



ISAÍAS LIMA
CARLOS A. M. PINHEIRO
FLÁVIA A. OLIVEIRA SANTOS

Inteligência Artificial



LTC

Inteligência Artificial

Isaías Lima

Carlos A.M. Pinheiro

Flávia A. Oliveira Santos

OceanofPDF.com



Sumário

Capa

Folha de rosto

Cadastro

Copyright

Autores

Prefácio

Capítulo Um. Introdução

1.1 Definições

1.2 Histórico

1.3 Aprendizado de Máquina

1.4 Abordagens usuais em Inteligência Artificial

1.5 Áreas de aplicação da Inteligência Artificial

1.6 Consideração final

Referências bibliográficas

Capítulo Dois. Lógica Fuzzy

2.1 Introdução

2.2 Conjuntos nebulosos (*Fuzzy Sets*)

2.3 Operações básicas sobre conjuntos nebulosos

2.4 Relações Fuzzy e Lógica Fuzzy

2.5 Exemplos

2.6 Exercícios

Referências bibliográficas

Capítulo Três. Redes Neurais Artificiais

3.1 Introdução

3.2 Conceituação

3.3 Modelos de Redes Neurais Artificiais – RNAs

3.4 Topologias

3.5 Treinamento e generalização

3.6 Redes Clássicas – Perceptron e Adaline

3.7 Perceptrons de múltiplas camadas

3.8 Treinamento não supervisionado

3.9 Redes de Kohonen

3.10 Redes RBF

3.11 Outros tipos de Redes Neurais Artificiais

Referências bibliográficas

Capítulo Quatro. Desenvolvimento de Aplicações Utilizando Redes Neurais

4.1 Introdução

- 4.2 Definição do problema
- 4.3 Coleta de dados
- 4.4 Pré-Processamento dos dados
- 4.5 Separação em conjunto de dados
- 4.6 Configuração da rede
- 4.7 Treinamento da rede
- 4.8 Teste da rede
- 4.9 Integração da Rede ao Sistema e Manutenção
- Referências bibliográficas

Capítulo Cinco. Sistemas Neuro-Fuzzy

- 5.1 Introdução
- 5.2 Características dos sistemas Neuro-Fuzzy
- 5.3 Vantagens e desvantagens dos sistemas Neuro-Fuzzy
- 5.4 Arquitetura Neuro-Fuzzy
- 5.5 O algoritmo Fuzzy C-Means
- 5.6 Aplicações dos sistemas Neuro-Fuzzy
- Referências bibliográficas

Capítulo Seis. Conjuntos Aproximados (*Rough Sets*)

- 6.1 Introdução
- 6.2 Definições básicas
- 6.3 Matriz e função de discernibilidade, redutos e regras de decisão
- 6.4 Exemplos de aplicação
- 6.5 Outras abordagens de IA
- 6.6 Exercícios
- Referências bibliográficas

Índice

OceanofPDF.com

Cadastro



OceanofPDF.com

Copyright

© 2014, Elsevier Editora Ltda.

Todos os direitos reservados e protegidos pela Lei n. 9.610, de 19/02/1998.

Nenhuma parte deste livro, sem autorização prévia por escrito da editora, poderá ser reproduzida ou transmitida sejam quais forem os meios empregados: eletrônicos, mecânicos, fotográficos, gravação ou quaisquer outros.

Copidesque: Vânia Coutinho

Revisão Gráfica: Flor de Letras Editorial

Editoração Eletrônica: Thonsom Digital

Capa: Usina Digital

Elsevier Editora Ltda.

Conhecimento sem Fronteiras

Rua Sete de Setembro, 111 – 16º andar

20050-006 – Centro – Rio de Janeiro – RJ – Brasil

Rua Quintana, 753 – 8º andar

04569-011 – Brooklin – São Paulo – SP – Brasil

Serviço de Atendimento ao Cliente

0800-0265340

gendigital@grupogen.com

ISBN : 978-85-352-7808-8

ISBN (versão eletrônica) : 978-85-352-7809-5

Nota

Muito zelo e técnica foram empregados na edição desta obra. No entanto, podem ocorrer erros de digitação, impressão ou dúvida conceitual. Em qualquer das hipóteses, solicitamos a comunicação ao nosso Serviço de Atendimento ao Cliente, para que possamos esclarecer ou encaminhar a questão.

Nem a editora nem o autor assumem qualquer responsabilidade por eventuais danos ou perdas pessoas ou bens, originados do uso desta publicação.

CIP-BRASIL. CATALOGAÇÃO NA PUBLICAÇÃO SINDICATO NACIONAL DOS EDITORES DE LIVROS, RJ

I48

Inteligência artificial : Isaia Lima Lopes ; Flávia Aparecida Oliveira Santos ; Carlos Alberto Murari Pinheiro - 1. ed. - Rio de Janeiro : Elsevier, 2014.

il. ; 24 cm.

Inclui bibliografia

ISBN 978-85-352-7808-8

1. Inteligência artificial. 2. Computação. I. Título.

14-14298 CDD: 006.3

CDU: 004.81

OceanofPDF.com

Autores

Isaías Lima Lopes

B. Isaías Lima Lopes é professor Associado na Universidade Federal de Itajubá. Membro dos Grupos de Pesquisas: GESIS e GEEA. Professor dos cursos de pós-graduação em Engenharia Elétrica e Engenharia de Controle e Automação. Coordenador de projetos de pesquisa e desenvolvimento e de parcerias com Instituições e Empresas Nacionais e Internacionais. Pesquisador nas áreas de estabilidade e controle de sistemas dinâmicos e em Sistemas Elétricos de Potência, desenvolvimento e aplicações de técnicas de Inteligência Artificial em Sistemas de Controle e Automação. Professor do Curso de Especialização em Sistemas Elétricos de Potência (CESE). Membro do Instituto Nacional de Ciência e Tecnologia em Energia Elétrica (INERGE). Membro do Comitê Técnico da International Conference on Environment and Electrical Engineering IEEEIC. Membro efetivo da Red Iberoamericana de generación distribuida y microrredes eléctricas inteligentes (CYTED 2012) pertencente à instituição Universidad Politécnica de Cataluña. Revisor técnico e tradutor da Pearson Education do Brasil Ltda. Consultor dos cursos de graduação em Engenharias da Revista Guia do Estudante. Consultor e perito técnico contra pirataria de software da Microsoft Corporation.

Flávia Aparecida Oliveira Santos

Bacharel em Ciência da Computação pela Universidade José do Rosário Vellano (UNIFENAS) (1999). Especialista em Informática em Educação pela Universidade Federal de Lavras (UFLA), (2004). Especialista em Metodologias Inovadoras Aplicadas à Educação pela Faculdade Internacional de Curitiba (FACINTER), (2006). Mestre em Engenharia Elétrica pela Universidade Federal de Itajubá (UNIFEI), (2010). Doutoranda em Sistemas Elétricos de Potência e pesquisadora da Universidade Federal de Itajubá. Tem experiência em Análise de Sistemas, Engenharia de Software e Inteligência Artificial, atuou na pesquisa de ferramentas tecnológicas para Educação :

Distância. Pesquisa técnicas de IA aplicadas em Aproximações de Funções Modelagem de Sistemas Dinâmicos Não Lineares e Previsões de Séries Temporais. Atua como Professora do Instituto Federal de Educação, Ciência e Tecnologia do Sul de Minas – Câmpus Machado.

Carlos Alberto Murari Pinheiro

Engenheiro Eletricista (Ênfase Eletrônica) pela Escola Federal de Engenharia de Itajubá (EFEI). Mestre em Sistemas de Potência pela EFEI. Doutor em Sistemas de Controle pela Universidade Estadual de Campinas. Professor na Escola Federal de Engenharia de Itajubá (atual Universidade Federal de Itajubá) desde 1983. Trabalha nas áreas de Sistemas de Controle, Processamento Digital de Sinais, Modelagem e Identificação de Sistemas Dinâmicos, Eletrônica de Potência e Inteligência Artificial Aplicada.

OceanofPDF.com

Prefácio

Este livro foi escrito com a finalidade de ser um texto inicial sobre Inteligência Artificial em disciplinas de graduação e pós-graduação em cursos de Computação, Engenharia, Matemática etc.

O conteúdo foi desenvolvido a partir das experiências dos autores ao lecionar disciplinas e orientar dissertações/teses relacionadas a temas de Inteligência Artificial em programas de extensão, graduação e pós-graduação nos cursos de Engenharia Elétrica, Automação e Computação da Universidade Federal de Itajubá.

Técnicas de Inteligência Artificial são empregadas com sucesso em várias áreas como Aprendizado de Máquinas, Automação de Processos, Jogos, Mineração de Dados, Robótica, Processamento de Imagens, Sons e Outros Sinais, Processos de Decisão, Sistemas de Controle, e muitas outras.

A área de Inteligência Artificial engloba atualmente várias abordagens como a Lógica Fuzzy, as Redes Neurais Artificiais, os Algoritmos Evolutivos etc.

Nesse contexto, é importante que os alunos de programas de graduação e pós-graduação em cursos de Computação, Engenharia, Matemática, e outros saibam os conceitos básicos relacionados com estas abordagens para exercerem plenamente as suas tarefas acadêmicas ou profissionais.

Este livro foi escrito com o objetivo de oferecer um texto didático, de fácil leitura e com uma conceituação básica relativamente simples e direta. Para esta finalidade foram compilados exemplos em contextos práticos, objetivos e de fácil entendimento, que possuem exemplificações numéricas e computacionais, objetivando ilustrar as aplicações dos conceitos apresentados. Alguns dos exemplos possuem listagens específicas que ilustram suas realizações computacionais. Para completar o aprendizado, alguns exercícios são propostos no final de cada capítulo, onde o leitor é encorajado a resolvê-los com o objetivo de aprimorar os conhecimentos adquiridos. No final de

cada capítulo desta publicação, encontram-se também as referências bibliográficas citadas no texto, nas quais os leitores podem buscar mais detalhes sobre os conceitos apresentados.

O nosso sincero desejo que este livro possa conduzir os leitores ao fascinante mundo da Inteligência Artificial.

Os Autores

Julho de 2014

OceanofPDF.com

CAPÍTULO

UM

Introdução

1.1 Definições

O termo Inteligência Artificial (IA) constitui vários procedimentos computacionais cujas funções realizadas, caso um ser humano as executasse seriam consideradas inteligentes. O conceito é amplo e recebe tantas definições quanto os significados diferentes da palavra inteligência.

O ato de forçar definições para as coisas que nós não entendemos geralmente causa mais danos do que benefícios. Além disso, apenas em lógica e matemática é que as definições detêm perfeitamente os conceitos.

As coisas com as quais lidamos na vida prática são frequentemente muito complicadas para permitirem uma representação clara baseada em expressões compactas. Em todo caso, não podemos nos privar de buscar uma definição para as coisas, no sentido de entender o que elas são.

Marvin Minsky, The Society of Mind, 1985

Esta área de pesquisa da ciência da computação tem como objetivo busca métodos ou sistemas computacionais que possuam ou reforcem a capacidade de comportamentos inteligentes do ser humano, como a de resolver problemas, adquirir e representar conhecimentos, reconhecer padrões etc.

Sabe-se, de forma genérica, que um sistema inteligente é aquele que apresenta capacidades como:

- aquisição de conhecimentos;
- planejamento de eventos;
- resolução de problemas;
- representações de informações;

- armazenamento de conhecimento;
- comunicação através de linguagens coloquiais;
- aprendizado.

1.2 Histórico

Os primeiros estudos sobre Inteligência Artificial surgiram na década de 1940 marcada pela Segunda Guerra Mundial. Neste período houve a necessidade de desenvolver uma tecnologia voltada para a análise de balística, quebra de códigos e cálculos para projetos de armas nucleares. Surgiam, então, os primeiros grandes projetos de construção de computadores, assim chamados por serem máquinas utilizadas para fazer cálculos (*cômputos*).

Após a Segunda Guerra Mundial, o computador não ficou restrito aos âmbitos militar e científico, sendo gradualmente utilizado em empresas, indústrias, universidades etc. A diversidade de aplicações estimulou pesquisas de software, hardware e linguagens de programação.

O desenvolvimento do computador, primeiramente impulsionado pela aplicabilidade militar e posteriormente comercial, mostrou-se extremamente relevante. Seu rápido e contínuo desenvolvimento, desde o surgimento dos primeiros computadores com válvulas eletrônicas (Collossus na Inglaterra, em 1943, e Eniac nos Estados Unidos, em 1946), até o surgimento dos microcomputadores com circuitos integrados (microprocessadores) na década de 1970, demonstra que essa área contemplou muitas pesquisas e recebeu grandes investimentos.

Em 1956, John McCarthy reuniu em uma conferência proferida na *Dartmouth College*, na Universidade de New Hampshire, vários pesquisadores de renome para estudar o que foi denominado por Minsky, McCarthy, Newell e Simon de *Inteligência Artificial*, expressão utilizada para designar um tipo de inteligência construída pelo homem para dotar máquinas de comportamentos inteligentes. A partir da estruturação desse novo campo de conhecimento, essa área começou a ser pesquisada de forma intensa. Vários esforços foram e têm sido feitos no sentido de simular os tipos de comportamentos considerados inteligentes, utilizados pelo ser humano, e realizá-los em computadores via técnicas de IA.

A área da Inteligência Artificial é atualmente bastante utilizada em vários campos de aplicações, auxiliando a capacidade humana em projetos e no

desenvolvimento de novos sistemas e produtos e, ainda, substituindo muitas atividades repetitivas e enfadonhas de operadores humanos em diversas funções produtivas. Isso se tornou possível graças ao desenvolvimento dos sistemas especialistas, da Lógica Fuzzy, das Redes Neurais Artificiais, dos Algoritmos Evolucionários e outros.

As áreas de IA são descritas em várias publicações (livros, artigos em revistas científicas, dissertações e teses), gerando milhares de estudos relacionados, produtos e serviços diversos, consumindo bilhões de dólares em projetos e pesquisas e envolvendo pesquisadores de instituições governamentais, militares, industriais e universitárias de todo o mundo. Um exemplo comercial consagrado da área de IA é o procedimento de pesquisa utilizado pelo buscador da Google.

As datas indicadas a seguir ilustram um breve histórico da evolução da Inteligência Artificial ao longo dos anos.

(1943-1950) Pesquisas voltadas ao desenvolvimento de modelos de neurônios artificiais (McCulloch & Pitts, 1943), que possibilitariam o desenvolvimento de máquinas que fossem capazes de aprender.

1.2.1 A Máquina de Turing

Em 1950 Alan Turing publicou um artigo chamado “Computing Machine and Intelligence”. Neste artigo, Turing apresentou, pela primeira vez, o que hoje é conhecido por Teste de Turing, com o qual se pretendia descobrir se uma máquina podia ou não emular o pensamento humano. O Teste de Turing funciona da seguinte forma: um interrogador (humano) fará perguntas a duas entidades ocultas; uma delas é um humano, e a outra é um computador. A comunicação entre o interrogador e as entidades é feita de modo indireto, pelo teclado, por exemplo. O interrogador tentará, através do “diálogo” realizado entre ele e as entidades, decidir qual dos dois é o humano. O computador será programado para se passar por humano, e o humano responderá de forma a confirmar a sua condição. Se, no final do teste, o interrogador não conseguir distinguir quem é o humano, então conclui-se que o computador pode “pensar” segundo o Teste de Turing. Em um de seus ensaios, Turing disse “acredito que no fim do século o uso da palavra e a opinião geralmente educada terão se alterado tanto que alguém será capaz de falar de máquinas pensantes sem ser contraditado”. Porém já chegamos ao fim do século e

nenhuma máquina conseguiu passar, consistentemente, pelo Teste de Turing. Alguns computadores, devidamente programados, conseguiram passar por versões simplificadas do teste, contudo sempre esteve ausente o atributo mental do entendimento. Como Marvin Minsky, do MIT, disse, “o maior desafio é dar bom senso às máquinas, e bom senso é essencial para passar no Teste de Turing”. [Russell e Norvig \(1995\)](#) observaram que programar um sistema de computador para passar no Teste de Turing é uma tarefa muito difícil. Tal sistema precisaria ter pelo menos as seguintes capacidades:

- processamento de linguagem natural para se comunicar com o usuário;
- representação de conhecimento para armazenar o que sabe ou aprende;
- raciocínio automatizado para usar o conhecimento armazenado com a finalidade de responder perguntas ou tirar novas conclusões;
- aprendizado de máquina para se adaptar a novas circunstâncias, detectar e extrapolar padrões, a fim de atualizar o seu conhecimento armazenado.

(1951-1969) Os primeiros programas capazes de jogar xadrez (Shannon 1950, e Turing, 1953). Prova de teoremas de lógica e emulação da forma de raciocínio do ser humano (Newell & Simon, 1956). Planejamento de tarefa (Green, 1963), comunicação em linguagem natural (Weizenbaum, 1965) aprendizado por analogia (Evans, 1968) e análises estruturais de moléculas químicas (Buchan *et al.*, 1969). Foi uma fase de grande entusiasmo, já que, até bem pouco tempo atrás, o computador era visto meramente como uma simples máquina de calcular.

(1970-1980) Nessa fase os pesquisadores começaram a esbarrar em problemas relacionados ao armazenamento de informações e ao tempo de processamento dos dados. Com o surgimento da Teoria da Complexidade Computacional (Cook, 1971), ficou comprovado que a solução de problemas computacionais não dependia apenas de memórias adicionais ou de processadores mais rápidos. Em 1981, pesquisadores japoneses anunciaram um projeto de computador de quinta geração, que teria o Prolog como linguagem de programação e seria capaz de realizar milhões de inferências por segundo. Recebendo o domínio japonês, grandes investimentos começaram a ser feitos na Europa e nos Estados Unidos. Em decorrência, muitas aplicações práticas foram desenvolvidas em áreas específicas como manufatura, robótica, processamento de imagens etc.

1.3 Aprendizado de Máquina

Nas últimas décadas, devido ao aumento do poder computacional e do crescimento das técnicas de IA, a linha de pesquisa conhecida como Aprendizado de Máquina vem ganhando importância. Essa linha de pesquisa tem por objetivo estudar e desenvolver métodos computacionais para a obtenção de sistemas capazes de adquirir conhecimento de forma automática. O desafio principal dos algoritmos de aprendizagem é maximizar a capacidade de generalização de seu aprendiz.

Nesse contexto, as Redes Neurais Artificiais, a Lógica Fuzzy, os Algoritmos Evolucionários, os Modelos Híbridos e a Teoria dos Conjuntos Aproximados (Rough Sets) apresentam-se como alternativas promissoras para a resolução desse desafio.

Partindo dessa premissa, este livro tem por objetivo explicar os conceitos básicos e mostrar algumas aplicações práticas destas técnicas em contextos de exemplos numéricos e computacionais. O desafio principal dos algoritmos de aprendizagem é maximizar a capacidade de generalização do seu aprendiz (Braga *et al.*, 2000). Entende-se por generalização a habilidade de uma máquina responder, de forma satisfatória, a dados ou amostras do mapeamento de entrada/saída não conhecidos durante o processo de aprendizagem, como no critério de desempenho baseado em observações de um determinado problema (Mitchell, 1997).

Algoritmos de Aprendizado de Máquina têm como objetivo descobrir o relacionamento entre as variáveis de um sistema (entrada/saída) a partir de dados amostrados. Estes algoritmos se originam de muitas áreas de aplicações: Estatística, Física, Engenharia, Computação etc.

Quando os relacionamentos entre todas as variáveis do problema (entrada/saída) não são completamente compreendidos, os Algoritmos de Aprendizagem de Máquina são necessários; este é o caso de muitos dos problemas reais com os quais nos defrontamos no cotidiano.

O Aprendizado de Máquina pode ser classificado da seguinte maneira:

- Aprendizado Supervisionado;
- Aprendizado por Reforço;
- Aprendizado Não Supervisionado.

1.4 Abordagens usuais em Inteligência Artificial

Com base nos diversos campos de estudos que fundamentaram os princípios teóricos da Inteligência Artificial, têm-se as três abordagens características citadas a seguir.

- **Simbólica:** baseada na hipótese de um sistema de símbolos definido, em que há um conjunto de estruturas simbólicas e um conjunto de regras de manipulação dessas estruturas, pelos quais têm-se os meios necessários e suficientes para se criar comportamento inteligente. Essa abordagem trata de problemas bem definidos (p.ex., planejamento de tarefas), e sua principal contribuição são os sistemas especialistas. Também denominada descendente ou cognitiva, dá ênfase aos processos cognitivos, ou seja, a forma como o ser humano raciocina. Objetiva encontrar uma explicação para comportamentos inteligentes com base em aspectos psicológicos e processos algorítmicos. Os pioneiros dessa corrente foram John McCarthy, Marvin Minsky, Newell e Simon. As primeiras modelagens da inteligência surgiram na década de 1950 e tiveram como base as regras de produção e a lógica dos predicados. A formalização da lógica facilitou o processo de formalização e representação dos conhecimentos a serem utilizados pelos programas de computador. Inicialmente, esses conhecimentos se restringiram a esquemas de raciocínios para jogos, aplicações em provas de teoremas matemáticos e simuladores específicos.
- **Conexionista:** baseia-se na hipótese de causa-efeito, segundo a qual um modelo suficientemente preciso de neurônios biológicos basta para reproduzir a inteligência humana. Essa abordagem trata de problemas imprecisos, mas que podem ser definidos através de exemplos (p.ex., reconhecimento de caligrafia), e sua principal contribuição são as redes neurais artificiais. Também denominada biológica ou ascendente, dá ênfase ao modelo de funcionamento do cérebro, dos neurônios e das conexões neurais. Os pioneiros dessa corrente foram McCulloch, Pitts, Hebb, Rosenblatt e Widrow. Em 1943 introduziu-se a representação matemática dos neurônios artificiais, que fez surgir os primeiros modelos de redes neurais artificiais. A corrente *conexista* sofreu grande impacto quando os cientistas Marvin Minsky e Seymour Papert publicaram (1969) o livro *Perceptrons*, no qual criticaram e garantiram que os modelos das

redes neurais não tinham sustentação matemática suficiente para que lhes fosse possível atribuir alguma confiabilidade. Apesar das pesquisas nesta área não terem parado, foi apenas na década de 1980 que o físico e biólogo do Instituto de Tecnologia da Califórnia, John Hopfield, conseguiu recuperar a credibilidade da utilização das redes neurais.

- **Evolucionária:** baseia-se na teoria evolutiva de Darwin. A hipótese é que podemos modelar sistemas inteligentes simulando a evolução de uma população de indivíduos, que carregam genes com informação suficiente para dar origem à solução de um problema. São utilizados procedimentos computacionais com base em recombinações e mutações genéticas. Essa abordagem é aplicada com sucesso em problemas de otimização, e sua vertente mais conhecida são os algoritmos genéticos.
- **Conjuntos Difusos, Conjuntos Aproximados:** abordagens baseadas na teoria dos conjuntos difusos ou nebulosos (*fuzzy sets*) e dos conjuntos aproximados (*rough sets*). Os conjuntos nebulosos são apropriados a interpretações linguísticas que facilitam as representações dos modelos matemáticos resultantes. Os conjuntos aproximados são adequados ao tratamento de informações que apresentem inconsistências.
- A combinação das abordagens citadas é denominada Modelagem Híbrida (Barreto, 2001), na qual são associadas diversas técnicas de IA para se obter uma solução para um determinado problema.

1.5 Áreas de aplicação da Inteligência Artificial

Algumas das áreas de aplicação da Inteligência Artificial:

- **Matemática:** demonstração de teoremas, resolução simbólica de equações, aproximação de funções complexas etc.;
- **Pesquisa operacional:** otimização e busca heurística em geral;
- **Jogos:** xadrez, damas, go etc.;
- **Processamento de linguagem natural:** tradução automática, verificadores ortográficos e sintáticos, interfaces para BDs etc.;
- **Sistemas tutores:** modelagem de aprendizados, escolha de estratégias pedagógicas etc.;
- **Percepção:** visão, tato, audição, olfato, paladar e outras;
- **Robótica:** visão, navegação, controle etc.;

- **Sistemas especialistas:** atividades que exigem conhecimento especializado, e não formalizado; extração de conhecimentos; regras ou informações em tarefas como diagnósticos, previsão, monitoramento, análises, planejamentos e projeto; e em áreas como Medicina, Finanças, Engenharia, Artes, Computação e outras.

1.6 Consideração final

A Inteligência Artificial é uma ampla área de pesquisa que se subdivide em diversas subáreas, cada uma delas adota diferentes abordagens e trata variados problemas que, em geral, são de alta complexidade e para os quais ainda não há soluções convencionais satisfatórias.

Referências bibliográficas

1. BARRETO JM. *Inteligência Artificial no Limiar do Século XXI – Abordagem Híbrida, Simbólica*. Conexionista e Evolucionária: UFSC; 2001.
2. BRAGA AP, CARVALHO ACPLF, LUDERMIR TB. *Redes Neurais Artificiais: teoria e aplicações*. Editora LTC 2000.
3. GENESERETH MR, NILSSON NJ. *Logical Foundations of Artificial Intelligence*. Morgan Kaufmann Publishers 1988.
4. MITCHELL TM. *Machine Learning*. McGraw-Hill 1997.
5. RICH E, KNIGHT K. *Inteligência Artificial*. 2ª ed. Makron Books 1995.
6. RUSSELL S, NORVIG P. *Artificial Intelligence – A Modern Approach*. Prentice Hall 1995.

CAPÍTULO

DOIS

Lógica Fuzzy

2.1 Introdução

A Teoria dos Conjuntos Difusos (*Fuzzy Sets*) foi proposta por [Zadeh \(1965\)](#) como alternativa na representação de informações imprecisas ou vagas. A partir de uma das primeiras aplicações práticas dos conceitos desta teoria por [Mamdani \(1974\)](#), esta área da IA contabiliza milhares de publicações entre artigos, livros e teses relacionadas a desenvolvimentos teóricos realizados neste campo, e às diversas aplicações desenvolvidas em áreas como tomada de decisões, agrupamentos de dados, classificações de padrões, sistemas de diagnósticos, processamento de sinais (sons, imagens etc.), sistemas de controle, e muitas outras.

A Teoria dos Conjuntos Nebulosos (ou difusos) pode ser classificada como uma extensão da teoria dos conjuntos, e a forma como as informações são representadas e processadas neste contexto pode ser vista como uma extensão da lógica clássica, ficando conhecida como Lógica *Fuzzy*.

2.2 Conjuntos nebulosos (*Fuzzy Sets*)

Na lógica clássica as informações são quantificadas como “verdadeiras” ou “falsas”, e em representações correspondentes na álgebra booleana são classificadas como “1” ou “0”, respectivamente. Seja a representação baseada em regras [\(2.1\)](#), que assumiremos estar relacionada a um exemplo simplificado associado a um sistema especialista ([Patterson, 1990](#)) desenvolvido para diagnóstico de doenças. A informação da variável “ x_1 ” referente à temperatura corporal, por exemplo, associada ao estado febril (temperatura corpórea tipicamente entre 37° e 38°), está representada graficamente na [Figura 2.1](#). O grau de pertinência associado à informação booleana correspondente assume o valor “1” para um universo de discurso em

valores compreendidos no intervalo $[37, 38]$, ou o grau de pertinência “0” para temperaturas nos intervalos $T^\circ < 37$ e $T^\circ > 38$. O conjunto de valores associados ao “Estado Febril” é chamado de convencional ou *crisp*, estando de acordo com as representações típicas da teoria (clássica) dos conjuntos.

$$r_i : \text{IF } x_1 = \text{Estado Febril} \quad \text{AND } x_2 = \dots \text{AND } \dots \text{ THEN Diagnóstico} = Z_i \quad (2.1)$$

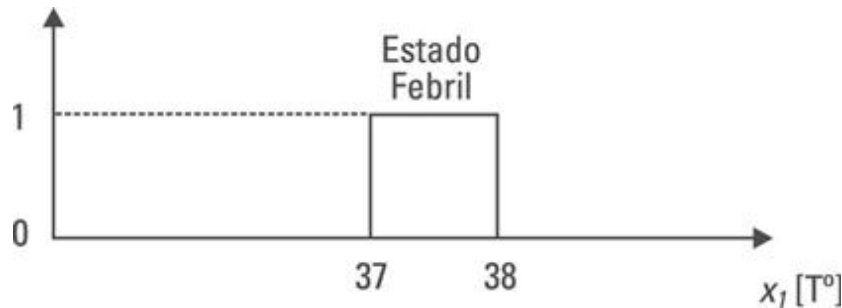


FIGURA 2.1 Ilustração gráfica de um conjunto clássico relacionado ao estado febril.

Em um conjunto nebuloso o grau de pertinência associado a um determinado conjunto também possui os valores limites “0” e “1”, mas pode assumir infinitos valores na faixa $[0, 1]$. A proposta dos conjuntos difusos difere de outra anterior apresentada por Lukasiewicz (Klir e Folger, 1988) cujos valores de pertinência também foram definidos no intervalo $[0, 1]$, mas utilizando-se de valores discretos entre “0” e “1”.

Formalmente um conjunto *fuzzy* F em um universo de discurso U é caracterizado por uma função de pertinência de grau μ_F que assume valores no intervalo $[0, 1]$, ou seja, $\mu_F: U \rightarrow [0, 1]$. Genericamente um conjunto *fuzzy* F em U pode ser representado como um conjunto de pares ordenados compostos de valores de uma variável u e os respectivos valores de grau de pertinência $\mu(u)$, com a notação expressa por (2.2).

$$F = \{(u, \mu_F(u)) \mid u \in U\} \quad (2.2)$$

Os conjuntos nebulosos são expressos por funções de pertinência diversas como as ilustradas na [Figura 2.2](#) (neste caso, conhecidas como triangulares), e aqui estão relacionadas ao exemplo referenciado na [Figura 2.1](#), onde são considerados agora três possíveis estados de temperatura corporal: *Normal*; *Febril*; *Febre*.

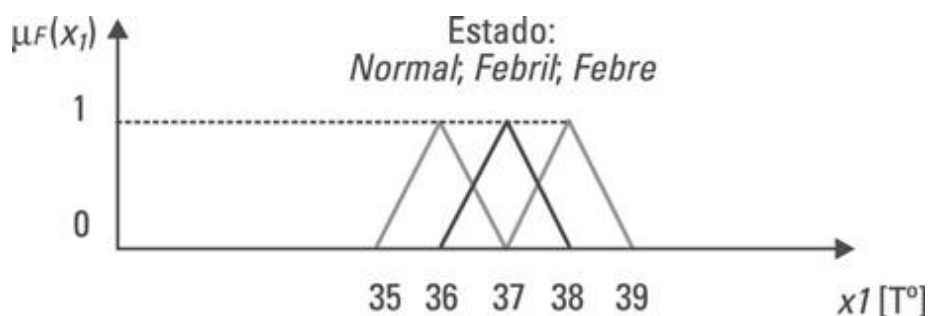


FIGURA 2.2 Ilustração gráfica de conjuntos nebulosos.

A [Figura 2.3](#) ilustra os valores modais (parâmetros) de uma função de pertinência triangular típica, e a expressão (2.3) representa os valores associados a ela.

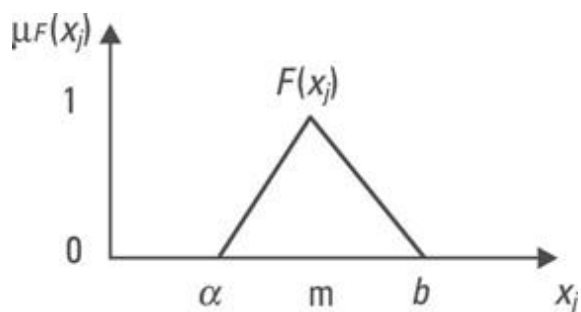


FIGURA 2.3 Função de pertinência triangular.

$$\mu_F(x_j) = \begin{cases} 0, & \text{se } x_j < a; \\ \frac{x_j - a}{m - a}, & \text{se } x_j \in [a, m]; \\ \frac{b - x_j}{b - m}, & \text{se } x_j \in [m, b]; \\ 0, & \text{se } x_j > b. \end{cases} \quad (2.3)$$

No exemplo ilustrado na [Figura 2.2](#), o grau de pertinência relativo a cada conjunto nebuloso associado à temperatura e os seus respectivos estados (*Normal*, *Febril* e *Febre*) é simbolizado por $\mu_F(x_1)$. Por exemplo, para $x_1 = 37$ têm-se: $\mu_{\text{Normal}}(x_1) = 0$; $\mu_{\text{Febril}}(x_1) = 1$; $\mu_{\text{Febre}}(x_1) = 0$.

Esta conversão da informação de temperatura (ou de outra grandeza qualquer) em um grau de pertinência entre $[0, 1]$ é conhecida como **fuzificação**.

Considerando agora $x_1 = 37,5$ têm-se: $\mu_{\text{Normal}}(x_1) = 0$; $\mu_{\text{Febril}}(x_1) = 0,5$; $\mu_{\text{Febre}}(x_1) = 0,5$. Onde neste caso os conjuntos correspondentes aos estados “*Febril*” e “*Febre*” possuem o mesmo grau de pertinência (0,5). Já para $x_1 = 37,75$ temos: $\mu_{\text{Normal}}(x_1) = 0$, $\mu_{\text{Febril}}(x_1) = 0,25$ e $\mu_{\text{Febre}}(x_1) = 0,75$ (que agora apresenta um maior grau de pertinência para o estado em questão do que em relação aos outros estados).

As escolhas ou atribuições de valores dos parâmetros das funções de pertinência **possibilitam a modelagem de informações imprecisas ou vagas** de uma forma adequada, sendo esta uma das vantagens da utilização dos conjuntos difusos.

Outra vantagem diz respeito aos nomes ou denominações atribuídas aos conjuntos nebulosos, que possibilitam representar informações linguísticas na forma de etiquetas ou rótulos como *Normal*, *Alta*, *Baixa*, *Pequena*, *Média*, *Grande* etc. que também facilitam a modelagem de informações de uma forma **qualitativa** e não somente quantitativa.

Outros exemplos de funções de pertinências frequentemente utilizadas são descritas a seguir, lembrando também que é possível empregar conjuntos

convencionais como o **crisp** exemplificado na [Figura 2.1](#), ou de valor singular (**singleton**) na qual o grau de pertinência é unitário para o valor modal em questão e nulo, caso contrário.

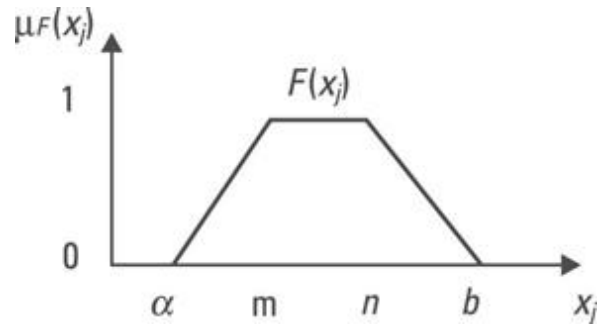


FIGURA 2.4 Função de pertinência trapezoidal.

$$\mu_F(x_j) = \begin{cases} 0, & \text{se } x_j < a; \\ \frac{x_j - a}{m - a}, & \text{se } x_j \in [a, m]; \\ 1, & \text{se } x_j \in [m, n]; \\ \frac{b - x_j}{b - n}, & \text{se } x_j \in [n, b]; \\ 0, & \text{se } x_j > b. \end{cases} \quad (2.4)$$

$$\mu_F(x_j) = e^{-\frac{(x_j - m)^2}{a^2}} \quad (2.5)$$

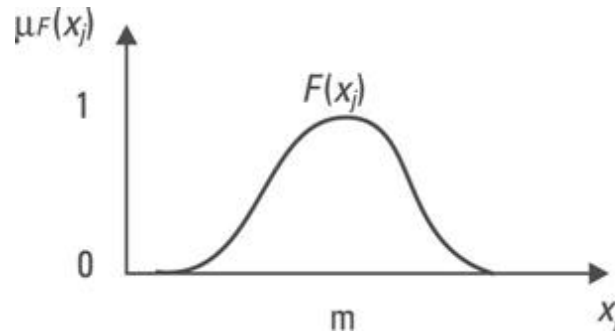


FIGURA 2.5 Função de pertinência *gaussiana*.

A [Figura 2.6](#) ilustra dois conjuntos nebulosos (A e B) relacionados com a velocidade (v) de um veículo, por exemplo, que receberam as denominações “Baixa” e “Média”. Os valores de pertinência e os dados que definem estes conjuntos podem ser representados na forma a seguir com seus respectivos pares ordenados.

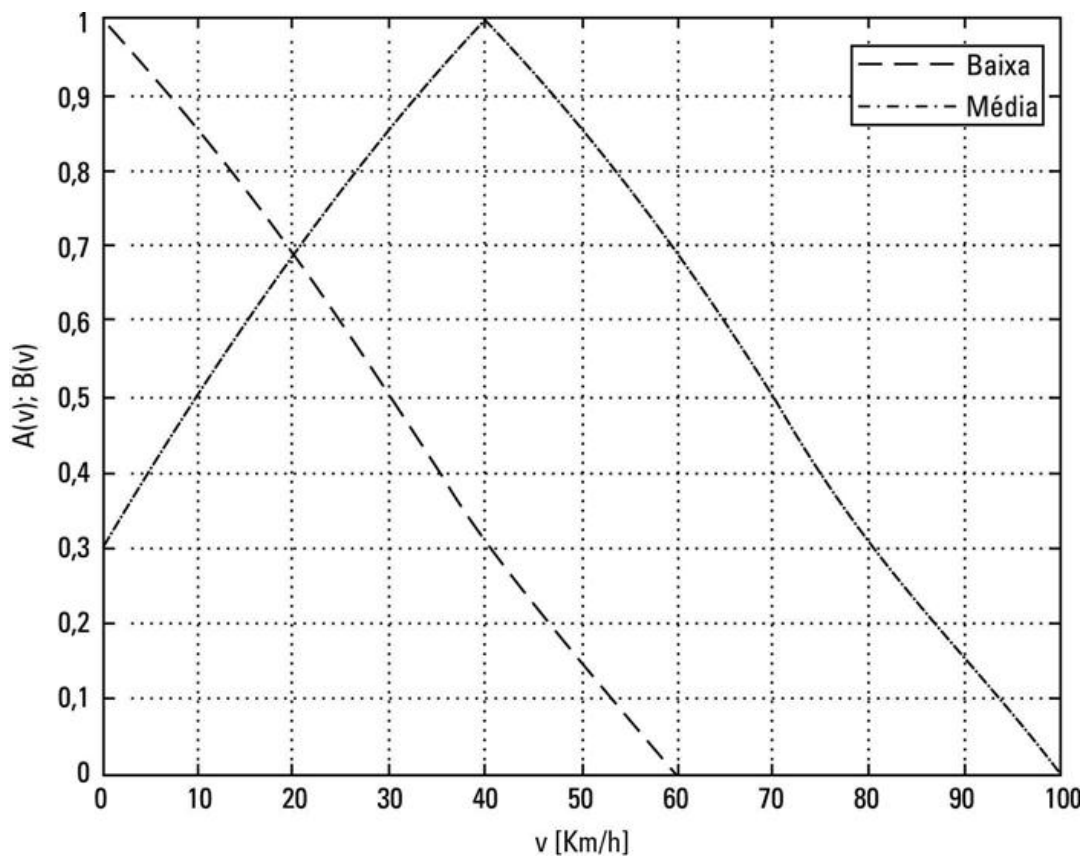


FIGURA 2.6 Ilustração dos dados de dois conjuntos nebulosos.

2.3 Operações básicas sobre conjuntos nebulosos

As operações básicas sobre conjuntos nebulosos são definidas a partir de normas conhecidas da teoria dos conjuntos, como as descritas a seguir.

2.3.1 T-norma

Uma t-norma é uma operação binária que satisfaz as condições de:

- Comutatividade: $x \text{ t } y = y \text{ t } x$;
- Associatividade: $x \text{ t } (y \text{ t } z) = (x \text{ t } y) \text{ t } z$;
- Monotonicidade: Se $x \leq y$ e $w \leq z$ então $x \text{ t } w \leq y \text{ t } z$;
- Condições limite: $0 \text{ t } x = 0$; $1 \text{ t } x = x$.

