Utilização do PSOM para o Projeto de Receptores FH-CDMA

Uma das possibilidades do serviço de comunicações digitais é a investigação da técnica de modulação de espalhamento espectral utilizando chaveamento por deslocamento em freqüência. Um sistema FH-CDMA pode ser representado por matrizes de transmissão e recepção. A regra de decisão que minimiza a probabilidade de erro de mensagem no receptor é conhecida por lógica majoritária.

A partir de um diagrama de blocos para o receptor de lógica majoritária, JÚNIOR (1998) desenvolveu uma análise utilizando o PSOM com o objetivo de mostrar que suas operações no cálculo de valores intermediários são equivalentes aos efetuados em um outro modelo de rede construtiva por ele proposto. Observando a matriz de pesos sinápticos, resultante do treinamento do PSOM, verificou-se que as colunas ficaram responsáveis pela classificação das mensagens de entrada. Concluiu-se que o modelo PSOM apresenta resultados compatíveis com outras arquiteturas que utilizam treinamento supervisionado.

9.8.1.7 Comentários sobre o Método PSOM

Os resultados apresentados mostram que o método proposto é capaz de definir uma estrutura reduzida que representa os dados com melhor desempenho do que o algoritmo original. Isso pode ser explicado pelo fato de que o método proposto é reinicializado cada vez que uma unidade de saída é retirada, ou seja, a taxa de aprendizagem retoma o valor inicial e

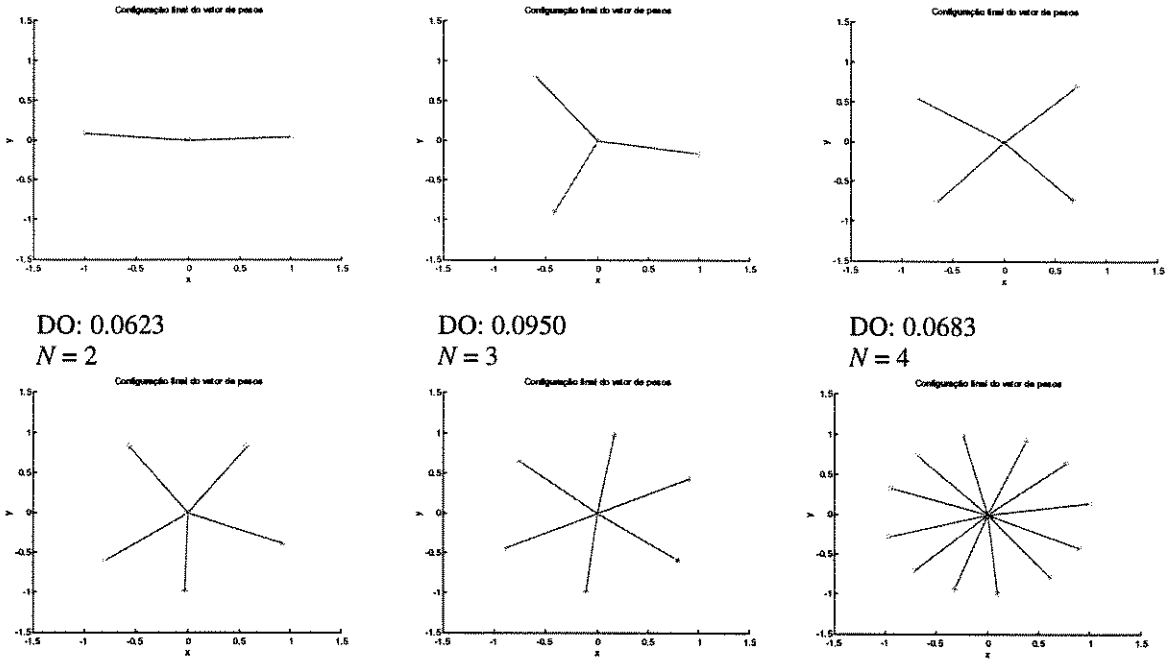
o raio de vizinhança a metade do número atual de saídas da rede. Este procedimento permite que o conjunto de pesos definido ao final do procedimento de ajuste anterior seja utilizado como a nova condição inicial do algoritmo após a poda.

A aplicação do algoritmo PSOM ao projeto de receptores FH-CDMA indica a importância da utilização desta arquitetura a problemas de mundo real.

9.8.2 Método de Inicialização dos Modelos Baseados em Busca de Projeção (KODIR)

Para que seja mais fácil visualizar os resultados obtidos pelo método proposto no Capítulo 8, serão apresentados os resultados considerando-se apenas um espaço de saída bidimensional.

A Figura 9.6 apresenta a configuração final dos vetores de pesos variando-se a quantidade de vetores distribuídos em um espaço de saída de dimensão 2.



DO: 0.0623
$N = 2$

DO: 0.0950
$N = 3$

DO: 0.0683
$N = 4$

DO: 0.0705
$N = 5$

DO: 0.0206
$N = 6$

DO: 0.0356
$N = 12$

Figura 9.6: Simulações do método KODIR para geração de vetores aproximadamente uniformemente distribuídos em um espaço de dimensão 2, onde DO é a distância do vetor resultante à origem e N o número de vetores (direções).

9.8.2.1 Comentários sobre o método KODIR

Através de várias experimentações verificou-se que o método desenvolvido é capaz de gerar vetores aproximadamente uniformemente distribuídos em espaços de dimensão elevada. Esta característica permite a sua aplicação como método ótimo de inicialização dos modelos baseados em busca de projeção.

Capítulo 10

Conclusões

Neste capítulo visamos apresentar, de forma sucinta, as diversas contribuições deste trabalho e propor algumas possíveis perspectivas de projetos a serem desenvolvidos como extensão ou complemento desta dissertação.

10.1 Contribuições

O estudo de redes neurais artificiais com base em conceitos de teoria de aproximação de funções e análise numérica conduz a resultados de análise e síntese fundamentais para a aplicação destas estruturas conexionistas a variados tipos de problemas não-lineares.

A arquitetura de rede neural do tipo perceptron de múltiplas camadas (MLP – do inglês *multilayer perceptron*) para treinamento supervisionado foi investigada detalhadamente, resultando:

- em estratégias otimizadas de treinamento das arquiteturas MLP;
- no estudo dos princípios envolvidos no projeto de algoritmos de treinamento eficientes para redes do tipo MLP;
- na compreensão dos méritos dos algoritmos de segunda ordem, derivados de teorias avançadas de otimização e comparados com os métodos convencionais de treinamento;
- na implementação de uma grande variedade de métodos de segunda ordem (incluindo métodos de gradiente conjugado e quase-Newton);
- no estudo da capacidade de generalização dos modelos de redes neurais resultantes;
- na comparação de desempenho de diferentes algoritmos de treinamento quando aplicados a vários problemas clássicos e de natureza distinta;
- em estratégias de redução dos efeitos indesejáveis associados a mínimos locais;
- no estudo de estratégias de determinação da dimensão ótima das arquiteturas empregadas;
- em estudos preliminares sobre a implementação em paralelo destas arquiteturas

Este trabalho foi finalizado com estudos sobre os *mapas auto-organizáveis de Kohonen* (SOM, do inglês *Self-Organizing Maps*), cujo treinamento é do tipo não-supervisionado, e foi

proposto um método de determinação da arquitetura mínima, ou quase mínima, para este tipo de rede. Ainda utilizando os SOM's apresentou-se uma nova estratégia de inicialização para o método de aprendizagem baseado em modelos de projeção. Este método visa explorar ao máximo o espaço inicial de soluções, gerando vetores aproximadamente uniformemente distribuídos em um espaço multidimensional.

As redes auto-organizáveis de Kohonen são muito utilizadas para mapear um conjunto de dados de entrada em regiões definidas em um mapa de saída. Em outras palavras, seu objetivo é agrupar dados que possuem características semelhantes em regiões próximas do espaço de saída. Neste contexto, a determinação da dimensão do espaço de mapeamento de saída permanece uma incógnita, e uma escolha inadequada desta dimensão pode levar a redundâncias nas soluções, assim como a um desperdício do potencial computacional utilizado na modelagem da estrutura. A estratégia desenvolvida efetua uma análise estatística do mapeamento feito pela rede durante o treinamento e verifica a existência ou não da necessidade de se eliminar unidades redundantes, ou seja, que representam o mesmo grupo de dados.

Os resultados deste trabalho são caracterizados por apresentarem uma vigorosa formalização teórica e uma proximidade marcante com outras linhas de pesquisa e atuação científica já estabelecidas, particularmente em aspectos vinculados às ferramentas mais modernas e versáteis empregadas nestas áreas. Estes aspectos se justificam pelo caráter multidisciplinar das redes neurais artificiais, pela maturidade que caracteriza todas as contribuições envolvendo a área e pela sua natureza intrinsecamente voltada ao tratamento de problemas de complexidade elevada.