2.3.2 S-norma

Uma s-norma é uma operação binária que satisfaz as condições de:

- Comutatividade: $x \text{ s } y = y \text{ s } x$;
- Associatividade: $x \text{ s } (y \text{ s } z) = (x \text{ s } y) \text{ s } z$;
- Monotonicidade: Se $x \leq y$ e $w \leq z$ então $x \text{ s } w \leq y \text{ s } z$;
- Condições-limite: $0 \text{ s } x = x$; $1 \text{ s } x = 1$.

Na álgebra booleana as operações típicas compreendem as seguintes funções lógicas básicas: “AND”; “OR”; “NOT”. Em conjuntos nebulosos foram definidas operações típicas baseadas em t-normas e s-normas, onde as operações básicas são o mínimo (**min**), o produto algébrico (*) e o máximo (**max**). O conceito do **complemento** de um conjunto nebuloso $A(x)$ é definido como $C(x) = 1 - A(x)$.

2.3.3 Algumas t-normas utilizadas em conjuntos nebulosos

- Interseção: $x \text{ t } y = \min(x, y)$;
- Produto Algébrico: $x \text{ t } y = x * y$;

$$x \text{ t } y = \begin{cases} x, & \text{se } y = 1 \\ y, & \text{se } x = 1 \\ 0, & \text{c.c.} \end{cases}$$

- Produto Drástico:
- Etc.

2.3.4 Algumas s-normas utilizadas em conjuntos nebulosos

- União: $x \text{ s } y = \max(x, y)$;
- Soma Algébrica: $x \text{ s } y = x + y - x * y$;

$$x \text{ s } y = \begin{cases} x, & \text{se } y = 0 \\ y, & \text{se } x = 0 \\ 1, & \text{c.c.} \end{cases}$$

- Soma Drástica:
- Etc.

Tomando como base o exemplo ilustrado graficamente na [Figura 2.6](#), a [Figura 2.7](#) ilustra a aplicação da operação **min** entre os conjuntos A e B definindo outro conjunto D ilustrado em linha cheia na figura, cujos respectivos pares ordenados estão expressos a seguir.

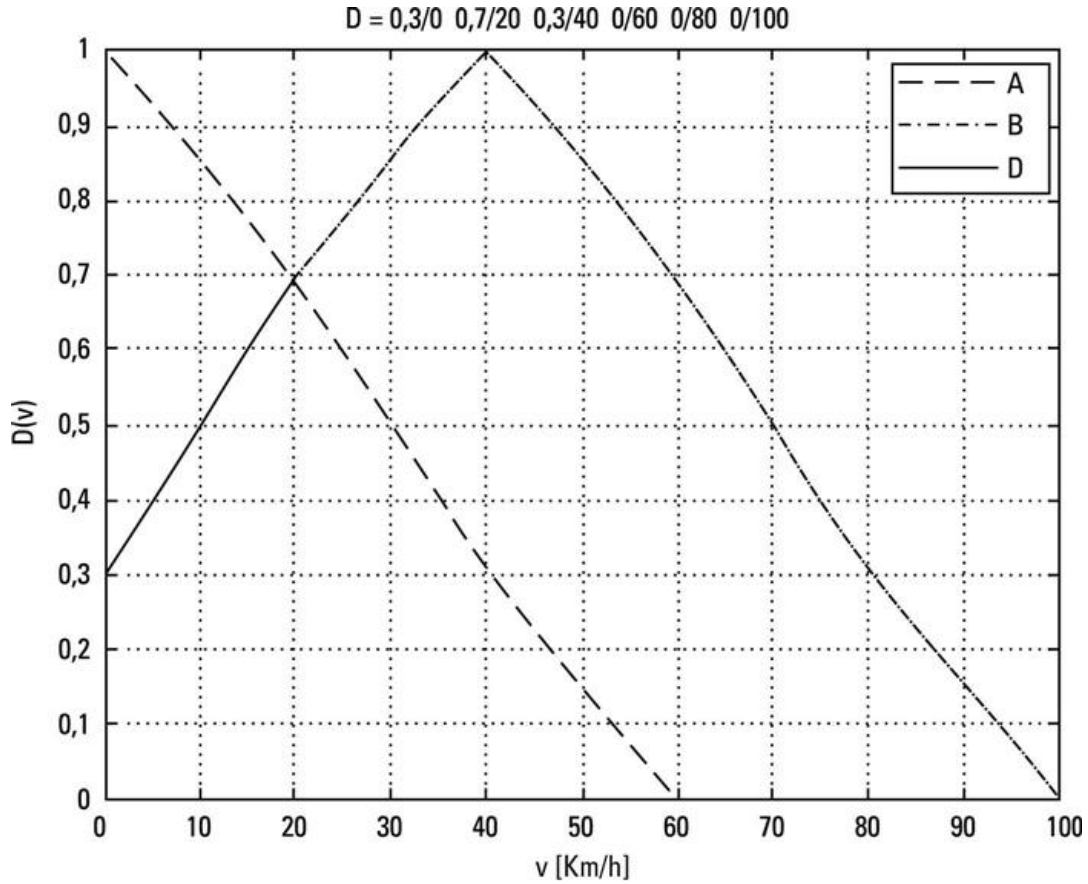


FIGURA 2.7 Ilustração gráfica da operação $\min$ entre os conjuntos A e B.

2.3.5 Medidas de possibilidade e necessidade

Objetivando comparações entre conjuntos nebulosos foram definidas as medidas de possibilidade e necessidade expressas por (2.6) e (2.7) respectivamente, onde “ $\sup$ ” denota valor supremo e “ $\inf$ ” valor inferior.

$$\text{Poss}(A, B) = \sup[\min(A(x), B(x))] \quad (2.6)$$

$$\text{Nec}(A, B) = \inf[\max(A(x), 1 - B(x))] \quad (2.7)$$

2.4 Relações Fuzzy e Lógica Fuzzy

Sejam X e Y universos de discurso de um conjunto de dados. A relação R em $X \times Y$ é um conjunto do produto cartesiano $R: X \times Y \rightarrow \{0, 1\}$. Exemplo: Para $U = \{1, 2, 3\}$, $V = \{1, 2, 3, 4\}$, $\mu_A(u)/u = 1/1 + 0,7/2 + 0,3/3$, $\mu_B(v)/v = 0,8/1 + 0,6/2 + 0,4/3 + 0,2/4$, e considerando que seja adotado o produto algébrico como operador entre os conjuntos, então o produto cartesiano resultante $A \times B$ é dado por:

$$A \times B = 0,8 / (1, 1) + 0,6 / (1, 2) + 0,4 / (1, 3) + 0,2 / (1, 4) + 0,56 / (2, 1) + 0,42 / (2, 2) + 0,28 / (2, 3) + 0,14 / (2, 4) + 0,24 / (3, 1) + 0,18 / (3, 2) + 0,12 / (3, 3) + 0,06 / (3, 4).$$

O resultado anterior pode ser representado por uma matriz de relação R conforme indicado na forma tabular ilustrada a seguir.

u/v	1	2	3	4	
1	0,8	0,6	0,4	0,2	
2	0,56	0,42	0,28	0,14	
3	0,24	0,18	0,12	0,06	

2.4.1 Composição multirrelacional

Se R e S são relações *fuzzy* nos espaços cartesianos $U \times V$ e $V \times W$, a composição de R e S é uma relação dada por uma *sup-t* composição expressa por (2.8).

$$R \circ S = \{[\sup(\mu_R(u, v) \text{ t } \mu_S(v, w))], u \in U, v \in V, w \in W\} \quad (2.8)$$

Considerando o exemplo de sentença ou regra (2.9), tem-se a composição relacional (2.10).

$$IF \ x_1 = Febre \ THEN \ Diagnóstico \ Z \quad (2.9)$$

$$x_1 = Febre \circ R \rightarrow y = Diagnóstico \ Z \quad (2.10)$$

Se x e y estão associados a conjuntos difusos A e B , pode-se escrever a composição geral (2.11).

$$B_i(\gamma) = \sup[A(x) \circ R_i(x, \gamma)] \quad (2.11)$$

Os chamados sistemas *fuzzy* possuem representações baseadas em modelos relacionais ou por intermédio de regras na forma de (2.12) e (2.13), que são conhecidas como regras linguísticas ou de Mamdani (1976), e funcionais ou de Takagi-Sugeno (1985). Onde $\alpha_i, \beta_i, \varphi_i, \sigma_i$ são conjuntos nebulosos e $c_{i,0}, c_{i,1}, c_{i,2}, \dots, c_{i,N}$ definem coeficientes numéricos da função do modelo *fuzzy* considerado.

$$r_i : IF \ x_1 = \alpha_i \ AND \ x_2 = \beta_i \ AND \ \dots \ AND \ x_N = \varphi_i \ THEN \ y = \sigma_i \quad (2.12)$$

$$r_i : IF \ x_1 = \alpha_i \ AND \ x_2 = \beta_i \ AND \ \dots \ AND \ x_N = \varphi_i \quad (2.13)$$

$$THEN \ y_i = c_{i,0} + c_{i,1}x_1 + c_{i,2}x_2 + \dots + c_{i,N}x_N$$

Dada uma relação difusa $R:U \rightarrow V$, a qual define uma relação entre u e v para um dado valor u_0 de u , a composição resultante produz um valor v em B . Um procedimento denominado **defuzificação** fornece um valor não *fuzzy* resultante das inferências das regras. Os principais procedimentos de *defuzificação* constituem o método do máximo, da média dos máximos e do centro de área, que são definidos por (2.14), (2.15) e (2.16), respectivamente.

$$v_0 = \max[\mu_{r_i}(v)] \quad (2.14)$$

$$v_o = \frac{1}{N} \sum_{n=1}^N \max[\mu_{r_i}(v_n)] \quad (2.15)$$

$$v_o = \frac{\sum_{n=1}^N v_n \mu_{r_i}(v_n)}{\sum_{n=1}^N \mu_{r_i}(v_n)} \quad (2.16)$$

Para regras do tipo Mamdani, os valores v_n correspondem aos centroides das funções de pertinência utilizadas nos consequentes das regras, e para regras do tipo Takagi-Sugeno, estes valores correspondem aos valores y_i das funções associadas.

2.4.2 Obtenção de Regras Fuzzy

A obtenção de regras *fuzzy* pode ser realizada de várias formas, conforme listado a seguir:

- informações de especialistas;
- análise qualitativa de dados;
- técnicas de regressão, método dos mínimos quadrados etc.;
- métodos de otimização;
- técnicas de agrupamento;
- métodos de treinamento, aprendizado de máquina etc.;
- associação com outras técnicas de IA;
- outras.

Similarmente aos sistemas especialistas, sistemas *fuzzy* podem ser obtidos por meio de informações fornecidas por especialistas de uma determinada área. As regras difusas são construídas por meio de variáveis linguísticas cujas denominações dos conjuntos nebulosos recebem etiquetas ou rótulos relacionados às informações do problema ou processo a ser modelado.

Os procedimentos relacionados com técnicas de análise qualitativa de dados possuem certa similaridade à abordagem por especialistas. As informações

relativas aos dados de um problema ou processo são analisadas de forma mais qualitativa do que quantitativa, fornecendo deste modo as regras *fuzzy* para o sistema analisado. Wang e Mendel (1992) apresentaram um algoritmo que permite a geração de regras *fuzzy* em processos de modelagem em geral. Na primeira etapa do algoritmo, os dados do processo são divididos em determinadas partições para a definição dos conjuntos difusos utilizados. Na segunda etapa as regras *fuzzy* são inicialmente geradas. Na terceira etapa define-se um grau de representação para cada regra obtida. Na quarta etapa a base de conhecimento é refinada. Na quinta parte ocorre a inferência do valor do modelo para as entradas correspondentes. Apesar de simples, o processo de determinação de regras *fuzzy* por este método pode ser bastante tedioso e demorado.

Técnicas conhecidas como regressão, o método dos mínimos quadrados, os métodos de otimização e as técnicas de agrupamento são aplicados com sucesso na seleção e ajustes de parâmetros de funções de pertinência, e dos coeficientes de modelos *fuzzy*.

Técnicas de treinamento baseadas em aprendizado de máquina, como o método do gradiente descendente (similar ao método utilizado no treinamento de redes neurais artificiais), também são usadas com sucesso em ajustes de parâmetros de modelos *fuzzy* (Wang, 1994; Kosko 1997).

Nas seções seguintes deste livro serão apresentados mais detalhes sobre os tópicos apresentados anteriormente. Também serão mostrados alguns exemplos numéricos com a finalidade de consolidar os conceitos vistos.

2.5 Exemplos

Uma das características dos sistemas *fuzzy* que possibilitam muitas das suas aplicações em modelagem de processos complexos, reconhecimento de padrões, sistemas de controle, entre outras, é a capacidade de aproximação de características dos mesmos. É conhecido que os sistemas *fuzzy* são aproximadores universais de funções (Kosko, 1994; Wang, 1994).

Exemplo 2.1

A Figura 2.8 contém dados de medidas de um processo que deve ser representado por um determinado modelo.

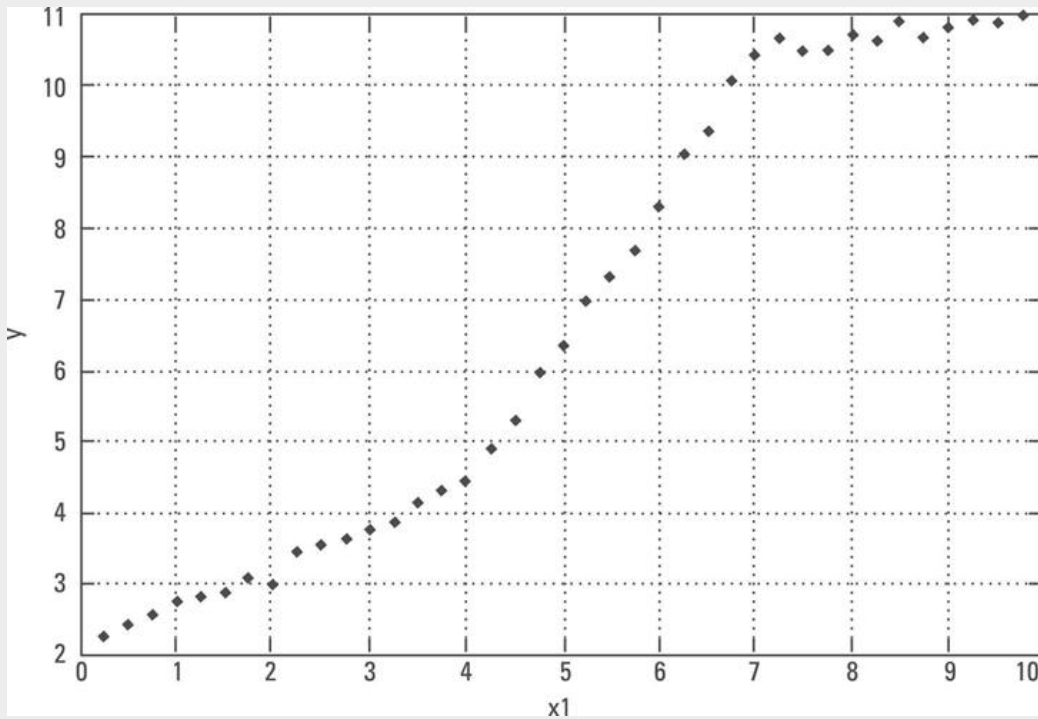


FIGURA 2.8 Gráfico relacionado com os dados de um determinado processo.

Para uma modelagem matemática conhecida como linear por partes (*piecewise*), o sistema em questão pode ser representado (aproximado) por três funções lineares (f_1 , f_2 e f_3), por exemplo. A ilustração na [Figura 2.9](#) mostra as aproximações em questão, onde cada função (f_i) modela um submodelo da representação geral.

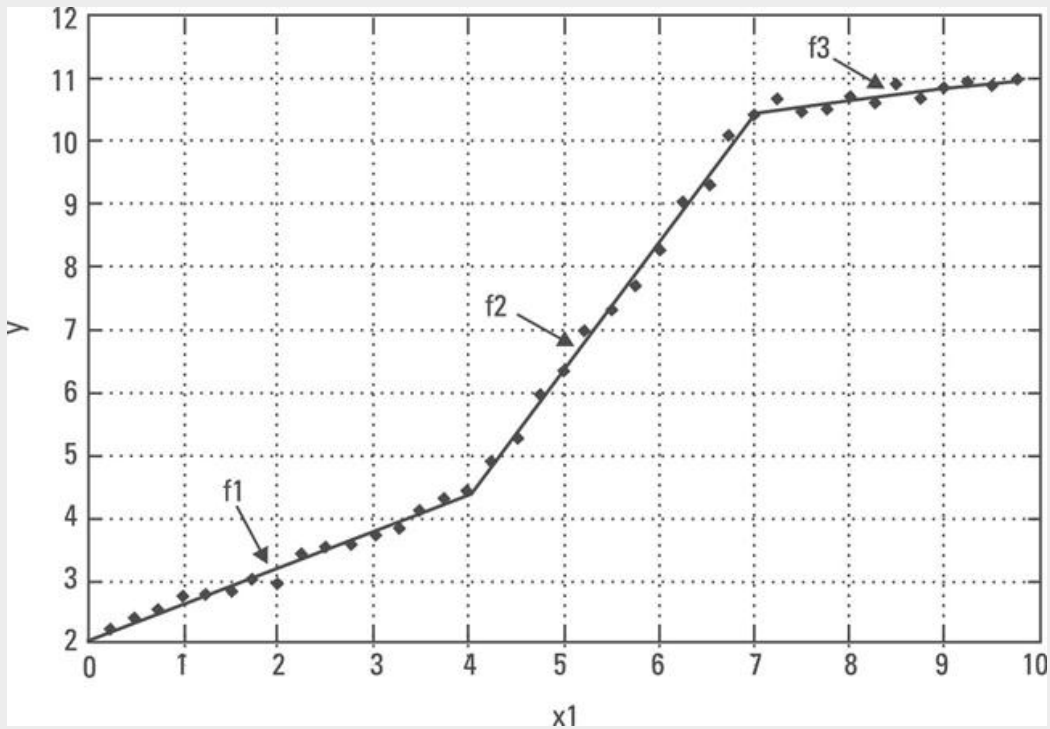


FIGURA 2.9 Modelagem linear por partes do sistema em questão.

Cada função de aproximação é definida por um determinado conjunto de dados (x_1), neste exemplo: $0 \leq x_1 \leq 4$; $4 \leq x_1 \leq 7$; $7 \leq x_1 \leq 10$. Utilizando os correspondentes conjuntos de dados da variável dependente (y) e o método dos mínimos quadrados, obtêm-se com (2.17) os coeficientes (w_i) associados às funções de modelagem. A representação do sistema modelado desta forma é expressa por (2.18).

$$F_r = [x_1 \ 1]; \quad (2.17)$$

$$w_i = [F_r^T \cdot F_r]^{-1} \cdot F_r^T \cdot y.$$

$$f_i(x_1) = \begin{cases} 0,5761x_1 + 2,0425 & \text{se } 0 \leq x_1 \leq 4; \\ 2,0206x_1 - 3,7412 & \text{se } 4 \leq x_1 \leq 7; \\ 0,1864x_1 + 9,1489 & \text{se } 7 \leq x_1 \leq 10. \end{cases} \quad (2.18)$$

Agora será utilizado um **modelo fuzzy** para a **modelagem do mesmo sistema** em questão, cujas funções de pertinência adotadas (Takagi e Sugeno, 1985) estão ilustradas na Figura 2.10, com as regras funcionais dadas por (2.19).

$$\begin{aligned} r_1: & \text{IF } x_1 = A_1 \text{ THEN } y_1 = 0,56x_1 + 2,17; \\ r_2: & \text{IF } x_1 = A_2 \text{ THEN } y_2 = 0,11x_1 + 9,6. \end{aligned} \quad (2.19)$$

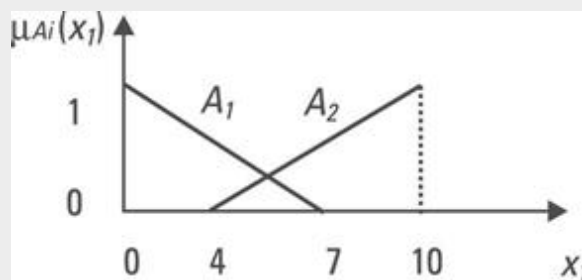


FIGURA 2.10 Funções de pertinência do exemplo de aproximação.

O modelo *fuzzy* resultante com apenas duas funções polinomiais possui a capacidade de modelar o universo de discurso do processo em questão, demonstrando a característica de aproximação de funções dos modelos nebulosos. Objetivando exemplificar a computação numérica do modelo será calculado o valor inferido do sistema *fuzzy* considerado para $x_1 = 5,5$. Calculando os valores das funções de pertinência correspondentes (*fuzificação*), vem: $\mu_{r_1} = 0,21$; $\mu_{r_2} = 0,25$. Os valores das funções polinomiais associadas são calculados diretamente a partir do valor considerado para x_1 : $y_1 = 5,25$; $y_2 = 10,2$. Computando o valor inferido pelas regras com o procedimento de *defuzificação*, temos:

$$y = (0,21 * 5,25 + 0,25 * 10,2) / (0,21 + 0,25) = 7,92.$$

O valor estimado obtido está próximo do dado original do sistema em questão, indicando a capacidade de interpolação do modelo *fuzzy* associado.

Computando outros valores e colocando-os em um gráfico, temos na [Figura 2.11](#) as informações estimadas (em linha cheia), juntamente com os dados do processo (representados por pontos).

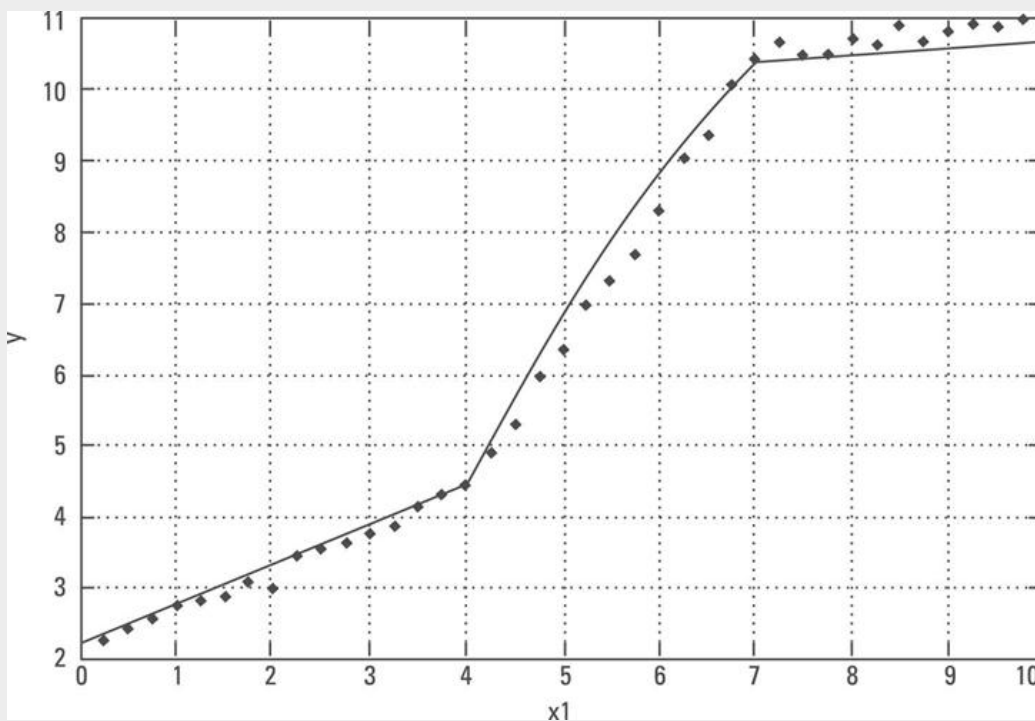


FIGURA 2.11 Resultado da modelagem *fuzzy* associada a [\(2.19\)](#).

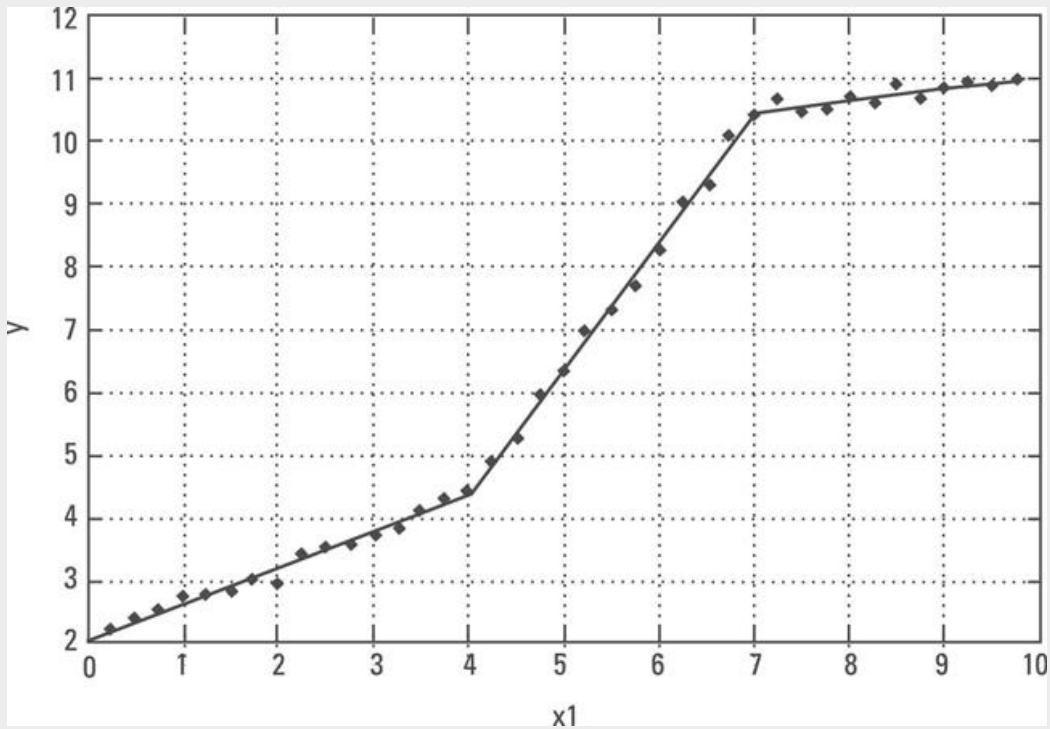


FIGURA 2.12 Resultado das aproximações com o procedimento de treinamento citado.

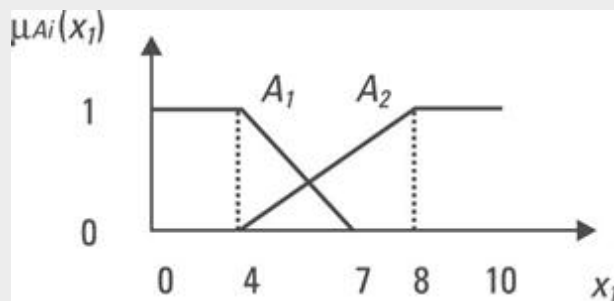


FIGURA 2.13 Funções de pertinência ajustadas para o exemplo em questão.

Utilizando técnicas de treinamento são obtidos modelos *fuzzy* mais precisos, e neste exemplo será utilizado a *Toolbox Anfis* do MatLab para este propósito (mais detalhes sobre a estrutura *Anfis* estão presentes no [Anexo 2.1](#)). A listagem a seguir mostra os comandos necessários para realizar o procedimento em questão.

Anexo 2.1

Anfis: Adaptive Network-Based Fuzzy Inference System

Um Anfis constitui um sistema de inferência *fuzzy* (FIS), cujos parâmetros são ajustados por intermédio de uma rede neural adaptativa (Jang, 1993). A Figura 1 ilustra a estrutura de uma rede neural adaptativa, onde se pode notar a presença de dois tipos distintos de neurônios: os neurônios representados por quadrados são denominados neurônios adaptativos; enquanto os representados por círculos são chamados de padrão.

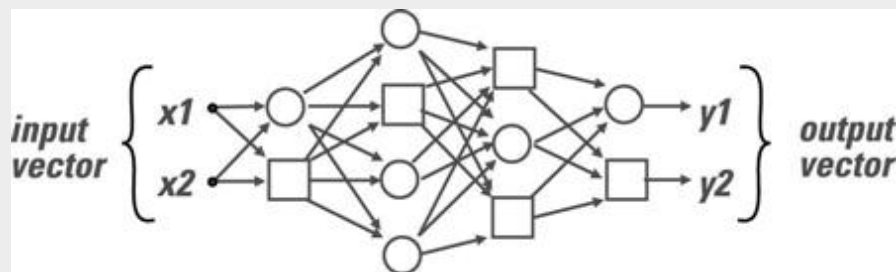


FIGURA 1 Exemplo de uma rede neural adaptativa. Fonte: (Jang, 1993).

O Anfis serve de base para a construção de um conjunto de regras nebulosas e de funções de pertinência correspondentes a um modelo FIS. Os parâmetros das funções de pertinência e a estrutura das regras *fuzzy* são obtidos via processo de treinamento supervisionado, que utiliza as informações de entrada e saída do sistema em questão.

Podem-se encontrar diferentes arquiteturas para um Anfis, cada qual baseada nos modelos de regras *fuzzy* utilizadas. A Figura 2 ilustra a arquitetura de um Anfis composto por duas regras, contendo duas variáveis de entrada que são mapeadas em dois conjuntos nebulosos distintos.

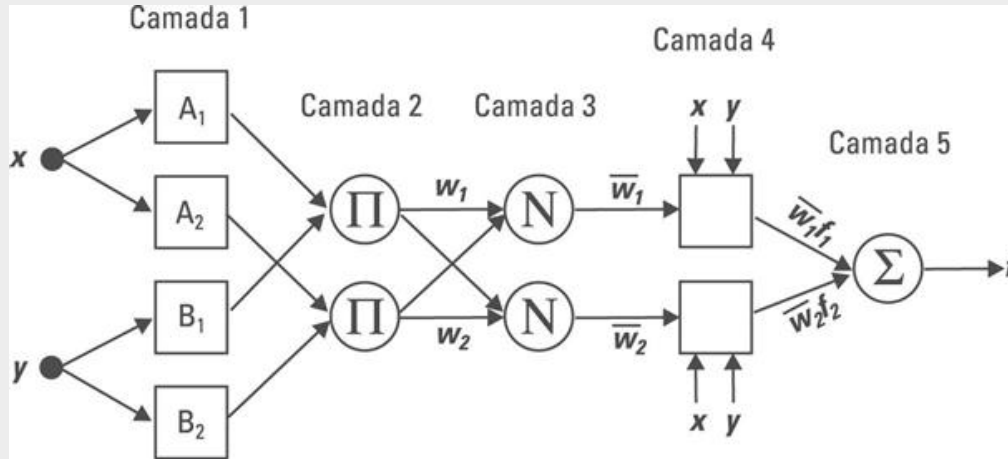


FIGURA 2 Arquitetura Anfis com duas entradas. Fonte: (Jang, 1993).

A estrutura Anfis em questão recebe as variáveis de entrada (x , y), e por meio de interações em suas camadas, realiza as etapas de *fuzificação* e *defuzificação* dos dados relacionados, resultando no valor de saída correspondente ao padrão de entrada. Na primeira camada ocorre o mapeamento das variáveis de entrada do universo de discurso considerado para o grau de pertinência correspondente, ou seja, é realizado o processo de *fuzificação* por meio da utilização de uma função de pertinência associada. Os neurônios da primeira camada são do tipo adaptativo, sendo assim, os parâmetros, que nesse caso dizem respeito aos valores modais das funções de pertinência são ajustados via processo de treinamento. O valor resultante (w_i) dos neurônios dessa camada é obtido por meio da aplicação de uma t-norma via um operador mínimo ou produto, por exemplo. A terceira camada é responsável pela normalização dos valores gerados na segunda camada. A saída dos neurônios que compõem essa camada indica o grau de ativação de uma regra em relação às outras. O valor de saída de cada neurônio é dado por (1).

$$\bar{w}_i = \frac{w_i}{\sum_{k=1}^k w_k} \quad (1)$$

Na quarta camada encontram-se as composições dos consequentes das regras (neste caso, do tipo funcional ou Takagi-Sugeno); os valores de

saída dos neurônios dessa camada são expressos por meio de (2), onde os parâmetros (p_i, q_i, r_i) são estimados por técnicas de treinamento ou pelo método dos mínimos quadrados.

$$f_i = \bar{w}_i g_i = \bar{w}_i (p_i x + q_i y + r_i) \quad (2)$$

Por fim, a quinta camada finaliza o processo de *defuzificação*, por meio da aplicação da equação (3), que consiste em uma ponderação dos valores obtidos na quarta camada da estrutura considerada.

$$z_i = \sum_i f_i = \sum_i \bar{w}_i g_i \quad (3)$$

O processo de treinamento pode ser realizado dentre três procedimentos possíveis:

- *Backproagation*: propaga-se o erro do modelo por meio do cálculo do gradiente descendente para as camadas anteriores ajustando-se os parâmetros dos nós adaptativos da primeira e quarta camadas.
- *Backpropagation* + Método dos Mínimos Quadrados: aplicação da técnica do gradiente descendente para ajustar os parâmetros dos neurônios da primeira camada e o método dos mínimos quadrados para ajustar os da quarta camada.
- *Backpropagation* + Método dos Mínimos Quadrados uma única vez: aplicação da técnica do gradiente descendente para ajustar os parâmetros dos neurônios da primeira camada e o método dos mínimos quadrados para ajustar os da quarta camada somente na primeira interação.

Outros detalhes da ferramenta Anfis podem ser encontrados na documentação correspondente do MatLab.

Exemplo 2.2

Neste exemplo será mostrado um modelo *fuzzy* com regras Mamdani relacionadas com o mapeamento de uma função com duas variáveis independentes. As regras em questão estão representadas na [Tabela 2.1](#). Este padrão de regras é comum em algumas aplicações relacionadas a sistemas de controles com lógica difusa. As funções de pertinência utilizadas estão indicadas na [Figura 2.14](#), onde os valores de entrada estão normalizados no intervalo $[-1, +1]$. As etiquetas correspondentes aos conjuntos possuem as seguintes denominações: N = Negativo; Z = Zero; P = Positivo; G = Grande Positivo; M = Muito Negativo.

Tabela 2.1
Regras Fuzzy

y x_2/x_1			
	N	Z	P
N	¹ M	⁴ N	⁷ Z
Z	² N	⁵ Z	⁸ P
P	³ Z	⁶ P	⁹ G

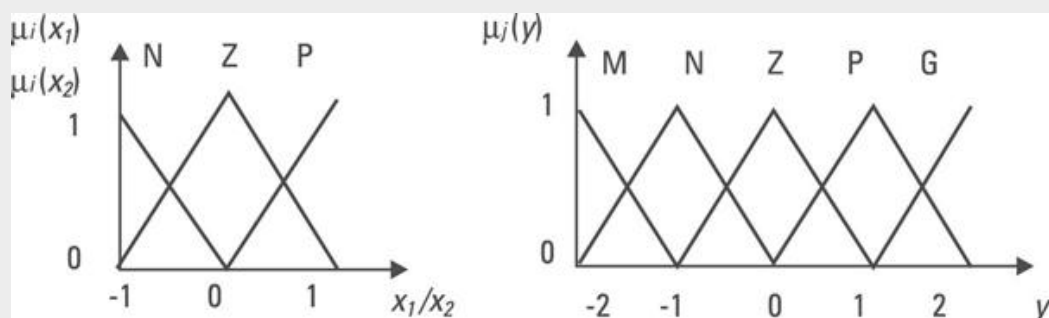


FIGURA 2.14 Funções de pertinência do [Exemplo 2.2](#).

Para exemplificar o processamento de informações deste sistema *fuzzy* serão computados os valores do modelo com o propósito de estimar o valor inferido do mesmo para $x_1 = 0,8$ e $x_2 = -0,6$. Estes valores estão relacionados com as seguintes regras ativadas (aquelas que produzem pertinências não nulas):

$$r_4: IF \ x_1 = Z \ AND \ x_2 = N \ THEN \ y = N;$$

$$r_5: IF \ x_1 = Z \ AND \ x_2 = Z \ THEN \ y = Z;$$

$$r_7: IF \ x_1 = P \ AND \ x_2 = N \ THEN \ y = Z;$$

$$r_8: IF \ x_1 = P \ AND \ x_2 = Z \ THEN \ y = P.$$

Realizando a etapa de *fuzificação* obtêm-se: $\mu_Z(x_1) = 0,2$; $\mu_P(x_1) = 0,8$; $\mu_N(x_2) = 0,6$; $\mu_Z(x_2) = 0,4$. Utilizando o operador produto têm-se as composições: $\mu_{r_4} = 0,2 * 0,6 = 0,12$; $\mu_{r_5} = 0,2 * 0,4 = 0,08$; $\mu_{r_7} = 0,8 * 0,6 = 0,48$; $\mu_{r_8} = 0,8 * 0,4 = 0,32$. Realizando o procedimento de *defuzificação* com o método do centro de área, temos o valor inferido do sistema:

$$y = (0,12 * -1 + 0,08 * 0 + 0,48 * 0 + 0,32 * 1) / (0,12 + 0,08 + 0,48 + 0,32) = 0,2.$$

Este sistema *fuzzy* mapeia uma função correspondente a $f(x_1, x_2) = x_1 + x_2$ que fornece um valor idêntico (0,2) ao modelo, dados os mesmos valores das variáveis independentes. Esta característica está de acordo com artigos ([Chen e Kuo, 1995](#); [Pinheiro e Gomide, 1999](#)) que mostram que modelos *fuzzy* com estrutura similar apresentam mapeamentos lineares quando se utiliza funções de pertinência triangulares, simétricas e igualmente espaçadas.

Exemplo 2.3

Neste exemplo será ilustrada a obtenção de regras *fuzzy* por meio de um procedimento de análise qualitativa e quantitativa de dados, os quais estão relacionados a um sistema de controle de estacionamento automático de veículos como carro, caminhão, e outros (Wang, 1994). A Figura 2.15 ilustra o contexto do problema conhecido como *truck backer-upper control*. A área definida pelas cotas y (com um determinado tamanho) e $x = [0, 20]$ representa uma área de um estacionamento no qual se deseja estacionar um veículo em posição perpendicular ($\Phi = 90^\circ$) na cota $x = 10$. O veículo se encontra inicialmente em uma posição (x,y) qualquer, e com um dado ângulo Φ entre seu eixo longitudinal e o eixo da cota x . O ângulo θ indica a inclinação das rodas do veículo em relação ao seu eixo longitudinal, que representa a ação de manobra sobre ele com o objetivo de comandar o estacionamento na posição desejada.

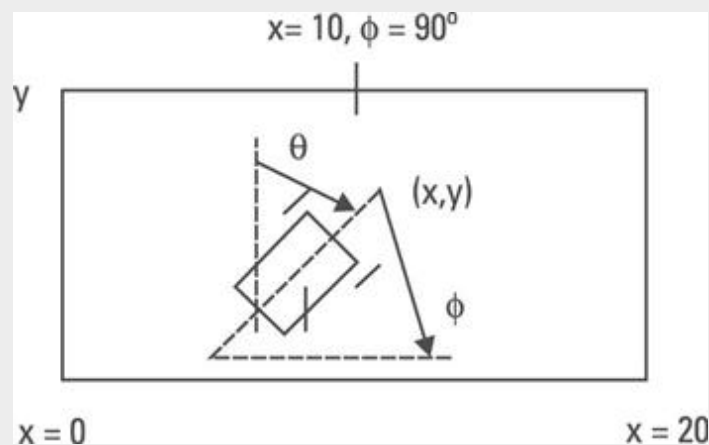


FIGURA 2.15 Representação relativa ao Exemplo 2.3.