Outra contribuição relevante deste trabalho foi a proposta de um método de inicialização do conjunto de pesos das redes MLP que visa explorar correlações presentes no conjunto de dados e proporcionar aos neurônios a capacidade de utilizar informações contidas nas próprias amostras de treinamento para gerar um vetor inicial de parâmetros mais adequado a tarefa a ser desempenhada. Grande parte dos diferentes métodos de inicialização do processo de treinamento que se encontram na literatura desprezam as informações existentes nos dados e consideram apenas a arquitetura de rede que está sendo utilizada para definir os pesos iniciais. A abordagem é baseada em procedimentos estatísticos, validados pelos resultados experimentais obtidos.

10.2 Extensões

Existem vários tópicos relacionados ao tema deste trabalho, cuja abordagem pode ser viabilizada a partir dos resultados obtidos e apresentados nesta dissertação. Dentre eles podemos citar:

- estudos sobre a capacidade de generalização das redes utilizando-se teoria de regularização e comparação com os resultados obtidos utilizando-se procedimentos de validação cruzada;
- comparação de desempenho (incluindo complexidade computacional do processo de busca) de diferentes procedimentos de busca ótima do passo de ajuste, como Fibonacci, Seção Áurea e Falsa Posição;
- utilização de estratégias que não utilizam diferenciação, como algoritmos genéticos (AG) e *simulated annealing* (SA), na determinação de arquiteturas e pesos ótimos da rede;
- verificação de desempenho de outros tipos de função de ativação para os neurônios das redes do tipo MLP, por exemplo, extraídos de um conjunto de funções candidatas;
- análise mais detalhada do efeito *bias/variância*;
- estudo de técnicas eficientes para partição do conjunto amostral, em treinamento, validação e teste, garantindo uma maior capacidade de generalização da rede treinada;
- implementação computacional em arquiteturas paralelas dos métodos de segunda ordem;
- estudo de métodos construtivos para os mapas auto-organizáveis de Kohonen;
- verificação de desempenho do método de poda proposto para as redes de Kohonen utilizando mapas com dimensão igual ou superior a 2;
- verificação mais abrangente da aplicação do algoritmo KODIR (Capítulo 8) para a geração das direções iniciais das redes construtivas baseadas em modelos de busca por projeção.

Referências Bibliográficas

[1] **Amari S., Murata N., Müller K.-R., Finke M. & Yang H.**, "Statistical Theory of Overtraining - Is Cross-Validation Assymptotically Effective?", *Technical Report*, 1996, URL: ftp://archive.cis.ohio-state.edu/pub/neuroprose/amari.overtraining.ps.Z.

[2] **Atiya A. & Ji C.**, "How Initial Conditions Affect Generalization Performance in Large Networks", *IEEE Trans. on Neural Networks*, vol. 8, n° 2, 1997.

[3] **Barnard E.**, "Optimization for Training Neural Nets", *IEEE Trans. on Neural Networks*, vol. 3, n° 2, 1992.

[4] **Barreiros J.A.L., Ribeiro R.R.P., Affonso C.M. & Santos E.P.**, "Estabilizador de Sistemas de Potência Adaptativo com Pré-Programação de Parâmetros e Rede Neural Artificial", *Third Latin-American Congress: Eletricity generation and transmission*, November 9th-13th, Campos do Jordão – SP/Brasil, pp.538-542, 1997.

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

[6] **Battiti R.**, "Learning with First, Second, and no Derivatives: A Case Study in High Energy Physics", *Neurocomputing*, NEUCOM 270, vol. 6, pp. 181-206, 1994, URL: ftp:// ftp.cis.ohio-state.edu/pub/neuroprose/battiti.neuro-hep.ps.Z.

[7] **Bazaraa M., Sherali H.D. & Shetty C.M.**, "Nonlinear Programming – Theory and Algorithms", 2° edição, John Wiley & Sons Inc., 1993.

[8] **Belew R.K., McInerney J., Schraudolph N.N.**, "Evolving Networks: Using the Genetic Algorithm with Connectionist Learning", *CSE Tech. Report #CS90-174*, 1990.

[9] **Bello M.G.**, "Enhanced Training Algorithms, and Integrated Training/Architecture Selection for Multilayer Perceptron Networks", *IEEE Trans. on Neural Netowrks*, vol. 3, n° 6, pp. 864-875, 1992.

[10] **Bishop C.**, "Exact Calculation of the Hessian Matrix for the Multilayer Perceptron," *Neural Computation*, vol. 4, pp. 494-501, 1992.

[11] **Boers E.G.W., & Kuiper H.**, "Biological Metaphors and the Design of Modular Artificial Neural Networks", Tese de Mestrado, Leiden University, Leiden, Netherlands, 1992, URL: http://www.wi.leidenuniv.nl/~boers/.

[12] **Bromberg M. & Chang T.S.**, "One Dimensional Global Optimization Using Linear Lower Bounds," *In C. A. Floudas & P. M. Pardalos (Eds.), Recent advances in global optimization*, pp. 200-220, Princeton University Press, 1992.

[13] **Castro L.N., Lima W. da S. & Martins W.**, "Redes Neurais Artificiais Aplicadas a Previsão de Produtividade de Soja", *Anais do III Congresso Brasileiro de Redes Neurais*, vol. único, pp. 308-312, 1997, URL: http://www.dca.fee.unicamp.br/~lnunes.

[14] **Castro L.N., Lima W. da S. & Martins W.**, "Classificador Neural para Previsão de Carga a Curto Prazo". *Anais do IV Simpósio Brasileiro de Redes Neurais*, vol. único, pp. 68-70, 1997, URL: http://www.dca.fee.unicamp.br/~lnunes.

[15] **Castro L.N., Von Zuben F.J. & Martins W.**, "Hybrid and Constructive Learning Applied to a Prediction Problem in Agriculture". *Proceedings of the IEEE International Joint Conference on Neural Networks*, Alaska/USA, vol. 3, pp. 1932-1926, Maio de 1998a, URL: http://www.dca. fee.unicamp.br/~lnunes.

[16] **Castro L.N., & Von Zuben F.J.**, "A Hybrid Paradigm for Weight Initialization in Supervised Feedforward Neural Network Learning", *aceito para o ICS'98 Workshop on*

Artificial Intelligence, Tainan/Taiwan/R.O.C., Dezembro, 1998b, URL: http://www.dca.fee.unicamp.br/~lnunes.

[17] **Castro L.N., Yioda E.M., Von Zuben F.J., & Gudwin R.R.**, "Feedforward Neural Network Initialization: an Evolutionary Approach", *submetido ao V Brazilian Symposium on Neural Networks*, Belo Horizonte - MG/Brasil, Dezembro de 1998c, URL: http://www.dca. fee.unicamp.br/~lnunes.

[18] **Chen S., Chang E.S. & Alkadhimi K.**, "Regularised Orthogonal Least Squares Algorithm for Constructing Radial Basis Function Networks", *Int. Joint Conference on Control*, v. 64, n° 5, pp.829-837, 1996, URL: http://www2.ee.port.ac.uk/~schen-www/paper/ijcrols.ps

[19] **Chen C.-T.**, "Linear Systems Theory and Design", New York: Holt, Rinehart and Winston, 1984.

[20] **Cho S.-B.**, "Self-Organizing Map with Dinamical Node Splitting: Application to Handwritten Digit Recognition", *Neural Computation*, vol. 9, pp. 1345-1355, 1997.

[21] **Cybenko G.**, "Approximation by Superpositions of a Sigmoidal Function", *Mathematics of Control Signals and Systems*, vol. 2, pp. 303-314, 1989.

[22] **Darken C., & Moody J.**, "Note on Learning Rate Schedules for Stochastic Optimization", *Neural Information Processing Systems*, Lippmann R. P. e Moody J. E. e Touretzky D. S. (Editors), pp. 832-838, 1991, URL: ftp://neural.cse.ogi.edu/pub/neural/papers/darkenMoody90.rates.tar.Z

[23] **Demuth H. & Beale M.**, "Neural Network Toolbox – For use with Matlab", The Math Works Inc., 1993.

[24] **Doering A., Galicki M. & Witte H.**, "Structure Optimization of Neural Networks with the A*-Algorithm", *IEEE Trans. on Neural Networks*, vol. 8, n° 6, pp. 1434-1445, 1997.

[25] **Eaton H.A., & Olivier T.L.**, "Learning Coefficient Dependence on Training Set Size", *Neural Networks*, vol. 5, pp. 283-288, 1992.

[26] **Ensley D. & Nelson D.E.**, "Extrapolation of Mackey-Glass data using Cascade Correlation", *SIMULATION*, 58:5, pp. 333-339, 1992.

[27] **Fahlman S.E.**, "An Empirical Study of Learning Speed in Back-Propagation Networks", *Technical Report*, September 1988, URL: ftp://archive.cis.ohio-state.edu/pub/neuroprose/ fahlman.quickprop-tr.ps.Z

[28] **Fahlman S.E., & Lebiere C.**, "The Cascade Correlation Learning Architecture", *Advances in Neural Information Processing Systems 2 (NIPS 2)*, pp. 524-532, 1990, URL: ftp://archive.cis.ohio-state.edu/pub/neuroprose/fahlman.cascor-tr.ps.Z

[29] **Fausett L.**, "Fundamentals of Neural Networks - Architectures, Algorithms and Applications", 1994.

[30] **Fiesler E.**, "Comparing Parameterless Learning Rate Adaptation Methods," *Proceedings of the ICNN'97*, pp. 1082-1087, 1997.

[31] **Finschi L.**, "An Implementation of the Levenberg-Marquardt Algorithm", *Technical Report*, April 1996, URL: http://www.ifor.math.ethz.ch/staff/finschi/Papers/LevMar.ps.gz.

[32] **Friedman J.H., & Stuetzle W.**, "Projection Pursuit Regression", *Journal of the American Statistical Association*, vol. 76, n° 376, pp. 817-823, 1981.

[33] **Fritzke B.**, "Growing Cell Structures – A Self-Organizing Network for Unsupervised and Supervised Learning", *Technical Report TR 26/93*, 1993, URL: http://www.neuroinformatik.ruhr-uni-bochum.de/ini/PEOPLE/fritzke/papers/fritzke.tr93-26.ps.gz

[34] **Geman S., Bienenstock E. & Doursat R.**, "Neural Networks and the Bias/Variance Dilemma", *Neural Computation*, vol. 4, pp. 1-58, 1992.

[35] **Girosi F., Jones M. & Poggio T.**, "Regularization Theory and Neural Networks Architectures", *Neural Computation*, vol. 7, pp.219-269, 1995.

[36] **Groot C. de & Würtz D.**, "Plain Backpropagation and Advanced Optimization Algorithms: A Comparative Study", *Neurocomputing*, NEUCOM 291, vol. 6, pp.153-161, 1994.

[37] **Gudwin R.**, "Contribuição ao Estudo Matemático de Sistemas Inteligentes", Tese de Doutorado, Faculdade de Engenharia Elétrica, Unicamp, 1996.

[38] **Hamey L.G.C.**, "Benchmarking Feed-forward Neural Networks", *Technical Report*, 1992, URL: ftp://ftp.mpce.mq.edu.au/pub/comp/papers/hamey.nips91.ps.Z.

[39] **Hamey L.G.C.**, "A Comparison of Back Propagation Implementations", *Technical Report C/TR95-06*, 1995, URL: ftp://ftp.mpce.mq.edu.au/pub/comp/techreports/950006.gibb.ps.Z

[40] **Hagan M.T.**, "Training Feedforward Networks with the Marquardt Algorithm", *IEEE Trans. on Neural Networks*, vol. 5, n° 6, pp. 989-993, 1994.

[41] **Hagan M.T., Demuth H.B. & Beale M.**, "Neural Network Design", PWS Publishing Company, 1995.

[42] **Haykin S.** "Neural Networks – A Comprehensive Foundation", Macmillan College Publishing Inc., 1994.

[43] **Hebb D.O.**, *Organization of Behaviour: A Neuropsychological Theory*. Wiley, New York, 1949.

[44] **Hertz J., Krogh A. & Palmer R.G.**, Introduction to the Theory of Neural Computation. Addison-Wesley Publishing Company, 1991.

[45] **Hopfield J.**, "Neural Networks and Physical Systems with Emergent Collective Computational Capabilities", *Proceedings of the National Academy of Science 79*, pp. 2554-2558, 1982.

[46] **Huang S.H., & Endsley M.R.**, "Providing Understanding of the Behaviour of Feedforward Neural Networks", *IEEE Trans. On Systems, Man, and Cybernetics-Part B*, vol. 27, n° 3, pp. 465-474, 1997.

[47] **Hwang J.N., Lay, S.R., Maechler M., Martin R.D. & Schimert J.**, "Regression Modeling in Back-Propagation and Projection Pursuit Learning", *IEEE Trans. On Neural Networks*, vol. 5, n° 3, pp. 342-353, 1994.

[48] **Jacobs R.A.**, "Increased Rates of Convergence Through Learning Rate Adaptation", *Neural Networks*, vol. 1, pp. 295-307, 1988, URL: http://www.cs.umass.edu/Dienst/UI/2.0/Describe/ncstrl.umassa_cs%2fUM-CS-1987-117

[49] **Jondarr C.G.H.**, "Back Propagation Family Album", *Technical Report C/TR96-5*, 1996, URL: ftp://ftp.mpce.mq.edu.au/pub/comp/techreports/96C005.gibb.ps.

[50] **Joost M. & Schiffman W.**, "Speeding Up Backpropagation Algorithms by Using Cross-Entropy Combined With Pattern Normalization", *International Journal of*

Uncertainty, Fuzzyness and Knowledge-Based Systems, 1993, URL: http://www.uni-koblenz.de/~schiff/ cenprop_eng.ps.gz

[51] **Júnior G.A. de D.,** "A Utilização de redes Neurais Artificiais no Projeto de receptores FH-CDMA", *Dissertação de Mestrado*, Faculdade de Engenharia Elétrica, Unicamp, 1998.

[52] **Kaski S. & Kohonen T.**, "Exploratory Data Analysis by the Self-Organizing Map: Structures of Welfare and Poverty in the World", *Proceedings of the Third International Conference on Neural Networks in the Capital Markets*, Londres, Inglaterra, pp. 498-507, 1995, URL: http:// www.cis.hut.fi/~sami/nncm95.ps.gz.

[53] **Kaski S.**, "Data Exploration Using Self-Organizing Maps", *Tese de Doutorado*, Escola Politécnica Escandinávia, Finlândia, 1997, URL: http://www.cis.hut.fi/~sami/ thesis.ps.gz.

[54] **Kim Y.K., & Ra J.B.**, "Weight Value Initialization for Improving Training Speed in the Backpropagation Network", *Proceedings of the IEEE International Joint Conference on Neural Networks,* vol. 3, pp. 2396-2401, 1991.

[55] **Kohonen T.**, "Self-Organized Formation of Topologically Correct Feature Maps", *Biological Cybernetics*, 43, pp. 59-69, 1982.

[56] **Kolen J.F., & Pollack J.B.**, "Back Propagation is Sensitive to Initial Conditions", *Technical Report TR 90-JK-BPSIC*, 1990, URL: ftp://ftp.cs.brandeis.edu/pub/faculty/ pollack/bpsic.tar.Z.

[57] **Kosko B.**, "Neural Networks and Fuzzy Systems – A dynamical Systems Approach to Machine Intelligence", University of Southern California, Prentice-Hall International, Inc., 1992.

[58] **Kwok T.Y., & Yeung. D.Y.**, "Constructive Algorithms for Structure Learning in Feedforward Neural Networks for Regression Problems", *IEEE Trans. On Neural Networks*, vol. 8, n° 3, pp. 630-645, 1997, URL: ftp://ftp.cs.ust.hk/pub/dyyeung/paper/ yeung.tnn97a.ps.gz.

[59] **Kwok T.Y., & Yeung. D.Y.**, "Use of a Bias Term in Projection Pursuit Learning Improves Approximation and Convergence Properties", *IEEE Transactions on Neural Networks,* vol. 7, n° 5, pp. 1168-1183, 1996, URL: ftp://ftp.cs.ust.hk/pub/dyyeung/ paper/yeung.tnn96.ps.gz

[60] **Kwok T.Y., & Yeung. D.Y.**, "Constructive Neural Networks: Some practical considerations", *Proceedings of the IEEE International Conference on Neural Networks*, pp.198-203, Orlando, Florida, USA, 1994, URL: ftp://ftp.cs.ust.hk/pub/ dyyeung/paper/yeung.icnn94.ps.gz

[61] **Kwok T.Y., & Yeung D.Y.**, "Theoretical Analysis of Constructive Neural Networks", *Technical Report HKUST-CS93-12*, 1993, URL: http://www.comp.hkbu.edu.hk/~jamesk/ publication.html.

[62] **Lawrence S., Back A.D., Tsoi A.C., Giles C.L.**, "On the Distribution of Performance from Multiple Neural Network Trials", *IEEE Trans. on Neural Networks*, vol. 8, n° 6, pp.1507-1517, 1997, URL: http://www.neci.nj.nec.com/homepages/lawrence/papers/ results-tnn97/results-tnn97-letter.ps.gz.

[63] **Lehtokangas M., Saarinen J., Kaski K. & Huuhtanen P.**, "Initializing Weights of a Multilayer Perceptron by Using the Orthogonal Least Squares Algorithm", *Neural Computation*, vol. 7, pp. 982-999, 1995.