A dinâmica do posicionamento em questão é representada pelas equações (2.20), onde “b” representa o tamanho do veículo. A Tabela 2.2 contém dados relativos aos variáveis do sistema provenientes de medidas efetuadas no mesmo. Estas medições são relativas a manobras de estacionamento realizadas por um motorista, as quais representam numericamente a dinâmica em questão de comportamento não linear.

$$\begin{aligned}x(k+1) &= x(k) + \cos[\phi(k) + \theta(k)] + \sin\phi(k) \sin\theta(k); \\y(k+1) &= y(k) + \sin[\phi(k) + \theta(k)] - \cos\phi(k) \sin\theta(k); \\ \phi(k+1) &= \phi(k) - \sin^{-1}\left[\frac{2\sin\theta(k)}{b}\right].\end{aligned}\tag{2.20}$$

Tabela 2.2

Medidas do processo

x	Φ	θ
1,00	0,00	-19,00
1,95	9,37	-17,95
2,88	18,23	-16,90
3,79	26,59	-15,85
4,65	34,44	-14,80
5,45	41,78	-13,75
6,18	48,60	-12,70
7,48	54,91	-11,65
7,99	60,71	-10,60
8,72	65,99	-9,55
9,01	70,75	-8,50
9,28	74,98	-7,45
9,46	78,70	-6,40
9,59	81,90	-5,34
9,72	84,57	-4,30
9,81	86,72	-3,25
9,88	88,34	-2,20
9,91	89,44	0,00

O procedimento citado utiliza uma análise qualitativa e quantitativa nos dados da tabela em questão (incluindo outras medidas de manobras), com o objetivo de obter as regras necessárias para controlar o estacionamento do

veículo de forma automática. As regras a serem obtidas podem ser processadas no computador de bordo do veículo. Um dispositivo GPS (ou outros tipos de transdutores), por exemplo, pode informar as coordenadas (x,y) e o ângulo Φ do sistema. Um dispositivo eletromecânico pode movimentar de um ângulo θ a direção do veículo com o objetivo de manobrá-lo.

Utilizando o procedimento proposto por Wang e Mendel (1992), são obtidas as regras básicas de controle (indicadas na Tabela 2.3). As funções de pertinência correspondentes aos conjuntos nebulosos adotados estão ilustradas na Figura 2.16, onde os seus valores modais estão incluídos na Tabela 2.4. As etiquetas b_1 , b_2 , b_3 , ce , s_1 , s_2 e s_3 indicam os conjuntos nebulosos em questão (os índices “ - ” e “ + ” simbolizam os valores extremos dos parâmetros das funções correspondentes).

Tabela 2.3**Regras *fuzzy* do controle**

θ		x				
		s2	s1	ce	b1	b2
	s3	¹ s2	⁶ s3	—	—	—
	s2	² s2	⁷ s3	¹² s3	¹⁷ s3	—
	s1	³ b1	⁸ s1	¹³ s2	¹⁸ s3	²³ s2
φ	ce	⁴ b2	⁹ b2	¹⁴ ce	¹⁹ s2	²⁴ s2
	b1	⁵ b2	¹⁰ b3	¹⁵ b2	²⁰ b1	²⁵ s1
	b2	—	¹¹ b3	¹⁶ b3	²¹ b3	²⁶ b2
	b3	—	—	—	²² b3	²⁷ b2

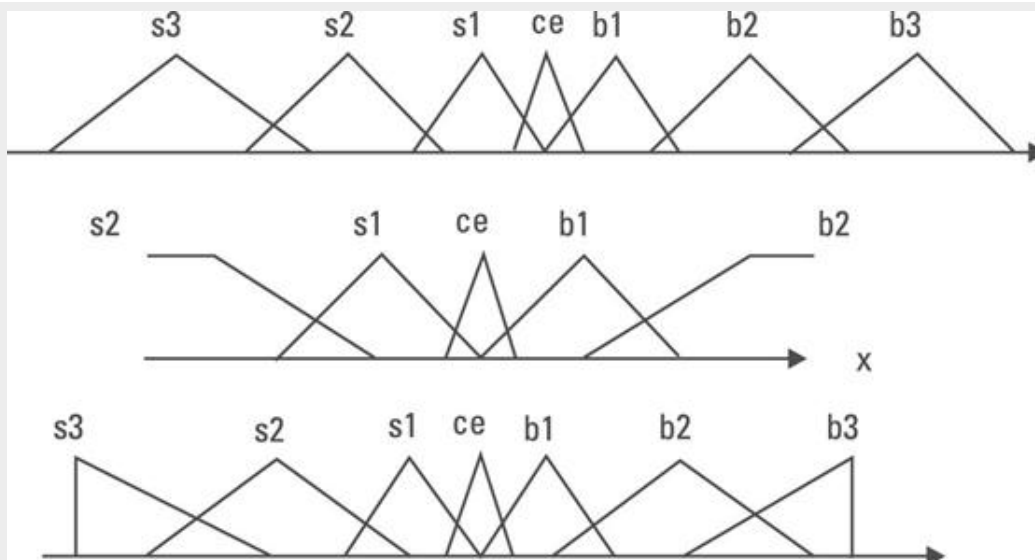


FIGURA 2.16 Funções de pertinência do [Exemplo 2.3](#).

Tabela 2.4Valores modais das funções indicadas na [Figura 2.16](#)

	ϕ_-	ϕ_+	x_-	x_+	θ_-	θ_+
s3	-115	-15			-40	-20
s2	-45	45	1,5	7	-33	-7
s1	15	90	4	10	-14	0
ce	80	100	9	11	-4	4
b1	90	165	10	16	0	14
b2	135	225	13	18,5	7	33
b3	195	295			20	40

As regras (tipo Mamdani) deste problema são semelhantes as do exemplo anterior, e algumas delas estão ilustradas abaixo (estão diretamente relacionadas à [Tabela 2.3](#), que contém as regras básicas de controle obtidas):

$$r_1: IF \ x = s2 \ AND \ \phi = s3 \ THEN \ \theta = s2;$$

$$r_2: IF \ x = s1 \ AND \ \phi = s3 \ THEN \ \theta = s3;$$

$$r_{27} : IF \ x=b2 \ AND \ \phi=b3 \ THEN \ \theta=b2.$$

Na [Tabela 2.3](#) as posições indicadas por “-” representam regras que não são necessárias ou que não foram totalmente identificadas. O MatLab possui um exemplo similar a este como demonstração em sua *toolbox* sobre lógica *fuzzy* (que contém uma animação gráfica do sistema em questão).

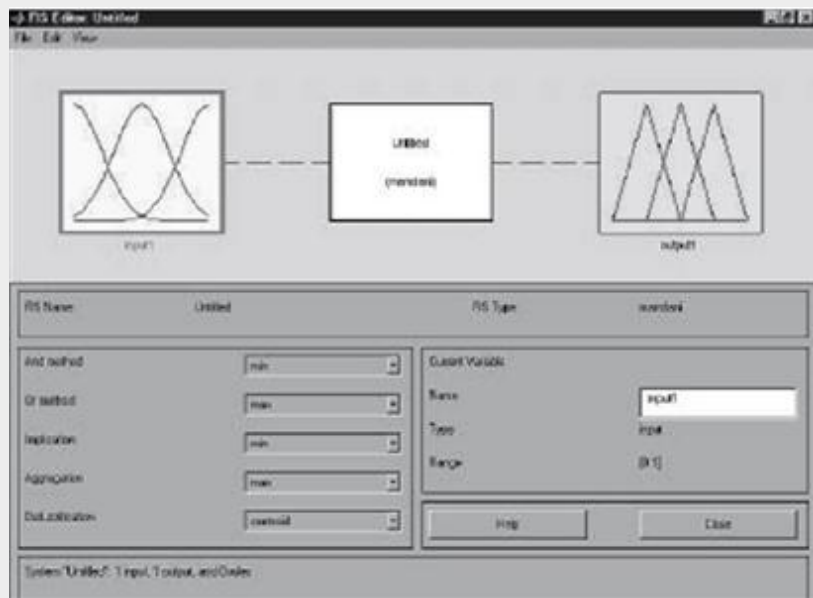
No [Anexo 2.3](#) encontram-se as listagens de programas compilados com o objetivo de simular o processo em questão, onde estão incluídas todas as 35 regras possíveis para modelar o sistema *fuzzy* correspondente (no lugar das 27 regras do artigo original). A vantagem da inclusão das regras adicionais é que o sistema consegue estacionar o veículo para qualquer condição inicial dele, dentro dos limites estabelecidos. A [Figura 2.17](#) mostra, na forma de pontos, a trajetória do veículo para as condições iniciais $x = 5$, $y = 10$ e $\Phi = 30^\circ$, onde se verifica que o sistema realizou adequadamente o posicionamento do veículo na cota $x = 10$ e com $\Phi = 90^\circ$.

Anexo 2.2

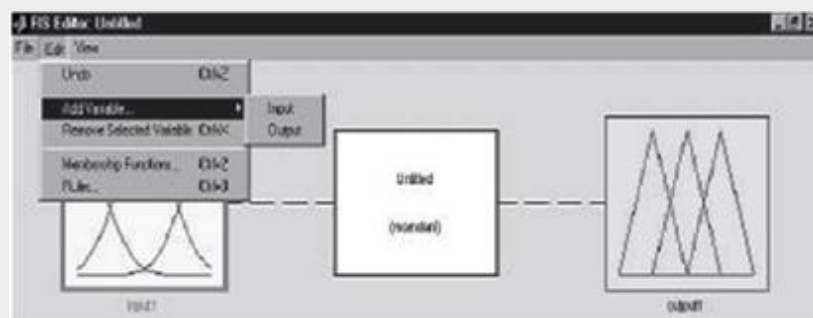
Toolbox Fuzzy

O *toolbox* “*fuzzy*” é uma ferramenta desenvolvida para a edição, execução e gravação de modelos difusos em geral. Em relação ao exemplo correspondente à [Listagem 2.1](#), o modelo difuso resultante pode ser carregado, verificado, executado e gravado por intermédio da ferramenta em questão. Mais detalhes deste *toolbox* encontram-se na documentação correspondente do MatLab.

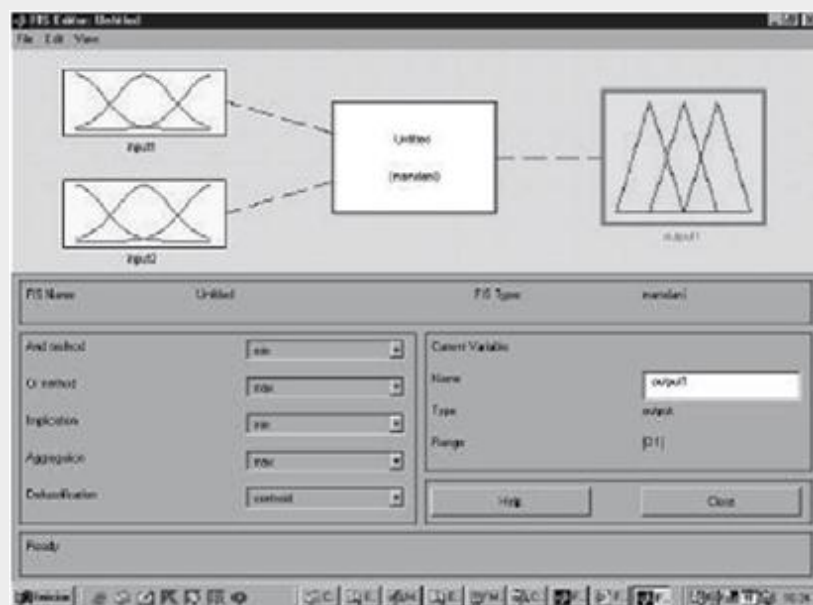
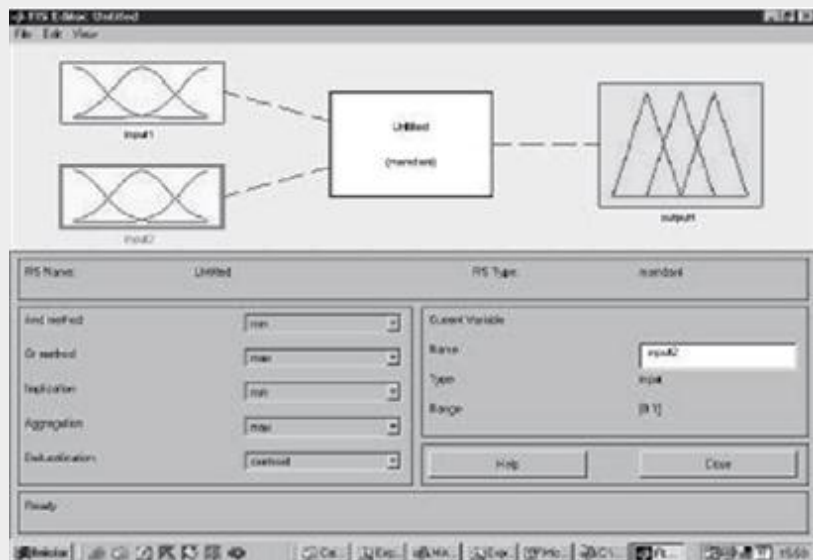
Nas próximas páginas deste anexo são fornecidas algumas informações sobre a utilização da ferramenta em questão. Este aplicativo é acionado digitando na área de trabalho o comando *anfis* <Enter>. A figura a seguir ilustra a tela típica do aplicativo.



A edição das variáveis de um modelo *fuzzy* é realizada na janela correspondente, ilustrada na próxima figura.



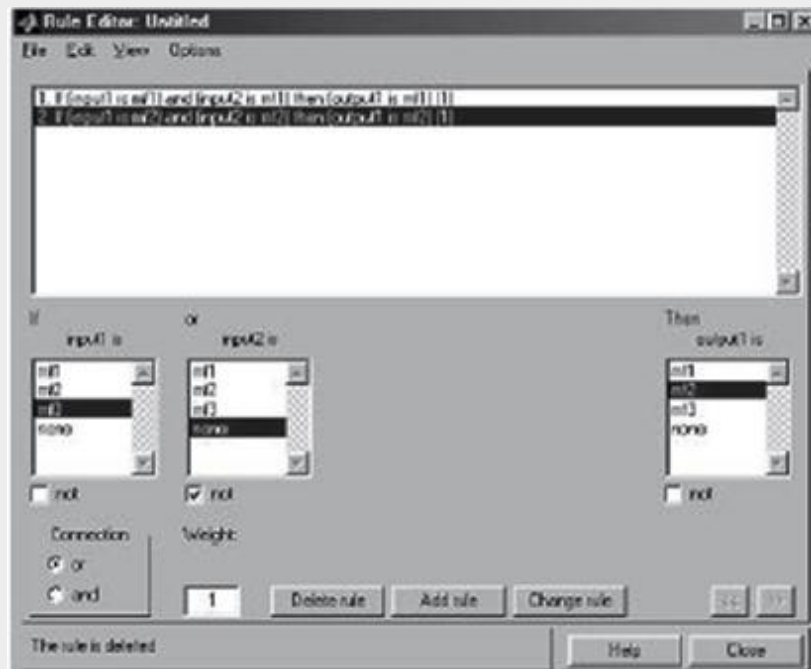
Os nomes das variáveis podem ser editados conforme as ilustrações seguintes.



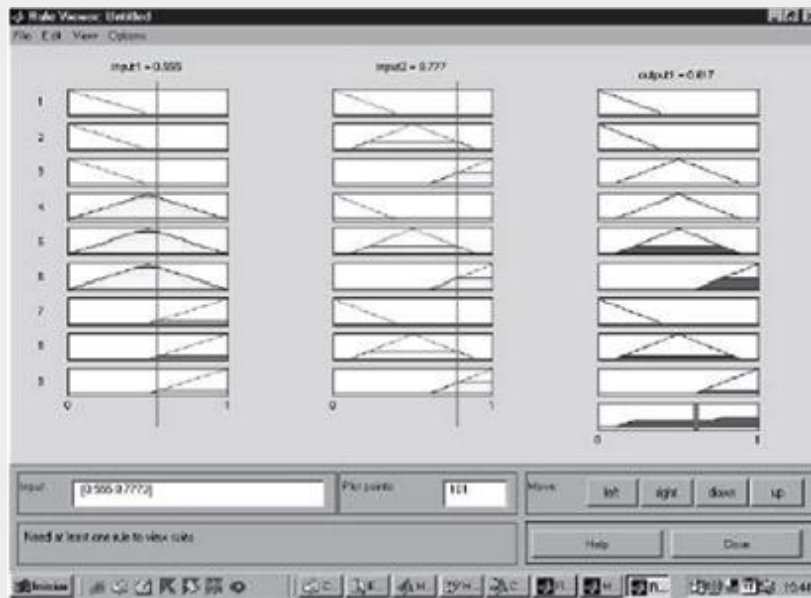
As funções de pertinência associadas às variáveis editadas podem ser inseridas, modificadas, removidas etc., conforme ilustrado na figura a seguir.



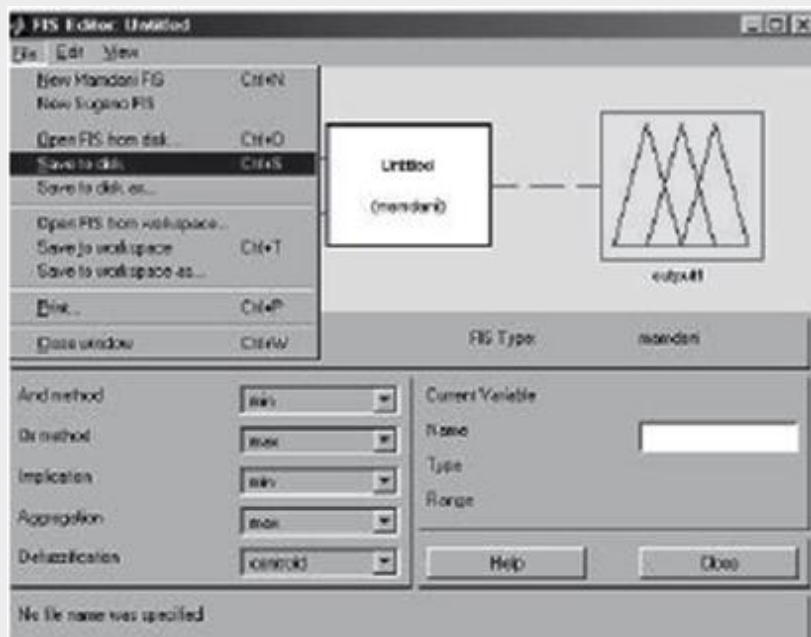
O banco de regras *fuzzy* é editado na tela correspondente do aplicativo, conforme exemplificado na próxima figura.



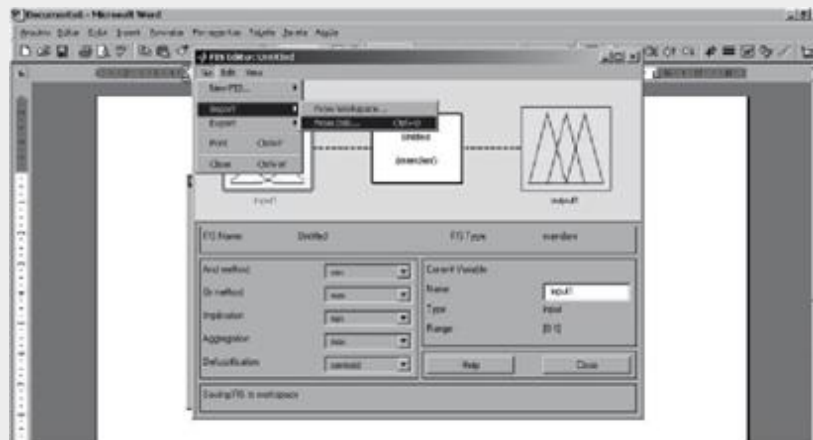
Um modelo difuso devidamente editado ou carregado pode ser testado por intermédio de uma tela correspondente (ver ilustração a seguir).



Um modelo editado pode ser gravado segundo a indicação abaixo.



Um modelo já gravado ou gerado pode ser carregado, conforme indicado na tela ilustrada a seguir.



No caso do exemplo associado à [Listagem 2.1](#), a carga do modelo *fuzzy* é realizada com a opção “From Workspace”, onde se digita na tela correspondente o nome do modelo. No exemplo em questão: *ModFuzzy* <Enter>.

As instruções a seguir mostram como editar o exemplo referente às regras (2.19) e as funções de pertinência associadas à [Figura 2.10](#):

- Na área de trabalho do MatLab digitar *fuzzy* <Enter>. Será aberta a janela da ferramenta em questão. No ícone da janela principal do aplicativo, acionar no campo *File* => *New FIS* => “Sugeno”, objetivando editar o modelo correspondente.
- Clicar com o mouse uma vez sobre o ícone da variável de entrada, e no campo “Name” redefini-la como *x1*, por exemplo. Idem na variável de saída (*output*) para defini-la como *y*.
- Clicar duas vezes na variável de entrada para editar as suas funções de pertinência. Será aberta outra janela. Ajustar a faixa no campo “Range” com os valores [0 10]. Clicar nas funções de pertinências para editar os seus nomes (p. ex., *A1* e *A2*), e os respectivos valores [0 0 7] e [4 10 10] que definem as mesmas. Para remover alguma função basta clicar na mesma e acionar a tecla “del”.
- No ícone referente à variável “*y*” clicar nas funções associadas para editar os seus nomes (p. ex., *y1* e *y2*). Selecionar os tipos (no campo “Type”), como “linear” e definir os respectivos valores dos coeficientes [0.56 2.17] e [0.11 9.6].
- Para editar as regras do modelo, basta acionar a opção *Edit* => *Rules*, e na janela aberta associar os conjuntos nebulosos editados

para a variável de entrada x_1 com as funções da variável de saída y , acionando o botão “Add Rule” para a inclusão de cada regra do modelo (duas regras neste exemplo).

- f) Para avaliar o modelo, acionar no ícone “View” a opção “View rules”. Será aberta uma janela, e no campo “Input”, basta digitar um valor (na faixa de 0 a 10) e pressionar <Enter>, para ter-se o valor estimado da variável y , onde os procedimentos de *fuzificação* e *defuzificação* são ilustrados graficamente.
- g) O aplicativo gerado pode ser gravado conforme ilustrado anteriormente neste anexo, e utilizado posteriormente como rotina de outro programa.

Anexo 2.3

Listagens referentes ao Exemplo 2.3.

```

% Exemplo de controle fuzzy do problema Truck Backer-Upper.

clear all;      % Zerar variáveis.
cor = 'k.';    % Define cor preta e linha tracejada.

tr = pi/180; % Fatores de ângulo em radianos-graus-radianos.
ta = 180/pi;

Xo = input('Valor inicial da coordenada X do carro = ');
Yo = input('Valor inicial da coordenada Y do carro = ');
Ao = input('Valor inicial do angulo horizontal do carro = ');

% Condições iniciais.
An = tr * Ao;
Xn=Xo; Yn=Yo; n=1;

while (Yn < 40), % Loop de simulação.

    Xk = Xn;      % Variável da coordenada X do carro.
    Ak = ta * An; % Variável do angulo horizontal do carro.

    Defuz = funcfuzzy(Xk,Ak); % Função de controle Fuzzy.

    vX(n) = Xn;    % Dados da coordenada X.
    vY(n) = Yn;    % Dados da coordenada Y.
    vA(n) = Defuz; % Dados de atuação.
    vn(n) = n;     % Número de passos do programa.

    Ac = tr * Defuz; % Ação de controle.

    % Equações do movimento do veículo.
    Xn = Xn + cos(An + Ac) + sin(An) * sin(Ac);
    Yn = Yn + sin(An + Ac) - cos(An) * sin(Ac);
    An = An - asin(0.5*sin(Ac));
    n = n + 1;

end

% Resultado da simulação.
plot(vX,vY,cor); grid; ylabel('Yk'); xlabel('Xk');
title('Trajetória do Veículo');
% Implantação do controle fuzzy do Truck Backer-Upper.

function Defuz=funcfuzzy(Xk,Ak) % Definição da function.

rs=0.01; % Valor para evitar denominador nulo.

% Parâmetros das funções de pertinência triangulares:
% cxyz=centro, bxyz=base,
% N=negative, Z=zero, P=positive, S=small, M=medium, L=large.
ceNM=1.5; ceNS=7; ceZE=10; cePS=13; cePM=18.5;
beNM=5.5; beNS=3; beZE=1; bePS=3; bePM=5.5;
cvNL=-65; cvNM=0; cvNS=52.5; cvZE=90; cvPS=127.5; cvPM=180; cvPL=245;

```



```

bvNL=50; bvNM=45; bvNS=37.5; bvZE=10; bvPS=37.5; bvPM=45; bvPL=50;
caNL=-40; caNM=-20; caNS=-7; caZE=0; caPS=7; caPM=20; caPL=40;

e = Xk; % Variável da coordenada X.
v = Ak; % Variável da coordenada angulo.

% Fuzzyficação;
if (((ceNM - beNM) <= e) & (e <= (ceNM + beNM))),
    ueNM = 1 - (abs(ceNM - e)) / beNM;
else
    ueNM = 0;
end
if e < ceNM,
    ueNM = 1;
end

if (((ceNS - beNS) <= e) & (e <= (ceNS + beNS))),
    ueNS = 1 - (abs(ceNS - e)) / beNS;
else
    ueNS = 0;
end

if (((ceZE - beZE) <= e) & (e <= (ceZE + beZE))),
    ueZE = 1 - (abs(ceZE - e)) / beZE;
else
    ueZE = 0;
end

if (((cePS - bePS) <= e) & (e <= (cePS + bePS))),
    uePS = 1 - (abs(cePS - e)) / bePS;
else
    uePS = 0;
end

if (((cePM - bePM) <= e) & (e <= (cePM + bePM))),
    uePM = 1 - (abs(cePM - e)) / bePM;
else
    uePM = 0;
end
if e > cePM,
    uePM = 1;
end

if (((cvNL - bvNL) <= v) & (v <= (cvNL + bvNL))),
    uvNL = 1 - (abs(cvNL - v)) / bvNL;
else
    uvNL = 0;
end
if v < cvNL,
    uvNL = 1;
end

if (((cvNM - bvNM) <= v) & (v <= (cvNM + bvNM))),

```



```

    uvNM = 1 - (abs(cvNM - v)) / bvNM;
else
    uvNM = 0;
end

if (((cvNS - bvNS) <= v) & (v <= (cvNS + bvNS))),
    uvNS = 1 - (abs(cvNS - v)) / bvNS;
else
    uvNS = 0;
end

if (((cvZE - bvZE) <= v) & (v <= (cvZE + bvZE))),
    uvZE = 1 - (abs(cvZE - v)) / bvZE;
else
    uvZE = 0;
end

if (((cvPS - bvPS) <= v) & (v <= (cvPS + bvPS))),
    uvPS = 1 - (abs(cvPS - v)) / bvPS;
else
    uvPS = 0;
end

if (((cvPM - bvPM) <= v) & (v <= (cvPM + bvPM))),
    uvPM = 1 - (abs(cvPM - v)) / bvPM;
else
    uvPM = 0;
end

if (((cvPL - bvPL) <= v) & (v <= (cvPL + bvPL))),
    uvPL = 1 - (abs(cvPL - v)) / bvPL;
else
    uvPL = 0;
end
if v > cvPL,
    uvPL = 1;
end

% Total de regras: 35.
% Operador produto:
r1 = ueNM * uvNL;

r2 = ueNM * uvNM;

r3 = ueNM * uvNS;

r4 = ueNM * uvZE;

r5 = ueNM * uvPS;

r6 = ueNM * uvPM;

r7 = ueNM * uvPL;

```



```
r8 = ueNS * uvNL;  
r9 = ueNS * uvNM;  
r10 = ueNS * uvNS;  
r11 = ueNS * uvZE;  
r12 = ueNS * uvPS;  
r13 = ueNS * uvPM;  
r14 = ueNS * uvPL;  
r15 = ueZE * uvNL;  
r16 = ueZE * uvNM;  
r17 = ueZE * uvNS;  
r18 = ueZE * uvZE;  
r19 = ueZE * uvPS;  
r20 = ueZE * uvPM;  
r21 = ueZE * uvPL;  
r22 = uePS * uvNL;  
r23 = uePS * uvNM;  
r24 = uePS * uvNS;  
r25 = uePS * uvZE;  
r26 = uePS * uvPS;  
r27 = uePS * uvPM;  
r28 = uePS * uvPL;  
r29 = uePM * uvNL;  
r30 = uePM * uvNM;  
r31 = uePM * uvNS;  
r32 = uePM * uvZE;  
r33 = uePM * uvPS;  
r34 = uePM * uvPM;
```

```

r35 = uePM * uvPL;

% Defuzificação.
d1 = caNM*r1+caNM*r2+caPS*r3+caPM*r4+caPM*r5;

d2 = caNL*r8+caNL*r9+caNS*r10+caPM*r11+caPL*r12+caPL*r13;

d3 = caNL*r16+caNM*r17+caZE*r18+caPM*r19+caPL*r20;

d4 = caNL*r23+caNL*r24+caNM*r25+caPS*r26+caPL*r27+caPL*r28;

d5 = caNM*r31+caNM*r32+caNS*r33+caPM*r34+caPM*r35;

d8 = r1+r2+r3+r4+r5+r8+r9+r10+r11+r12+r13+r16+r17+r18+r19+r20+r23+r24+r25;

d9 = r26+r27+r28+r31+r32+r33+r34+r35+rs;

d = (d1 + d2 + d3 + d4 + d5 ) / ( d8 + d9 );

Defuz = d;

```

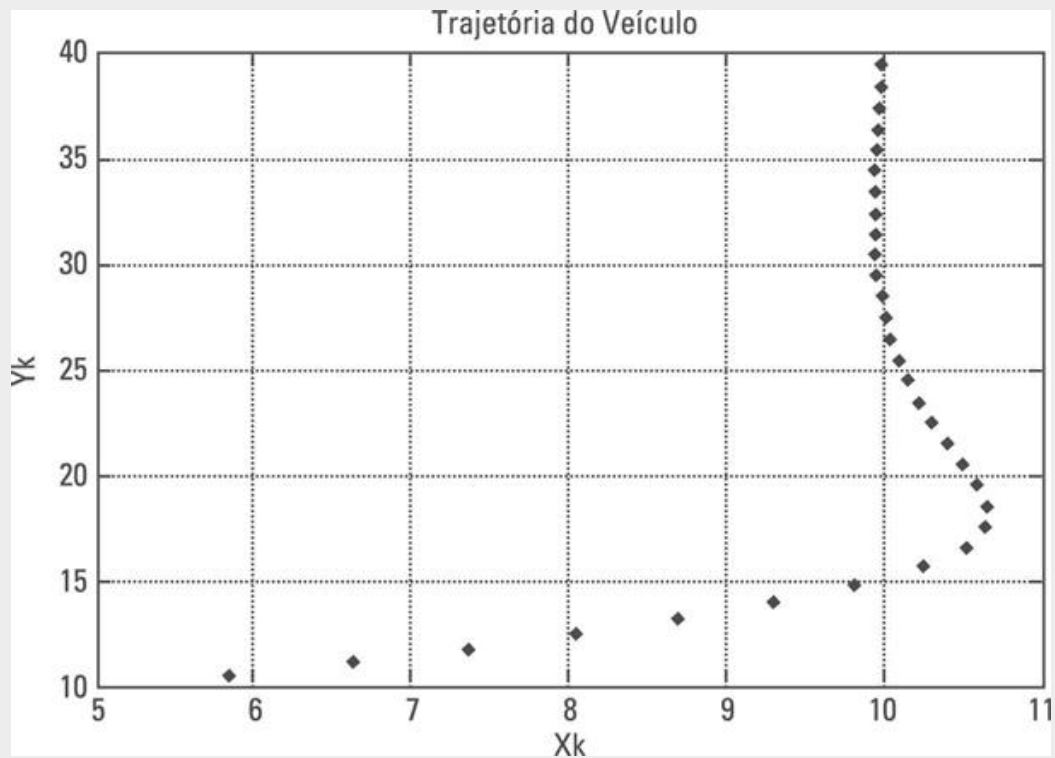


FIGURA 2.17 Resultado da simulação referente ao [Exemplo 2.3](#).

Exemplo 2.4

Este exemplo refere-se a um sistema de suspensão ativa para automóveis. Será mostrado como regras *fuzzy* podem ser obtidas por meio de uma análise qualitativa de uma estratégia de controle conhecida. A [Figura 2.18](#) ilustra a estrutura típica de um sistema de suspensão veicular ([Pinheiro, 2004](#); [Pinheiro et al., 2007](#)).

O feixe de mola e o amortecedor passivo de uma estrutura de suspensão convencional são representados pelos seus coeficientes K_f e B_a , respectivamente. O parâmetro M_s corresponde à massa suspensa (carroceria) do veículo, sustentada pelo sistema de suspensão, sendo M_r a massa do conjunto roda mais pneu e K_p a sua elasticidade. Os deslocamentos d_p , d_r e d_s são movimentos verticais do pneu, da roda e da carroceria, respectivamente. A força F_a representa a atuação exercida por um mecanismo com a finalidade de impor determinadas características adicionais de amortecimento na suspensão. Esse dispositivo pode ser um amortecedor ativo que substitui um amortecedor passivo convencional. As equações (2.21) expressam o modelo linearizado do sistema em questão.

$$\begin{aligned} M_s \frac{d_s^2(t)}{dt^2} + B_a \left(\frac{d_s(t)}{dt} - \frac{d_r(t)}{dt} \right) + K_f (d_s(t) - d_r(t)) - F_a &= 0; \\ M_r \frac{d_r^2(t)}{dt^2} + B_a \left(\frac{d_r(t)}{dt} - \frac{d_s(t)}{dt} \right) + K_f (d_r(t) - d_s(t)) + K_p (d_p(t) - d_r(t)) - F_a &= 0. \end{aligned} \quad (2.21)$$

A expressão (2.22) define uma estratégia de controle para sistemas de suspensão ativa, que é conhecida como continuamente variável. Onde V_{abs} é a velocidade absoluta da massa suspensa e V_{rel} a velocidade relativa entre a massa suspensa e a massa do conjunto roda mais pneu. Se a velocidade absoluta e a velocidade relativa possuem o mesmo sinal algébrico, então um estado de alto amortecimento é selecionado por meio de um coeficiente C_{on} . Agora, se os sinais forem opostos, um estado de baixo amortecimento é

selecionado por meio de um coeficiente C_{off} . Velocidade relativa positiva é definida com as massas M_s e M_r (Figura 2.18) se afastando. Velocidade absoluta positiva é definida com o movimento para cima da massa suspensa (M_s).

$$F_a = \begin{cases} C_{on} V_{abs} + C_{off} V_{rel} & \text{se } V_{abs} V_{rel} \geq 0, \\ C_{off} V_{rel} & \text{se } V_{abs} V_{rel} < 0. \end{cases} \quad (2.22)$$

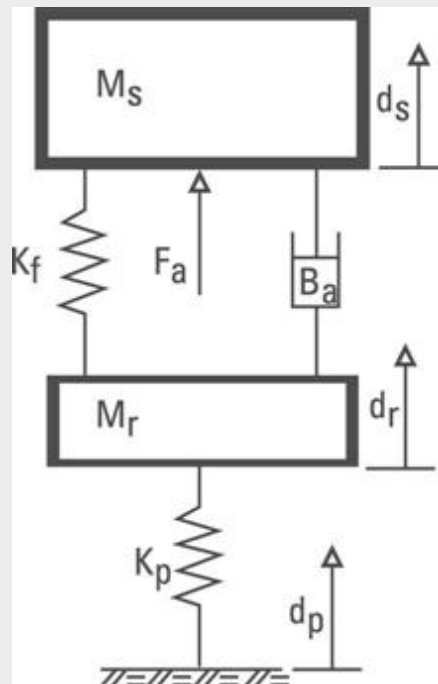


FIGURA 2.18 Estrutura típica de um sistema de suspensão veicular.

A Figura 2.19 ilustra a resposta dinâmica de um sistema de suspensão ativa (a aceleração da carroceria), considerando um deslocamento repentino de cinco centímetros na roda do veículo. Os parâmetros do sistema são: $M_s = 400$ [Kg]; $M_r = 50$ [Kg]; $K_f = 20000$ [N/m]; $K_p = 250000$ [N/m]; $C_{off} = 500$ [Ns/m]; $C_{on} = 1400$ [Ns/m].

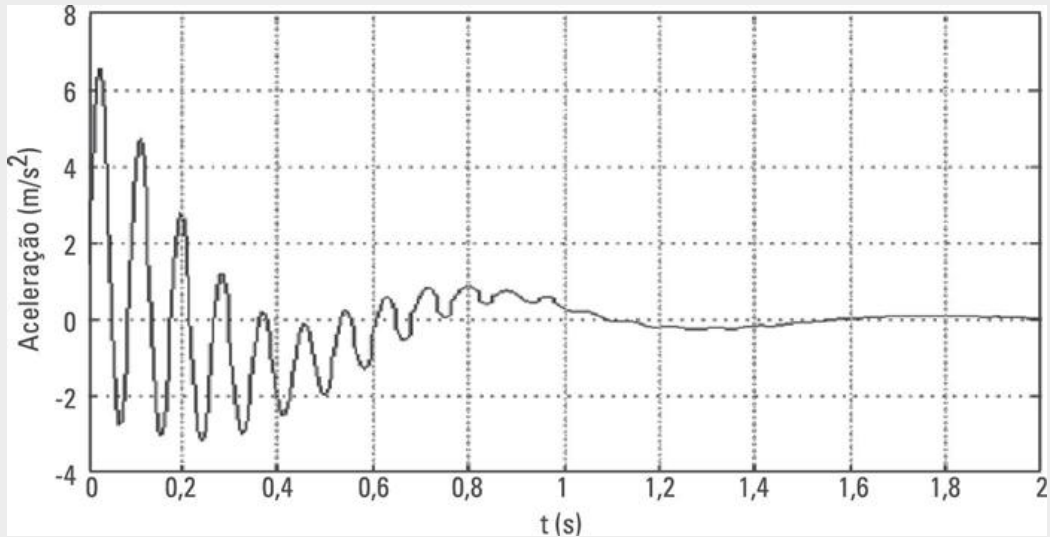


FIGURA 2.19 Resposta do sistema de suspensão veicular com a estratégia clássica.

A estratégia de controle convencional apresenta algumas limitações, pois a força (F_a) adicional de atuação do amortecedor ativo atua em duas faixas em relação aos valores dos coeficientes (C_{off} e C_{on}) ajustados. Este comportamento pode provocar solavancos e um ruído audível indesejável no sistema de suspensão.

Objetivando melhorar a estratégia de controle (2.22), as suas informações serão mapeadas em regras *fuzzy*, com a finalidade de se obter uma melhor atuação no sistema de suspensão.

A escolha adequada das funções de pertinência dos conjuntos nebulosos (simétricas e igualmente espaçadas como nas Figuras 2.20, 2.21 e 2.22), pode suavizar eventuais discontinuidades na força de atuação (isto tende a minimizar problemas de solavancos e ruídos indesejáveis). Os universos de discurso dos conjuntos nebulosos foram escolhidos em razão da faixa de excursão das variáveis utilizadas no modelo de suspensão ativa fornecido. Os conjuntos difusos para as variáveis utilizadas (V_{abs} , V_{rel} e F_a) terão partições denotadas linguisticamente como: *NG* (negativa grande); *NM* (negativa média); *ZE* (zero); *PM* (positiva média); *PG* (positiva grande).

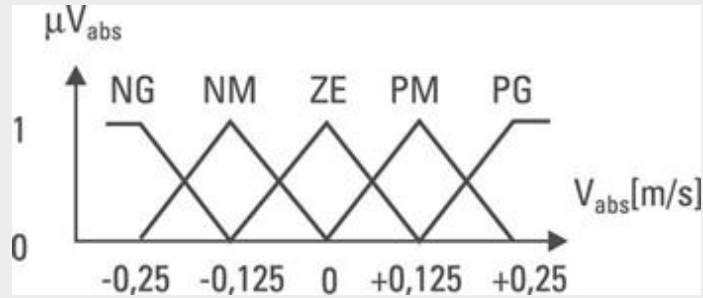


FIGURA 2.20 Funções de pertinência para V_{abs} .

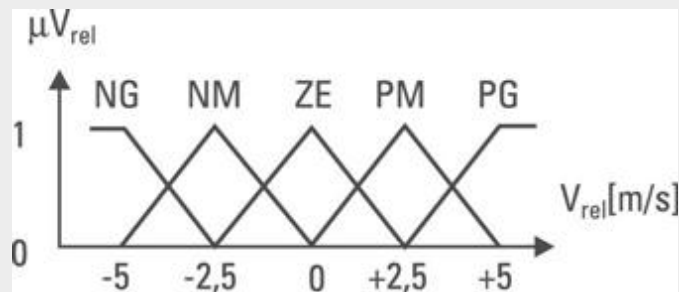


FIGURA 2.21 Funções de pertinência para V_{rel} .

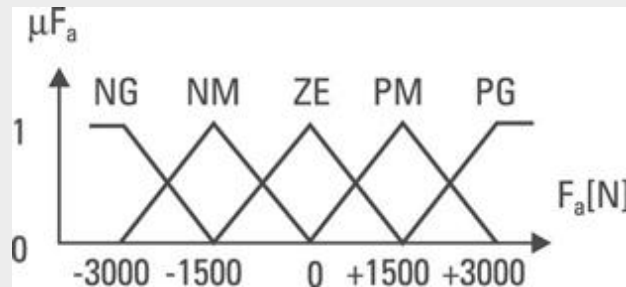


FIGURA 2.22 Funções de pertinência para F_a .

Genericamente, as regras *fuzzy* podem ser expressas por (2.23), onde G_i , H_i e I_i representam os conjuntos nebulosos do sistema em questão. As regras difusas serão representadas de forma tabular, como ilustrado na Tabela 2.5.

$$r_j : IF V_{abs} = G_j \text{ AND } V_{rel} = H_j \text{ THEN } F_a = I_j \quad (2.23)$$

Tabela 2.5**Representação tabular genérica das regras *fuzzy***

F_a		V_{abs}				
		NG	NM	ZE	PM	PG
V_{rel}	NG	I_1	I_6	I_{11}	I_{16}	I_{21}
	NM	I_2	I_7	I_{12}	I_{17}	I_{22}
	ZE	I_3	I_8	I_{13}	I_{18}	I_{23}
	PM	I_4	I_9	I_{14}	I_{19}	I_{24}
	PG	I_5	I_{10}	I_{15}	I_{20}	I_{25}

A seguir, será visto como obter as regras *fuzzy* para o sistema de suspensão ativa em questão. Analisando o algoritmo (2.22) é fácil constatar as condições (2.24) e (2.25).

$$V_{abs} V_{rel} \geq 0 \text{ se } \begin{cases} V_{abs} = 0 \text{ ou } V_{rel} = 0, \\ V_{abs} > 0 \text{ e } V_{rel} > 0, \\ V_{abs} < 0 \text{ e } V_{rel} < 0. \end{cases} \quad (2.24)$$

$$V_{abs} V_{rel} < 0 \text{ se } \begin{cases} V_{abs} > 0 \text{ e } V_{rel} < 0, \\ V_{abs} < 0 \text{ e } V_{rel} > 0. \end{cases} \quad (2.25)$$

Da primeira condição de (2.24) em (2.22) é fácil estabelecer que a regra I_{13} deve gerar um efeito nulo (ou próximo disto), ou seja, a força de atuação deve ter a atribuição de um conjunto nebuloso ZE . Com a última condição de (2.25) em (2.22) fica claro que I_5 deve ser PM , pois V_{rel} é intensa no sentido positivo e o coeficiente de amortecimento é baixo, assim a força tende a ser média e positiva. Com a primeira condição de (2.25) em (2.22) fica estabelecido que I_{21} é NM , pois V_{rel} é intensa no sentido negativo e o coeficiente de amortecimento é baixo, assim a força de atuação tende a ser média e negativa. Da terceira condição de (2.24) em (2.22) fica fácil perceber que I_1 é NG , pois V_{abs} é intensa no sentido negativo e o coeficiente de amortecimento é alto, assim a força tende a ser grande e negativa. De forma similar, analisando os limites de (2.24) e (2.25) em (2.22), torna-se simples estabelecer as outras regras *fuzzy* indicadas na Tabela 2.6. A computação dessas regras é semelhante às dos Exemplos 2.3 e 2.4

Tabela 2.6**Regras *fuzzy* para o sistema de suspensão ativa**

F_a		V_{abs}				
		NG	NM	ZE	PM	PG
	NG	NG_1	NG_6	NM_{11}	NM_{16}	NM_{21}
	NM	NG_2	NM_7	NM_{12}	NM_{17}	NM_{22}
V_{rel}	ZE	NM_3	NM_8	ZE_{13}	PM_{18}	PM_{23}
	PM	PM_4	PM_9	PM_{14}	PM_{19}	PG_{24}
	PG	PM_5	PM_{10}	PM_{15}	PG_{20}	PG_{25}

A [Figura 2.23](#) ilustra a resposta dinâmica do sistema de suspensão ativa (a aceleração da carroceria), agora com o algoritmo definido pelas regras *fuzzy*, considerando um deslocamento repentino de cinco centímetros em uma das rodas do veículo. Nota-se que os valores de aceleração são agora menores do que aqueles da [Figura 2.20](#), o que ocasiona um deslocamento mais suave na carroceria ([Figura 2.24](#): curva em linha cheia referente ao controle *fuzzy*; linha tracejada para a estratégia convencional). Nos artigos citados, essas regras foram implantadas na prática em um sistema real, em que os resultados práticos obtidos comprovaram o melhor desempenho da estratégia *fuzzy*.

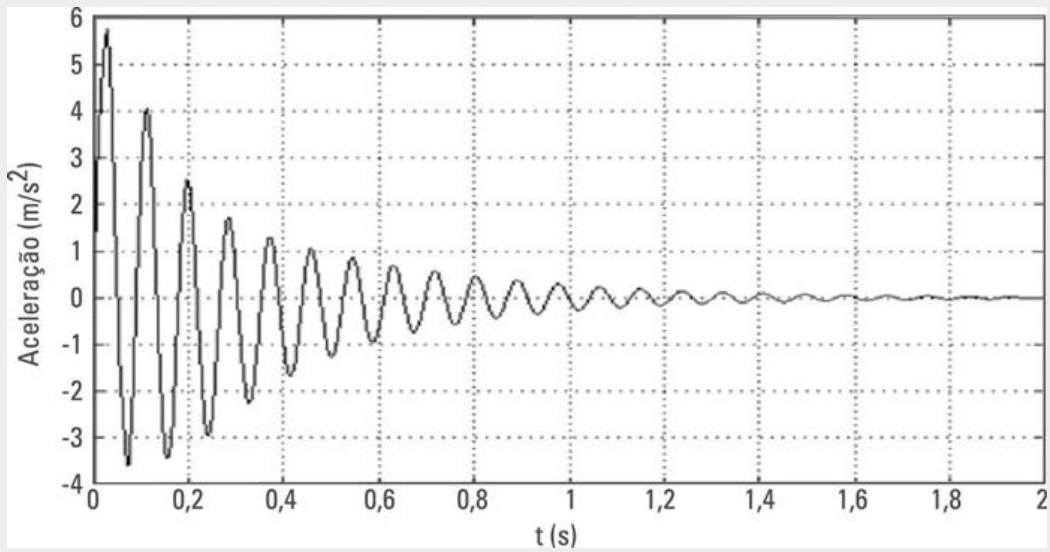


FIGURA 2.23 Resposta do sistema de suspensão veicular com a estratégia *fuzzy*.

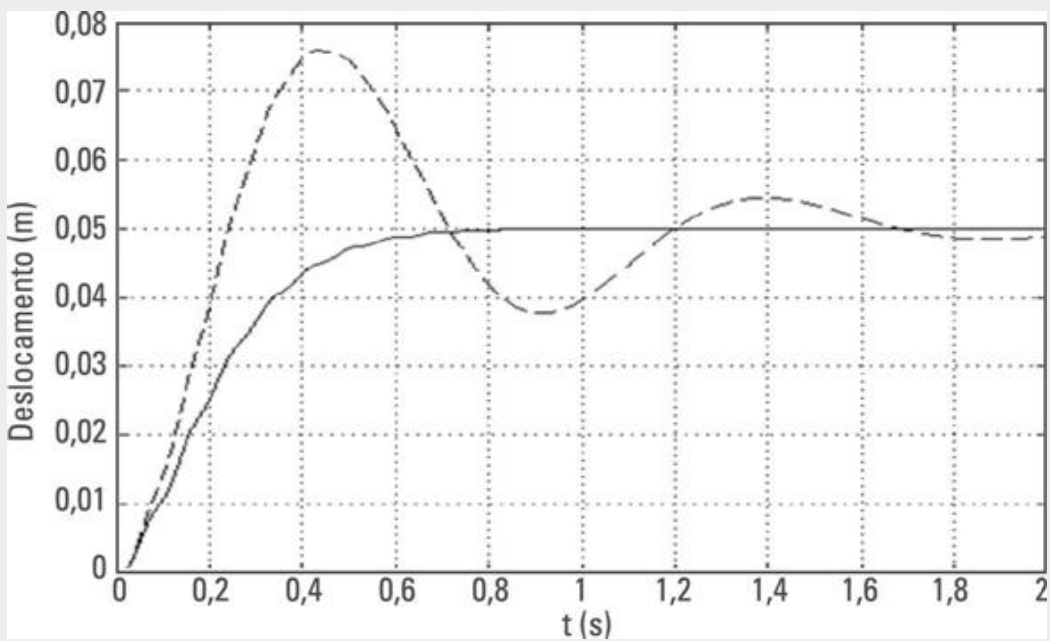


FIGURA 2.24 Deslocamento dos chassis com a estratégia convencional e *fuzzy* (linha cheia).

Exemplo 2.5

Este exemplo está relacionado com modelagem via regras *fuzzy* de um processo dinâmico de nível com comportamento não linear, onde será utilizado um procedimento de treinamento para a obtenção do modelo *fuzzy* correspondente.

Sistemas de nível são processos usuais em plantas industriais, e a equação (2.26) constitui um modelo contínuo típico destes sistemas, onde “*u*” representa a variável de comando do sistema e “*y*” a variável de saída (nível). Neste exemplo serão considerados os seguintes coeficientes para (2.26): $R = 1$ [m]; $r = 0,1$ [m]; $A = \pi R^2$ [m²]; $a = \pi r^2$ [m²]; $g = 9,8$ [m/s²]; $c = 0,1$ [m³/s/V].

$$A \frac{dy(t)}{dt} = c \cdot u(t) - a \sqrt{2g} \cdot \sqrt{y(t)} \quad (2.26)$$

Com a utilização frequente de instrumentos digitais em processos industriais, a representação (2.27) é mais adequada para efeito de modelagens práticas, onde o índice “*k*” denota informações atuais de entrada e saída do modelo, “*k – m*” simboliza as informações anteriores, e *f(.)* representa uma relação entre os valores amostrados.

$$y(k) = f(y(k-1), \dots, y(k-m), u(k), \dots, u(k-m)) \quad (2.27)$$

As figuras seguintes ilustram dados de entrada e saída do sistema em questão que serão utilizados neste exemplo. Os dados nas duas figuras representam as informações de entrada (*u*) e de saída (*y*) do sistema nas unidades de tensão [V] (de comando) e de nível [m], respectivamente. Cada amostra possui um intervalo de tempo de um segundo em relação à próxima. Os dados da Figura 2.25 serão utilizados para a obtenção dos modelos deste exemplo, e os da Figura 2.26 servirão para testar a modelagem em questão.

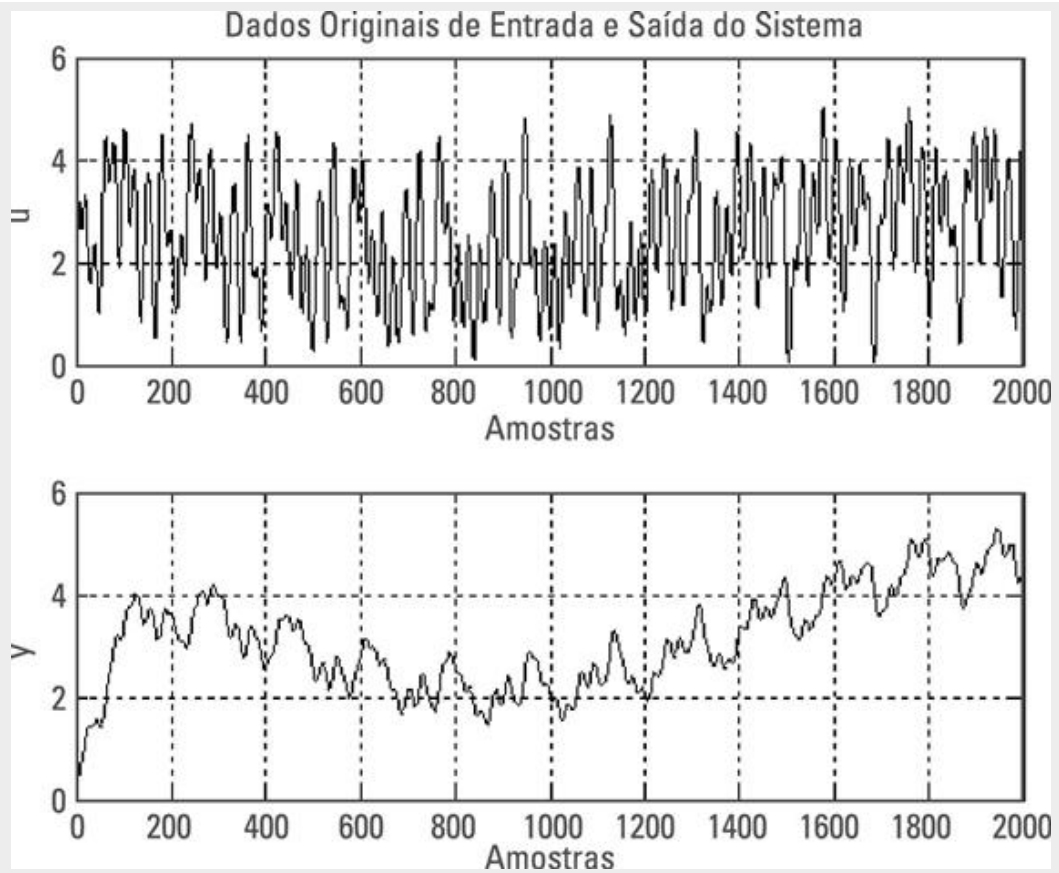


FIGURA 2.25 Dados originais de entrada e saída de um sistema de nível.

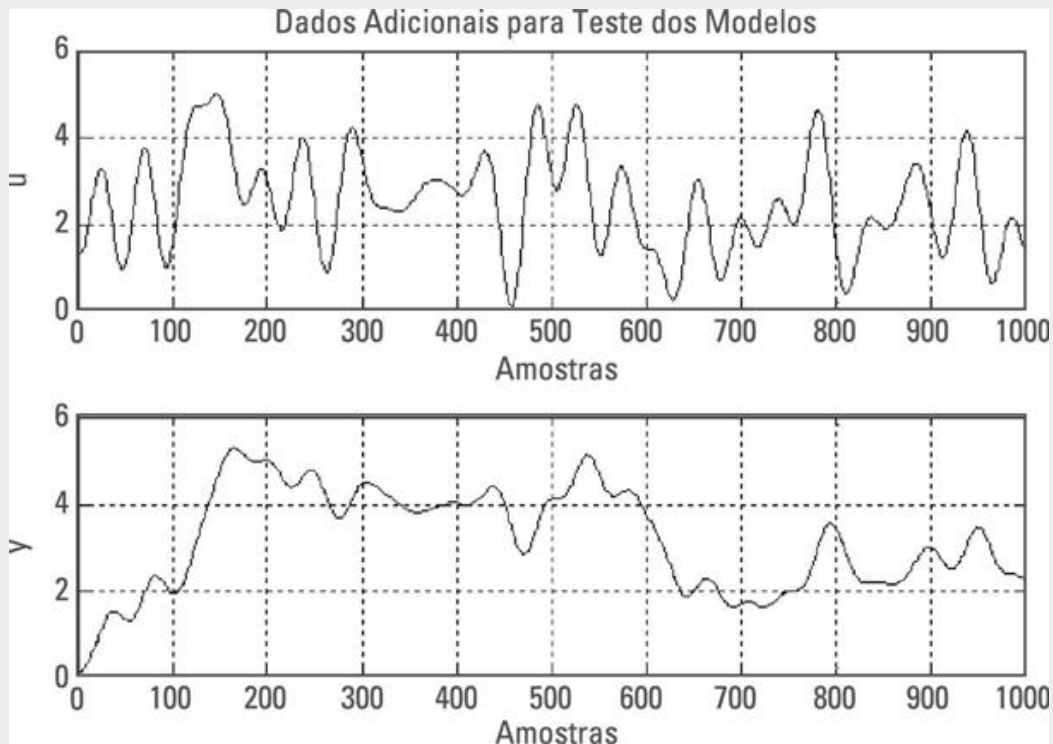


FIGURA 2.26 Dados de entrada e saída para teste do modelo de nível.

A [Figura 2.27](#) mostra uma forma de se utilizar um sistema *fuzzy* para modelar um determinado processo dinâmico, onde o operador “ z^{-1} ” simboliza amostras anteriores ($k-1$) das variáveis do sistema em questão. Neste exemplo será utilizada a notação: $x_1 = u(k-1)$; $x_2 = y(k-1)$; $y = y(k)$.

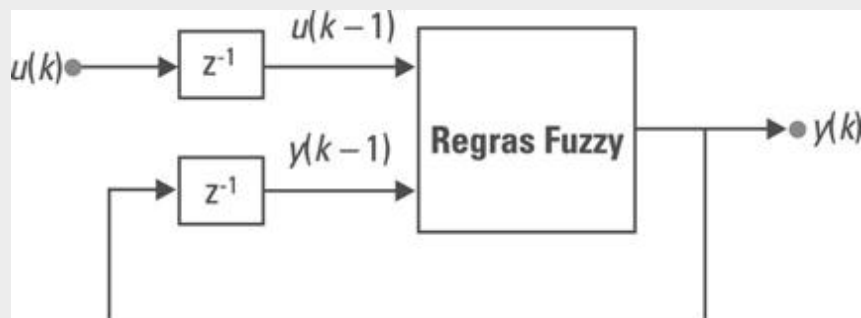


FIGURA 2.27 Exemplo de estrutura utilizada para modelagem *fuzzy* do processo.

A segunda listagem no [Anexo 2.4](#) contém comandos do *toolbox Anfis* do MatLab, onde é utilizado um procedimento baseado em treinamento

(aprendizado via retropropagação) para a obtenção das regras difusas e das funções de pertinência do modelo. Os dados referentes à [Figura 2.25](#) são usados para a etapa de treinamento, e os da [Figura 2.26](#) para o teste do modelo *fuzzy* resultante. A [Figura 2.28](#) ilustra o resultado do aprendizado utilizando os mesmos valores de entrada (da [Figura 2.25](#)). Os resultados na [Figura 2.29](#) estão relacionados ao teste do modelo (para os mesmos valores de entrada da [Figura 2.26](#)), cujo objetivo é verificar a capacidade de generalização do sistema *fuzzy* obtido. Nota-se que o modelo difuso apresentou excelente aproximação em relação aos dados do processo original (que tem comportamento não linear). Este tipo de modelagem é uma opção interessante para modelos Narx ou Narmax clássicos ([Aguirre, 2000](#)).

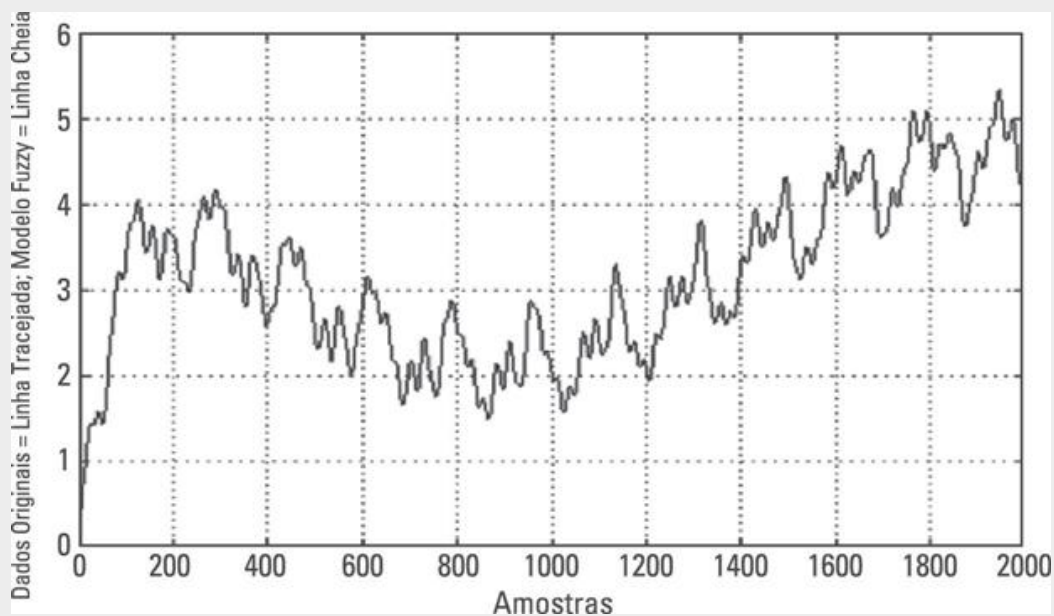


FIGURA 2.28 Dados (de treinamento) originais do processo e do modelo *fuzzy*.

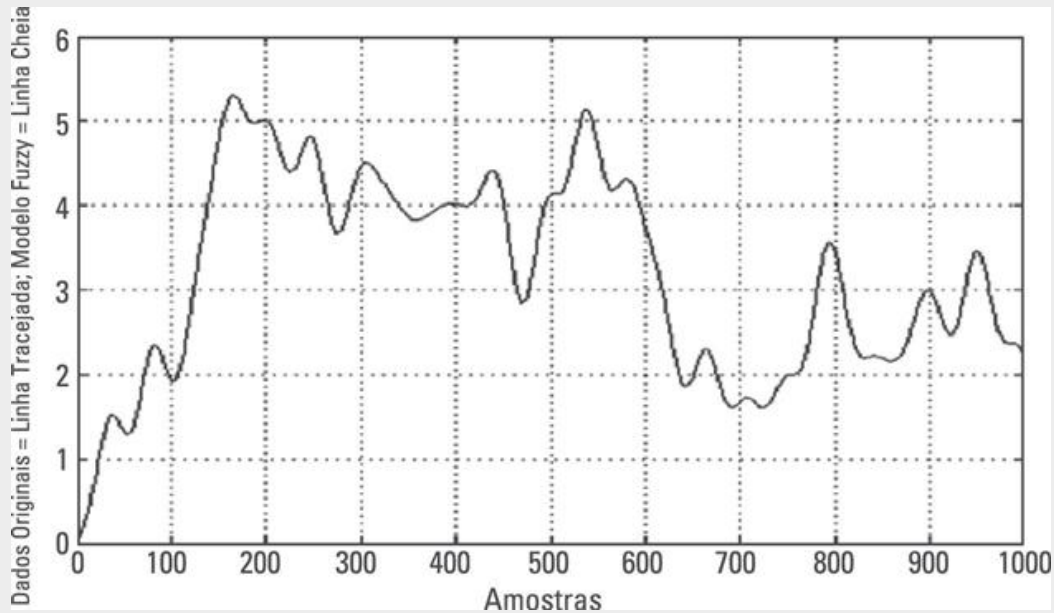


FIGURA 2.29 Dados (de teste) originais do processo e do modelo fuzzy.

Utilizando o *toolbox* “fuzzy” do MatLab podemos carregar, visualizar e gravar o modelo difuso obtido. Os valores modais dos parâmetros das funções de pertinência ficaram ajustados em: 2,109 e 4,032 para a variável x_1 ; 1,746 e 4,079 para a variável x_2 . Os coeficientes polinomiais para as quatro regras geradas são, respectivamente: 0,03178, 0,9843, -0,02191; 0,03194, 0,989, -0,04471; 0,03454, 0,9776, -0,01823; 0,03181, 0,989, -0,04434.

Exemplo 2.6

Este exemplo tem a estrutura similar ao anterior e a modelagem via regras fuzzy estará relacionada com a previsão de uma série temporal. A [Figura 2.30](#) contém dados de uma série temporal relativa a valores de arrouba de gado, onde se têm registros de valores (em reais) relativos a 73 meses. Serão utilizados valores dos primeiros 62 meses com a finalidade de projetar um modelo fuzzy, para depois verificar a capacidade deste de estimar os valores dos 11 meses seguintes.

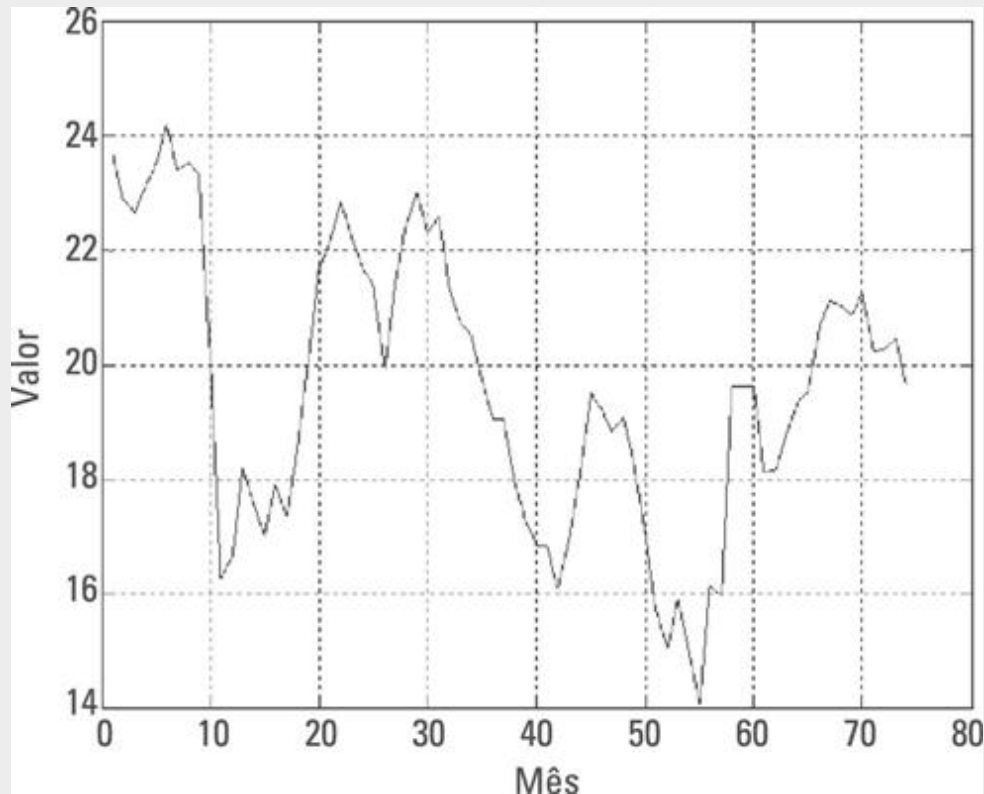


FIGURA 2.30 Dados da série temporal.

Será utilizada uma amostra atual (k) de valor de preço e outra de valor anterior ($k-1$) com a finalidade de estimar o preço do mês seguinte ($k + 1$). Assim, a estrutura básica da [Figura 2.26](#) será utilizada para modelar os dados da série em questão, onde as variáveis de modelagem são: x_1 = Valor do Preço Anterior; x_2 = Valor do Preço Atual; e Y = Valor do Mês Seguinte. Utilizando o mesmo procedimento de treinamento abordado no exemplo anterior, com um programa bem semelhante ao da segunda listagem do [Anexo 2.4](#), podemos obter o modelo *fuzzy* pretendido.

Anexo 2.4

Listagens referentes ao Exemplo 2.5.

```

% Modelo não linear de sistema de nível

clear all;

% Parâmetros do Modelo
R=1; A=pi*R^2; r=0.1; a=pi*r^2; g=9.8; c=0.1;

% Informações de Simulação
dt=1; h=0;
np=2000; %np=1000;

U = idinput(np,'sine',[0,1/10],[0,5]); % Entrada do Sistema (treino)
%U = idinput(np,'sine',[0,1/20],[0,5]); % Entrada do Sistema (teste)

for n=1:np %Laço de Simulação

    u = U(n); % Informação de Entrada do Sistema

    vu(n)=u; vh(n)=h; vn(n)=n-1; % Armazenamento de Dados

    dh = (c*u - a*sqrt(2*g*h))/A; % Equação do Modelo Não Linear

    h = h + dh*dt; % Integração da Variável de Saída

end

% Gráficos
subplot(211);
plot(vn,vu,'b');grid;
xlabel('Amostras'); ylabel('u');
subplot(212);
plot(vn,vh,'b');grid;
xlabel('Amostras'); ylabel('y');

% Dados do Modelo
x1=vu(1:np-1); x2 = vh(1:np-1); y = vh(2:np);
DadosNivel_Treino = [x1' x2' y'];
save DadosNivel_Treino;
%DadosNivel_Testes = [x1' x2' y'];
%save Dados Nivel_Testes;
% Exemplo de obtenção do modelo fuzzy do processo de nível via Anfis

clear all;

% Dados do Sistema
load DadosNivel_Treino;
DadosSist = DadosNivel_Treino;
x1=DadosSist(:,1); x2=DadosSist(:,2); y=DadosSist(:,3);

% Obtenção de Modelo Fuzzy
dadosT = [x1 x2 y]; % Dados de Treinamento
n_epoca = 100; % Número de Épocas
NFP = 2; % Número de Funções de Pertinência (FP)

```

```

fismat = genfis1(dadosT,NFP,'trapmf'); % Definição das FP - Modelo
mod_fuzzy = anfis(dadosT,fismat,n_epoca); % Procedimento de Treino

% Verificação do Treinamento
Xn = [x1 x2]; % Dados de Entrada do Modelo
Ye = evalfis([Xn],mod_fuzzy); % Estimações do Modelo Fuzzy

% Gráfico do Treinamento
Np=size(y); np=Np(1); vK=1:np; % Número de Amostras
plot(vK',y,'k--',vK',Ye,'k'); grid;
xlabel('Amostras'); ylabel('Dados Originais = Linha Tracejada; Modelo
Fuzzy = Linha Cheia');
figure

% Somatória do Erro Quadrático do Modelo
er = y - Ye; eq= er.*er;
seqm = sum(eq)

% Dados do Sistema
load DadosNivel_Testes;
DadosSist = DadosNivel_Testes;
x1=DadosSist(:,1); x2=DadosSist(:,2); y=DadosSist(:,3);

% Teste de Validação do Modelo Fuzzy
Xn = [x1 x2]; % Dados de Entrada do Modelo
Ye = evalfis([Xn],mod_fuzzy); % Estimações do Modelo Fuzzy

% Gráfico do Teste
Np=size(y); np=Np(1); vK=1:np; % Número de Amostras
plot(vK',y,'k--',vK',Ye,'k'); grid;
xlabel('Amostras'); ylabel('Dados Originais = Linha Tracejada; Modelo
Fuzzy = Linha Cheia');

% Somatória do Erro Quadrático do Modelo
er = y - Ye; eq= er.*er;
seqm = sum(eq)

```

A [Figura 2.31](#) ilustra o resultado do aprendizado com os de treinamento, onde em linha tracejada estão os dados originais da série e em linha cheia temos os dados do modelo *fuzzy*. Na [Figura 2.32](#) temos o resultado do modelo com os dados de teste (em linha tracejada estão os dados originais e em linha cheia temos os dados do modelo *fuzzy*), cujo objetivo é verificar a capacidade de previsão do sistema *fuzzy* obtido. Nota-se que o modelo difuso apresentou uma razoável capacidade de previsão, que pode ser melhorada com a inclusão no modelo de mais entradas referentes a amostras passadas, e até mesmo a entradas externas como informação do mês, taxa de juros etc. A modelagem baseada em regras é uma opção

interessante para previsão de séries temporais (Faustino et al., 2011; Faustino et al., 2012).

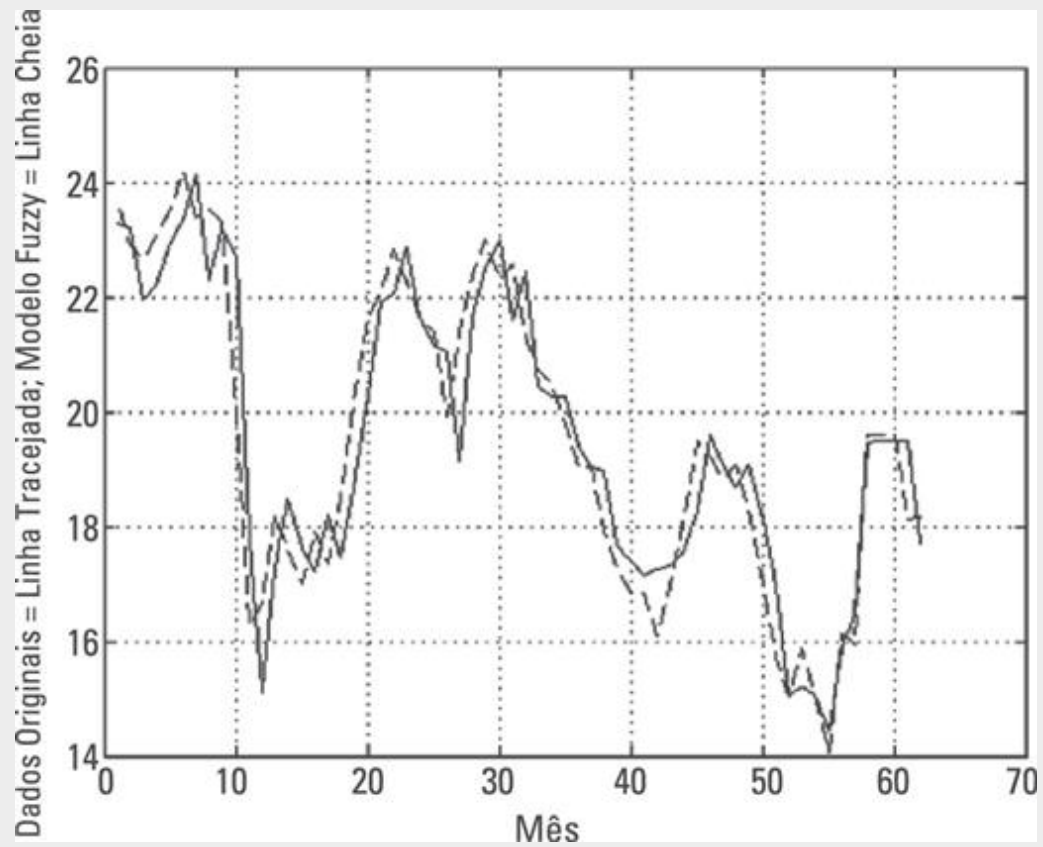


FIGURA 2.31 Dados (de treinamento) originais da série temporal e do modelo *fuzzy*.

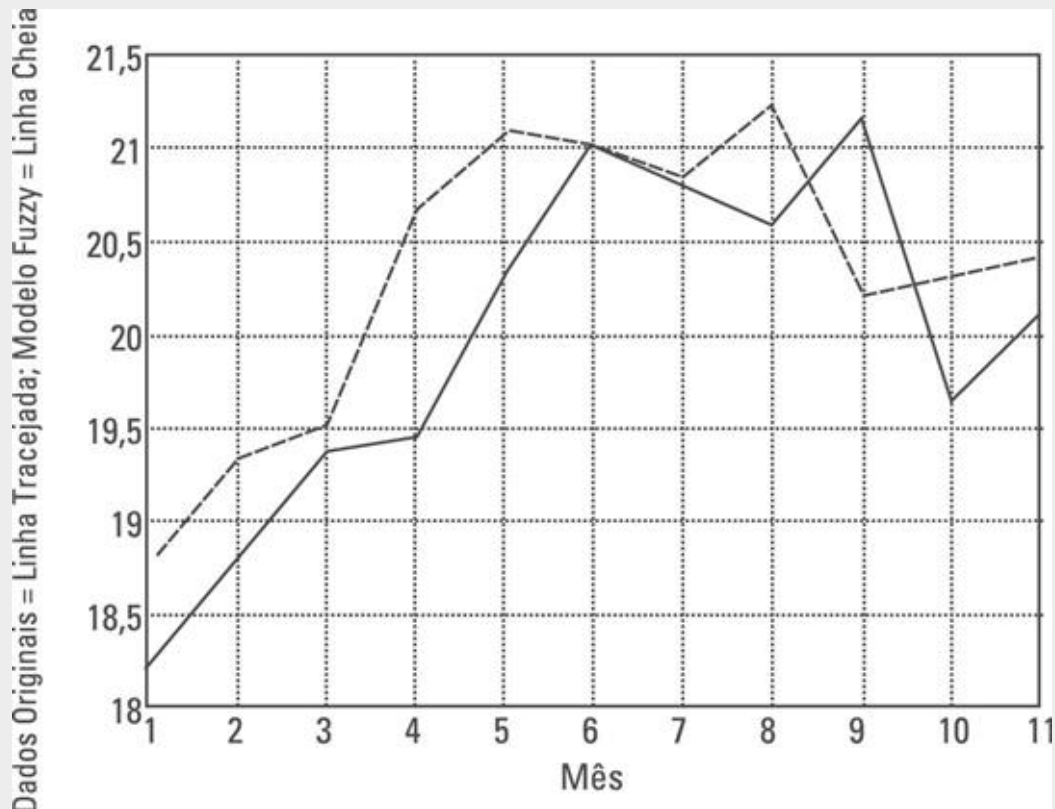


FIGURA 2.32 Dados (de teste) originais da série temporal e de previsões do modelo *fuzzy*.

2.6 Exercícios

Exercício 2.1

Tomando como base os conjuntos nebulosos representados na [Figura 2.6](#), processar a união dos conjuntos *A* e *B*. Apresentar o resultado obtido na forma gráfica e por meio dos dados expressos via seus respectivos pares ordenados.

Exercício 2.2

Tomando como base os conjuntos nebulosos representados na [Figura 2.6](#), processar o complemento do conjunto *B*. Apresentar o resultado obtido na

forma gráfica e por meio dos dados expressos via seus respectivos pares ordenados.

Exercício 2.3

Em relação ao exercício anterior, processar a união do conjunto A com o complemento do conjunto B .

Exercício 2.4

Utilizando o modelo *fuzzy* definido pelas regras (2.19) e com as informações da Figura 2.10, calcular o valor de saída do sistema para $x_1 = 3,5$ e 6 .

Exercício 2.5

Em relação aos resultados ilustrados na Figura 2.12, utilizar agora funções de pertinência do tipo gaussiana na Listagem 2.1 (parâmetro: TipoFunc = “gaussmf”), e verificar se a aproximação obtida ficou mais suave.

Listagem 2.1

```
DadosTreinamento = [x1 y];  
    NumFuncPertinen = 2; TipoFunc = 'trapmf'; EpocasTreinamento =  
100;  
    ModF = genfis1(DadosTreinamento, NumFuncPertinen, TipoFunc); %  
Modelo Fuzzy  
    ModFuzzy = anfis(DadosTreinamento, ModF, EpocasTreinamento); %  
Treinamento  
    y = evalfis(x1,ModFuzzy); plot(x1,vety,'k.',x1,y,'k'); xlabel('x1');  
ylabel('y'); grid;
```

A Figura 2.12 ilustra os resultados obtidos com a ferramenta descrita. Verifica-se que as aproximações foram bem melhores para este modelo *fuzzy* obtido com esse procedimento de aprendizado de máquina.

O sistema nebuloso resultante pode ser arquivado via *toolbox* “fuzzy” (mais detalhes sobre esta *toolbox* estão presentes no [Anexo 2.2](#)), e os parâmetros do modelo podem ser visualizados nesse ambiente. As funções de pertinência obtidas estão ilustradas na [Figura 2.13](#) (em um exercício proposto no final deste capítulo será sugerida a utilização de funções gaussianas objetivando a obtenção de uma aproximação mais “suave”). As regras *fuzzy* de modelagem são similares as de (2.19), mas com valores dos coeficientes distintos:

$$r_1 : IF \ x_1 = A_1 \ THEN \ y_1 = 0,5711x_1 + 2,049;$$

$$r_2 : IF \ x_1 = A_2 \ THEN \ y_2 = 0,1875x_1 + 9,139.$$

Exercício 2.6

Utilizando o modelo *fuzzy* definido pelas regras da Tabela 2.19 e informações da [Figura 2.10](#), calcular o valor de saída do sistema para $x_1 = -0,8$ e $x_2 = 0,3$.

Exercício 2.7

Em relação ao [Exemplo 2.3](#), calcular o valor de comando do sistema para $x = 5$ e $\Phi = 30^\circ$.

Exercício 2.8

Em relação à listagem do Anexo 3 referente ao [Exemplo 2.3](#), trocar a composição tipo produto nas regras pelo operador min (mínimo) e verificar

se a resposta do sistema é semelhante ao gráfico da [Figura 2.17](#) para as mesmas condições iniciais: $x = 5$; $y = 10$; $\Phi = 30^\circ$.

Exercício 2.9

Em relação ao [Exemplo 2.4](#), calcular o valor de comando (F_a) do sistema para $V_{abs} = 0,125$ [m/s] e $V_{rel} = -2,5$ [m/s].