[64] **Littmann E., & Ritter H.**, "Cascade LLM Networks", *Proc. of the International Conference on Neural Networks*, Orlando-Florida, vol. 1, pp. 253-257, 1992, URL: ftp://neuro.informatik. uni-ulm.de/ni/enno/littmann.icann92.ps.gz.

[65] **Luenberger D.G.**, "Optimization by Vector Space Methods", New York: John Wiley & Sons, 1969.

[66] **Luenberger D.G.**, "Linear and Nonlinear Programming", 2° edição, 1989.

[67] **Mardia K.V., Kent J.T. & Bibby J.M.**, "Multivariate Analysis", Academic Press, 1979.

[68] **McClelland J.L., & Rumelhart D.E.**, "Parallel Distributed Processing: Explorations in the Microstructure of Cognition, volume 1: Foundations", M.I.T. Press, Cambridge, Massachusetts, 1986.

[69] **McClelland J.L., & Rumelhart D.E.**, "Parallel Distributed Processing: Explorations in the Microstructure of Cognition, volume 2: Psychological and Bilogical Models", M.I.T. Press, Cambridge, Massachusetts, 1986.

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

[71] **McKeown J.J., Stella F. & Hall G.**, "Some Numerical Aspects of the Training Problem for Feed-Forward Neural Nets", *Neural Networks*, vol. 10, n° 8, pp.1455-1463, 1997.

[72] **Misra M.**, "Parallel Environments for Implementing Neural Networks", *Neural Computing Survey*, vol. 1, pp. 48-60, 1997. URL: http://www.isci.berkeley.edu/~jagota/NCS.

[73] **Minsky M.L. & Papert S.A.**, "Perceptrons: An Introduction to Computational Geometry", M.I.T. Press, Cambridge, Massachusets, 1969.

[74] **Moller M.F.**, "A Scaled Conjugate Gradient Algorithm for Fast Supervised Learning", *Neural Networks*, vol. 6, pp. 525-533, 1993.

[75] **Narendra K.S. & Pathasarathy K.**, "Identification and Control of Dynamical Systems Using Neural Networks", *IEEE Trans. on Neural Networks*, vol. 1, n. 1, pp. 4-27, 1990.

[76] **Nguyen D., & Widrow B.**, "Nguyen-Widrow Initialization", *Em FAUSETT, 1994*, pp. 297, 1990.

[77] **Opitz D.W., & Shavlik J.W.**, "Connectionist Theory Refinement: Genetically Searching the Space of Network Topologies", *Journal of Artificial Intelligence Research*, vol. 6, pp. 177-209, 1997, URL: http://www.jair.org/abstracts/opitz97a.html

[78] **Patterson D.W.**, "Introduction to Artificial Intelligence and Expert Systems", University of Texas, El Paso, Prentice Hall, 1990.

[79] **Pearlmutter B.A.**, "Gradiente Descent: Second-Order Momentum and Saturating Error", *In Advances in Neural Information Processing Systems 4*, pp. 887-894, 1992, URL: http://www.cs.unm.edu/~bap/papers/nips91-descent.ps.gz.

[80] **Pearlmutter B.A.**, "Fast Exact Calculation by the Hessian", *Neural Computation*, vol. 6, pp. 147-160, 1994, URL: ftp://ftp.cis.ohio-state.edu/pub/neuroprose/pearlmutter.hessian.ps.Z.

[81] **Prechelt L.**, "Automatic Early Stopping Using Cross Validation: Quantifying the Criteria", *Neural Networks*, vol. 11, n° 4, pp. 761-767, 1998.

[82] **Prechelt L.,** "Early Stopping - but when?", *Technical Report*, 1997, URL: http:// wwwipd.ira.uka.de/~prechelt/Biblio/stop_tricks1997.ps.gz

[83] **Prechelt L.,** "A Quantitative Study of Experimental Evaluations of Neural Network Learning Algorithms: Current Research Practice. Neural Networks" 9(3):457-462, 1996, URL: http:// wwwipd.ira.uka.de/~prechelt/Biblio/neurnetw96.ps.gz

[84] **Prechelt L.,** "Proben 1 - A Set of Neural Networks Benchmark Problems and Benchmarking Rules", *Technical Report 21/94*, 1994, URL: ftp://ftp.ira.uka.de/pub/ papers/techreports/1994/ 1994-21.ps.gz.

[85] **Reed R.**, "Pruning Algorithms – A Survey", *IEEE Trans. on Neural Networks*, vol. 4, n° 5, pp. 740-747, 1993.

[86] **Rosenblatt F.**, "The Perceptron: A Perceiving and Recognizing Automation", *Technical Report 85-460-1*, Cornell Aeronautical Laboratory, 1957.

[87] **Rumelhart D.E., McClelland J.L., and the PDP Research Group**. "Parallel Distributed Processing: Exploration in the Microstructure of Cognition, vol. 1. MIT Press, Cambridge, Massachussetts, 1986.

[88] **Saarinen S., Bramley R., e Cybenko G.**, "Ill-Conditioning in Neural Network Training Problems", *CSRD Report 1089*, 1991, URL: http://polaris.cs.uiuc.edu/ reports/1089.ps.gz.

[89] **Saarinen S., Bramley R., e Cybenko G.**, "Neural Networks, Backpropagation, and Automatic Differentiation", *In Automatic Differentiation of Algorithms: Theory, Implementation, and Application*, Philadelphia, PA, pp. 31-42, 1992, URL: http://www.siam. org/catalog/mcc07/griewank.htm

[90] **Sarkar M., & Yegnanarayana B.**, "Feedforward Neural Networks Configuration Using Evolutionary Programming", *Proceedings of the IEEE Int. Conference on Neural Networks*, pp. 438-443, 1997.

[91] **Sarle W.S.**, "Neural Networks and Statistical Models", *Proceedings of the Nineteenth Annual SAS Users Group International Conference*, 1994, URL: ftp://ftp.sas.com/pub/ sugi19/neural/ neural1.ps.

[92] **Sarle W.S.**, "Stopped Training and Other Remedies for Overfitting", *Proceedings of the Symposium on the Interface*, 1995, URL: ftp://ftp.sas.com/pub/neural/inter95.ps.Z.

[93] **Shepherd A.J.**, "Second-Order Methods for Neural Networks – Fast and Reliable Methods for Multi-Layer Perceptrons", Springer, 1997.

[94] **Shewchuk, J.R.**, "An Introduction to the Conjugate Gradient Method Without the Agonizing Pain", *Technical Report*, 1994, URL: http://www.cs.cmu.edu/afs/cs/ project/quake/public/papers/painless-conjugate-gradient.ps.

[95] **Schiffman W., Joost M., & Werner R.**, "Optimization of the Backpropagation Algorithm for Training Multilayer Perceptrons", *Technical Report*, 1994, URL: ftp://archive.cis.ohio-state. edu/pub/neuroprose/schiff.bp_speedup.ps.Z.

[96] **Sjöberg J., & Ljung L.**, "Overtrainning, Regularization and Search for Minimum with Application to Neural Networks", *Technical Report*, 1994 URL: http://www.control.isy.liu.se.

[97] **Stäger F., & Agarwal M.**, "Three Methods to Speed up the Training of Feedforward and Feedback Perceptrons", *Neural Networks*, vol. 10, n° 8, pp. 1435-1443, 1997.

[98] **Strang G.**, "Linear Algebra and its Applications", San Diego: Harcourt Brace Jovanovich Publishers, 1988.

[99] **Tetko I.V., & Villa A.E.P.**, "Efficient Partition of Learning Data Sets for Neural Network Training", *Neural Networks*, vol. 10, n° 8, pp. 1361-1374, 1997.

[100] **Thimm G., & Fiesler E.**, "High-Order and Multilayer Perceptron Initialization", *IEEE Trans. on Neural Networks*, vol. 8, n° 2, 1997, URL: http://www.idiap.ch/cgi-bin/w3-msql/ publications/thimm-97.1.bib.

[101] **Thimm G., Moerland P. & Fiesler E.**, "The Interchangeability of Learning Rate Gain in Backpropagation Neural Networks", *Neural Computation*, vol. 8, n° 2, 1996 ftp://ftp.cis.ohio-state.edu/pub/neuroprose/thimm.gain.ps.Z.