Exercício 2.10

Em relação aos resultados ilustrados na [Figura 2.29](#), utilizar agora funções de pertinência triangulares na listagem do anexo correspondente (agora com opção *trimf* ao invés de *trapmf*), e verificar os resultados da modelagem. Anotar também os parâmetros do modelo *fuzzy* obtido.

Referências bibliográficas

1. AGUIRRE, L.A. (2000). **Introdução à identificação de sistemas: técnicas lineares e não lineares aplicadas a sistemas reais**. Belo Horizonte, Editora UFMG.
2. CHEN CL, KUO FC. Design and analysis of a fuzzy logic controller. *Int J System Sci*. 1995;26:1.223–48.
3. FAUSTINO, C.P. (2011). **Previsão de séries temporais via modelos baseados em regras**. Universidade Federal de Itajubá. Dissertação de Mestrado, 127 p.
4. FAUSTINO CP, PINHEIRO C, CARPINTEIRO O, LIMA I. Time series forecasting through rule-based models obtained via rough sets. *Artificial Intelligence Review* 2011;(n. 36, p):299–310.
5. FAUSTINO CP, NOVAES CP, PINHEIRO C, CARPINTEIRO O. Improving the performance of fuzzy rules-based forecasters through application of FCM algorithm. *Artificial Intelligence Review*. 2012;1:10–24.
6. JANG J. Anfis: Adaptive-network-based fuzzy inference system. *IEEE Transaction on Systems, Man and Cybernetics*. 1993;23(n. 3,

- p):665–685.
7. KLIR GJ, FOLGER TA. Fuzzy Sets, Uncertainty and Information Nova Jersey. *Prentice Hall* 1998.
 8. KOSKO B. Fuzzy Systems as Universal Approximators. *IEEE Transactions on Computers*. 1994;43(11):1.329–1.333.
 9. ———. *Fuzzy Engineering*. Nova Jersey: Prentice Hall; 1997.
 10. MAMDANI EH. Applications of fuzzy algorithms for control of simple dynamic plant. *Proc IEE*. 1974;121(12):1.585–1.588.
 11. ———. Advances in the linguistic synthesis of fuzzy controllers. *International Journal of Man-Machine Studies*. 1976;8(n.6 p):78–669.
 12. PATTERSON DW. *Introduction to Artificial Intelligence & Expert Systems*. Nova Jersey: Prentice Hall; 1990.
 13. PINHEIRO C, GOMIDE F. On the Equivalence Between Basic Fuzzy and Classical Controllers. *Proceedings of the Eighth International Fuzzy Systems Association World Congress*. 1999;2:594–597 Taipei.
 14. PINHEIRO, C. (2004). **Estratégia de controle fuzzy para suspensão ativa automobilística**. XV Congresso Brasileiro de Automática.(em CD).
 15. PINHEIRO C, MACHADO J, BOMBARD A, LIMA J, DIAS J. ***Fuzzy Logic Control for Magnetorheological Damper***. *Electrorheological Fluids and Magnetorheological Suspensions*. Cingapura: World Scientific; 2007; p. 09-603.
 16. TAKAGI T, SUGENO M. Fuzzy identification of systems and its applications to modelling and control. *IEEE Transactions on Systems Man and Cybernetics*. 1985;15(1):32–116.
 17. WANG LX, MENDEL JM. Generating fuzzy rules from examples. *IEEE Transactions on Systems, Man and Cybernetics*. 1992;22(6):1414–1427.
 18. WANG LX. *Adaptive Fuzzy Systems and Control*. Nova Jersey: Prentice Hall; 1994.
 19. ZADEH LA. Fuzzy Sets. *Information and Control* 1965;(n. 8, p):338–353.

CAPÍTULO

TRÊS

Redes Neurais Artificiais

3.1 Introdução

Redes Neurais Artificiais ou simplesmente Redes Neurais (também conhecidas como modelos conexionistas) têm sido, ao longo dos últimos anos uma área da IA de grande desenvolvimento. Redes Neurais podem ser caracterizadas como modelos computacionais com capacidades de adaptar, aprender, generalizar, agrupar ou organizar dados, nos quais a estrutura operacional é baseada em processamento paralelo. A origem das pesquisas nesta área data da época em que foi apresentado o modelo computacional de um neurônio biológico por McCulloch e Pitts (1943). Quando Minsky e Papert publicaram o livro *Perceptrons* em 1969, no qual eles mostraram as deficiências do modelo Perceptron, que possuía um algoritmo de treinamento básico, a maioria dos estudos em redes neurais foram redirecionados e muitos pesquisadores abandonaram a área. Uns poucos continuaram suas pesquisas notadamente Teuvo Kohonen, Stephen Grossberg, James Anderson e Kunihiko Fukushima. O interesse pela área somente ressurgiu depois que alguns resultados teóricos importantes foram atingidos, dentre os quais se destaca o algoritmo de retropropagação (*backpropagation*) em redes Perceptrons de múltiplas camadas. Hoje, a maioria das universidades tem um grupo de pesquisas em redes neurais dentro dos departamentos de Psicologia, Física, Informática, Biologia ou Engenharia.

Neste contexto é realizada uma introdução a Redes Neurais Artificiais (RNA). O ponto de vista desta abordagem é computacional. Não há preocupações com implicações biológicas, neurológicas etc.

3.2 Conceituação

As redes neurais possuem arquiteturas baseadas em blocos construtivos semelhantes entre si e que realizam o processamento de forma paralela. Nestas

capítulo discutiremos, primeiramente, estas unidades de processamento e a seguir os tipos de redes e suas diferentes topologias. Posteriormente, serão apresentadas algumas estratégias de aprendizado.

3.2.1 Fundamentação do modelo

Uma rede neural é formada por um conjunto de unidades básicas de processamento que se comunicam enviando informações uma para a outra por meio de determinadas conexões. O componente elementar desse modelo são as unidades de processamento, também chamadas nós, neurônios ou células. Essa unidade de processamento é um modelo matemático que possui inspiração no modelo biológico de um neurônio.

3.2.2 Modelo de neurônio biológico

Um sistema nervoso central possui milhões de células nervosas (neurônios) que se interligam entre si, formando uma rede disposta em camadas. Nessas redes são processadas todas as informações referentes a uma determinada funcionalidade, como visão, audição, aprendizado etc. A capacidade de pensar, memorizar e resolver problemas tem levado muitos cientistas a tentar modelar o funcionamento de várias partes do cérebro humano ou de outras entidades biológicas. Diversos pesquisadores de áreas distintas como Psicologia, Neurociências, Computação, Engenharia e Matemática estão reunindo esforços conjuntos neste sentido. A [Figura 3.1](#) contém uma microfotografia de neurônios humanos.

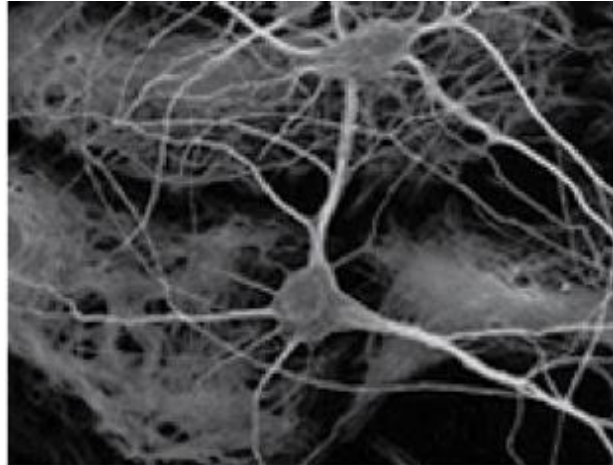


FIGURA 3.1 Microfotografia de neurônios humanos. Fonte: Science Photo Library Site.

Um neurônio é uma célula nervosa cuja representação básica está ilustrada na [Figura 3.2](#).

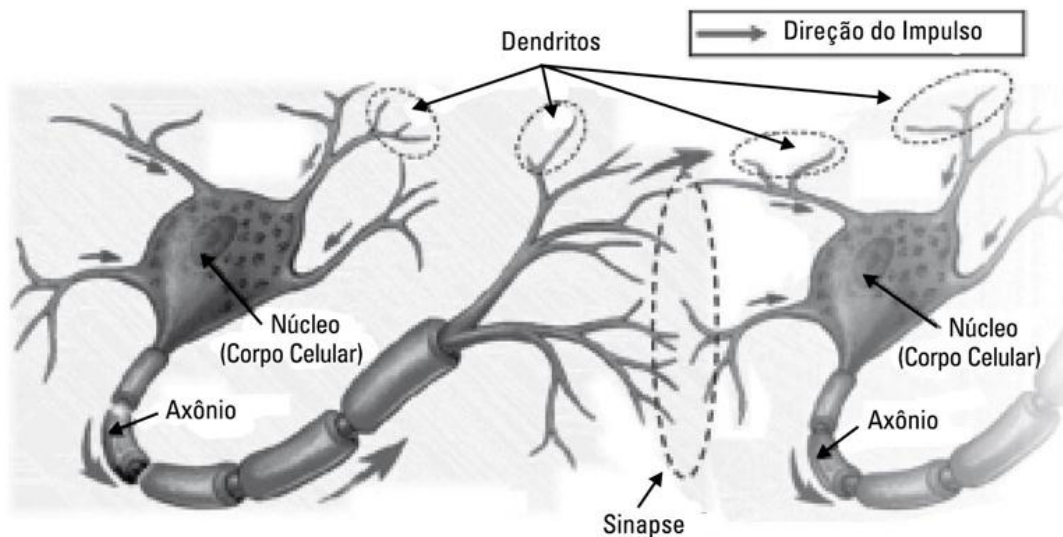


FIGURA 3.2 Ilustração de um neurônio biológico. Fonte: Acervo de conteúdo livre da Wikimedia Foundation.

Os estímulos entram nos neurônios através das sinapses que se conectam aos dendritos (ramificações de entrada de um neurônio) com axônios de outros neurônios.

As sinapses regulam as quantidades de informações que passam dos dendritos para a célula nervosa em questão. As informações são então

passadas para o somador (corpo celular), que as adiciona e as aplica a um discriminador, onde se determina o nível mínimo de entrada acima do qual o neurônio dispara uma carga elétrica como resposta a um ou mais estímulos. Entretanto, se a soma for menor que um determinado limiar, nenhuma informação é propagada para os neurônios vizinhos.

As sinapses têm um papel fundamental na memorização, pois é nelas que são armazenadas as informações.

Em média, os neurônios mais complexos de um cérebro humano possuem entre mil e 10 mil sinapses. O cérebro humano possui cerca de 100 bilhões de neurônios, e o número possível de sinapses é de mais de 100 trilhões possibilitando a formação de uma rede neural muito complexa.

3.2.3 Modelo de neurônio artificial

Objetivando simular o comportamento de um neurônio biológico, McCulloch e Pitts (1943) propuseram um modelo matemático, cuja representação básica está ilustrada na [Figura 3.3](#). Neste modelo cada informação de entrada é multiplicada por um número ou peso, sendo devidamente totalizada (somadas) em um nó ou ponto de soma. Caso a informação resultante exceda certo limiar definido por uma função chamada ativação, é gerada uma resposta na saída do modelo.

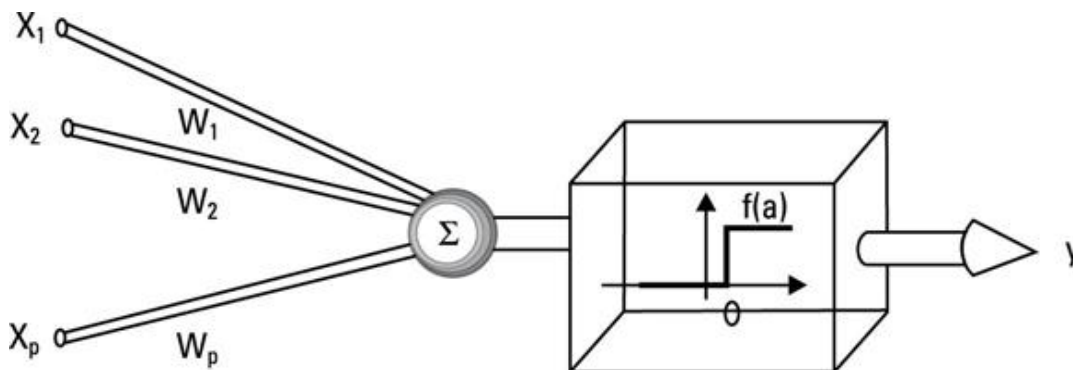


FIGURA 3.3 Modelo matemático básico de um neurônio. Fonte: Tafner et al., 1996.

Onde:

$X_1, X_2, \dots, X_p$ são as informações de entrada.

$W_1, W_2, \dots, W_p$ são os pesos aplicados nas entradas.

Σ é um nó ou ponto de soma, tendo como saída uma informação “a” (função líquida).

$f(a)$ é a função de ativação que, no caso do neurônio de McCulloch-Pitts, é uma função de limiar.

Y é a saída do modelo.

3.2.4 Funções de ativação

Outros modelos de neurônio possuem funções de ativação distintas. O papel da função de ativação é determinar a forma e a intensidade de alteração dos valores transmitidos de um neurônio a outro. É preciso quantificar a influência de cada entrada na ativação da unidade. Para isso se define uma função de ativação f_i , na qual uma determinada informação $a_i(t)$ produz um valor para a ativação da unidade i da rede em questão.

Diferentes tipos de funções de ativação têm sido utilizados, conforme detalhado a seguir.

3.2.5 Função de ativação limiar

A primeira rede com neurônios artificiais proposta utilizava uma função de ativação por limiar (Rosenblatt, 1958). O modelo de neurônio usado por Rosenblatt foi o mesmo sugerido por McCulloch-Pitts (1943).

Neste modelo, as unidades são ditas binárias, isto é, suas saídas assumem normalmente o valor “0” ou “1”. A unidade correspondente responde com uma saída igual a 1 se o valor da entrada líquida for superior a um determinado valor chamado *threshold* (que na maioria das vezes é zero), caso contrário, a saída da unidade assume um valor nulo (Figura 3.4).

$$y = f_i(a_i(t)) = \begin{cases} 1 & \text{se } a_i(t) \geq 0 \\ 0 & \text{se } a_i(t) < 0 \end{cases}$$

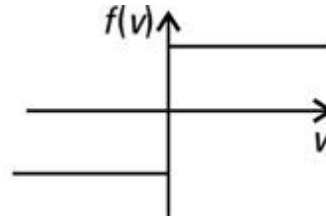


FIGURA 3.4 Função de ativação limiar.

Em vez dos valores 0 e 1, são utilizados também os limiares -1 e 1 . Uma rede de camada simples utilizando este tipo de unidade é denominada *Perceptron Simples*.

3.2.6 Função de ativação linear

Em 1972, Kohonen definiu um novo modelo de redes neurais conhecido como Mapa Auto-Organizável de Caracterização que, ao contrário do Perceptron não era limitado a valores binários. Os valores das entradas, dos pesos e das saídas poderiam ser contínuos.

A saída do neurônio neste caso é representada por uma função linear da forma descrita na [Figura 3.5](#).

$$y = f_i(a_i(t)) = \begin{cases} 1 & \text{se } a_i(t) \geq 0.5 \\ a_i(t) & \text{se } -0.5 < a_i(t) < 0.5 \\ -1 & \text{se } a_i(t) \leq -0.5 \end{cases}$$

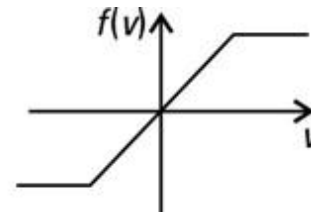


FIGURA 3.5 Função de ativação linear.

As redes que usam este tipo de função de ativação apresentam apenas uma camada de entrada e uma de saída. Tais redes possuem uma série de limitações quanto ao que são capazes de representar ([Hertz et al., 1991](#)).

Não existe nenhuma vantagem de uma rede com mais de duas camadas ou com conexões do tipo *feedback*. A capacidade de aprendizado de uma rede de multicamadas (mais de um neurônio) com neurônios, usando este tipo de função de ativação, pode também ser realizada por uma de camadas simples equivalente ([Hertz et al., 1991](#)). Na verdade, uma rede multicamadas que utilize neurônios artificiais com funções de ativação, com algum tipo de não

linearidade, tem um potencial de aprendizado muito maior que uma rede com unidades lineares (Minsky e Paper, 1969). Com unidades não lineares em uma rede com pelo menos uma camada escondida é possível representar algumas associações que não são possíveis de representar com uma rede linear ou uma camada simples que utilize a mesma função de ativação. Entretanto, Minsky e Paper (1969) mostraram que não existe um algoritmo que assegure o treinamento desse tipo de rede multicamadas.

3.2.7 Funções de ativação semilineares

Uma das respostas à falta de um algoritmo de treinamento para o tipo de rede citado é a utilização de redes com unidades semilineares (Hertz et al., 1991). As funções de ativação mais usadas por este tipo de unidades são a função logística (Figura 3.6) e a tangente hiperbólica. A popularidade dessas funções está ligada ao fato de suas derivadas (necessárias nas etapas de treinamento) poderem ser expressas a partir das próprias funções.

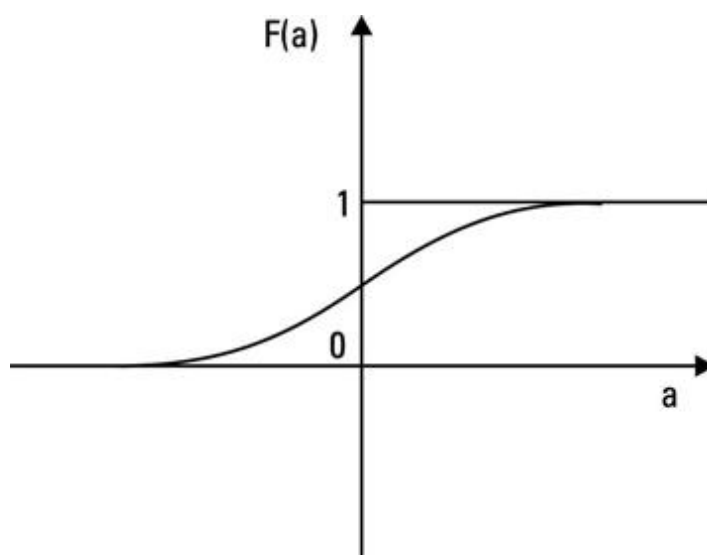


FIGURA 3.6 Função de ativação logística.

Estas funções podem ser expressas pelas equações abaixo, onde a representa a entrada líquida da unidade.

$$f(a) = \frac{1}{1 + e^{(-2\beta a)}} \quad g(a) = \tanh(\beta a)$$

As respectivas derivadas das funções acima são dadas por:

$$f'(a) = 2\beta f(1 - f) \quad g'(a) = \beta(1 - g^2)$$

3.3 Modelos de Redes Neurais Artificiais – RNAs

Uma RNA é um modelo matemático simplificado de um neurônio biológico que consiste na conexão de várias unidades básicas denominadas elementos de processamento. Estes elementos são organizados em camadas, onde cada elemento tem conexões ponderadas com outros elementos. Essas conexões podem tomar diferentes configurações, dependendo da aplicação desejada. Uma RNA ([Figura 3.7](#)) é, geralmente, constituída de uma camada de entrada que recebe os estímulos do modelo, e de uma camada de saída, que produz a resposta correspondente. Algumas redes podem ter uma ou mais camadas internas (também chamadas ocultas ou escondidas). A capacidade computacional de uma rede neural está nas conexões entre os elementos processadores. Nos pesos que ponderam cada conexão são armazenadas as informações que a rede “aprendeu”.

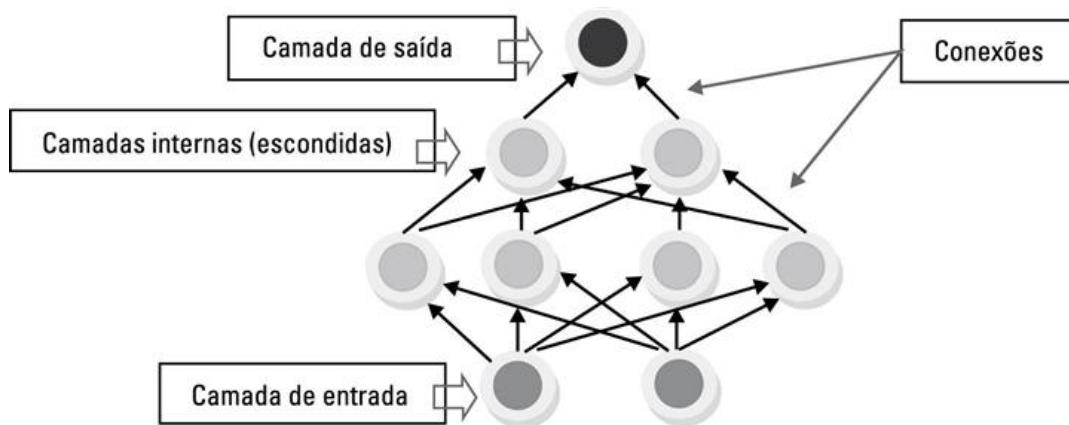


FIGURA 3.7 Representação de uma RNA típica.

Da combinação entre elementos de processamento e conexões surgem as diferentes topologias de Redes Neurais Artificiais. Pode-se fazer uma analogia com um grafo onde os nodos são os elementos de processamento e as arestas são as conexões.

3.4 Topologias

Em uma RNA, os neurônios podem estar agrupados por camadas direcionadas ou não, com ligações em um sentido (para a frente) ou em ambos (para a frente e para trás). As redes com ligações somente à frente, também são chamadas de diretas ou *feed-forward*, e se dividem em redes monocamada e multicamadas. As redes com ligações em mais de um sentido, também são chamadas de redes com ciclos, se dividem em redes recorrentes e redes competitivas.

3.4.1 Redes diretas (*feed-forward*)

São aquelas cujo grafo não apresenta ciclos. Seu aspecto geral é o da [Figura 3.8a](#). Frequentemente, é possível representar essas redes em camadas, que neste caso, são chamadas de redes multicamadas. No caso de redes organizadas por camadas, um neurônio de uma camada só pode se conectar a um neurônio da camada seguinte, como na [Figura 3.8b](#). No caso mais simples as redes em camadas possuem uma só camada, como na [Figura 3.8c](#).

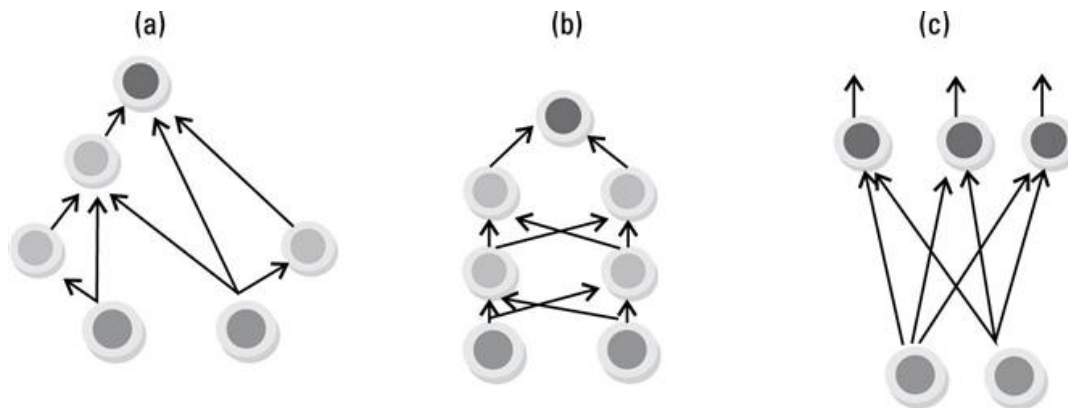


FIGURA 3.8 Redes diretas.

3.4.2 Redes com ciclos

Redes com ciclos são aquelas cujo grafo de conectividade contém ao menos um ciclo, e cujos neurônios apresentam a possibilidade de realimentarem outros neurônios. Além disso, podem envolver neurônios dinâmicos chamados de recorrentes ([Figura 3.9a](#)). Quando os neurônios competem entre si, de forma que só as saídas de alguns são consideradas nos processamentos resultantes, as redes resultantes são chamadas de competitivas ([Figura 3.9b](#)).

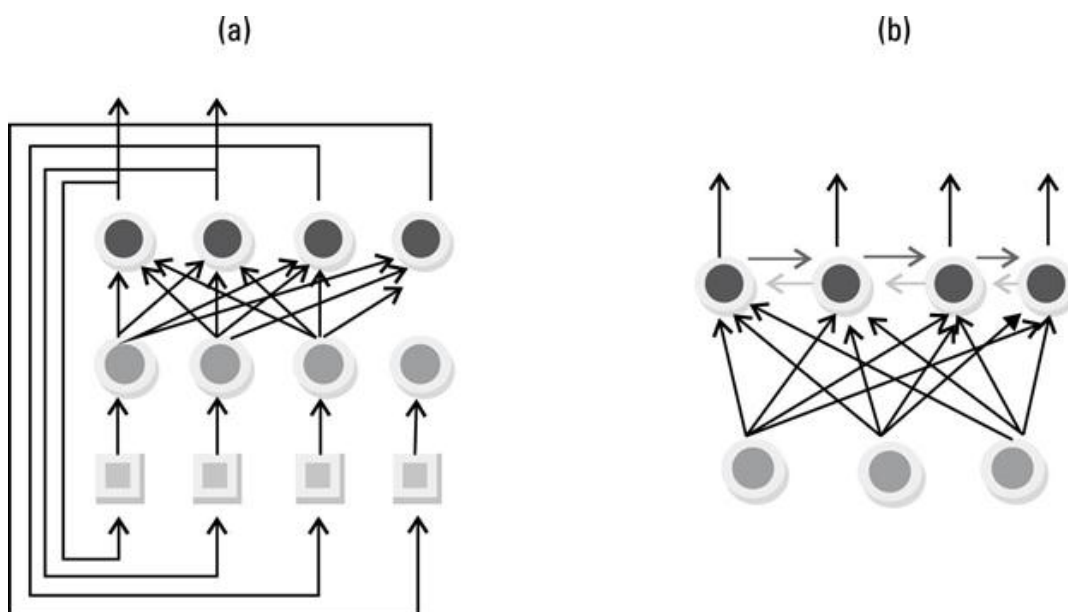


FIGURA 3.9 Redes com ciclos.

3.5 Treinamento e generalização

Os sistemas de computação neural são dinâmicos e autoadaptativos. Eles são autoadaptáveis porque os elementos de processamento têm capacidade de autoajuste. São também capazes de utilizar uma “experiência” para modificar as respostas das redes em determinadas situações. São dinâmicos porque podem estar mudando constantemente para adaptar-se a novas condições, de forma a responder satisfatoriamente a novas entradas ou estímulos. Três processos compreendem a adaptabilidade dos sistemas neurais: o aprendizado, o treinamento e a generalização.

3.5.1 Aprendizado

Aprendizado é a autoadaptação ao nível de elemento de processamento. No processo de aprendizado as conexões ponderadas são ajustadas para encontrar resultados específicos. Através de um processo que chamamos de treinamento ou aprendizado de máquina, e que discutiremos a seguir, o sistema se adapta ajustando as conexões ponderadas entre os elementos de processamento.

Regras de aprendizagem são esquemas de atualização dos valores dos pesos das sinapses de uma rede neural, de forma a obter da rede um padrão de processamento desejado. O processo de aprendizado pode ser feito basicamente de duas maneiras: aprendizado supervisionado ou não supervisionado. No aprendizado supervisionado, os exemplos das entradas e suas respectivas saídas (conjunto de padrões) são usados no treinamento da rede neural. Geralmente, este método se baseia na minimização do erro entre o valor desejado de saída das unidades da camada de saída e o valor computado pela rede. Baseado nessa diferença, os pesos da rede são ajustados a cada apresentação de um padrão de treinamento, de forma a minimizar o erro na saída para o conjunto total das informações fornecidas. O processo é repetido diversas vezes até que o procedimento tenha incorporado o conhecimento relativo às informações fornecidas.

No aprendizado não supervisionado, apenas os exemplos de entradas são fornecidos no procedimento de treinamento da rede neural em questão. As unidades respondem a características de “interesse” das entradas apresentadas no processo de treinamento. Este tipo de treinamento está intimamente ligado a topologias de redes com conexão competitiva.

3.5.2 Treinamento

Treinamento é o modo pelo qual o sistema computacional neural aprende a respeito da informação que ele necessitará, a fim de resolver certos problemas. O treinamento envolve a exposição do sistema a um conjunto específico de informações para encontrar um estado de auto-organização particular. [Hebb \(1949\)](#) introduziu a conceituação básica neste contexto. Sua ideia consistia no fato de que, se um neurônio biológico recebe uma entrada de outro neurônio e ambos se encontram altamente ativados, então, a ligação entre eles é reforçada. Aplicando-se esta observação ao esquema de Redes Neurais Artificiais poderíamos dizer que se uma unidade i recebe uma entrada de outra unidade k altamente ativada, então, a importância desta conexão deve ser aumentada, isto é, o valor do peso entre as unidades deve ser acrescido. A maneira mais simples de se representar matematicamente esta relação é por intermédio da equação (3.1), onde O_i é a saída da unidade i , O_k é a saída da unidade k e Δw_{ik} é a alteração a ser feita no peso W_{ik} que liga a unidade k à unidade i .

$$\Delta W_{ik} = O_i O_k \quad (3.1)$$

Entretanto, se as duas unidades tiverem saídas negativas, isto também causará o aumento do peso entre elas e esta reação não combina com a proposição realizada por Hebb. A equação (3.1) foi estendida e modificada na forma de (3.2), conhecida como regra de Hebb generalizada.

$$\Delta W_{ik} = g(O_i, D_i) h(O_k, W_{ik}) \quad (3.2)$$

A equação (3.1) pode ser vista como um caso particular da equação (3.2) onde as funções g e h simplesmente dependem do primeiro argumento, sendo o parâmetro η igual a 1 (o parâmetro η é denominado taxa de aprendizagem). Outras variações desta regra geral foram propostas.

Uma regra de aprendizado importante, que pode ser vista como uma variante da equação (3.2), é chamada processo de correção do erro. Nesse procedimento supervisionado, o algoritmo de aprendizado consiste em altera

o valor do peso entre unidades numa forma proporcional à diferença entre a saída desejada e a computada pela rede. Esta ideia, só pode ser aplicada diretamente a uma rede de camada simples, já que no caso de existirem camadas escondidas, o valor desejado não é conhecido. Em sua forma mais simples, esta relação é matematicamente descrita pela equação (3.3).

$$\Delta W_{ik} = (D_i - O_i)O_k \quad (3.3)$$

Esta regra pode ser vista como uma variação particular da regra generalizada de Hebb (equação (3.2)). Neste caso, a função g é expressa como a diferença entre o valor desejado e o valor computado pela rede, enquanto a função h é definida como sendo igual ao valor de entrada (independente do valor do peso). A equação (3.3) pode ser generalizada, a fim de poder ser aplicada a redes de multicamadas. Esta generalização chama-se regra de retropropagação (*back-propagation*).

Outras regras de treinamento foram desenvolvidas, como, por exemplo, a Estocástica e a Competitiva.

3.5.3 Generalização

Um dos principais problemas do treinamento é a capacidade de generalização isto é, a capacidade da rede de responder adequadamente a padrões que não fizeram parte do conjunto de treinamento. Além da habilidade de aprendizado os sistemas computacionais neurais são também capazes de generalizar informações específicas de entrada para produzir uma solução de saída. Ao se fornecer uma entrada que não foi apresentada anteriormente, o sistema deve ser capaz de generalizar esta entrada para fornecer uma resposta correspondente. Este conceito é muito importante, porque permite que o sistema ofereça soluções, mesmo quando as informações de entrada são incompletas ou imprecisas. A topologia, o tamanho da rede e a forma de executar o treinamento da rede são bastante determinantes na sua capacidade de generalização. A rede é dita bem generalizada quando a relação entrada-saída aprendida no treinamento é representativa do problema. Por sua vez uma rede dita supertreinada pode não apresentar uma boa capacidade de generalização. A regularidade do mapeamento está associada a uma boa

generalização. Ao adicionarmos complexidade ao modelo, por exemplo utilizando um modelo mais complexo ao invés de um modelo mais simples estamos aumentando a possibilidade de obter uma generalização pobre. Por ser demasiadamente complexo, o modelo capta excessivos detalhes dos dados de treinamento em detrimento de propriedades mais gerais.

Nas redes neurais os parâmetros principais a serem estimados são os pesos e, assim, quanto mais camadas e número de neurônios nas camadas ocultas houver, mais complexo será o modelo resultante. Quanto menor a quantidade de dados, mais simples deve ser o modelo. Em geral, haverá generalização na condição dada por (3.4).

$$N > \frac{W}{\epsilon} \quad (3.4)$$

Onde:

W é o número de pesos da rede.

ϵ é o erro admitido na fase de teste.

N é o número de exemplos de treinamento.

Por exemplo, se o erro admitido é de 10%, o número de exemplos deve ser maior que $10 \times W$. Para evitar que o treinamento se especialize em relação aos dados para os quais uma rede está sendo treinada, utiliza-se um conjunto de dados de validação, que não contribui para o treinamento. Esse conjunto é usado para medir a capacidade de generalização. Os dados desse conjunto, que tipicamente possui 20% ou 30% do tamanho do conjunto de dados de treinamento, são usados para validar a rede obtida até aquela iteração. Isso é feito comparando-se os erros em uma determinada iteração (época de treinamento) do conjunto de treinamento e do conjunto de validação. Os pesos das ligações são modificados durante o treinamento de forma a minimizar o erro para os padrões apresentados. Se a quantidade de padrões apresentados é suficientemente representativa da função, a tendência durante o treinamento é que o erro, tanto do conjunto de padrões de treinamento quanto do conjunto de

padrões de validação, vá diminuindo à medida que a rede vá generalizando o aprendizado, como ilustrado no gráfico da [Figura 3.10](#).

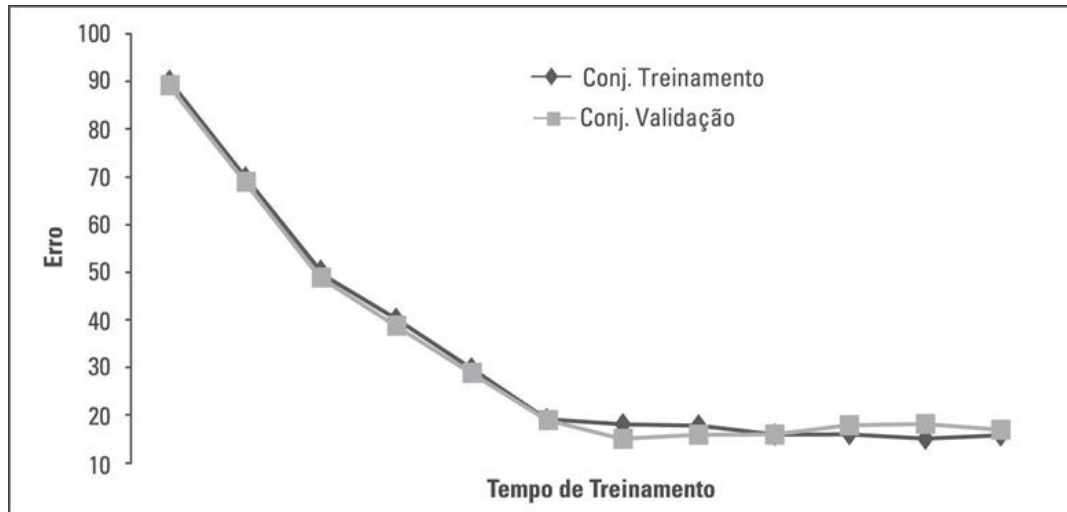


FIGURA 3.10 Treinamento de uma RNA.

Entretanto, após certo tempo, o erro do conjunto de validação atinge um mínimo e começa a crescer, enquanto o erro do conjunto de treinamento continua a cair, indicando uma tendência da rede de incorporar informações que são específicas do conjunto de treinamento, tal como, por exemplo, os erros de medidas daquele conjunto. Uma solução é parar o treinamento quando a tendência de erro do conjunto de validação começa a divergir da tendência do conjunto de treinamento. Dessa forma, o conjunto de treinamento é usado para modificar os pesos enquanto o conjunto de validação é usado para estimar a capacidade de generalização.

Outra abordagem para evitar o excesso de treinamento é limitar a capacidade da rede de absorver as correlações espúrias entre os dados de entrada. Isto acontece basicamente quando a rede possui mais graus de liberdade (que são proporcionais ao número de ligações) do que o número de padrões de treinamento. Assim, o problema da generalização está diretamente ligado ao problema do dimensionamento da rede. Ou seja, basicamente a ideia é conseguir a menor rede que acomode as informações contidas no conjunto de dados de treinamento. O problema é justamente estimar qual é esse tamanho mínimo que permite que a rede ainda aprenda sem incorporar os dados espúrios do conjunto de treinamento. A abordagem é simples, embora

bastante ineficiente, consiste em executar várias redes de diversos tamanhos e verificar qual é a rede mínima que consegue aprender os padrões de entrada. Mesmo que seja possível determinar a menor rede por este processo, ainda se fica muito susceptível aos parâmetros de treinamento, de forma que pode ser muito difícil determinar se a rede é muito pequena para aprender os padrões se ela apenas aprende lentamente ou se devido a valores de iniciação ou parâmetros de treinamento mal-escolhidos o procedimento caiu em algum mínimo local. A abordagem escolhida por vários autores, e que trataremos mais à frente neste capítulo, parte do princípio comum de iniciar o treinamento com uma rede maior do que o necessário e ir podando (removendo) partes da rede que não sejam necessárias. Como inicialmente a rede é grande, possui graus de liberdade suficientes para acomodar rapidamente as características gerais dos dados de entrada de uma forma pouco sensível às condições iniciais e a questões de mínimos locais. Após a acomodação inicial, então a rede pode ser podada de forma a eliminar características específicas do conjunto de treinamento, favorecendo os critérios desejáveis de generalidade.

3.6 Redes Clássicas – Perceptron e Adaline

3.6.1 Redes com funções de ativação de limiar

Uma rede simples de uma camada consiste em um ou mais neurônios de saída j , conectados às entradas i , através de pesos W_{ij} . No caso mais simples, a rede possui um só neurônio k , como indicado na [Figura 3.11](#).

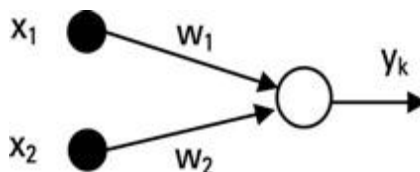


FIGURA 3.11 Neurônio simples com duas entradas e uma saída.

A saída y_k do neurônio é dada pela expressão abaixo.

$$Y_k = \varphi(u_k) \quad \text{onde} \quad u_k = \sum_{i=1}^n W_{ki} X_i$$

A função de ativação proposta originalmente por McCullock e Pitts é uma função de limiar:

$$y_k = \varphi(u_k) = \begin{cases} 1, & \text{se } u_k \geq 0 \\ 0, & \text{se } u_k < 0 \end{cases}$$

Para este caso, o limiar de ativação do neurônio é zero, mas o modelo permite estabelecer um limiar de ativação diferente de zero. Uma modificação introduzida nesse modelo de neurônio foi a introdução de um parâmetro polarizador (*bias* ou *offset*) b_k (Figura 3.12), cuja finalidade é deslocar o valor da informação referente à entrada líquida u_k , de forma a transladar a função de ativação no eixo correspondente ao valor de u_k .

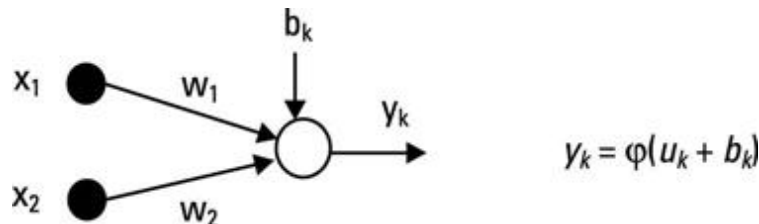


FIGURA 3.12 Adição de um *offset* no modelo do neurônio.

O polarizador pode ser tratado como mais um peso da rede. Para isso, basta considerarmos uma nova entrada do tipo $x_0 = 1$, com um peso associado $w_0 = b_k$, assim a representação fica sendo:

$$u_k = w_0 + w_1 x_1 + w_2 x_2; \quad y_k = \varphi(u_k)$$

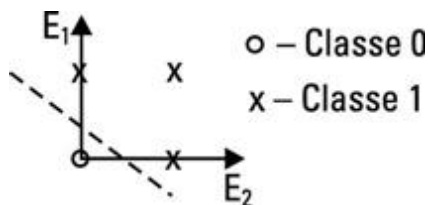
Esse neurônio pode ser empregado para separar classes distintas de padrões de entradas para aplicações de classificações de padrões. Se a entrada líquida

for maior que o limiar, o padrão dessa entrada pertence à classe 1, caso contrário, pertence à classe 0. Portanto, podemos usá-lo para emular o comportamento de uma porta lógica, por exemplo.

Veja os dados típicos de uma porta lógica “OU”, cuja tabela-verdade está indicada a seguir.

Entrada E_1	Entrada E_2	Saída	
0	0	0	} Classe 0
0	1	1	
1	0	1	} Classe 1
1	1	1	

Poderíamos usar para representá-la uma rede neural similar a da [Figura 3.12](#), com um neurônio de duas entradas ($E_1 = x_1$, $E_2 = x_2$) e uma saída (y_k). Assumindo os valores dos pesos e bias $w_1 = 1$, $w_2 = 1$ e $b_k = -0,5$. A rede de fato modela uma função discriminante linear que separa as classes do problema, como ilustrado a seguir.



A separação entre as duas classes é dada pela equação $E_1w_1 + E_2w_2 + b_k = 0$, que é a equação de uma família de retas que podem separar as classes em questão. Resta o questionamento de como foram encontrados os valores dos pesos para o exemplo dado. Ou seja, como treina a rede para obter a característica de classificação desejada. A seguir, veremos dois métodos de treinamento básicos.

3.6.2 Regra de treinamento de Hebb

Os algoritmos de treinamento basicamente calculam, a cada passo, um novo valor para os pesos dos nós, ou seja: $w_{\text{novo}} = w_{\text{antigo}} + \Delta w$. Hebb propôs uma regra de cálculo para cada acréscimo Δw , que é proporcional à entrada associada, mais uma função de transferência (ou de ativação), que é ligeiramente diferente da função de limiar vista anteriormente. Esta função é conhecida como função de limiar bipolar, e está representada na expressão abaixo.

$$y_k = \varphi(u_k) = \begin{cases} 1, & \text{se } u_k \geq 0 \\ -1, & \text{se } u_k < 0 \end{cases}$$

O algoritmo de treinamento proposto está indicado a seguir:

1. Inicie todos os pesos em zero.
2. Para cada padrão de entrada e saída fornecido faça os passos a seguir.
 - 2.1 Para cada conexão faça: (onde d é a saída desejada do nó)
 - 2.1.1 $\Delta W_i = X_i \times d$; (considerar $w_0 = b$, $x_0 = 1$ e $\Delta W_0 = \Delta b$);
 - 2.1.2 $W_i = W_i + \Delta w_i$.

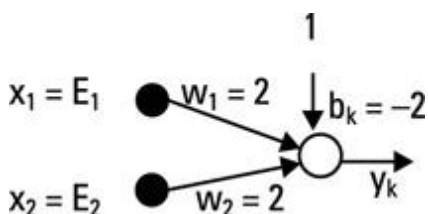
Ao final do treinamento, espera-se que $y = d$, ou seja, a saída obtida seja igual à desejada. Vamos exemplificar aplicando o procedimento para uma função AND, com dados bipolares representados pela tabela seguinte.

Entrada E_1	Entrada E_2	Saída	
1	1	1	} Classe 0
1	-1	-1	
-1	1	-1	} Classe 1
-1	-1	-1	

Aplicando o algoritmo e realizando os cálculos correspondentes, a tabela abaixo ilustra os resultados obtidos.

E1	E2	Saída (d)	$\Delta w_0 =$ (1 × d)	$\Delta w_1 =$ (E1 × d)	$\Delta w_2 =$ (E2 × d)	novo b	novo w_1	novo w_2
						0	0	0
1	1	1	1	1	1	1	1	1
1	-1	1	-1	-1	-1	0	0	2
-1	1	1	-1	-1	1	-1	1	1
-1	-1	1	-1	-1	1	-2	2	2

Ou seja, a rede resultante está ilustrada a seguir.



3.6.3 Regra de treinamento do Perceptron

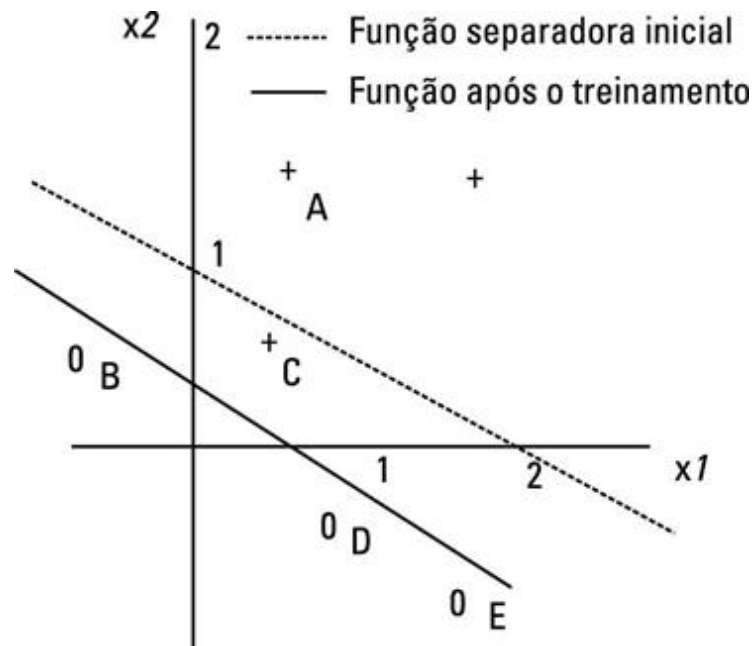
Esta regra foi proposta por [Roseblatt \(1958\)](#) e apresenta algumas mudanças em relação à regra de Hebb. A função de ativação passou a ter um limiar diferente de zero. Foi introduzido um fator de aprendizado η ($0 < \eta \leq 1$), que regula a velocidade de modificação dos pesos. O algoritmo de treinamento foi modificado, de forma que não são realizadas correções nos pesos quando a rede responde corretamente a um padrão desejado. O algoritmo completo é o seguinte:

1. Inicie aleatoriamente os pesos da rede.
2. Repita para cada padrão se $y \neq d$:

$$\Delta W_i = \eta \times X_i \times d;$$

$$W_i = W_i + \Delta W_i.$$

Como exemplificação, veja a figura a seguir.



Os pontos marcados com “+” pertencem a uma classe, e os pontos marcados com círculo, pertencem a outra. O Perceptron deve ser capaz de achar a função separadora das classes em questão. Uma tabela com os valores das entradas, saídas e pesos encontra-se abaixo, onde os pesos foram iniciados com valores arbitrários diferentes de zero, e correspondem à reta separadora inicial (reta pontilhada na figura). A taxa de aprendizado é unitária e o valor do limiar da função de ativação é nulo.

$+$ o	x_1	x_2	1	Saída (d)	$\Delta w_0 =$ (1 $\times$ d)	$\Delta w_1 =$ ($x_1 \times$ d)	$\Delta w_2 =$ ($x_2 \times$ d)	novo b	novo w_1	novo w_2
								-2	1	2
A	0,5	1,5	1	1	-	-	-	-	-	-
B	-0,5	0,5	1	-1	-	-	-	-	-	-
C	0,5	0,5	1	1	1	0,5	0,5	-1	1,5	2,5
D	0,5	-0,3	1	-1	-	-	-	-	-	-
E	1,5	1	1	1	-	-	-	-	-	-

Como houve uma mudança na época de treinamento, o algoritmo ainda calcularia os pesos uma segunda vez, e só então, como não haveria mudanças o algoritmo pararia. A reta separadora (linha cheia) correspondente aos novos parâmetros de pesos e bias encontrados. Na prática, pode acontecer de a convergência para todos os padrões não seja possível, isto é, o erro não tende a zero. Assim, pode-se modificar o algoritmo para parar quando um percentual desejado (de respostas corretas) atingir uma tolerância especificada.

3.6.4 Redes Adaline

Uma importante alteração no procedimento de treinamento foi proposta por [Widrow e Hoff \(1960\)](#). A rede por eles proposta é uma rede adaptativa (ADaptative Linear Element), treinada por um algoritmo baseado no método dos mínimos quadrados, que ficou conhecido como Regra Delta. A arquitetura do neurônio tem a mesma forma do Perceptron.

Do mesmo modo que em um Perceptron, a rede pode ter vários neurônios recebendo as mesmas entradas em uma única camada de saída. Entretanto, se os nós são combinados de forma que a saída de um esteja conectada à entrada de outro, temos uma rede de múltiplas camadas, chamada rede Madaline (Many ADaptative Linear Elements). A saída de cada nó da rede é dada por [\(3.5\)](#), onde M é o número de entradas da rede.

$$Y_k = \sum_{i=0}^M W_{ki} X_i \quad (3.5)$$

A ideia básica da regra Delta é ajustar os pesos a partir do erro na saída da rede. O erro (também chamado custo) para um nó (referente a um determinado padrão p) é expresso por (3.6).

$$e_k^p = d_k^p - Y_k^p \quad (3.6)$$

Onde d_k é a saída desejada (fornecida no padrão p de entrada/saída) e y_k é a saída realmente obtida (fornecida pela rede quando é apresentado o padrão p). Se existir mais de um nó na saída, é preciso considerar todos os erros dos diferentes nós. Entretanto, se simplesmente somarmos os erros, a tendência é que valores de erros positivos anulem os negativos, fornecendo uma informação inadequada do erro total. Assim, a proposta é minimizar o valor do erro dado pela somatória quadrática dos erros (3.7).

$$E = \frac{1}{2} \sum_{k=1}^N e_k^2 = \frac{1}{2} \sum_{k=1}^N (d^p - y^p)^2 \quad (3.7)$$

O fator $\frac{1}{2}$ na equação é usado com a finalidade de simplificar o algoritmo de treinamento. Como o erro E é função da saída da rede y_k e esta é função dos pesos w_{ki} , pode-se dizer que o erro E é uma função que varia com os pesos da rede, ou seja, $E = f(w_{k1}, w_{k2}, w_{k3}, \dots, w_{kN})$. Assim, a ideia é minimizar o erro ajustando os pesos, ou seja, encontrar o mínimo da função relativa ao erro. Este mínimo é dado pela derivada parcial da função em questão em relação a cada peso.

O gradiente da função do erro é o vetor composto por $\partial_{w_{ki}}$, que representa as derivadas parciais da função em relação a cada peso, apontando para a direção de crescimento da função. Assim, a ideia é variar os pesos de forma a caminhar na direção contrária ao gradiente, que é a direção de maior diminuição do valor da função de erro, ou seja:

$$\Delta W_{ki} = \frac{-\partial E}{\partial w_{ki}} \quad (3.8)$$

Onde η é uma constante de proporcionalidade, também chamada de taxa de treinamento. A derivada parcial da equação (3.8) pode ser decomposta pela regra da cadeia na forma de (3.9).

$$-\frac{\partial E}{\partial w_{ki}} = -\frac{\partial \left(\sum_{k=1}^N (d^p - y^p)^2 \right)}{\partial w_{ki}} = (d^p - y^p) \frac{\partial y^p}{\partial w_{ki}} \quad (3.9)$$

Sabe-se que $\frac{\partial y^p}{\partial w_{ki}} = x_i$, o que resulta em (3.10).

$$\Delta w_{ki} = (d^p - y^p) x_i \quad (3.10)$$

A regra Delta para atualização dos pesos é capaz de atualizar corretamente os pesos a cada iteração, seja com entradas binárias ou contínuas. O algoritmo de treinamento pode ser assim resumido:

1. Inicie os pesos com valores aleatórios pequenos.
2. Para cada padrão de entrada e saída fornecido:
 Calcule y^p ;
 Para cada conexão faça:

$$\begin{aligned} \Delta W_i &= \eta \times x_i \times (d^p - y^p); \\ W_i &= W_i + \Delta W_i; \end{aligned}$$

Até que um erro pequeno tenha sido atingido.

Caso desejemos obter saídas binárias, podemos introduzir uma função de limiar na saída do nó. Um exemplo seria a emulação de uma porta ou função

lógica como a OR, definindo uma saída bipolar, com os dados representados pela tabela abaixo.

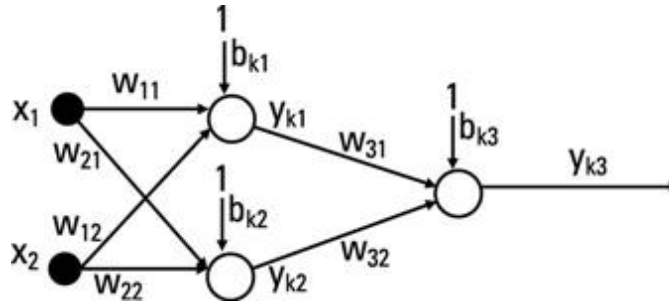
Entrada E ₁	Entrada E ₂	Saída
1	1	1
1	-1	1
-1	1	1
-1	-1	-1

Aplicando o algoritmo proposto e realizando os cálculos correspondentes, a tabela a seguir ilustra os resultados obtidos, onde no segundo passo do treinamento a saída da rede já está com o valor correto de classificação.

#	1	E1	E2	d	y	$\delta = (d-y)$	$\Delta w_0 = 1 \times \delta \times \eta$	$\Delta w_1 = E1 \times \delta \times \eta$	$\Delta w_2 = E2 \times \delta \times \eta$	novo $w_0(b)$	novo w_1	novo w_2
										0,10	0,10	0,10
	1	1	1	1	0,30	0,70	0,18	0,18	0,18	0,28	0,28	0,28
1	1	1	-1	1	0,28	0,72	0,18	0,18	-0,18	0,46	0,46	0,09
	1	-1	1	1	0,09	0,91	0,23	-0,23	0,23	0,68	0,23	0,32
	1	-1	-1	-1	0,13	-1,13	-0,28	0,28	0,28	0,40	0,51	0,60
	1	1	1	1	1,51	-0,51	-0,13	-0,13	-0,13	0,27	0,38	0,47
2	1	1	-1	1	0,18	0,82	0,21	0,21	-0,21	0,48	0,59	0,26
	1	-1	1	1	0,15	0,85	0,21	-0,21	0,21	0,69	0,38	0,47
	1	-1	-1	-1	-0,16	-0,84	-0,21	0,21	0,21	0,48	0,59	0,68

3.6.5 Redes Madaline

Nas redes com múltiplas estruturas Adaline (Many Adaline), as mesmas estão conectadas em uma ou mais camadas, de forma a combiná-las de alguma forma, como ilustrado na figura abaixo.



Esta combinação pode ser uma “regra da maioria”, em que o Adaline de saída não possui pesos e somente propaga a informação das saídas dos Adalines ocultos. Pode ser também uma combinação lógica dos Adalines ocultos, em que o Adaline de saída possui pesos fixos responsáveis por essa função lógica. Por exemplo, com $w_{31} = 0,5$, $w_{32} = 0,5$ e $b_{k3} = 0,5$ temos uma combinação correspondente a uma porta OU. Já com $w_{31} = 0,5$, $w_{32} = 0,5$ e $b_{k3} = -0,5$ temos uma combinação correspondente a uma função AND.

Widrow e Hoff (1960) desenvolveram o algoritmo MR1 para o treinamento de redes Madaline com entradas bipolares. Uma extensão desse algoritmo conhecida como MR2, foi desenvolvida por Widrow, Winter e Baxter (1987) possibilitando mais flexibilidade ao algoritmo de treinamento. Entretanto, o melhor algoritmo para treinamento de redes com múltiplas camadas é o conhecido como *backpropagation*, que será explicado na próxima sessão.

A versão do algoritmo de treinamento MR1 possui a seguinte forma:

1. Inicie os pesos e bias com valores aleatórios pequenos para os nós ocultos e com valores fixos para o nó de saída.
2. Para cada padrão de entrada e saída fornecido:

Calcule u^p (entrada líquida) e y^p para todos os nós

Se $y_{\text{nó de saída}} \neq d$, então:

Caso a função do nó de saída seja OR:

Se $d = 1$, então, para cada conexão do nó k , cujo valor da entrada líquida u_k é o mais próximo de

zero, faça:

$$\Delta w_{ki} = (\times x_i \times (1 - U_k));$$

$$w_{ki} = w_{ki} + \Delta w_{ki}.$$

Caso a função do nó seja AND:

Se $d = 1$, então, para cada conexão de nó k , cujo valor da entrada líquida u_k seja negativo, faça:

$$\Delta w_{ki} = (x_i \times (1 - U_k));$$

$$w_{ki} = w_{ki} + \Delta w_{ki}.$$

Se $d = -1$, então, para cada conexão do nó k , cujo valor da entrada líquida u_k é o mais próximo de zero, faça:

$$\Delta w_{ki} = (x_i \times (-1 - U_k));$$

$$w_{ki} = w_{ki} + \Delta w_{ki};$$

(até que um erro pequeno tenha sido atingido, que os pesos não estejam mais variando, ou executar o algoritmo um número de vezes).

Como exemplo, empregar uma rede Madaline para emular uma porta lógica XOR que não pode ser representada com uma rede de uma só camada, dado que a função não é linearmente separável. Podemos definir uma função de saída bipolar com limiar nulo e aplicar a tabela-verdade de uma função XOR com valores bipolares no procedimento de treinamento, como a ilustrada a seguir.

3.7 Perceptrons de múltiplas camadas

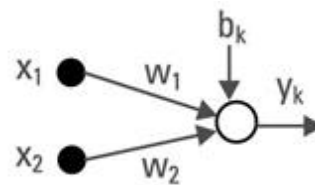
3.7.1 Limitações em redes de camada única

Até agora foram vistas redes neurais com apenas uma camada (além da camada de entrada) ou, no máximo, como é o caso das redes Madaline, rede com duas camadas, na qual a camada de saída possui seus pesos fixos funcionando como uma combinação lógica (OR ou AND) dos nós da camada anterior.

Não foram abordadas ainda as limitações deste tipo de redes, como mostrado por Minsky e Papert, onde se provou que estas topologias não são capazes de separar classes de determinadas funções.

Seja a função “ou exclusivo” (XOR), cuja tabela-verdade (bipolar) e representação neural estão indicadas abaixo.

x_1	x_2	y_k
-1	-1	-1
-1	1	1
1	-1	1
1	1	-1



Se usarmos uma rede com uma só camada de um só neurônio, com duas entradas e uma saída, teríamos como entrada líquida do neurônio o valor expresso pela equação abaixo.

$$u_k = \sum_{i=1}^n w_{ki} x_i = w_{k1} x_1 + w_{k2} x_2 + b_k$$

Com uma função de ativação de limiar zero, os casos de valores da entrada líquida u_k , que correspondem à saída abaixo do limiar zero, estarão situados na fronteira de separação das classes correspondentes à condição:

$$y_k = 0 \quad w_{k1} x_1 + w_{k2} x_2 + b_k = 0.$$

Portanto, a fronteira de separação é dada por uma reta, cuja equação é dada por:

$$w_{k1}x_1 + w_{k2}x_2 = -b_k.$$

Entretanto, os valores de w e b da equação que definem uma reta que separa adequadamente os valores de saída da função XOR em duas classes distintas não podem ser encontrados, haja vista que não há nenhuma reta que separe estes valores, como ilustrado na [Figura 3.13](#) (ao contrário de retas de separação para as funções AND e OR, que podem ser encontradas, como indicado na mesma figura).

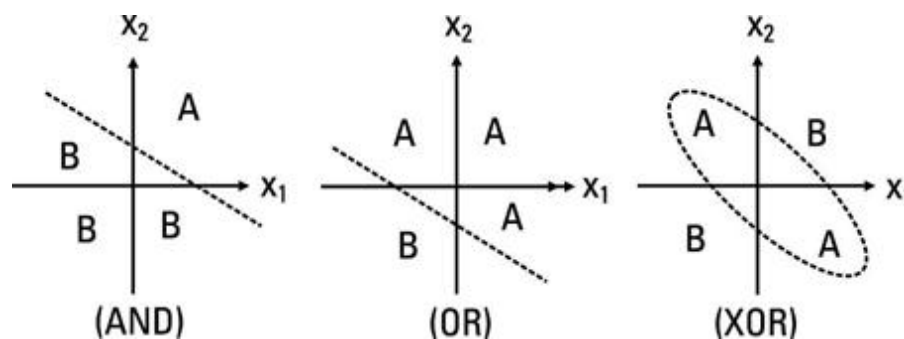


FIGURA 3.13 Retas de separação de classes em funções lógicas típicas.

3.7.2 Perceptrons Multicamadas

Minsky e Papert mostraram que esta limitação de classificação pode ser contornada acrescentando mais uma camada de nós, formando assim uma rede conhecida como Perceptron de Múltiplas Camadas. Para o exemplo específico citado (XOR), basta acrescentar outra camada, constituída de um único nó (situado na saída do já existente), para que o problema passe a ter solução, isto é, para que a rede passe a ser capaz de separar os valores de entrada em duas classes de saída distintas. A arquitetura da rede está representada na [Figura 3.14a](#), onde os valores dentro dos nós são os limiares, e a superfície de separação que a arquitetura gera está ilustrada na [Figura 3.14b](#).

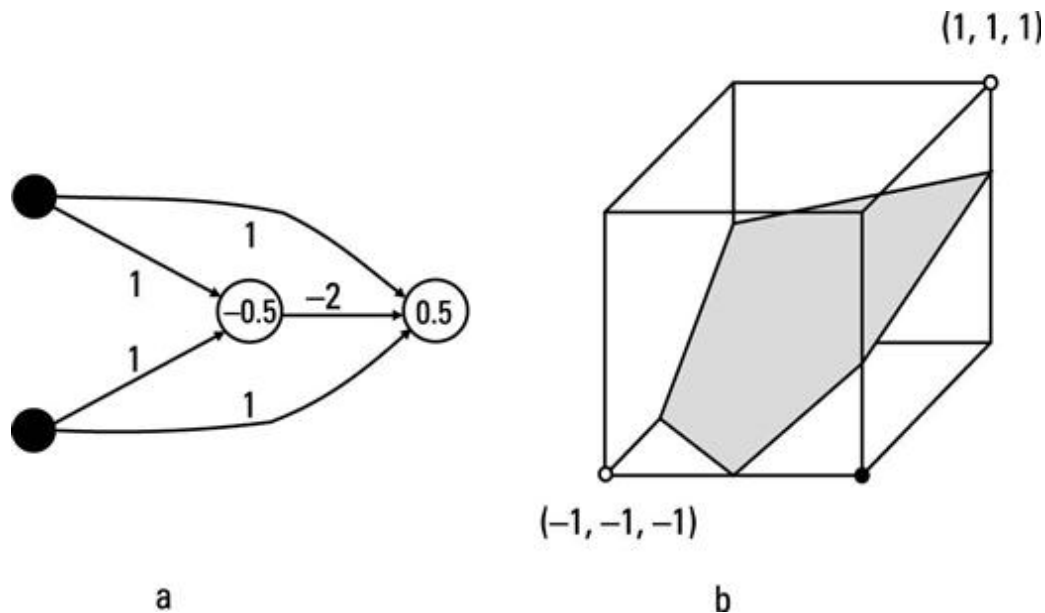


FIGURA 3.14 Arquitetura da rede e superfície separadora do exemplo.

A rede resultante definiu uma superfície separadora (plano separador), que separou os resultados da combinação de entradas possíveis em um plano tridimensional.

Foi mostrado de forma genérica que para qualquer sistema de dimensão N é possível achar uma superfície separadora para as classes do problema usando uma rede de múltiplas camadas. Entretanto, encontrar os valores dos pesos necessários ou mesmo a arquitetura adequada constitui outro problema. A dificuldade configura-se em não saber o valor desejado na(s) saída(s) do(s) nó(s) da camada oculta e, portanto, não se poder usar a regra de atualização do Perceptron para atualizar os pesos que ligam a camada de entrada à camada oculta.

3.7.3 Treinamento de Perceptrons Multicamadas

Apesar das descobertas interessantes de [Minsky e Papert \(1969\)](#) quanto à aplicabilidade das redes, os autores não sugeriram nenhum método que fosse capaz de encontrar os parâmetros delas. Um algoritmo para esta finalidade surgiu a partir das pesquisas independentes de [Paul Werbos \(1974\)](#) e [Rumelhart, Williams e Hinton \(1986\)](#). O que estes autores propuseram foi um

método de propagar o erro da camada de saída (que se conhece) para a(s) camada(s) oculta(s). Dessa forma, tornou-se novamente possível alterar os parâmetros de todas as camadas da rede, a partir do erro na saída, ou seja procurando-se, como nas redes de uma só camada, qual é a alteração dos parâmetros que minimiza o erro na saída de uma determinada rede.

Este algoritmo de treinamento ficou conhecido como *Backpropagation*, e consiste basicamente nos seguintes passos:

1. Apresentar um padrão à rede.
2. Calcular o erro na saída da rede.
3. Retropropagar o erro na rede, calculando de que forma as mudanças nos pesos afetam o erro.
4. Modificar os pesos de forma a minimizar o erro médio.
5. Retornar ao passo 1 enquanto ainda houver padrões que não foram apresentados nesta iteração.
6. Se um erro máximo desejado não tiver sido atingido, retornar ao passo 1 para a próxima iteração (apresentação de todos os padrões novamente).

Com o objetivo de minimizar o erro médio para todos os padrões apresentados, são feitas, portanto, modificações nos pesos a cada padrão. O algoritmo de treinamento trata de forma diferenciada os neurônios da camada de saída e os das camadas ocultas, baseando-se em uma rede genérica (Figura 3.15).

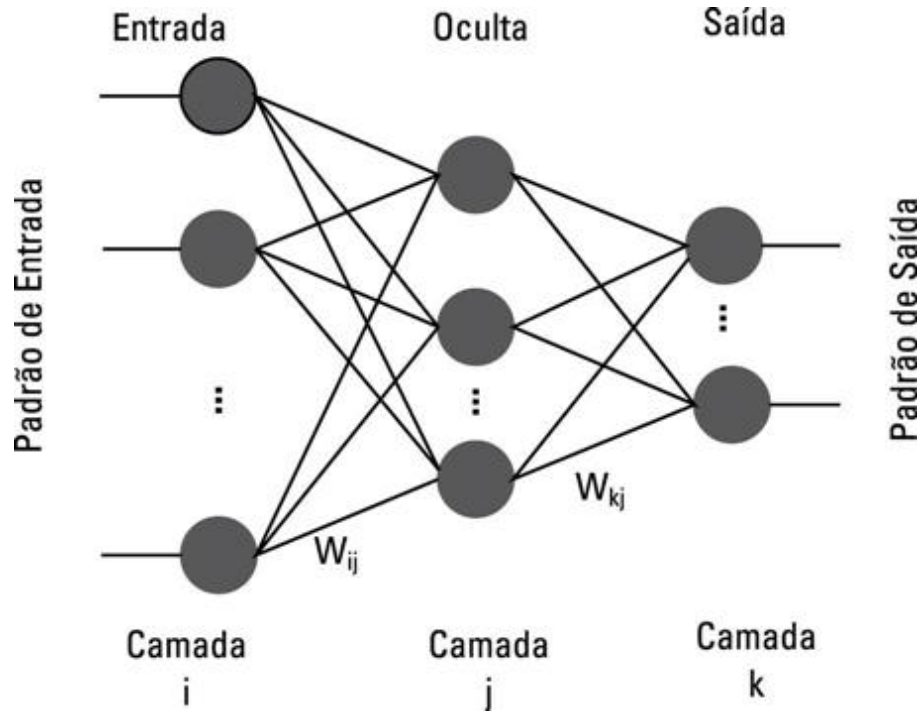


FIGURA 3.15 Topologia típica dos perceptrons em múltiplas camadas.

Em um neurônio k da camada de saída, o erro da n -ésima iteração é dado por (3.11).

$$e_k(n) = d_k(n) - \gamma_k(n) \quad (3.11)$$

A soma dos erros quadráticos na camada de saída pode ser escrita como (3.12).

$$\xi(n) = \frac{1}{2} \sum_k e_k^2(n) \quad (3.12)$$

Onde k varia de 1 até o número de elementos da camada de saída. O erro médio para os N padrões apresentados no treinamento é dado por (3.13).

$$\xi_{md} = \frac{1}{N} \sum_{n=1}^N \xi(n) \quad (3.13)$$

A entrada líquida do neurônio K é expressa por (3.14).

$$u_k(n) = \sum_{j=0}^p w_{kj}(n) \gamma_j(n) \quad (3.14)$$

Onde p é o número de elementos da camada oculta e $j = 0$ simboliza a polarização do nó k . A informação de saída do neurônio k na iteração n é definida por (3.15).

$$\gamma_k(n) = \varphi[u_k(n)] \quad (3.15)$$

As modificações nos pesos W_{kj} são proporcionais ao valor do gradiente do erro em relação aos pesos (3.16).

$$\frac{\partial(n)}{\partial w_{kj}(n)} \quad (3.16)$$

O desenvolvimento da expressão (3.16) é efetuado de acordo com a regra da cadeia, obtendo-se (3.17).

$$\frac{\partial(n)}{\partial w_{kj}(n)} = \frac{\partial(n)}{\partial e_k(n)} \frac{\partial e_k(n)}{\partial \gamma_k(n)} \frac{\partial \gamma_k(n)}{\partial u_k(n)} \frac{\partial u_k(n)}{\partial w_{kj}(n)} \quad (3.17)$$

O primeiro fator de (3.17) é dado por (3.18), e assim por diante para os outros termos, obtendo-se (3.19), (3.20) e (3.21).

$$\frac{\partial(n)}{\partial e_k(n)} = e_k(n) \quad (3.18)$$

$$\frac{\partial e_k(n)}{\partial \gamma_k(n)} = -1 \quad (3.19)$$

$$\frac{\partial \gamma_k(n)}{\partial u_k(n)} = \varphi'_k(u_k(n)) \quad (3.20)$$

$$\frac{\partial u_k(n)}{\partial w_{kj}(n)} = \gamma_j(n) \quad (3.21)$$

Substituindo os respectivos termos em (3.17), tem-se (3.22).

$$\frac{\partial(n)}{\partial w_{kj}(n)} = -e_k(n) \varphi'_k(u_k(n)) \gamma_j(n) \quad (3.22)$$

A computação do gradiente dá a direção de máximo crescimento do erro em relação à variação dos pesos. Assim, se deseja que a variação dos pesos leve na direção contrária (regra delta), resultando em (3.23).

$$\Delta w_{kj}(n) = -\frac{\partial n(n)}{\partial w_{kj}(n)} \quad (3.23)$$

Logo, tem-se (3.24).

$$\Delta w_{kj}(n) = \delta_k(n) \gamma_j(n) \quad (3.24)$$

Onde:

$$\delta_k(n) = e_k(n) \varphi'_k(u_k(n)). \quad (3.25)$$

Ou seja, se k é um nó da camada de saída, é necessário atualizar os pesos que o ligam à camada anterior usando as fórmulas acima. Para um nó j da camada oculta, o problema é que não se conhece $e_j(n)$. Agora, se deseja obter (3.26).

$$\frac{\partial(n)}{\partial w_{ji}(n)} \quad (3.26)$$

Usando novamente a regra da cadeia:

$$\frac{\partial(n)}{\partial w_{ji}(n)} = \frac{\partial(n)}{\partial \gamma_j(n)} \frac{\partial \gamma_j(n)}{\partial u_j(n)} \frac{\partial u_j(n)}{\partial w_{ji}(n)}. \quad (3.27)$$

E sabendo que:

$$\frac{\partial \gamma_j(n)}{\partial u_j(n)} = \varphi'_j(u_j(n)). \quad (3.28)$$

Derivando a expressão do erro, tem-se (3.29):

$$\frac{\partial(n)}{\partial \gamma_j(n)} = \sum_k e_k(n) \frac{\partial e_k(n)}{\partial \gamma_j(n)} \quad (3.29)$$

Que expandida pela regra da cadeia resulta em (3.30):

$$\frac{\partial(n)}{\partial \gamma_j(n)} = \sum_k e_k(n) \frac{\partial e_k(n)}{\partial u_k(n)} \frac{\partial u_k(n)}{\partial \gamma_j(n)} \quad (3.30)$$

Sabe-se que:

$$e_k(n) = d_k(n) - \gamma_k(n) - \varphi'_k(u_k(n)) \quad (3.31)$$

Derivando (3.31), obtém-se (3.32):

$$\frac{\partial e_k(n)}{\partial u_k(n)} = \varphi'_k(u_k(n)) \quad (3.32)$$

Sabe-se também que:

$$\frac{\partial(u_k(n))}{\partial(\gamma_i(n))} = w_{kj}(n). \quad (3.33)$$

Substituindo (3.32) e (3.33) em (3.30), obtém-se (3.34):

$$\frac{\partial(n)}{\partial \gamma_j(n)} = -\sum_k e_k(n) \varphi'_k(u_k(n)) w_{kj}(n) = -\sum_k \delta_k(n) w_{kj}(n) \quad (3.34)$$

Logo:

$$u_j(n) = \sum_{i=0}^q w_{ji}(n) \gamma_j(n). \quad (3.35)$$

Onde q é o número de elementos da camada de entrada, e $i = 0$ simboliza a polarização do nó j .

Sabendo que:

$$\frac{\partial u_j(n)}{\partial w_{ji}(n)} = \gamma_i(n). \quad (3.36)$$

Substituindo os termos, obtém-se (3.37):

$$\frac{\partial(n)}{\partial w_{ji}(n)} = \left(- \sum_k \delta_k(n) w_{kj}(n) \right) \phi'_j(u_j(n)) \gamma_i(n) \quad (3.37)$$

Logo, têm-se:

$$\Delta w_{ji}(n) = - \frac{\partial(n)}{\partial w_{ji}(n)}; \quad (3.38)$$

$$\Delta w_{ji}(n) = j(n) \gamma_j(n). \quad (3.39)$$

Onde:

$$\delta_j(n) = \varphi'_j(u_j(n)) \sum_k \delta_k(n) w_{kj}(n) \quad (3.40)$$

Ou seja, se j é um nó da camada oculta, deve-se atualizar os pesos que o ligam à camada de entrada usando as fórmulas (3.39) e (3.40).

É possível mostrar que estes cálculos são extensíveis para N camadas. O algoritmo abaixo enumera os passos correspondentes:

1. Apresentar um padrão de entrada.
2. Propagar a informação à frente calculando as saídas correspondentes.
3. Calcular os δ_j para os neurônios da camada de saída.
4. Retropropagar o erro da saída calculando os δ_i das camadas ocultas.
5. Atualizar os pesos, apresentar novo padrão e retornar ao passo 1.

Alguns aspectos relevantes sobre a função de ativação a ser empregada dadas as características do algoritmo:

- a) A função de ativação precisa ser diferenciável em todo o seu domínio.
- b) A função de ativação precisa ser não decrescente, para que a sua derivada não troque de sinal de modo a comprometer a convergência do algoritmo.

Por estes motivos e também pela facilidade de obtenção da derivada da função de ativação utilizada (que pode ser expressa em termos da própria função), é comum que sejam empregadas funções, tais como a função sigmoide (Figura 3.16) e a tangente hiperbólica (cuja diferença é a variação entre -1 e 1).

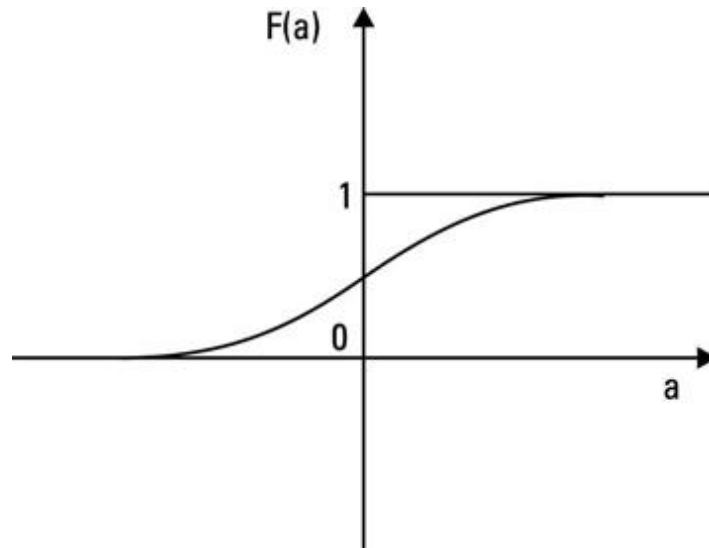


FIGURA 3.16 Ilustração da função sigmoide.

A função sigmoide é definida por:

$$\gamma_j = \varphi(u_j) = \frac{1}{1 + e^{-u_j}}, \text{ onde } u_j > 0.$$

Diferenciando ambos os lados, tem-se:

$$\frac{\delta \gamma_j}{\delta u_j} = \varphi'(u_j) = \frac{e^{-u_j}}{(1 - \gamma_j(n))^2}.$$

Ou ainda:

$$\varphi'(u_j) = \gamma_j(n)[1 - \gamma_j(n)].$$

3.7.4 Alguns exemplos de aplicações

Os exemplos a seguir estão relacionados com as aplicações dos conceitos apresentados sobre redes MLPs e o procedimento de treinamento por retropropagação.

Exemplo 3.1

Neste exemplo será considerado o problema de aproximação de função apresentado no [Capítulo 2](#), no qual se empregou regras *fuzzy* na modelagem de um determinado sistema, sendo que agora será utilizada uma rede neural com o mesmo propósito.

O processo a ser modelado possui os seguintes dados provenientes de medidas realizadas no mesmo: $x_1 = [0,00; 0,25; 0,50; 0,75; 1,00; 1,25; 1,50; 1,75; 2,00; 2,25; 2,50; 2,75; 3,00; 3,25; 3,50; 3,75; 4,00; 4,25; 4,50; 4,75; 5,00; 5,25; 5,50; 5,75; 6,00; 6,25; 6,50; 6,75; 8,00; 8,25; 8,50; 8,75; 9,00; 9,25; 9,50; 9,75; 10,00]$; $y = [2,0000; 2,2197; 2,3811; 2,5136; 2,7310; 2,7827; 2,8327; 3,0351; 2,9551; 3,3973; 3,5117; 3,5909; 3,7345; 3,8419; 4,0952; 4,2879; 4,4000; 4,8764; 5,2843; 5,9241; 6,3302; 6,9608; 7,3044; 7,6791; 8,2819; 9,0139; 9,3387; 10,0420; 10,4000; 10,6437; 10,4786; 10,4928; 10,7082; 10,6233; 10,8862; 10,6830; 10,8393; 10,9186; 10,8814; 10,9779; 11,0000]$.

Neste exemplo será considerada uma rede neural MLP com três neurônios na camada escondida que utiliza funções de ativação do tipo sigmoide, e uma camada de saída com função de ativação linear. O fator de aprendizado escolhido foi de 0,005 e a somatória do erro quadrado do critério de parada ficou em 0,35. Os valores iniciais dos pesos e bias foram aleatórios na faixa $[-1, +1]$. O gráfico da [Figura 3.17](#) contém os dados originais do sistema e os resultantes do treinamento da rede neural em questão.