[102] **Thimm G., & Fiesler E.**, "Evaluating Pruning Methods", *Proceedings of the International Symposium on Artificial Neural Networks*, Taiwan/R.O.C., 1995, URL: ftp://ftp.idiap.ch/pub/papers/ neural/thimm.pruning-hop.ps.Z

[103] **URL 1:** Neural Networks: Frequently Asked Questions URL:http://camphor.elcom. nitech.ac.jp/~pear/neural/FAQ3.html

[104] **URL 2:** Neural Networks Benchmarks. URL: ftp://ftp.ics.uci.edu/pub/machine-leraning-databases

[105] **URL 3:** TSP BIB Home Page. URL: http://www.ing.unlp.edu.ar/cetad/mos/ TSPBIB_home. html

[106] **Van Der Smagt P., P**, "Minimization Methods for Training Feedforward Neural networks," Neural Networks, vol 1, n° 7, 1994, URL: http://www.op.dlr.de/~smagt/ papers/SmaTB92.ps.gz

[107] **Von Newmann J.**, "The Computer and the Brain", Yale University Press, New Haven, 1958.

[108] **Von Zuben F.J. & Netto M.L.A.**, "Unit-growing Learning Optimizing the Solvability Condition for Model-free Regression", *Proc. of the IEEE International Conference on Neural Networks*, vol. 2, pp. 795-800, 1995.

[109] **Von Zuben F.J. & Netto M.L.A.**, "Projection Pursuit and the Solvability Condition Applied to Constructive Learning", *Proc. of the IEEE International Joint Conference on Neural Networks*, vol. 2, pp. 1062-1067, 1997.

[110] **Von Zuben F.J.**, "Modelos Paramétricos e Não-Paramétricos de Redes neurais Artificiais e Aplicações", *Tese de Doutorado*, Faculdade de Engenharia Elétrica, Unicamp, 1996.

[111] **Von Zuben F.J.**, "Redes Neurais Aplicadas ao Controle de Máquinas de Indução", Tese de Mestrado, Faculdade de Engenharia Elétrica, Unicamp, 1993.

[112] **Wessels L.F.A., & Barnard E.**, "Avoiding False Local Minima by Proper Initialization of Connections", *IEEE Trans. on Neural Networks*, vol. 3, n° 6, 1992.

[113] **Widrow B.**, "Generalization and Information Storage in Networks of Adaline Neurons", Em Yovitz, M., Jacobi, G., Goldstein, G., editors, *Self-Organizing Systems*, pp. 435-461, Spartan Books, Washington DC, 1962.

[114] **Wiener N.**, *Cybernetics*, M.I.T. Press, Cambridge, MA, 1948.

[115] **Zheng Z.**, "A Benchmark for Classifier Learning", *Technical Report 474*, 1993, URL: http:// www.cm.deakin.edu.au/~zijian/Papers/ai93-benchmark.ps.gz.

[116] **Zurada J.M., Marks R.J. & Robinson C.J.**, "Computational Intelligence Imitating Life", *IEEE Press*, 1994.

Apêndice A

Conceitos Básicos de Álgebra Linear e Notação

As definições e axiomas apresentados a seguir representam uma coletânea de conceitos básicos de álgebra linear, fundamentais para a formulação de boa parte dos resultados apresentados neste estudo. A inclusão destes conceitos na forma de apêndice se justifica pela sua utilização generalizada na apresentação dos resultados principais deste trabalho. Sendo assim, no texto da tese, assume-se familiaridade com os tópicos aqui referenciados. Criam-se, portanto, condições para que o texto se ocupe apenas de tópicos mais avançados.

Na apresentação que se segue, os resultados são apenas referenciados, sendo que discussões mais detalhadas podem ser encontradas na literatura (CHEN, 1984; LUENBERGER, 1969; STRANG, 1988; BAZARAA *et. al.* , 1993).

A.1 Escalar

Uma variável que assume valores no eixo dos números reais é denominada escalar. Os escalares são descritos por letras minúsculas do alfabeto romano expressas em itálico. O conjunto de todos os escalares reais é representado por $\Re$.

- O módulo de um escalar real x é dado na forma: $|x| = \begin{cases} x & \text{se } x \geq 0 \\ -x & \text{se } x < 0 \end{cases}$

A.2 Vetor

Um arranjo ordenado de n escalares $x_i \in \Re$ (i=1,2,...,n) é denominado vetor de dimensão n. Os vetores são descritos por letras minúsculas do alfabeto romano expressas em negrito, e assumem a forma de coluna ou linha:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad \text{ou} \quad \mathbf{x} = \begin{bmatrix} x_1 & x_2 & \cdots & x_n \end{bmatrix}.$$

Para coerência de notação, salvo menção contrária, o texto desta dissertação considera os vetores na forma de coluna. O conjunto de todos os vetores de dimensão n com elementos reais é representado por $\Re^n$.

- Um escalar é um vetor de dimensão 1.
- Vetor $\mathbf{0}_n$: é o vetor nulo de dimensão n, com todos os elementos iguais a zero. O subscrito n é suprimido quando não há margem à dúvida.
- Vetor $\mathbf{1}_n$: é o vetor de dimensão n com todos os elementos iguais a 1.
- Vetor $\mathbf{e}_i$: é o vetor normal unitário de dimensão n (a dimensão deve ser indicada pelo contexto) com todos os elementos iguais a 0, exceto o i-ésimo elemento que é igual a 1. Neste caso, $1 \leq i \leq n$.

A.3 Matriz

Um arranjo ordenado de $m.n$ escalares x_{ij} (i=1,2,...,m; j=1,2,...,n) é denominado matriz de dimensão $m \times n$. As matrizes são descritas por letras maiúsculas do alfabeto romano expressas em negrito, e assumem a forma:

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m1} & x_{m2} & \cdots & x_{mn} \end{bmatrix}.$$

O conjunto de todas as matrizes $m \times n$ com elementos reais é representado por $\Re^{m \times n}$.

- As colunas da matriz $\mathbf{X}$ são vetores-coluna descritos por $\mathbf{x}_i = \begin{bmatrix} x_{1i} \\ x_{2i} \\ \vdots \\ x_{mi} \end{bmatrix}$, i=1,...,n.
- As linhas da matriz $\mathbf{X}$ são vetores-linha descritos por $\mathbf{x}_{(j)} = \begin{bmatrix} x_{j1} & x_{j2} & \cdots & x_{jn} \end{bmatrix}$, j=1,...,m.
- Um vetor é uma matriz com número unitário de linhas e/ou colunas.

A.4 Conjuntos e operações com conjuntos

Um conjunto pode ser definido como uma agregação de objetos. Os conjuntos são descritos por letras maiúsculas do alfabeto romano expressas em itálico. Por conveniência de notação, alguns conjuntos especiais são descritos por símbolos específicos. Exemplos:

- $\aleph$: conjunto dos números naturais
- $\Re$: conjunto dos números reais
- $\mathbf{C}$: conjunto dos números complexos

O estado lógico ou associação de um elemento $\mathbf{x}$ a um conjunto X qualquer é representado por

$\mathbf{x} \in X$: $\mathbf{x}$ pertence a X

$\mathbf{x} \notin X$: $\mathbf{x}$ não pertence a X

Um conjunto pode ser especificado listando-se seus elementos entre colchetes

$$X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$$

ou evidenciando uma ou mais propriedades comuns aos seus elementos

$$X_2 = \{\mathbf{x} \in X_1 \text{ tal que } \mathrm{P}(\mathbf{x}) \text{ é verdade}\} \quad \text{ou} \quad X_2 = \{\mathbf{x} \in X_1 : \mathrm{P}(\mathbf{x})\}$$

As principais operações entre conjuntos são:

- União: $X_1 \cup X_2 = \{\mathbf{x}: \mathbf{x} \in X_1 \text{ ou } \mathbf{x} \in X_2\}$;
- Interseção: $X_1 \cap X_2 = \{\mathbf{x}: \mathbf{x} \in X_1 \text{ e } \mathbf{x} \in X_2\}$;

$X_1 \cap X_2 = \varnothing$ (conjunto vazio) se X_1 e X_2 são conjuntos disjuntos.

O complemento de um conjunto X é representado por $\overline{X}$ e é definido na forma:

$$\overline{X} = \{\mathbf{x}: \mathbf{x} \notin X\}.$$

S é um subconjunto de X, se $\mathbf{x} \in S$ implica $\mathbf{x} \in X$. Neste caso, diz-se que S está contido em X ($S \subset X$) ou que X contém S ($X \supset S$). Se $S \subset X$ e S não é igual a X, então S é um subconjunto próprio de X.

A.5 Conjuntos especiais de números reais

Se a e b são números reais ($a,b \in \Re$), define-se:

$$[a,b]=\{x:\ a \le x \le b\}$$
$$(a,b]=\{x:\ a < x \le b\}$$
$$[a,b)=\{x:\ a \le x < b\}$$
$$(a,b)=\{x:\ a < x < b\}$$

Se X é um conjunto de números reais, então o menor limitante superior de X, dado por

$$\overline{x} = \sup_{x \in X} x = \sup\{x:\ x \in X\},$$

é o supremo de X, e o maior limitante inferior de X, dado por

$$\underline{x} = \inf_{x \in X} x = \inf\{x:\ x \in X\},$$

é o ínfimo de X. Se $\overline{x} = +\infty$ então X não é limitado superiormente. De forma análoga, se $\underline{x} = -\infty$ então X não é limitado inferiormente.