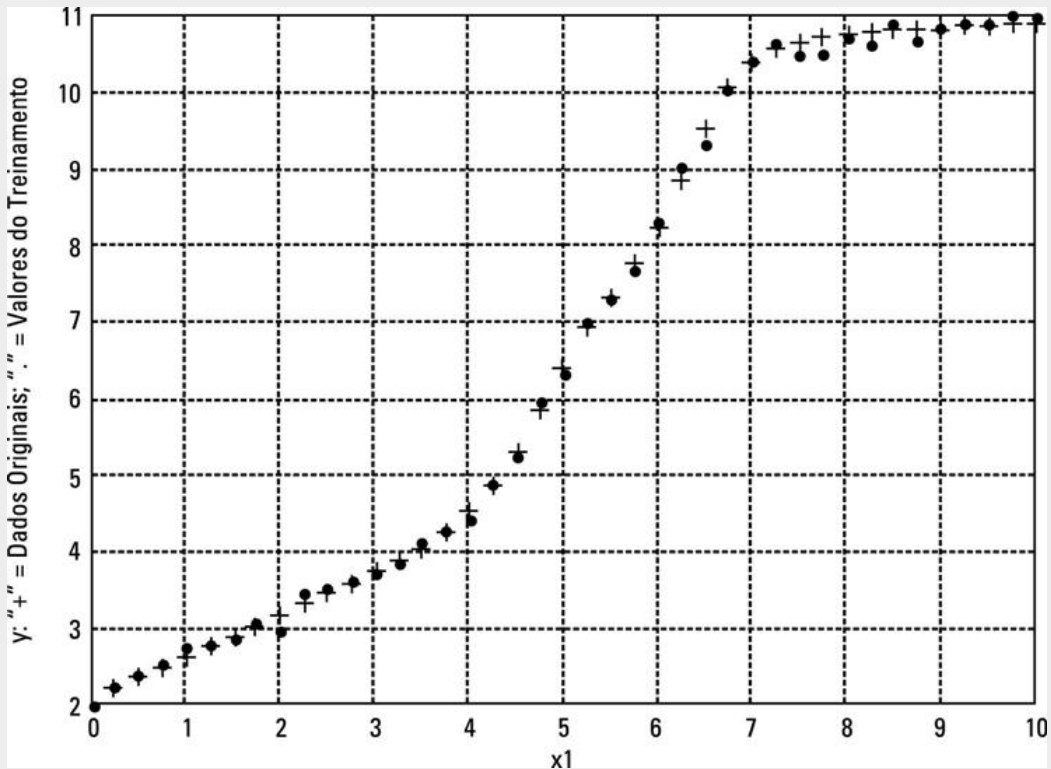


FIGURA 3.17 Resultado do treinamento.

No [Anexo 3.1](#) tem-se o código do programa desenvolvido para este exemplo. O treinamento resultou nos seguintes valores de pesos da camada escondida: -1,8742; -0,241; -1,201. Valores de bias da camada escondida: 11,9436; 0,431; 5,8041. Pesos da camada de saída: -1,2746; -2,2714; -1,5554. Bias da camada de saída: 5,8644. A [Figura 3.18](#) ilustra a capacidade de interpolação da RNA resultante para valores intermediários aos utilizados no treinamento.

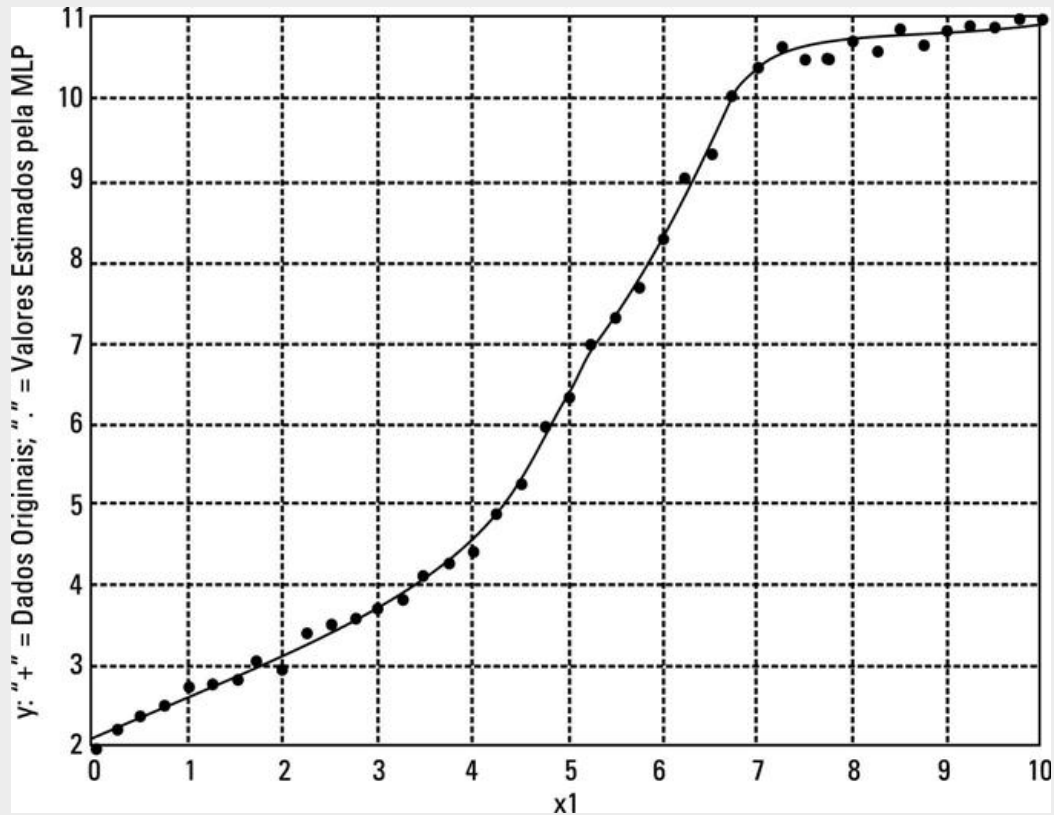


FIGURA 3.18 Teste da capacidade de interpolação da rede.

Como exemplo numérico da capacidade de estimação da rede resultante, seja o valor $x_1 = 5,1267$ que não consta do vetor de entrada de treinamento. Os valores dos dados totalizados por cada um dos três neurônios da camada escondida são:

$$11,9436 - 1,8742 \times 5,1267 = 2,3351;$$

$$0,4310 - 0,241 \times 5,1267 = -0,8044;$$

$$5,8041 - 1,201 \times 5,1267 = -0,3529.$$

Os valores de saída das funções de ativação são:

$$2 / (1 + \exp(-2 \times 2,3351)) - 1 = 0,9814;$$

$$2 / (1 + \exp(-2 \times -0,8044)) - 1 = -0,6665;$$

$$2 / (1 + \exp(-2 \times -0,3529)) - 1 = -0,339.$$

Logo, o valor estimado pela rede na sua camada de saída é $y = 5,8644 - 1,2746 \times 0,9814 - 2,2714 \times 0,6665 - 1,5554 \times -0,339 = 6,6546$ (um valor bem próximo de um dado eventualmente medido).

Esta capacidade de aproximação de funções é que possibilita aplicações de RNAs na modelagem e controle de sistemas dinâmicos complexos (Narendra e Parthasarathy, 1990).

Exemplo 3.2

Neste exemplo será considerado o problema da função XOR descrito anteriormente.

Foi especificada uma rede neural MLP com três neurônios na camada escondida que utiliza funções de ativação do tipo sigmoide, e uma camada de saída com função de ativação linear. O fator de aprendizado escolhido foi de 0,005 e a somatória do erro quadrado para critério de parada ficou em 0,0001. Os valores iniciais dos pesos e bias foram aleatórios na faixa $[-1, +1]$. Os valores dos vetores de treinamento são: $x_1 = [0; 0; 1; 1]$; $x_2 = [0; 1; 0; 1]$; $y = [0; 1; 1; 0]$.

No Anexo 3.2 tem-se o código do programa desenvolvido para este exemplo. Após 2600 épocas de treinamento atingiu-se o critério de erro de parada, resultando-se nos seguintes valores de pesos da camada escondida: 0,2157; 1,5795; -1,4056; 0,3791; -1,6998; 1,3715. Valores de bias da camada escondida: 0,7665; -0,8902; -0,7295. Pesos da camada de saída: 0,3134; 0,9981; 1,0155. Valor do *offset* da camada de saída: 1,1478.

Para exemplificar numericamente a capacidade de classificação da rede resultante, seguem os dados e computações correspondentes. Para $x_1 = 0$ e $x_2 = 1$, os valores dos dados totalizados por cada um dos três neurônios da camada escondida são:

$$\begin{aligned} &0,7665 + 0,2157 \times 0 + 0,3791 \times 1 = 1,1456; \\ &-0,8902 + 1,5795 \times 0 - 1,6998 \times 1 = -2,590; \\ &-0,7295 - 1,4056 \times 0 + 1,3715 \times 1 = 0,6421. \end{aligned}$$

Os valores de saída das funções de ativação associadas são:

$$\begin{aligned}2 / (1 + \exp(-2 \times 1,1456)) - 1 &= 0,8163; \\2 / (1 + \exp(-2 \times -2,590)) - 1 &= -0,9888; \\2 / (1 + \exp(-2 \times 0,6421)) - 1 &= 0,5663.\end{aligned}$$

Logo, o valor estimado pela rede na sua camada de saída é $y = 1,1478 + 0,3134 \times 0,8163 + 0,9981 \times -0,9888 + 1,0155 \times 0,5663 = 0,9918$ que é classificado como nível lógico “1”. De modo similar para $x_1 = 1$ e $x_2 = 0$, tem-se $y = 0,9930$ (nível lógico “1”). Para $x_1 = 1$ e $x_2 = 1$, tem-se $y = 0,0049$ classificado como nível lógico “0”. E finalmente para $x_1 = 0$ e $x_2 = 0$, tem-se $y = 0,0073$ (nível lógico “0”). Desta forma, a rede consegue mapear adequadamente os valores típicos da tabela-verdade da função XOR exemplificada.

Este tipo de exemplo serve de base em aplicações de redes neurais no reconhecimento de caracteres alfanuméricos de um texto ou da numeração de uma placa de um veículo, por exemplo. Valores unitários representam pixels acesos de um caractere de uma determinada imagem, e valores nulos, pixels apagados. Uma tabela com padrões de pixels de um caractere específico de uma imagem, por exemplo, fornece os dados de treinamento para uma RNA com este propósito. Depois de treinada, a rede em questão é capaz de reconhecer os caracteres fornecidos na etapa de treinamento, de modo que esta possa reconhecer ou classificar os dados de interesse em um conjunto de caracteres diversos de uma ou mais imagens em geral.

Os programas nos [Anexos 3.1](#) e [3.2](#) foram desenvolvidos (baseados no algoritmo descrito em [Fausett \(1994\)](#)) com finalidades didáticas, onde se utilizou variáveis singelas nos códigos-fonte correspondentes. A utilização de variáveis com representações matriciais e a elaboração de sub-rotinas específicas nos programas possibilitam a construção de aplicativos mais compactos e eficientes.

Pacotes computacionais como o MatLab possuem *toolboxes* para RNAs que disponibilizam linhas de comando ou ferramentas gráficas apropriadas para o desenvolvimento e testes de modelos neurais em geral.

Exemplo 3.3

Como ilustração, seja resolver o problema de aproximação de função similar ao [Exemplo 3.1](#).

Será utilizado o *toolbox nntool*. Inicialmente, carrega-se na área de trabalho do MatLab os vetores de dados de treinamento, por exemplo: $x1 = [0.00 \ 0.25 \ 0.50 \ 0.75 \ 1.00 \ 1.25 \ 1.50 \ 1.75 \ 2.00 \ 2.25 \ 2.50 \ 2.75 \ 3.00 \ 3.25 \ 3.50 \ 3.75 \ 4.00 \ 4.25 \ 4.50 \ 4.75 \ 5.00 \ 5.25 \ 5.50 \ 5.75 \ 6.00 \ 6.25 \ 6.50 \ 6.75 \ 8.00 \ 8.25 \ 8.50 \ 8.75 \ 9.00 \ 9.25 \ 9.50 \ 9.75 \ 10.00]$; $y = [2.0000 \ 2.2197 \ 2.3811 \ 2.5136 \ 2.7310 \ 2.7827 \ 2.8327 \ 3.0351 \ 2.9551 \ 3.3973 \ 3.5117 \ 3.5909 \ 3.7345 \ 3.8419 \ 4.0952 \ 4.2879 \ 4.4000 \ 4.8764 \ 5.2843 \ 5.9241 \ 6.3302 \ 6.9608 \ 7.3044 \ 7.6791 \ 8.2819 \ 9.0139 \ 9.3387 \ 10.0420 \ 10.4000 \ 10.6437 \ 10.4786 \ 10.4928 \ 10.7082 \ 10.6233 \ 10.8862 \ 10.6830 \ 10.8393 \ 10.9186 \ 10.8814 \ 10.9779 \ 11.0000]$.

Aciona-se a ferramenta em questão através do comando: *nntool* <Enter>. Na tela aberta do aplicativo aciona-se o botão “Import...”, e na janela correspondente seleciona-se o vetor $x1$, marcando a opção “Input Data”, pressionando o botão “Import” e, depois, “Close”. O procedimento anterior é repetido para selecionar o vetor y , marcando a opção “Target Data”, pressionando o botão “Import” e, depois, “Close”. A seguir é acionada a opção “New...”, para criar um modelo de rede neural, onde na janela correspondente é selecionado o tipo da rede (Feed-Forward backprop, neste caso), o “Input Data” e o “Target Data” ($x1$ e y , respectivamente), o número de camadas (“Number of layers = 2”, para uma camada escondida e outra de saída), o número de neurônios da camada escondida (“Number of neurons = 3”, neste caso), o tipo de função de ativação (“Transfer Function = “TANSIG”, neste exemplo), e finalmente se aciona os botões “Create” e “Close”. Na tela principal do aplicativo aparece o nome da rede especificada, cujo nome default é *network1*. Clicando no nome em questão, surge uma janela com a estrutura da rede e abas de opções do aplicativo.

Acionando a aba “Train”, inclui-se nos campos “Inputs” e “Targets”, os vetores $x1$ e y , respectivamente, e aperta-se o botão “Train Network”, para se treinar a rede (onde uma janela é aberta com as informações do treinamento: número de épocas utilizadas, erro obtido etc.). Na aba “View/Edit Weights” pode-se visualizar os valores dos pesos e bias obtidos na etapa de treinamento. Na tela principal do aplicativo aparece, no campo “Output Data”, os dados (com o nome default *network1_outputs*) obtidos pela rede treinada para o padrão de entrada de treinamento. Esses dados e os parâmetros da rede podem ser transferidos para a área de trabalho do

MatLab, ou gravados para posterior utilização. Para tanto, basta acionar o botão “Export...” e selecionar os arquivos a serem exportados (*Export* neste exemplo) ou salvos (*Save*) no computador. Os comandos de linha a seguir mostram como gerar um gráfico com os dados originais do sistema e os obtidos pelo treinamento da rede em questão:

```
Y = sim(network1,x1);  
plot(x1,y,'k.',x1,Y,'k+'); grid; xlabel('x1');  
ylabel('y: "." = Dados Originais; "+" = Valores do Treinamento'); <Enter>.
```

A [Figura 3.19](#) ilustra os dados correspondentes à resolução do problema de aproximação de função com a ferramenta citada.

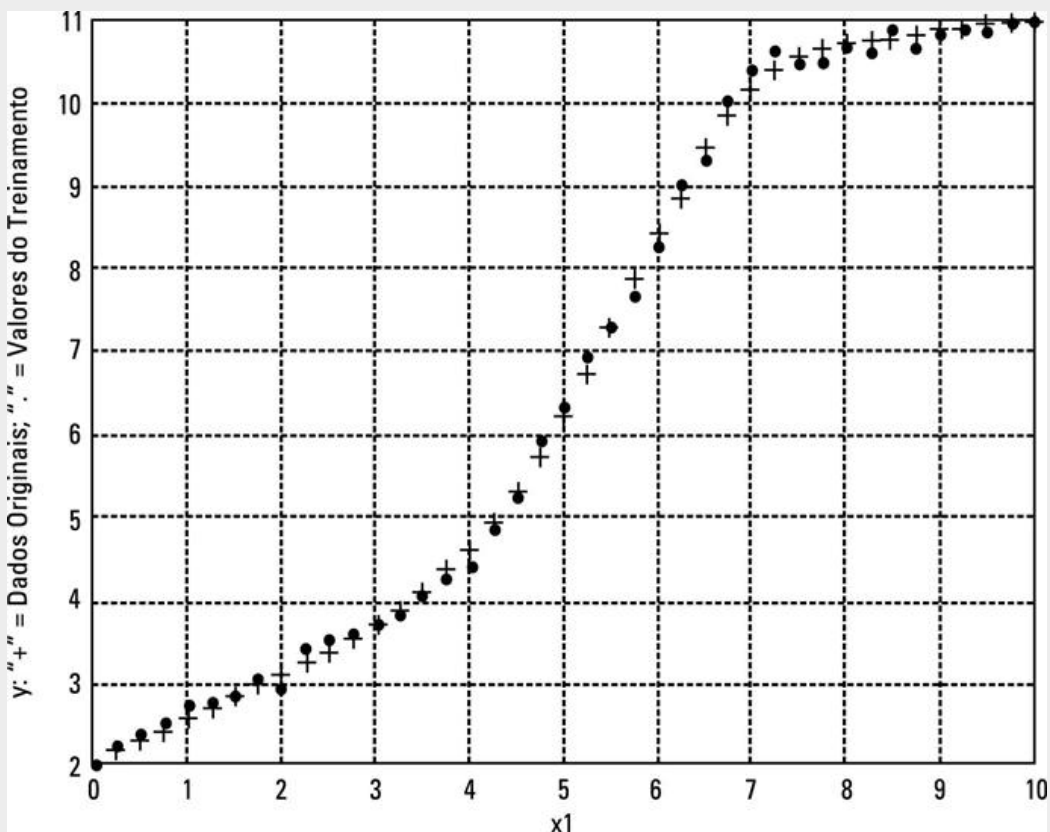


FIGURA 3.19 Resultado do treinamento com a ferramenta citada.

3.7.4.1 Características práticas do *backpropagation*: quanto ao instante de atualização dos pesos

Se a atualização dos pesos for realizada a cada apresentação de um novo padrão, ao final de uma iteração é razoável imaginar que a atualização dos pesos corresponda mais às alterações feitas pelos últimos padrões apresentados do que aos primeiros. Se a ordem de apresentação é a mesma a cada iteração, então a alteração será tendenciosa.

Portanto, caso se deseje alterar os pesos a cada apresentação de um novo padrão, é conveniente variar a ordem de apresentação dos padrões a cada iteração, por exemplo, sorteando essa ordem. A este esquema de apresentação dos padrões chamamos atualização iterativa.

Outra opção é não alterar o peso a cada apresentação de padrões após o cálculo das variações desejadas pelo padrão, acumular a alteração deste padrão com as alterações determinadas pelos demais padrões e realizar a alteração nos pesos somente ao final da iteração. A este método chamamos atualização por batelada. Quanto à parada do treinamento, este geralmente é encerrado quando ocorre um dos seguintes casos:

- I. Erro atingiu um valor predeterminado;
 - II. Erro está caindo a uma taxa suficientemente pequena de acordo com um parâmetro predeterminado; ou.
 - III. A atualização está prejudicando a generalização (medida com um conjunto de validação). Isto é, o erro está crescendo para o conjunto de validação, apesar de possivelmente estar caindo para o conjunto de treinamento.
- Quanto à escolha dos pesos iniciais, [Rumelhart \(1986\)](#) mostrou que uma rede neural pode jamais conseguir aprender se todos os pesos forem iniciados com um mesmo valor, uma vez que todos os pesos seriam alterados de uma mesma maneira. Os pesos são mudados proporcionalmente à derivada da função de ativação, portanto, quanto mais próximo do centro da função estiver, maior será a modificação. Nas extremidades da função de ativação (regiões de saturação), as derivadas são próximas de zero e, conseqüentemente, há pouca variação nos pesos, mesmo que o erro seja significativo. Isto sugere que o processo de aprendizado deva ser iniciado utilizando-se valores de pesos de pequena amplitude, que sejam aleatoriamente gerados a partir de uma distribuição uniforme. Sugere-se, por exemplo, tomar os valores iniciais dos pesos e das

polarizações a partir de uma distribuição uniforme no intervalo $[-0,5; +0,5]$.

3.7.4.2 Modificações no procedimento de retropropagação

O algoritmo de retropropagação possui algumas características que devem ser levadas em consideração para acelerar a convergência, evitar oscilações ou mesmo para impedir que o treinamento da rede falhe (não ocorrendo uma convergência efetiva).

3.7.4.3 O termo momento

Na prática, uma forma de aumentar a taxa de aprendizado sem causar oscilações é modificar a equação (3.39) para incluir um termo chamado termo de momento. Esta alteração consiste na utilização da parcela de atualização dos pesos (Δw) da iteração anterior no processo de atualização dos pesos da iteração corrente, conforme indicado a seguir (e de forma similar para o ajuste de pesos de outras camadas):

$$\Delta w_{ji}(n) = \alpha \Delta w_{ji}(n-1) + \delta_j(n) y_i(n)$$

O índice n indica a iteração (etapa de treinamento ou época), e o coeficiente α a taxa de momento que determina o efeito de alterações anteriores na direção do movimento no espaço dos pesos. O efeito dessa alteração é a aceleração do treinamento por um fator de $1/(1-\alpha)$.

3.7.4.4 Mínimos locais

Outro problema que pode afetar o treinamento de uma rede neural é a questão dos mínimos locais. O algoritmo do gradiente decrescente pode encontrar mínimos locais na busca da solução do problema de ajustes dos pesos de uma rede. Isto normalmente está ligado à amplitude do valor inicial dos pesos e ao fator de normalização. Se os valores estiverem concentrados nos extremos da função de ativação (perto de 0 ou 1 para a função logística, ou de -1 ou 1 para a função tangente hiperbólica), o processo de treinamento pode ficar parado ou estacionar em um mínimo local.

Uma maneira de minimizar este problema é escolher como faixa de trabalho uma porção mais central da função de ativação, onde a função apresenta um comportamento mais linear. Outra forma de se contornar o problema dos mínimos locais é estabelecer uma tolerância satisfatória para a convergência do algoritmo. O processo de aprendizado é interrompido quando a tolerância é atingida. Assim, o fato do mínimo ser global ou local torna-se irrelevante, uma vez que uma solução satisfatória foi encontrada. Se a solução não é satisfatória, é possível mudar os pesos iniciais e executar novamente o procedimento de treinamento com outros fatores de aprendizado e/ou taxa de momento.

3.7.4.5 Função de erro alterada

Alguns autores sugerem a alteração da função de erro a ser minimizada como forma de agilização do processo de aprendizado da rede. A utilização da função de erro descrita na equação abaixo incrementa o processo em 50%, em vez da tradicional função de erro médio quadrático.

$$E(W) = -\sum_{i,p} \ln(1 - (D_i - O_i)^2)$$

3.7.4.6 Alterações no algoritmo de minimização

Alguns autores propõem a substituição do algoritmo de minimização como forma de incrementar o processo de treinamento. [Waltrous \(1987\)](#) e [White \(1990\)](#), entre outros, utilizaram o algoritmo de Newton, enquanto [Nowlan & Hinton \(1992\)](#) utilizaram o algoritmo de gradiente conjugado. O problema na utilização do primeiro método é que o algoritmo requer o cálculo da inversa da matriz Hessiana, o que torna praticamente impossível seu emprego em redes de grandes dimensões. O segundo método apresenta algumas vantagens em relação à técnica do gradiente decrescente, principalmente em funções quadráticas.

Existem autores que sugerem regras empíricas para atualização dos valores do fator de aprendizado e da taxa de momento. [Jacobs \(1988\)](#) sugere a utilização de uma taxa de aprendizado por peso, enquanto outros — por exemplo, [Ritter \(1992\)](#) — sugerem a utilização de uma taxa variável ao longo

do treinamento, mas única para todos os pesos. Nenhuma destas técnicas se mostrou superior às demais, dado que o desempenho de todas as abordagens está relacionado ao problema estudado.

3.7.4.7 Algoritmos de poda em redes *feedforward*

Um dos principais problemas quando se usa treinamento supervisionado é a capacidade de generalização, isto é, a capacidade da rede de responder adequadamente a padrões que não fizeram parte do conjunto de treinamento. Os pesos das ligações são modificados durante o treinamento de forma a minimizar o erro para os padrões apresentados. Se a quantidade de padrões apresentados é suficientemente representativa da função, a tendência durante o treinamento com o algoritmo de retropropagação é que o erro, tanto do conjunto de padrões de treinamento quanto do conjunto de padrões de validação, vá diminuindo à medida que a rede vá generalizando o aprendizado (conforme ilustrado na [Figura 3.10](#)). Após certo tempo, o erro do conjunto de validação atinge um mínimo, e começa a crescer, enquanto o erro do conjunto de treinamento continua a cair, indicando uma tendência da rede de incorporar informações que são específicas do conjunto de treinamento, tal como, por exemplo, os erros de medidas daquele conjunto. Uma solução é parar o treinamento quando a tendência de erro do conjunto de validação começa a divergir da tendência do conjunto de treinamento. Dessa forma, o conjunto de treinamento é usado para modificar os pesos, enquanto o conjunto de validação é usado para estimar a capacidade de generalização. Entretanto, isso pode não ser possível se o conjunto de dados disponíveis for pequeno, na medida em que a divisão em dois conjuntos pode agravar a falta de representatividade do conjunto de padrões.

Outra abordagem para evitar o excesso de treinamento é limitar a capacidade da rede de absorver as correlações espúrias entre os dados de entrada. Isto acontece basicamente quando a rede possui mais graus de liberdade (que são proporcionais ao número de ligações entre os neurônios) do que o número de padrões de treinamento. Assim, o problema da generalização está diretamente ligado ao problema do dimensionamento da rede. Ou seja basicamente a ideia é conseguir a menor rede que consiga acomodar as informações contidas no conjunto de dados de treinamento. O problema é justamente estimar qual é esse tamanho mínimo que permite que a rede ainda aprenda sem incorporar os dados espúrios do conjunto de treinamento.

Estudos no sentido de estimar esse tamanho, relacionam a complexidade do sistema com o número de exemplos de treinamento necessários para aprender uma determinada função. Se o número de exemplos de treinamento for pequeno em relação ao tamanho da rede, o erro de generalização é alto. A abordagem mais simples, embora trabalhosa, é simplesmente executar várias redes de diversos tamanhos e verificar qual é a rede mínima que consegue aprender os padrões de entrada. Mesmo que seja possível determinar a menor rede por este processo, ainda fica-se muito susceptível aos parâmetros de treinamento, de forma que pode ser muito difícil determinar se a rede é muito pequena para aprender os padrões, se ela simplesmente aprende lentamente ou se devido a valores de iniciação ou parâmetros de treinamento mal escolhidos ela caiu em algum mínimo local. A abordagem escolhida por vários autores parte do princípio comum de iniciar o treinamento com uma rede maior do que o necessário e ir podando (removendo) partes da rede que não sejam necessárias. Como inicialmente a rede é grande, possui graus de liberdade suficientes para acomodar rapidamente as características gerais dos dados de entrada de uma forma pouco sensível às condições iniciais e aos mínimos locais. Após a acomodação inicial, então, a rede pode ser podada de modo a eliminar características específicas do conjunto de treinamento, favorecendo os critérios desejáveis de generalidade.

3.7.4.8 Eliminação de conexões

Uma ideia básica para simplificar a estrutura de uma rede neural é cortar as conexões entre os seus nós. Para tanto, uma primeira abordagem pode ser a de simplesmente zerar o peso de uma conexão e verificar qual a influência disso no erro do modelo resultante. Se o erro aumentar muito, então se restaura o peso desconsiderado, se não houver mudança significativa, remove-se definitivamente o peso e espera-se um determinado tempo para que as demais conexões absorvam as características das entradas que estavam representadas naquela ligação. Se a rede possui uma quantidade X de pesos, o tempo para cada propagação é da ordem de $O(X)$. Como a operação deve ser executada para cada peso e para cada um dos Y padrões de entrada, então cada passo de poda é da ordem de $O(Y \times X^2)$, sem contar o tempo de acomodação do processo de treinamento. Caso se queira adotar um procedimento mais cauteloso, que analise a influência no erro de cada peso para cada padrão, se remove apenas o peso de menor influência, repetindo-se o procedimento até

que a menor alteração de erro atinja um limiar de parada, então o tempo de propagação é da ordem de $O(Y \times X^3)$. Naturalmente, esses são procedimentos bastante lentos, principalmente para redes de maior porte. Assim, os algoritmos de poda citados a seguir adotam outras abordagens mais interessantes.

3.7.4.9 Métodos de cálculo de sensibilidade

Existem alguns métodos que procuram estimar a sensibilidade da função de erro perante a remoção de uma ligação, para então remover as ligações com menor influência. Em geral esses métodos atuam após o treinamento da rede. Ou seja, primeiramente a rede é treinada para um tamanho maior que o necessário, depois é estimada a sensibilidade de cada peso e então os de menor sensibilidade são retirados. O problema com esses métodos é que eles não levam em consideração a correlação entre os pesos da rede. A sensibilidade de cada peso w_{ij} é calculada como se ele fosse o único candidato a ser retirado, ou como se o nó i fosse o único candidato a ser podado. Quando um dos pesos é retirado, as demais sensibilidades calculadas não continuam necessariamente válidas para a nova configuração da rede. Isto pode ser facilmente compreendido imaginando um exemplo com dois nós que se cancelam mutuamente na contribuição para a informação de saída. Vistos como um par eles não têm contribuição para a saída em questão, mas são vistos individualmente, onde cada um tem uma forte contribuição e, portanto provavelmente não serão cortados. Com nós que estejam parcialmente correlacionados acontece o mesmo, e eles são bastante comuns na maioria das redes.

3.7.4.10 Métodos com termos de penalização

A ideia central do algoritmo de *backpropagation* é minimizar uma função de custo, no caso, o erro na saída da rede. Como a hipótese dos algoritmos de poda é de se ter a menor rede que seja capaz de responder aos padrões de treinamento, e que apresentará a melhor generalização, surgiu a ideia de dividir a função de custo em uma soma de dois termos, um representando o custo devido ao erro e outro representando o custo devido à complexidade da rede. Esse termo adicional é chamado termo de penalização, na medida em que representa uma penalização que a função de custo sofre devido à

complexidade da rede. Os métodos que introduzem esse termo procuram minimizar a nova função de custo (que inclui um termo de custo de complexidade). Assim, a tendência é que haja uma redução na complexidade da rede, forçada pela tendência de que pesos sejam levados para valores nulos tendendo à minimização do custo de complexidade.

3.7.4.11 Decaimento de pesos

Alguns dos métodos que usam fatores de penalização incluem termos que provocam um decaimento dos pesos durante o processo de treinamento. [Chauvin \(1989\)](#) adicionou um termo à equação de custo com esse objetivo cuja expressão é dada a seguir.

$$C = \mu_{er} \sum_j^P \sum_i^O (d_{ij} - O_{ip})^2 + \mu_w \sum_{ij}^W W_{ij}^2$$

A derivada do segundo termo da expressão em relação a W_{ij} é $2\mu_w w_{ij}$, o que introduz efetivamente um termo de decaimento dos pesos na equação do algoritmo. Pesos que não sejam essenciais à solução tendem a decair para zero e podem ser retirados. Este método é conhecido como Weight Decay. Ishikawa (1990) propôs outra função de custo na forma descrita a seguir.

$$C = \sum_{k \in T} (t_k - o_k)^2 + \left(\sum_{i,j} |w_{ij}| \right)$$

O segundo termo desta expressão introduz uma mudança na regra de atualização dos pesos que passa agora a ter um termo da forma $-\text{sgn}(w_{ij})$. Se $W_{ij} > 0$, então o peso é decrementado de λ , caso contrário, ($w_{ij} < 0$), o peso é incrementado de λ .

3.7.4.12 Método de Weight-Elimination

Este método foi proposto por [Weigend \(1991\)](#), e consiste na adição de um termo de penalização na forma do segundo termo abaixo.

$$\sum_{k \in P} (D_k - O_k)^2 + \sum_{i \in C} \frac{w_i^2 / w_0^2}{1 + w_i^2 / w_0^2}$$

Onde C é o conjunto de todas as conexões e w_0 é a escala para as conexões dos pesos da camada em questão. O primeiro termo desta equação mede o desempenho da rede, enquanto o segundo termo avalia o tamanho (complexidade) da rede. O parâmetro λ representa a importância relativa do termo de complexidade com relação ao termo de desempenho. Weigend mostra o custo de complexidade como função de w_i/w_0 (Figura 3.20). As regiões onde os valores absolutos das conexões $|w_i|$ são muito grandes ou muito pequenos são facilmente interpretadas. Para $\|w_i\| \gg w_0$, o custo de uma conexão aproxima-se de 1 (multiplicado por λ), e isso justifica a interpretação do termo de complexidade como um contador de conexões com magnitude significativa. Para $|w_i| \ll w_0$, o custo se aproxima de zero. Valores grandes ou pequenos são definidos em relação à escala w_0 , que é um parâmetro livre do procedimento para eliminação das conexões. Segundo Weigend, a escolha de w_0 de ordem unitária é um bom valor, e o parâmetro λ é ajustado dinamicamente durante o processo de treinamento.

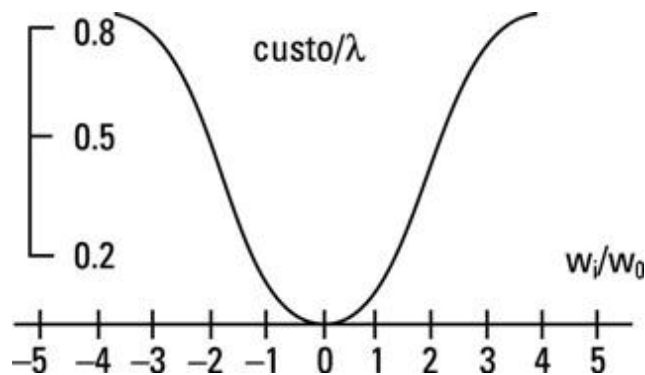


FIGURA 3.20 Custo complexidade versus pesos.

O método de Chauvin (1990) (Weight Decay) produz um decaimento proporcional ao quadrado do valor do peso, e tende a dar preferência a uma grande quantidade de pesos pequenos, em detrimento de poucos pesos com valores altos. O método de Weight-Elimination (Weigend, 1991) tende a

preservar poucos pesos grandes em relação à média dos pesos, eliminando os pesos menores. O parâmetro de escala w_0 permite expressar a preferência por muitos pesos pequenos (w_0 grande), ou poucos pesos com valores de pesos grandes (w_0 pequeno).

3.8 Treinamento não supervisionado

3.8.1 Redes auto-organizadas

Os procedimentos de treinamento vistos até agora consistiram na apresentação de padrões de dados que continham os dados da entrada e os dados desejáveis na saída. A partir desses padrões, as redes calculam a diferença entre o que é desejável e o que realmente aparece na sua saída, e, a partir desta diferença ou erro, ajustam-se os pesos de forma a minimizar o erro. Esse tipo de treinamento é conhecido como supervisionado, na medida em que é necessário mostrar qual é a saída desejada. Ocorre que em muitas aplicações não é possível ter exemplos de treinamento nesse estilo (entradas e saídas), estando disponíveis somente dados de entrada. Por exemplo, em problemas de agrupamentos (ou “clusterização”), em que não se conhece as classes dos dados e deseja-se que a rede separe os dados de entrada em grupos (clusters) ou em outros problemas de extração de características, constituem contexto desta natureza. Para esses casos, não havendo um “tutor” que ensine à rede quais são as saídas esperadas para um determinado padrão de entrada de dados, ela deve ser capaz, geralmente através de um processo de competição entre os nós, de mapear as entradas disponíveis em uma saída que, em geral possui uma dimensionalidade menor. Esse processo é conhecido como aprendizado auto-organizado.

Há muitos tipos de redes com aprendizado auto-organizado aplicáveis a uma vasta área de problemas. A rede de aprendizagem competitiva proposta por [Rumelhart e Zipser \(1985\)](#), as redes ART, propostas por Carpenter e Grossberg (1997), as redes *cognitron*, de [Fukushima \(1975, 1988\)](#) ou os Mapas Auto-organizáveis, de Kohonen, são exemplos de redes auto-organizadas.

3.8.2 Aprendizado competitivo

O aprendizado competitivo é um algoritmo que divide uma determinada série de dados de entradas em grupos (*clusters*) que são inerentes aos dados do problema. A informação é extraída sem que haja um par entrada/saída-alvo que sirva como um tutor. As redes relacionadas possuem uma camada de nós de saída que estão ligados a uma só camada (de entrada, portanto), de tal forma que existam tantos nós na entrada quantas sejam as características dos padrões de entrada do problema, e tantos nós de saída quantos sejam os *clusters* em que queiramos classificar as entradas. O procedimento de treinamento promove uma competição entre os nós da rede, de forma que somente um dos nós da saída seja ativado, para cada padrão de entrada apresentado. Este nó é justamente o nó representativo do *cluster* ao qual pertence aquele padrão de entrada. Por exemplo, podemos ter uma arquitetura como a mostrada na [Figura 3.21](#).

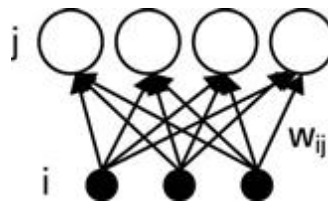


FIGURA 3.21 Exemplo de parte de uma rede competitiva.

Para cada padrão (vetor) de entrada apresentado à rede, somente um dos nós da saída, chamado nó vencedor, será ativado. Em uma rede já treinada, todos os vetores de entrada que pertencerem a um mesmo *cluster*, ou seja, que tiverem determinadas características comuns que possam ser identificados como pertencentes a um mesmo *cluster*, acionarão o mesmo nó de saída (representativo do *cluster*, por conseguinte).

A ideia básica é que, durante o processo de treinamento, a cada padrão apresentado, o algoritmo descubra qual dos nós da saída melhor representa aquele padrão de entrada (através de algum tipo de métrica de distância entre os vetores de entrada e os nós de saída), realizando os ajustes dos pesos que ligam a entrada a este nó de saída. Isto faz com que o nó escolhido fique ainda mais representativo desse padrão, ou seja, diminui a “distância” que separa esse padrão do nó de saída. Ao longo do treinamento, o nó de saída vai se tornando o representante típico dos padrões de entrada que o tornaram vencedor, ou seja, aquele nó passa a representar o *cluster* daquele grupo de

padrões de entrada. Para determinar a distância que existe entre o vetor de entrada (padrão de entrada) e cada um dos nós de saída são usados basicamente dois métodos.

3.8.3 Produto escalar

Este método pressupõe que tanto o vetor de entrada (x) quanto o vetor de pesos (w_i), que liga o nó i às entradas, estejam normalizados para o valor unitário (isto é, o comprimento desses vetores deve ser unitário). É calculado um valor de ativação a_i do nó i que corresponde ao produto escalar dos dois vetores. Uma vez que tenham sido calculados os valores de ativação de cada nó, é escolhido o nó k com o maior valor de ativação como o nó vencedor. Assim, somente os pesos que ligam a entrada a este nó serão atualizados. A Figura 3.22 ilustra o conceito em questão.

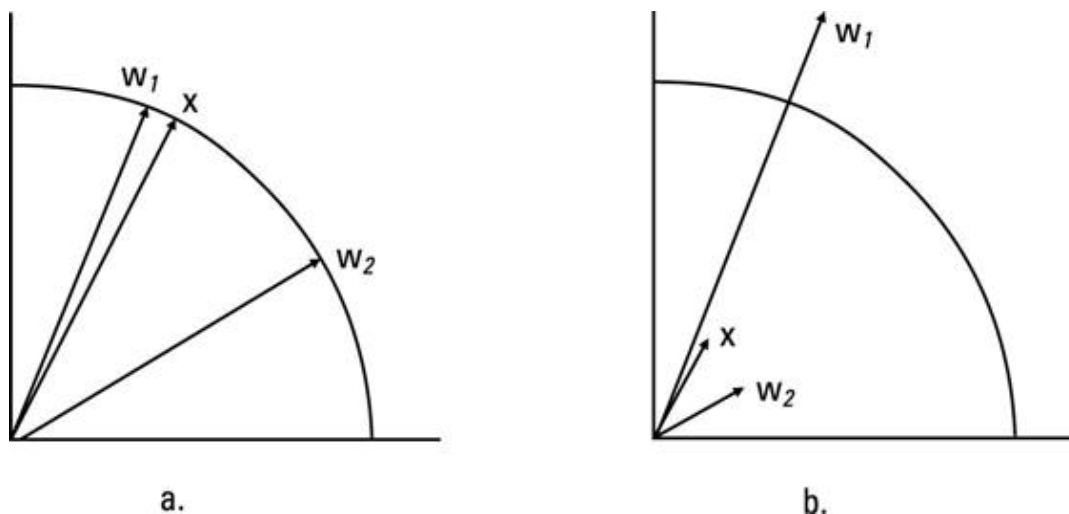


FIGURA 3.22 Ilustração gráfica de produto escalar.