A.6 Espaço vetorial linear

Um espaço vetorial linear $(X,\Im)$ consiste de um conjunto X de vetores e de um campo $\Im$, sobre os quais estão definidas duas operações:

- ***Adição*** (+): mapeamento $X \times X \rightarrow X$ tal que, $\forall\ \mathbf{x}, \mathbf{y} \in X$, $(\mathbf{x},\mathbf{y}) \rightarrow (\mathbf{x}+\mathbf{y})$ produz $\mathbf{x}+\mathbf{y} \in X$.
- ***Multiplicação por escalar*** ($\cdot$): mapeamento $\Re \times X \rightarrow X$ tal que, $\forall\ \mathbf{x} \in X$ e $\forall\ a \in \Im$, $(a,\mathbf{x}) \rightarrow (a \cdot \mathbf{x})$ produz $a \cdot \mathbf{x} \in X$.

Uma série de axiomas podem ainda ser deduzidos para o espaço $(X,\Im)$:

(1)	$\mathbf{x}+\mathbf{y} = \mathbf{y}+\mathbf{x}$	(comutatividade na adição)
(2)	$(\mathbf{x}+\mathbf{y})+\mathbf{z} = \mathbf{x}+(\mathbf{y}+\mathbf{z})$	(associatividade na adição)
(3)	$a \cdot (\mathbf{x}+\mathbf{y}) = a \cdot \mathbf{x} + a \cdot \mathbf{y}$	(distributividade na adição)
(4)	$\mathbf{x}+\mathbf{0} = \mathbf{x}$	(elemento nulo)
(5)	$(ab) \cdot \mathbf{x} = a \cdot (b \cdot \mathbf{x})$	(associatividade na multiplicação)
(6)	$(a+b) \cdot \mathbf{x} = a \cdot \mathbf{x} + b \cdot \mathbf{x}$	(distributividade na multiplição)

(7) $1 \cdot \mathbf{x} = \mathbf{x}$ (elemento neutro)

Exemplos: $(\Re,\Re)$,$(\Re^n,\Re)$, $(\Re^{m\times n},\Re)$, onde n e m são inteiros positivos.

A.7 Subespaço vetorial linear

Seja $(X,\Im)$ um espaço vetorial linear e S um subconjunto de X. Diz-se então que $(S,\Im)$ é um subespaço linear de $(X,\Im)$ se S forma um espaço linear sobre $\Im$ através das mesmas operações definidas sobre $(X,\Im)$.

A.8 Variedade linear

A translação de um subespaço é chamada de variedade linear. Uma variedade linear V pode ser descrita na forma:

$$V = \mathbf{x} + S,$$

onde $(S,\Im)$ é um subespaço linear de $(X,\Im)$ e $\mathbf{x}$ é qualquer elemento de X.

A.9 Conjunto convexo

Diz-se que um conjunto X em um espaço vetorial linear é convexo se dados quaisquer elementos $\mathbf{x}_1,\mathbf{x}_2 \in X$, todos os elementos da forma $a\mathbf{x}_1 + (1-a)\mathbf{x}_2$, com $0 \leq a \leq 1$ também estão em X. São conjuntos convexos:

- o conjunto vazio, por definição;
- subespaços;
- variedades lineares;
- interseção de conjuntos convexos.

As propriedades de convexidade e não convexidade estão ilustradas na Figura A.1.

(a) (b)

Figura A.1: Convexidade. (a) Convexo. (b) Não convexo.

A.10 Combinação linear e combinação convexa

Seja $S = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\}$ um subconjunto de um espaço vetorial linear $(X,\Im)$. Combinações lineares de elementos de S são formadas através de

$$a_1\mathbf{x}_1 + a_2\mathbf{x}_2 + \cdots + a_n\mathbf{x}_n,$$

onde $a_1, a_2, ..., a_n \in \Im$.

O conjunto de todas as combinações lineares de elementos de S é chamado de subespaço gerado e é representado por $[S]$. $[S]$ é um subespaço linear de $(X,\Im)$.

Se os escalares $a_1, a_2, ..., a_n \in \Im$ são tais que $a_i \geq 0$ $(i=1,2,...,n)$ e $\sum_{i=1}^{n} a_i = 1$, então a combinação linear é chamada combinação convexa dos elementos $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n \in X$.

A.11 Dependência linear

Diz-se que um vetor $\mathbf{x}$ é linearmente dependente em relação a um conjunto de vetores S se $\mathbf{x}$ pode ser expresso como uma combinação linear de elementos de S, ou seja, se $\mathbf{x} \in [S]$. Caso contrário, diz-se que $\mathbf{x}$ é linearmente independente em relação ao conjunto S. Um conjunto de vetores $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n$ é linearmente independente se e somente se a equação $\sum_{i=1}^{n} a_i\mathbf{x}_i = \mathbf{0}$ tem como única solução $a_i = 0$, $i = 1,2,...,n$.

A.12 Base e dimensão de um espaço vetorial linear

Um conjunto S de vetores linearmente independentes forma uma base para $(X,\Im)$ se S gera $(X,\Im)$. Se $S = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\}$ é uma base para $(X,\Im)$, então diz-se que o espaço vetorial $(X,\Im)$ é de dimensão n (ou n-dimensional). Qualquer outra base para o mesmo espaço $(X,\Im)$ possui o mesmo número n de elementos.

Qualquer $\mathbf{y} \in X$ pode ser expresso na forma:

$$\mathbf{y} = b_1\mathbf{x}_1 + b_2\mathbf{x}_2 + \cdots + b_n\mathbf{x}_n = \begin{bmatrix}\mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_n\end{bmatrix} \cdot \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix},$$

onde $\mathbf{b} = [b_1 \;\; b_2 \;\; \cdots \;\; b_n]^T$ é a representação de $\mathbf{y}$ na base S (o superscrito T representa a operação de transposição).

A representação de um vetor numa determinada base é única.

Se n é finito, diz-se que o espaço vetorial $(X,\Im)$ é de dimensão finita. Para n infinito, $(X,\Im)$ é dito ser de dimensão infinita.

A.13 Produtos entre vetores

Produto interno

O produto interno é uma função f_{PI}: $\Re^n \times \Re^n \rightarrow \Re$ que associa a cada par de vetores $\mathbf{x}, \mathbf{y} \in \Re^n$ um escalar dado por:

$$\langle \mathbf{x},\mathbf{y} \rangle = \mathbf{x}^T\mathbf{y} = x_1y_1 + x_2y_2 + \cdots + x_ny_n,$$

onde $\mathbf{x} = [x_1 \;\; x_2 \;\; \cdots \;\; x_n]^T$ e $\mathbf{y} = [y_1 \;\; y_2 \;\; \cdots \;\; y_n]^T$.

Produto externo

O produto externo é uma função f_{PE}: $\Re^m \times \Re^n \rightarrow \Re^{m \times n}$ que associa a cada par de vetores $\mathbf{x} \in \Re^m$, $\mathbf{y} \in \Re^n$ uma matriz de dimensão $m \times n$ definida na forma:

$$\rangle \mathbf{x},\mathbf{y} \langle = \mathbf{x}\mathbf{y}^T = \begin{bmatrix} x_1y_1 & x_1y_2 & \cdots & x_1y_n \\ x_2y_1 & \ddots & & \vdots \\ \vdots & & & \\ x_my_1 & \cdots & & x_my_n \end{bmatrix}$$

A.14 Norma, semi-norma e quase-norma

A norma é uma função f_N: $\Re^n \rightarrow \Re$ que associa a cada vetor $\mathbf{x} \in \Re^n$ um escalar representado por $\|\mathbf{x}\|$. A norma satisfaz os seguintes axiomas:

- $\|\mathbf{x}\| \geq \mathbf{0}, \;\; \forall \mathbf{x} \in X; \;\; \|\mathbf{x}\| = \mathbf{0} \Leftrightarrow \mathbf{x} = \mathbf{0}$;
- $\|\mathbf{x}+\mathbf{y}\| \leq \|\mathbf{x}\| + \|\mathbf{y}\|, \;\; \forall \mathbf{x},\mathbf{y} \in X$ (desigualdade triangular);
- $\|a\mathbf{x}\| = |a|\|\mathbf{x}\|, \;\; \forall \mathbf{x} \in X, \; \forall a \in \Im$.

Dentre as funções f_N que atendem a estes axiomas, é possível destacar:

- $\|\mathbf{x}\|_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p}$ (norma p);

- Para $p = 1$ tem-se $\|\mathbf{x}\|_1 = \sum_{i=1}^{n} |x_i|$;

- Para $p = 2$ tem-se $\|\mathbf{x}\|_2 = \langle \mathbf{x}, \mathbf{x} \rangle^{1/2} = \left(\mathbf{x}^T \mathbf{x} \right)^{1/2} = \left(\sum_{i=1}^{n} |x_i|^2 \right)^{1/2}$ (norma euclidiana);

- Para $p = \infty$ tem-se $\|\mathbf{x}\|_\infty = \max_i |x_i|$.

A semi-norma é uma função f_{SN}: $\Re^n \rightarrow \Re$ que satisfaz todas as propriedades de norma, com exceção do primeiro axioma. O subespaço linear $X_0 \subset \Re^n$ onde $\|\mathbf{x}\| = 0$ é denominado espaço nulo da semi-norma.

A quase-norma é uma função f_{QN}: $\Re^n \rightarrow \Re$ que satisfaz todas as propriedades de norma, com exceção do segundo axioma (desigualdade triangular), o qual assume a forma:

- $\|\mathbf{x} + \mathbf{y}\| \leq b(\|\mathbf{x}\| + \|\mathbf{y}\|), \quad \forall \mathbf{x}, \mathbf{y} \in X, \quad \text{com} \quad b \in \Re$.

Os conceitos de norma, semi-norma e quase-norma são fundamentais para a definição de propriedades topológicas.

A.15 Vetores ortogonais e ortonormais

Dois vetores $\mathbf{x}$, $\mathbf{y} \in \Re^n$ são ortogonais entre si, $\mathbf{x} \perp \mathbf{y}$, se o seu produto interno se anula:

$$\mathbf{x} \perp \mathbf{y} \Rightarrow \langle \mathbf{x}, \mathbf{y} \rangle = 0.$$

Além disso, se $\langle \mathbf{x}, \mathbf{x} \rangle = \langle \mathbf{y}, \mathbf{y} \rangle = 1$, então os vetores $\mathbf{x}$, $\mathbf{y} \in \Re^n$ são ortonormais entre si.

Um vetor $\mathbf{x}$ é ortogonal a um conjunto S ($\mathbf{x} \perp S$) se $\langle \mathbf{x}, \mathbf{s} \rangle = 0$, $\forall \mathbf{s} \in S$.

A.16 Relação entre produto interno e norma euclidiana

A desigualdade de Cauchy-Schwartz-Buniakowsky

$$\left|\langle \mathbf{x},\mathbf{y}\rangle\right|^2 \le \langle \mathbf{x},\mathbf{x}\rangle\langle \mathbf{y},\mathbf{y}\rangle \Rightarrow \left|\langle \mathbf{x},\mathbf{y}\rangle\right| \le \|\mathbf{x}\|_2 \cdot \|\mathbf{y}\|_2$$

estabelece uma importante relação entre produto interno e norma euclidiana. A igualdade é válida se e somente se os vetores **x** e **y** estiverem alinhados.

A.17 Projeção de um vetor em uma determinada direção

Dado um espaço vetorial linear X, seja $\mathbf{y} \in X$ um vetor que fornece uma determinada direção. A projeção de qualquer vetor $\mathbf{x} \in X$ na direção de **y** é dada na forma:

$$\text{proj}_{\mathbf{y}}(\mathbf{x}) = \frac{\langle \mathbf{x},\mathbf{y}\rangle}{\langle \mathbf{y},\mathbf{y}\rangle^{1/2}} \cdot \frac{\mathbf{y}}{\langle \mathbf{y},\mathbf{y}\rangle^{1/2}} = \frac{\mathbf{x}^T\mathbf{y}}{\mathbf{y}^T\mathbf{y}} \cdot \mathbf{y}$$

A.18 Vetores ortonormais gerados a partir de vetores linearmente independentes

Apesar de todo conjunto de vetores ortonormais não-nulos ser linearmente independente, nem todo conjunto de vetores linearmente independentes é ortonormal, mas pode passar por um processo de ortonormalização, como segue.

Dado um conjunto de m vetores n-dimensionais ($m \le n$) linearmente independentes $\{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_m\}$, é sempre possível estabelecer uma combinação linear adequada destes vetores que produza m vetores n-dimensionais mutuamente ortogonais $\{\mathbf{u}_1, \mathbf{u}_2, ..., \mathbf{u}_m\}$ que geram o mesmo espaço. Além disso, se os vetores $\mathbf{u}_i$ (i=1, ..., m) apresentarem norma unitária, eles são mutuamente ortonormais.

Um conjunto de vetores ortonormais $\{\mathbf{u}_1, \mathbf{u}_2, ..., \mathbf{u}_m\}$ pode ser obtido a partir de um conjunto de vetores linearmente independentes $\{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_m\}$ através do processo de ortogonalização de Gram-Schmidt, o qual pode ser dividido em duas etapas:

Etapa (1)
$$\mathbf{y}_1 = \mathbf{x}_1$$

$$\mathbf{y}_i = \mathbf{x}_i - \sum_{j=1}^{i-1} \frac{\langle \mathbf{x}_i,\mathbf{y}_j\rangle}{\langle \mathbf{y}_j,\mathbf{y}_j\rangle}\mathbf{y}_j, \quad i = 2, ..., m$$

Etapa (2)
$$\mathbf{u}_i = \frac{\mathbf{y}_i}{\langle \mathbf{y}_i,\mathbf{y}_i\rangle^{1/2}}, \quad i = 1, ..., m$$

Com isso tem-se

$$\mathbf{u}_1 = a_{11}\mathbf{x}_1$$

$$\mathbf{u}_2 = a_{21}\mathbf{x}_1 + a_{22}\mathbf{x}_2$$

$$\mathbf{u}_3 = a_{31}\mathbf{x}_1 + a_{32}\mathbf{x}_2 + a_{33}\mathbf{x}_3$$

$$\vdots \quad \vdots \quad \vdots \quad \ddots$$

com $a_{ii} > 0$ (i=1,...,m). Certamente existem outros processos de ortonormalização mais gerais, que não impõem qualquer tipo de restrição aos coeficientes a_{ij} ($i \geq j$; i,j=1,...,m).

A.19 Algumas propriedades de matrizes

Simetria

Uma matriz $\mathbf{A} = \{a_{ij}\}$ de dimensão $n \times n$ é dita ser simétrica se $a_{ij} = a_{ji}$, i,j=1,2,...,n.

Inversão

Dada uma matriz **A** de dimensão $n \times n$, se existe uma matriz **B** de mesma dimensão tal que $\mathbf{AB} = \mathbf{BA} = \mathbf{I}$ então **B** é única e é denominada a matriz inversa de **A**, ou seja, $\mathbf{B} = \mathbf{A}^{-1}$.

Pseudo-inversão

Dada uma matriz **A** de dimensão $m \times n$, sempre existe uma matriz **B** satisfazendo as seguintes igualdades:

- $\mathbf{ABA} = \mathbf{A}$;
- $\mathbf{BAB} = \mathbf{B}$;
- $(\mathbf{AB})^T = \mathbf{AB}$;
- $(\mathbf{BA})^T = \mathbf{BA}$;

A matriz **B**, geralmente denominada $\mathbf{A}^+$, é única e é conhecida como a inversa de Moore-Penrose ou a pseudo-inversa da matriz **A**. A matriz $\mathbf{A}^+$ pode ser diretamente determinada através da decomposição de **A** em valores singulares.

Se a matriz **A** é de posto completo, ou seja, se $\rho(\mathbf{A}) = \min(m,n)$ (veja definição mais adiante), a pseudo-inversa é dada na forma:

- se $m < n$, $\rho(\mathbf{A}) = m$ e $\mathbf{A}^+ = \mathbf{A}^T(\mathbf{A}\mathbf{A}^T)^{-1}$;
- se $m > n$, $\rho(\mathbf{A}) = n$ e $\mathbf{A}^+ = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T$;
- se $m = n$, $\rho(\mathbf{A}) = m = n$ e $\mathbf{A}^+ = \mathbf{A}^{-1}$.

Portanto, se **A** é uma matriz inversível (matriz quadrada e de posto completo), a pseudo-inversa é igual à inversa.

Cofator

Dada uma matriz **A** de dimensão $n \times n$, o cofator do elemento a_{ij} ($i,j=1,2,...,n$) é dado na forma:

$$c_{ij} = (-1)^{i+j} m_{ij},$$

onde m_{ij} é o determinante da matriz formada eliminando-se a i-ésima linha e a j-ésima coluna da matriz **A**.

Determinante

Dada uma matriz **A** de dimensão $n \times n$, o determinante de **A** é dado na forma:

$$|\mathbf{A}| = \det(\mathbf{A}) = \begin{cases} \sum_{i=1}^{n} a_{ij} c_{ij}, \text{ para qualquer } i \\ \text{ou} \\ \sum_{j=1}^{n} a_{ij} c_{ij}, \text{ para qualquer } j \end{cases}$$

onde c_{ij} é o cofator do elemento a_{ij}.

Seja $\mathbf{A} = [\mathbf{a}_1 \ \cdots \ \mathbf{a}_j \ \cdots \ \mathbf{a}_n]$ $(1 \leq j \leq n)$. O determinante de **A** possui as seguintes propriedades:

- Invariância: $\det([\mathbf{a}_1 \ \cdots \ \mathbf{a}_j \ \cdots \ \mathbf{a}_n]) = \det([\mathbf{a}_1 \ \cdots \ \mathbf{a}_j + \mathbf{a}_k \ \cdots \ \mathbf{a}_n]), \quad j \neq k,$ $1 \leq j,k \leq n$;
- Homogeneidade: $\det([\mathbf{a}_1 \ \cdots \ b\mathbf{a}_j \ \cdots \ \mathbf{a}_n]) = b.\det([\mathbf{a}_1 \ \cdots \ \mathbf{a}_j \ \cdots \ \mathbf{a}_n])$

Com isso, se **B** é uma matriz obtida de **A** pela troca de duas de suas colunas, então $\det(\mathbf{B}) = -\det(\mathbf{A})$.

Traço

Dada uma matriz **A** de dimensão $n \times n$, o traço de **A**, representado por $\text{tr}(\mathbf{A})$, é a soma dos elementos da diagonal de **A**, ou seja:

$$\text{tr}(\mathbf{A}) = \sum_{i=1}^{n} a_{ii}$$

Adjunta

Dada uma matriz **A** de dimensão $n \times n$, a adjunta de **A**, representada por adj(**A**), é dada na forma:

$$\text{adj}(\mathbf{A}) = \{ a'_{ij} \}$$

onde $a'_{ij} = c_{ji}$, o cofator do elemento a_{ji}.

São válidas as seguintes igualdades: $\begin{cases} \mathbf{A}.\text{adj}(\mathbf{A}) = \det(\mathbf{A}).\mathbf{I} \\ \text{adj}(\mathbf{A}).\mathbf{A} = \det(\mathbf{A}).\mathbf{I} \end{cases} \Rightarrow \mathbf{A}^{-1} = \dfrac{\text{adj}(\mathbf{A})}{\det(\mathbf{A})}$

Singularidade

Diz-se que uma matriz **A** de dimensão $n \times n$ é singular se $\det(\mathbf{A}) = 0$. Caso contrário, diz-se que **A** é não-singular. Como $\mathbf{A}^{-1} = \text{adj}(\mathbf{A}) / \det(\mathbf{A})$, **A** admite inversa se e somente se $\det(\mathbf{A}) \neq 0$, ou seja, se **A** é não-singular.

Range e posto

O range de uma matriz **A** de dimensão $m \times n$ é um subespaço do $\Re^m$ definido na forma:

$$R(\mathbf{A}) = \left\{ \mathbf{y} \in \Re^m : \mathbf{y} = \mathbf{A}\mathbf{x}, \text{ para algum } \mathbf{x} \in \Re^n \right\}.$$

Portanto, $R(\mathbf{A})$ é dado pelo conjunto de todas as possíveis combinações lineares das colunas de **A**. A dimensão de $R(\mathbf{A})$ é denominada posto da matriz **A** e é representado por $\rho(\mathbf{A})$. O posto é o número de colunas linearmente independentes de **A**.

Como $\rho(\mathbf{A}) = \rho(\mathbf{A}^T)$, o posto pode ser também considerado como sendo o número de linhas linearmente independentes de **A**. Se $n \geq m$ e $\rho(\mathbf{A}) = m$, então $R(\mathbf{A}) \equiv \Re^m$, donde se conclui que

$$\dim\{R(\mathbf{A})\} = \rho(\mathbf{A}) = \rho(\mathbf{A}^T) \leq \min(m,n).$$

Nulidade

O nulo de uma matriz **A** de dimensão $m \times n$ é um subespaço do $\Re^n$ definido na forma:

$$N(\mathbf{A}) \triangleq \left\{ \mathbf{x} \in \Re^n : \mathbf{A}\mathbf{x} = \mathbf{0} \right\}$$

A dimensão de $N(\mathbf{A})$ é denominada nulidade da matriz $\mathbf{A}$ e é representada por $\nu(\mathbf{A})$. A nulidade e o posto estão relacionados por:

$$\nu(\mathbf{A}) + \rho(\mathbf{A}) = n$$

São propriedades importantes: $N(\mathbf{A}) \perp R(\mathbf{A}^T)$ no $\Re^n$ $\quad N(\mathbf{A}^T) \perp R(\mathbf{A})$ no $\Re^m$

Autovalor e autovetor

Seja uma matriz $\mathbf{A}$ de dimensão $n \times n$. Diz-se que um escalar $\lambda \in \mathbf{C}$ (conjunto dos números complexos) é um autovalor de $\mathbf{A}$ se existe um vetor não-nulo $\mathbf{x} \in \mathbf{C}^n$, chamado de autovetor associado a λ, tal que

$$\mathbf{Ax} = \lambda\mathbf{x}.$$

- $\mathbf{Ax} = \lambda\mathbf{x}$ pode ser reescrito como $(\lambda\mathbf{I} - \mathbf{A})\mathbf{x} = \mathbf{0}$;
- $\exists\, \mathbf{x} \in \mathbf{C}^n,\ \mathbf{x} \neq \mathbf{0}$ tal que $(\lambda\mathbf{I} - \mathbf{A})\mathbf{x} = \mathbf{0}$ se e somente se $\det(\lambda\mathbf{I} - \mathbf{A}) = 0$;
- $\Delta(\lambda) \triangleq \det(\lambda\mathbf{I} - \mathbf{A})$ é o polinômio característico de $\mathbf{A}$;
- Como o grau de $\Delta(\lambda)$ é n, a matriz $\mathbf{A}$ possui n autovalores.

Positividade

Uma matriz $\mathbf{A}$ de dimensão $n \times n$ é dita ser definida positiva se e somente se

$$\langle \mathbf{x}, \mathbf{Ax} \rangle = \mathbf{x}^T\mathbf{Ax} > 0,\ \forall\, \mathbf{x} \in \Re^n,\ \mathbf{x} \neq \mathbf{0},$$

ou, de forma equivalente, se todos os seus autovalores forem positivos.

Uma matriz $\mathbf{A}$ de dimensão $n \times n$ é dita ser semi-definida positiva se e somente se

$$\langle \mathbf{x}, \mathbf{Ax} \rangle = \mathbf{x}^T\mathbf{Ax} \geq 0,\ \forall\, \mathbf{x} \in \Re^n,$$

ou, de forma equivalente, se todos os seus autovalores forem não-negativos.

Uma matriz $\mathbf{A}$ de dimensão $n \times n$ é dita ser definida negativa (semi-definida negativa) se $-\mathbf{A}$ é definida positiva (semi-definida positiva).

A.20 Teoremas envolvendo propriedades de matrizes

Teorema: Seja **A** uma matriz de dimensão $m \times n$ e considere o sistema de equações lineares $\mathbf{Ax} = \mathbf{y}$. É possível afirmar que:

1) Dados $\mathbf{A} \in \Re^{m \times n}$ e $\mathbf{y} \in \Re^m$, existe $\mathbf{x} \in \Re^n$ tal que $\mathbf{Ax} = \mathbf{y}$ se e somente se $\mathbf{y} \in R(\mathbf{A})$, ou seja, se e somente se **y** puder ser expresso como uma combinação linear das colunas de **A**.

   Nota: $\mathbf{y} \in R(\mathbf{A}) \Rightarrow \rho(\mathbf{A}) = \rho([\mathbf{A} : \mathbf{y}])$.

2) Dado $\mathbf{A} \in \Re^{m \times n}$, para todo $\mathbf{y} \in \Re^m$ existe $\mathbf{x} \in \Re^n$ tal que $\mathbf{Ax} = \mathbf{y}$ se e somente se $R(\mathbf{A}) \equiv \Re^m$.

   Nota: $R(\mathbf{A}) \equiv \Re^m \Rightarrow \rho(\mathbf{A}) = m$

Teorema: Seja **A** uma matriz de dimensão $m \times n$ tal que $\rho(\mathbf{A}) = m$. Então a matriz $\mathbf{Q} = \mathbf{AA}^T \in \Re^m$ é definida positiva.

Teorema: Seja **A** uma matriz de dimensão $n \times n$. Então os autovetores associados a autovalores distintos são linearmente independentes.

Teorema: Seja **A** uma matriz simétrica de dimensão $n \times n$. Então os autovalores de **A** são reais e autovetores associados a autovalores distintos são ortogonais.

Teorema: Toda matriz **A** de dimensão $m \times n$ que tenha posto unitário pode ser fatorada na forma $\mathbf{A} = \mathbf{uv}^T$, para algum $\mathbf{u} \in \Re^m$ e $\mathbf{v} \in \Re^n$.

Índice Remissivo de Autores