A necessidade dos vetores estarem normalizados é demonstrada graficamente na Figura 3.22a, notando que o produto escalar entre dois vetores é também um vetor (perpendicular ao plano dos dois vetores), cujo comprimento será tanto maior quanto mais próximos estiverem os vetores originais (desde que sejam do mesmo tamanho), já que o produto escalar é definido como $|x||w|\cos\theta$ (onde θ é o ângulo entre os vetores). O produto escalar de x por w_1 será maior do que o produto escalar de x por w_2 , dados que

a norma (tamanho) dos vetores é unitária e que o cosseno aumenta quando o ângulo se aproxima de zero. Já na [Figura 3.22b](#), os vetores não estão normalizados e o produto escalar dos vetores não é proporcional somente ao cosseno, mas também à norma dos vetores. Assim, o produto escalar maior não corresponde necessariamente ao menor ângulo entre os vetores. Uma representação gráfica do que acontece durante a etapa de treinamento está ilustrada na [Figura 3.23](#).

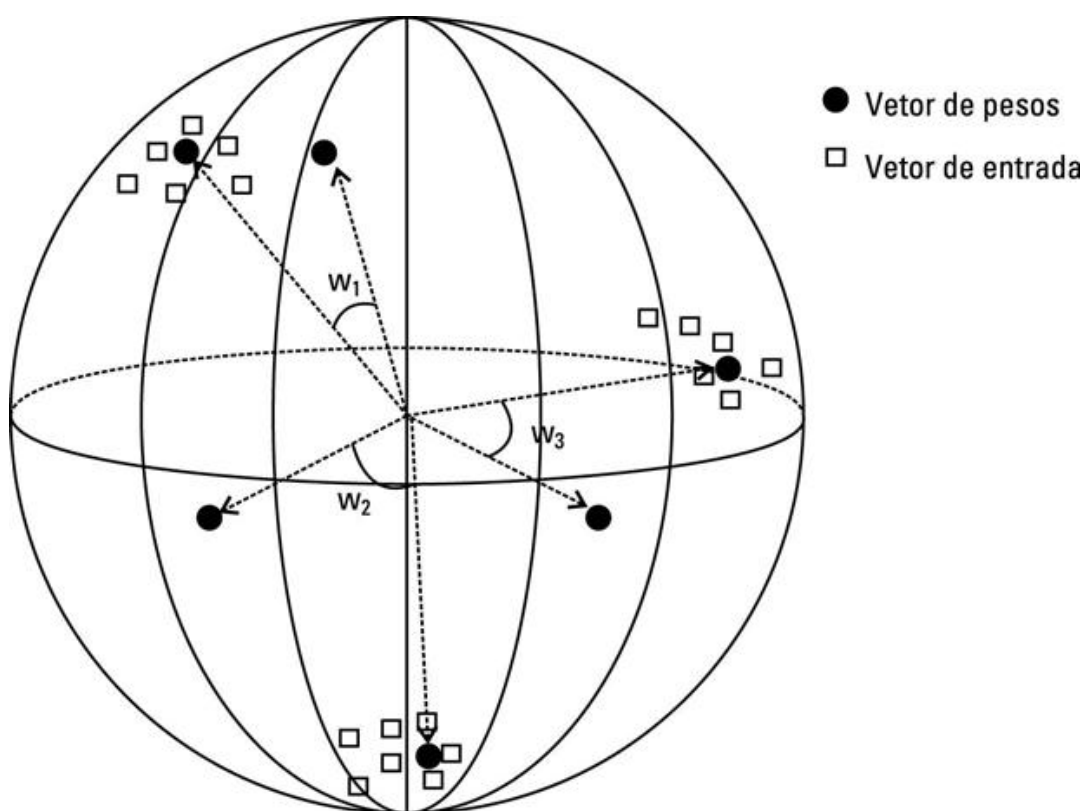


FIGURA 3.23 Os vetores pesos durante treinamento.

É útil lembrar que tanto o vetor de pesos w_j quanto o vetor de entrada x possuem a mesma dimensão (no caso, dimensão três) e estão normalizados (comprimento unitário). Assim, se os padrões de entrada possuem três características de agrupamento, a dimensão da entrada é três (três nós de entrada), o vetor de pesos também possui dimensão três e os vetores são representáveis no espaço tridimensional, dentro de uma esfera de raio unitário. A figura mostra os vetores de pesos representados por esferas pretas, e os vetores de entrada (padrões) representados por quadrados. Ao longo do

treinamento a atualização dos pesos vai rodando os vetores de pesos e fazendo com que cada vetor de pesos se aproxime do centro do grupo de vetores de entrada para os quais o respectivo nó i sai vencedor. Ou seja, o vetor de pesos é o típico representante daquele *cluster* relacionado com os vetores de entrada

3.8.4 Distância euclidiana

Este outro método determina qual o vetor k de pesos está mais próximo ao vetor de entrada e utiliza a informação da distância euclidiana entre os vetores dada por $\|w_k - x\|$, para tanto. O vetor w_k , com a menor distância euclidiana ganha a disputa. Assim, somente este vetor será ajustado segundo a fórmula (3.41), sendo η o coeficiente de aprendizagem. Este método geralmente é mais utilizado que o produto escalar.

$$w_k(\text{novo}) = w_k + (x - w_k) \quad (3.41)$$

3.9 Redes de Kohonen

As redes de Kohonen, também conhecidas como Mapas Auto-organizáveis de Kohonen, foram propostas por Teuvo Kohonen, em 1984, e utilizam um procedimento de aprendizado não supervisionado. A arquitetura da rede é constituída de uma só camada de nós, geralmente bidimensional, ligada aos nós da entrada (Figura 3.24). O novo conceito introduzido nessa estrutura de rede é que o comportamento de um determinado nó é diretamente afetado pelo comportamento dos nós vizinhos (vizinhança local). Cada neurônio está conectado a cada componente de entrada através de um vetor de pesos cuja dimensão é a mesma da entrada.

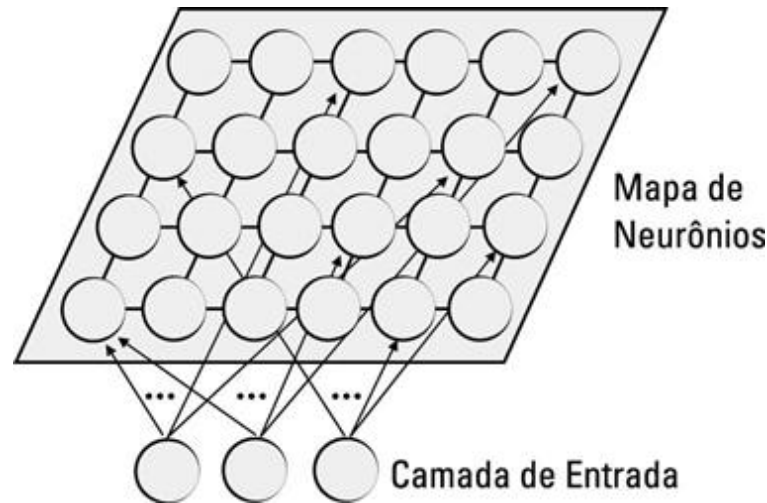


FIGURA 3.24 Arquitetura de uma rede de Kohonen.

Durante o procedimento de treinamento destas redes, os pesos são alterados não somente nos do neurônio vencedor, mas também nos pesos dos neurônios de sua vizinhança local. Padrões que estejam próximos no espaço original (que tenham características similares) acionarão um mesmo neurônio ou acionarão neurônios próximos na malha de Kohonen. Assim, é preciso definir quais são os modos de medir distâncias na estrutura, ou seja, definir qual a vizinhança que será afetada durante a etapa de treinamento. Duas formas comuns são por meio das distâncias de Manhattan e de Grid. Considerando dois neurônios em uma malha bidimensional $N1$ e $N2$, com coordenadas (i,j) e (k,h) , as respectivas métricas são expressas por:

$$\text{Dist}_{\text{Manhatan}}(N1, N2) = \text{abs}(h - j) + \text{abs}(k - i);$$

$$\text{Dist}_{\text{Grid}}(N1, N2) = \max\{|h - j|, |k - i|\}$$

As Figuras 3.25 e 3.26 ilustram essas distâncias em duas redes com 2×3 neurônios ordenados pelas linhas das estruturas.

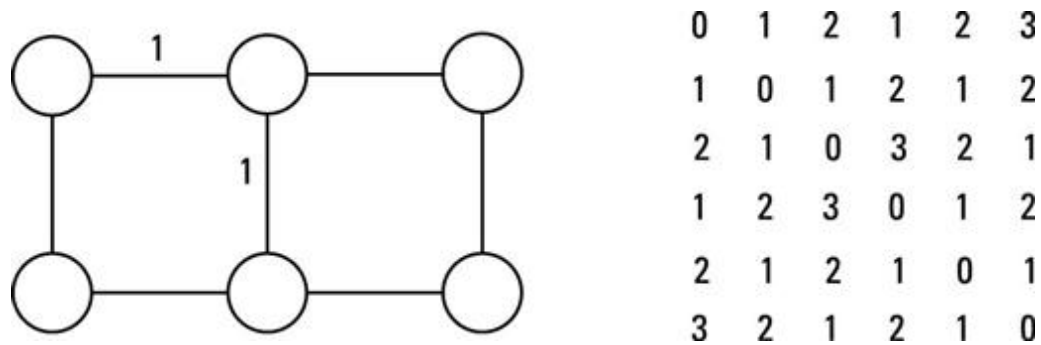


FIGURA 3.25 Distância de Manhattan em uma rede 2 × 3.

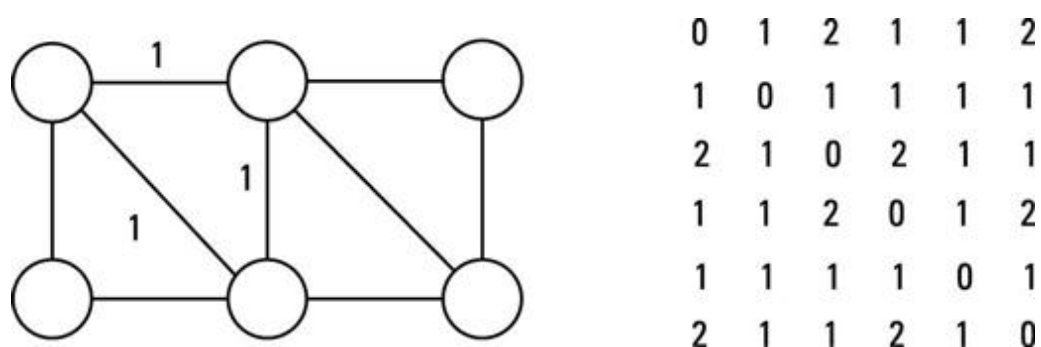


FIGURA 3.26 Distância de Grid em uma rede 2 × 3.

O algoritmo de treinamento desse tipo de rede está indicado a seguir:

1. Escolha aleatoriamente q vetores de pesos iniciais.
2. Apresente a entrada $x(k)$ para $k = 0$ na primeira iteração.
3. Calcule a distância de $x(k)$ a cada um dos q vetores de pesos $W_j(k)$, $j = 1...q$ e nomeie como neurônio vencedor aquele para o qual $(W_j(k) - x(k))^2$ ou $\|W_j(k) - x(k)\|$ é minimizado.
4. Escolha uma vizinhança $V_j(n)$ para o neurônio vencedor i .
5. Atualize os pesos de todos os neurônios de $V_i(n)$:

$$w_j(n+1) = \begin{cases} w_j(n) + ((n[x(n) - w_j(n)])j \in V_i(n) \\ w_j(n) & \text{caso contrário} \end{cases}$$

Onde η é a taxa de aprendizado, em geral decrescente ao longo do treinamento (sendo que a vizinhança $V_i(n)$ também pode diminuir).

6. Retorne ao passo 2 até que ocorra a convergência de valores.

A entrada de uma rede de Kohonen é um conjunto de vetores de padrões. Por exemplo, pode haver 10 mil padrões de clientes bancários, cada qual formado por um vetor de tamanho 5, contendo, para cada cliente, informações como: idade; imóvel próprio ou não; CEP; profissão; e renda. Na escolha de uma malha bidimensional 4×4 , têm-se 16 neurônios na malha, cada um deles ligado ao vetor de entrada através de um vetor de pesos de tamanho 5. Ao final da fase de treinamento, pode-se atribuir 1 aos 16 neurônios da malha a cada um dos 10 mil padrões.

Outro exemplo poderia ser um vetor de 13 características de animais e padrões de vetores correspondentes diversos, como aqueles ilustrados na tabela seguinte. A classificação de uma rede de Kohonen usando uma malha de 10×10 é calculada como na tabela posterior.

Animal	Pombo	Galinha	Pato	Ganso	Falcão	Coruja
Pequeno	1	1	1	1	1	1
Médio	0	0	0	0	0	0
Grande	0	0	0	0	0	0
2 pernas	1	1	1	1	1	1
4 pernas	0	0	0	0	0	0
Pêlo	0	0	0	0	0	0
Chifre	0	0	0	0	0	0
Crina	0	0	0	0	0	0
Penas	1	1	1	1	1	1
Caça	0	0	0	0	1	1
Corre	0	0	0	0	0	0
Voa	1	0	0	1	1	1
Nada	0	0	1	1	0	0

cão	.	.	raposa	.	.	gato	.	.	águia
.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	coruja
.	.	.	.	.	.	tigre	.	.	.
lobo	.	.	.	.	.	.	.	.	falcão
.	.	.	leão	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	pombo
cavalo	.	.	.	.	.	.	galinha	.	.
.	.	.	.	vaca	.	.	.	.	ganso
zebra	.	.	.	.	.	.	pato	.	.

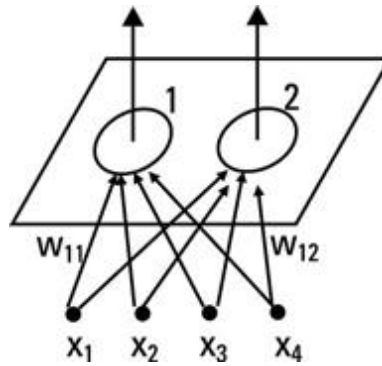
3.9.1 Exemplos de redes de Kohonen

Seja uma rede de Kohonen destinada a classificar quatro vetores, com quatro características cada, em dois *clusters* diferentes. Os vetores são: $a = (1,1,0,0)$ $b = (0,0,0,1)$; $c = (1,0,0,0)$; $d = (0,0,1,1)$.

A taxa de aprendizado η começará com 0,6 e será cortada pela metade a cada iteração, isto é, $\eta(t + 1) = \eta(t)/2$.

O treinamento se encerrará quando a taxa de treinamento cair abaixo de 0,01 ou o erro médio para os quatro vetores cair também abaixo de 0,01 (estes critérios são arbitrários). Como queremos classificar em dois *clusters* este problema, vamos usar, para efeito de simplificação dos cálculos correspondentes, uma arquitetura bem simples, com apenas dois nós e uma vizinhança de raio zero (onde somente o nó vencedor terá seus pesos alterados para cada padrão apresentado).

A figura seguinte ilustra a estrutura da rede em questão.



Os pesos são inicializados aleatoriamente com os valores $W_1 = (0,2 \ 0,6 \ 0,5 \ 0,9)$ e $W_2 = (0,8 \ 0,4 \ 0,7 \ 0,3)$, definindo a matriz de pesos abaixo.

$$W = \begin{bmatrix} 0,2 & 0,8 \\ 0,6 & 0,4 \\ 0,5 & 0,7 \\ 0,9 & 0,3 \end{bmatrix}$$

Calculando as distâncias dos vetores de entrada aos vetores de pesos de cada nó, tem-se para o vetor $a = (1,1,0,0)$:

$$D_1 = (0,2 - 1)^2 + (0,6 - 1)^2 + (0,5 - 0)^2 + (0,9 - 0)^2 = 1,86;$$

$$D_2 = (0,8 - 1)^2 + (0,4 - 1)^2 + (0,7 - 0)^2 + (0,3 - 0)^2 = 0,98.$$

Assim, o vetor que está mais próximo do nó 2 é, portanto, o nó vencedor. Atualizando os pesos do nó vencedor, temos:

$$W_{12}(t) = W_{12}(t-1) + 0,6(1 - 0,8) = 0,8 + 0,12 = 0,92;$$

$$W_{22}(t) = W_{22}(t-1) + 0,6(1 - 0,4) = 0,4 + 0,36 = 0,76;$$

$$W_{32}(t) = W_{32}(t-1) + 0,6(0 - 0,7) = 0,7 - 0,42 = 0,28;$$

$$W_{42}(t) = W_{32}(t-1) + 0,6(0 - 0,3) = 0,3 - 0,18 = 0,12.$$

Os novos valores dos pesos resultantes são, portanto:

$W_1 = (0,2 \ 0,6 \ 0,5 \ 0,9)$ e $W_2 = (0,92 \ 0,76 \ 0,28 \ 0,12)$, definindo a nova matriz W .

$$W = \begin{bmatrix} 0,2 & 0,92 \\ 0,6 & 0,76 \\ 0,5 & 0,28 \\ 0,9 & 0,12 \end{bmatrix}$$

Para o vetor $b = (0,0,0,1)$, têm-se:

$$D_1 = (0,2 - 0)^2 + (0,6 - 0)^2 + (0,5 - 0)^2 + (0,9 - 1)^2 = 0,66;$$

$$D_2 = (0,92 - 0)^2 + (0,76 - 0)^2 + (0,28 - 0)^2 + (0,12 - 1)^2 = 2,27.$$

Para este caso, o vetor b está mais próximo do nó 1, que é, portanto, o vencedor. Atualizando os pesos do nó vencedor, têm-se:

$$W_{11}(t) = W_{11}(t-1) + 0,6(0 - 0,2) = 0,2 - 0,12 = 0,08;$$

$$W_{21}(t) = W_{21}(t-1) + 0,6(0 - 0,6) = 0,6 - 0,36 = 0,24;$$

$$W_{31}(t) = W_{31}(t-1) + 0,6(0 - 0,5) = 0,5 - 0,30 = 0,20;$$

$$W_{41}(t) = W_{31}(t-1) + 0,6(1 - 0,9) = 0,9 + 0,06 = 0,96.$$

Os novos valores dos pesos são, portanto: $W_1 = (0,08 \ 0,24 \ 0,20 \ 0,96)$ e $W_2 = (0,92 \ 0,74 \ 0,28 \ 0,12)$, definindo a matriz de pesos atual.

$$W = \begin{bmatrix} 0,08 & 0,92 \\ 0,24 & 0,74 \\ 0,20 & 0,28 \\ 0,96 & 0,12 \end{bmatrix}$$

Para o vetor $c = (1,0,0,0)$, tem-se: $D_1 = 1,86$; $D_2 = 0,67$. O nó 2 será o vencedor e os novos pesos serão $W_1 = (0,08 \ 0,24 \ 0,20 \ 0,96)$ e $W_2 = (0,968 \ 0,304 \ 0,112 \ 0,048)$, com a nova matriz W .

$$W = \begin{bmatrix} 0,08 & 0,968 \\ 0,24 & 0,304 \\ 0,20 & 0,112 \\ 0,96 & 0,048 \end{bmatrix}$$

Para o vetor $d = (0,0,1,1)$, têm-se: $D_1 = 0,70$; $D_2 = 2,72$. O nó 1 será o vencedor e os novos pesos serão $W_1 = (0,032 \ 0,096 \ 0,680 \ 0,984)$ e $W_2 = (0,968 \ 0,304 \ 0,112 \ 0,048)$, com a matriz de pesos.

$$W = \begin{bmatrix} 0,032 & 0,968 \\ 0,096 & 0,304 \\ 0,680 & 0,112 \\ 0,984 & 0,048 \end{bmatrix}$$

No ciclo seguinte, a taxa de treinamento η será de $0,6 / 2 = 0,3$.

A aplicação do segundo ciclo de treinamento resulta em: $W_1 = (0,016 \ 0,047 \ 0,630 \ 0,999)$ e $W_2 = (0,980 \ 0,360 \ 0,155 \ 0,024)$.

$$W = \begin{bmatrix} 0,016 & 0,980 \\ 0,047 & 0,360 \\ 0,630 & 0,155 \\ 0,999 & 0,024 \end{bmatrix}$$

Após 100 épocas de treinamento, o algoritmo converge para $W_1 = (0,0 \ 0,0 \ 0,5 \ 1,0)$ e $W_2 = (1,0 \ 0,5 \ 0,0 \ 0,0)$.

$$W = \begin{bmatrix} 0,0 & 1,0 \\ 0,0 & 0,5 \\ 0,5 & 0,0 \\ 1,0 & 0,0 \end{bmatrix}$$

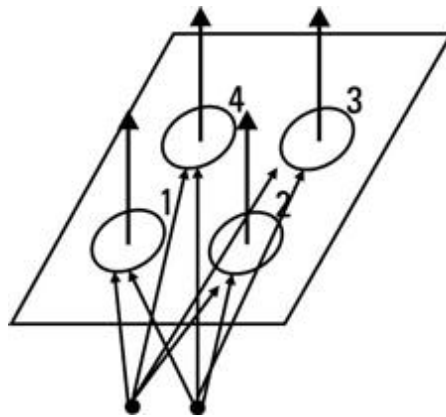
Com esses pesos a rede classifica os vetores a e b no *cluster* 1 e c e d no *cluster* 2 do problema em questão.

Seja agora uma rede simples para classificar veículos. Nesse exemplo simplificado será considerada a vizinhança sendo o próprio neurônio

utilizando-se duas iterações e fazendo $\eta = 0,8$ fixo. Como características serão usados o número de rodas do veículo e a existência ou não de motor nele (como numa bicicleta, por exemplo). O valor unitário indica a existência do motor, enquanto um valor nulo indica a ausência. Os padrões de entrada para os dois veículos usados no treinamento estão ilustrados abaixo.

	Rodas	Motor
Bicicleta	2	0
Carro	4	1

Como há duas características, a camada de entrada da rede consistirá de dois neurônios, e na camada de saída serão utilizados quatro neurônios. A figura a seguir ilustra a estrutura da rede considerada neste problema.



Os pesos são inicializados aleatoriamente em:

$$\text{Neurônio 1} = \{w_{11}, w_{21}\} = \{1, 2\};$$

$$\text{Neurônio 2} = \{w_{12}, w_{22}\} = \{2, 2\};$$

$$\text{Neurônio 3} = \{w_{13}, w_{23}\} = \{1, 3\};$$

$$\text{Neurônio 4} = \{w_{14}, w_{24}\} = \{3, 2\}.$$

Primeira iteração: apresentadas as características da bicicleta {2,0} e calculando as distâncias correspondentes, têm-se:

$$d1 = (2 - 1)^2 + (0 - 2)^2 = 5;$$

$$d2 = (2 - 2)^2 + (0 - 2)^2 = 4;$$

$$d3 = (2 - 1)^2 + (0 - 3)^2 = 10;$$

$$d4 = (2 - 3)^2 + (0 - 2)^2 = 5.$$

O neurônio vencedor é o segundo (d2). Ajustando seu peso, têm-se:

$$w_{12} = w_{12} + 0,8(x_1(t) - w_{12}(t)) = 2 + 0,8.(2 - 2) = 2;$$

$$w_{22} = w_{22} + 0,8(x_2(t) - w_{22}(t)) = 2 + 0,8.(0 - 2) = 0,4.$$

Apresentadas as características do automóvel {4,1} e calculando as distâncias correspondentes, têm-se:

$$d_1 = (4 - 1)^2 + (1 - 2)^2 = 10;$$

$$d_2 = (4 - 2)^2 + (1 - 0,4)^2 = 4,36;$$

$$d_3 = (4 - 1)^2 + (1 - 3)^2 = 13;$$

$$d_4 = (4 - 3)^2 + (1 - 2)^2 = 2.$$

O neurônio vencedor é o d_4 . Ajustando seu peso, têm-se:

$$w_{14} = w_{14} + 0,8(x_1(t) - w_{14}(t)) = 3 + 0,8.(4 - 3) = 3,8;$$

$$w_{24} = w_{24} + 0,8(x_2(t) - w_{24}(t)) = 2 + 0,8.(1 - 2) = 1,2.$$

Segunda iteração: apresentadas as características da bicicleta {2,0} e calculando as distâncias, têm-se:

$$d_1 = (2 - 1)^2 + (0 - 2)^2 = 5;$$

$$d_2 = (2 - 2)^2 + (0 - 0,4)^2 = 0,16;$$

$$d_3 = (2 - 1)^2 + (0 - 3)^2 = 10;$$

$$d_4 = (2 - 3,8)^2 + (0 - 1,2)^2 = 4,68.$$

Novamente o neurônio vencedor é o segundo (d_2). Ajustando seu peso, têm-se:

$$w_{12} = w_{12} + 0,8(x_1(t) - w_{12}(t)) = 2 + 0,8.(2 - 2) = 2;$$

$$w_{22} = w_{22} + 0,8(x_2(t) - w_{22}(t)) = 0,4 + 0,8.(0 - 0,4) = 0,08.$$

Apresentado as características do automóvel {4,1} e calculando as distâncias associadas, têm-se:

$$d_1 = (4 - 1)^2 + (1 - 2)^2 = 10;$$

$$d_2 = (4 - 2)^2 + (1 - 0,08)^2 = 4,8464;$$

$$d_3 = (4 - 1)^2 + (1 - 3)^2 = 13;$$

$$d_4 = (4 - 3,8)^2 + (1 - 1,2)^2 = 0,08.$$

Novamente o neurônio vencedor é o d_4 . Ajustando seu peso, têm-se:

$$w_{14} = w_{14} + 0,8(x_1(t) - w_{14}(t)) = 3,8 + 0,8.(4 - 3,8) = 3,96;$$

$$w_{24} = w_{24} + 0,8(x_2(t) - w_{24}(t)) = 1,2 + 0,8.(1 - 1,2) = 1,04.$$

Nova entrada: apresentando as características de uma moto que possui duas rodas e motor. Como as entradas são mais parecidas à da bicicleta, a rede deve classificar como bicicleta. A isto se dá o nome de generalização. Para classificar como moto, a rede deveria ser treinada para este fim, onde às vezes é necessário fornecer outras características de entrada.

Verificando a saída da rede treinada para a entrada de informação correspondente à moto {2,1}:

$$d_1 = (2 - 1)^2 + (1 - 2)^2 = 2;$$

$$d_2 = (2 - 2)^2 + (1 - 0,08)^2 = 0,8464;$$

$$d_3 = (2 - 1)^2 + (1 - 3)^2 = 5;$$

$$d_4 = (2 - 3,8)^2 + (1 - 1,2)^2 = 3,8432.$$

Como era de se esperar, o neurônio vencedor foi o segundo, indicando que o modelo apresentado à rede é da classe das bicicletas.

3.10 Redes RBF

As redes de função de base radial (RBF – Radial Basis Function) são utilizadas com sucesso em classificações de padrões, em aproximações de funções (Haykin, 2007), em previsões de séries temporais (Sun et al., 2005), e em outras aplicações.

Uma rede RBF em sua forma mais básica possui apenas uma única camada oculta, enquanto redes do tipo MLP apresentam uma ou mais camadas escondidas, segundo suas especificações (Haykin, 2007). A camada escondida de uma rede RBF é constituída por funções de ativação de base radial, onde as mais usuais são as gaussianas que são definidas por seus centros e larguras. A camada de saída da rede possui pesos que ponderam as informações das saídas de cada função de base radial, agregando-as linearmente na saída da rede (que

pode ter mais de uma saída, conforme a aplicação). A [Figura 3.27](#) ilustra a estrutura típica de uma rede RBF.

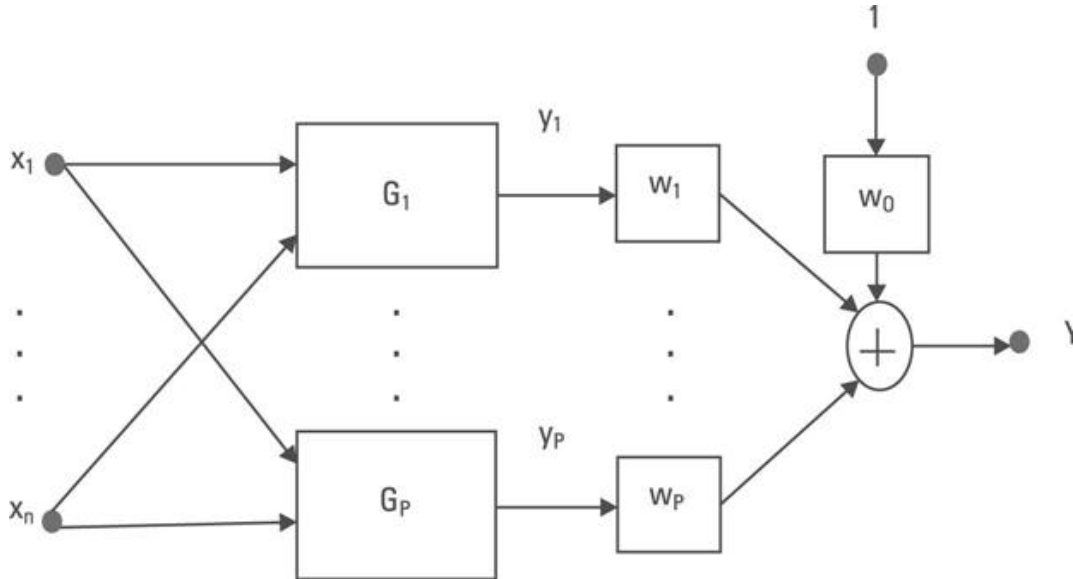


FIGURA 3.27 Estrutura típica de uma rede RBF.

Não há pesos na primeira camada da rede, as variáveis de entrada constituem os dados das funções de ativação (\$G_j\$), que por sua vez são os próprios nós da camada oculta da rede.

A expressão típica de uma função de base radial é dada por [\(3.42\)](#), onde \$m\$ denota o valor central (centro, esperança ou média) da função em questão, e \$\sigma\$ corresponde ao seu desvio-padrão (ou largura).

$$y_j = G_j(x) = e^{-\left(\frac{(x_1 - m_1)^2}{2\sigma_1^2} + \dots + \frac{(x_n - m_n)^2}{2\sigma_n^2}\right)} \quad (3.42)$$

Os pesos (\$w_j\$) da camada de saída da rede multiplicam os valores (\$y_j\$) fornecidos pelas funções de base radial da estrutura, que totalizados (somados) fornecem o valor (\$Y\$) de saída da rede, que é adicionado a um valor de *offset* unitário multiplicado pelo peso (bias) “\$w_0\$” associado. Uma rede RBF pode ter outras saídas associadas a outros pesos correspondentes. A informação de cada saída de uma determinada rede é modelada pela equação [\(3.43\)](#).

$$Y = \sum_{i=1}^P w_i G_j(x) \quad (3.43)$$

Existem algumas estratégias utilizadas para ajustar os parâmetros (centros, desvios-padrões e pesos) das redes RBF, sendo tipicamente classificadas como empíricas, auto-organizadas e supervisionadas. Nas estratégias empíricas, os centros das funções são ajustados aleatoriamente, e as suas larguras são dadas pela relação entre a distância máxima dos centros escolhidos pelo valor da raiz quadrada do número de funções de ativação adotadas ([Haykin, 2007](#)). O objetivo é obter distribuições uniformes para as funções, com a finalidade de mapear adequadamente os dados de entrada, de forma a promover uma boa capacidade de generalização da rede. Os pesos da camada de saída são calculados pelo método dos mínimos quadrados, a partir dos valores de saída das funções gaussianas estabelecidas, onde são utilizados um determinado conjunto de dados de entrada de treinamento e o correspondente padrão desejado para as informações de saída da rede em questão.

Como estratégia auto-organizada pode-se mencionar o procedimento citado em [Haykin \(2007\)](#), que utiliza um algoritmo tipo K-means de agrupamento de dados, com uma técnica de casamento de similaridade associada a um método de atualização iterativo, cujo objetivo consiste em selecionar e ajustar os centros das funções de base radial da rede. Os pesos da camada de saída são ajustados pelo método dos mínimos quadrados. O trabalho de [Sun et al \(2005\)](#) utiliza um método otimizado de partição de dados, baseado na distância entre os centros dos agrupamentos dos dados de treinamento da rede. Uma função de custo adicional é associada ao método de otimização utilizado cujo objetivo consiste em ajustar os centros e desvios (larguras) das funções da rede. As estratégias supervisionadas ([Haykin, 2007](#)) utilizam procedimentos de aprendizagem que empregam o método do gradiente descendente para ajustar todos os parâmetros (centros, desvios, padrões e pesos) de uma determinada rede RBF, onde a informação do erro entre a informação de saída da rede em questão e o padrão desejado para ela é utilizado no processo de treinamento.

3.10.1 Exemplos de redes RBF

Exemplo 3.4

Utilizar uma rede RBF para emular a função lógica OU Exclusiva.

Será considerada uma rede com dois nós na camada escondida, onde as funções de base radial possuem valores de desvios-padrão unitários, com valores dos seus centros em 0,25 e 0,75. Os valores dos pesos da camada de saída da RBF são calculados com a técnica dos mínimos quadrados, em que se utiliza os dados resultantes das funções gaussianas empregadas para o padrão de dados de entrada da rede, com os valores do padrão de saída da mesma. Os cálculos mostrados a seguir ilustram o procedimento em questão. Para os vetores de entrada $x_1 = [0;0;1;1]$ e $x_2 = [0;1;0;1]$, têm-se os seguintes valores de saída das funções gaussianas: $[0,88251 \ 0,3247; 0,5352 \ 0,5353; 0,5352 \ 0,5353; 0,3247 \ 0,8825]$. Com estes valores e com os dados do vetor de saída $y = [0;1;1;0]$ calcula-se os coeficientes (W) dos pesos da rede em questão (o símbolo " T " denota matriz transposta e " -1 " matriz inversa):

$$F = \begin{bmatrix} 0,8825 & 0,3247 & 1 \\ 0,5353 & 0,5353 & 1 \\ 0,5353 & 0,5353 & 1 \\ 0,3247 & 0,8825 & 1 \end{bmatrix}; \quad Y = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}; \quad W = [F^T F]^{-1} Y = \begin{bmatrix} -7,3206 \\ -7,3206 \\ +8,8375 \end{bmatrix}.$$

Todo o procedimento de obtenção dos parâmetros da rede RBF é do tipo não supervisionado (conforme citado anteriormente).

Exemplificando numericamente o mapeamento resultante da função XOR pela rede RBF obtida, sendo que para $x_1 = 1$ e $x_2 = 1$, têm-se: $y = 8,8375 - 7,3206 \times 0,3247 - 7,3206 \times 0,8825 = 0,0004$ (que corresponde ao nível lógico "0"). De forma similar para $x_1 = 0$ e $x_2 = 0$, tem-se $y = 0,0004$. Para $x_1 = 0$ e $x_2 = 1$, tem-se $y = 1,0006$, que corresponde ao nível lógico "1". Idem para $x_1 = 1$ e $x_2 = 0$ com $y = 1,0006$ (nível "1" também). No [Anexo 3.3](#) encontra-se o código-fonte de um programa básico desenvolvido para computar o modelo da rede RBF em questão.

Exemplo 3.5

Neste exemplo, uma rede RBF será empregada para aproximar a função não linear $y = x_1^2$ para $x_1 = [0, 1]$.

Será considerada uma rede com três nós na camada escondida, onde se empregará conceitos de agrupamentos de dados no estabelecimento dos parâmetros da rede. As respectivas funções gaussianas possuem os seguintes centros, considerando-se três *clusters* igualmente espaçados no intervalo de valores da variável x_1 : 0,175; 0,525; 0,85. Os desvios-padrão das funções de base radial foram estimados calculando os respectivos desvios dos três agrupamentos considerados. Estes valores foram multiplicados por um fator de valor igual a 10, com a finalidade de se ajustar adequadamente às dispersões das gaussianas para a modelagem da função em questão, obtendo-se: 1,2247; 0,9354; 1,0801. Os pesos da camada de saída da rede foram calculados de forma similar (pelo método dos mínimos quadrados) ao do exemplo anterior, gerando-se: $w_0 = 11,2323$; $w_1 = -15,2547$; $w_2 = 12,1$.

Exemplificando numericamente o mapeamento resultante da função não linear pela rede RBF obtida, para $x_1 = 0,5$ têm-se os valores das funções gaussianas correspondentes: 0,9654; 0,9996; 0,9489. Logo, o valor estimado pela rede é dado por:

$$11,2323 - 15,2547 \times 0,9654 + 12,1 \times 0,9996 - 8,7973 \times 0,9489 = 0,254$$
(sendo um valor próximo da função original $0,5^2 = 0,25$).

O gráfico da [Figura 3.28](#) contém os dados da função não linear original e os valores aproximados pela rede RBF resultante. A somatória do erro quadrático entre os valores originais e os estimados foi de 0,0003.

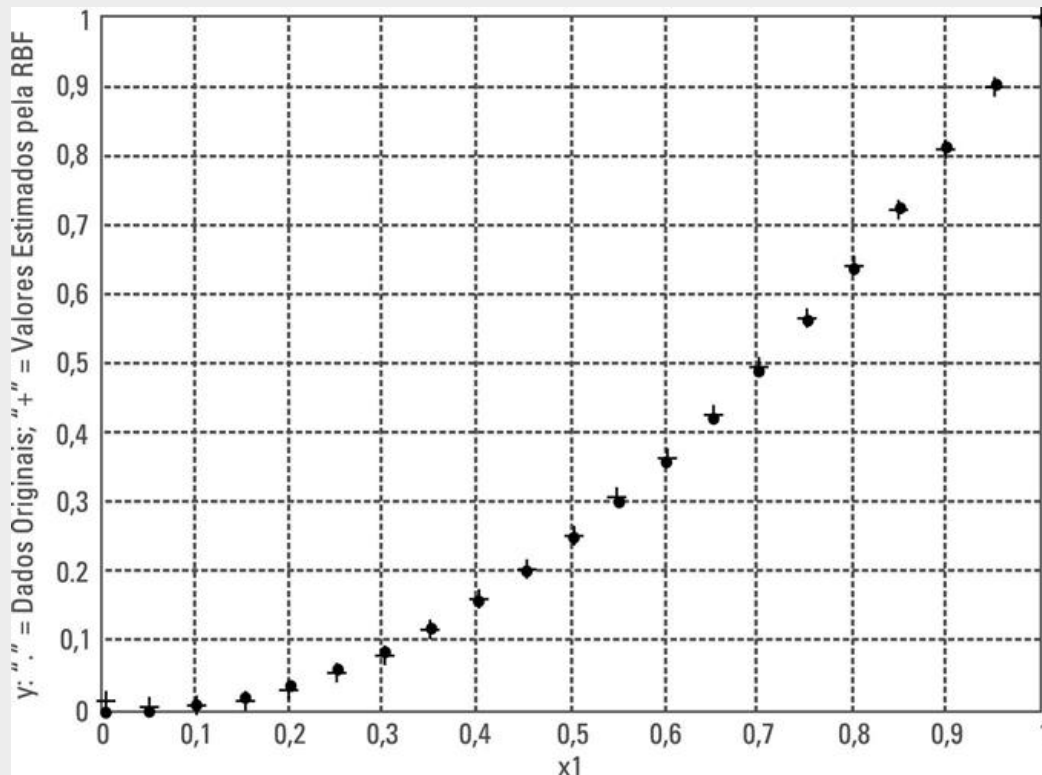


FIGURA 3.28 Aproximação da rede RBF para a função não linear.

Este exemplo será novamente resolvido por meio do *toolbox* de rede neural do MatLab, especificamente utilizando uma estrutura RBF. Os comandos a seguir mostram a resolução do problema:

```
x1=0:0.05:1;y=x1.*x1; % Padrões de Entrada e Saída
network=newrb(x1,y); % Comando para obtenção da RBF
Y=sim(network,x1); % Simulação da rede neural obtida.
plot(x1,y,'k.',x1,Y,'k+');grid;xlabel('x1'); % Gráficos Resultantes
ylabel('y: \".\" = Dados Originais; \"+\" = Valores do Treinamento');<Enter>.
```

A ferramenta em questão utiliza técnicas de aprendizagem supervisionada que possibilita uma exatidão melhor na modelagem neural obtida. O *toolbox nntool* (citado anteriormente) também pode ser utilizado para resolver o problema. A [Figura 3.29](#) ilustra, por meio de um gráfico, o resultado obtido após o treinamento da rede.

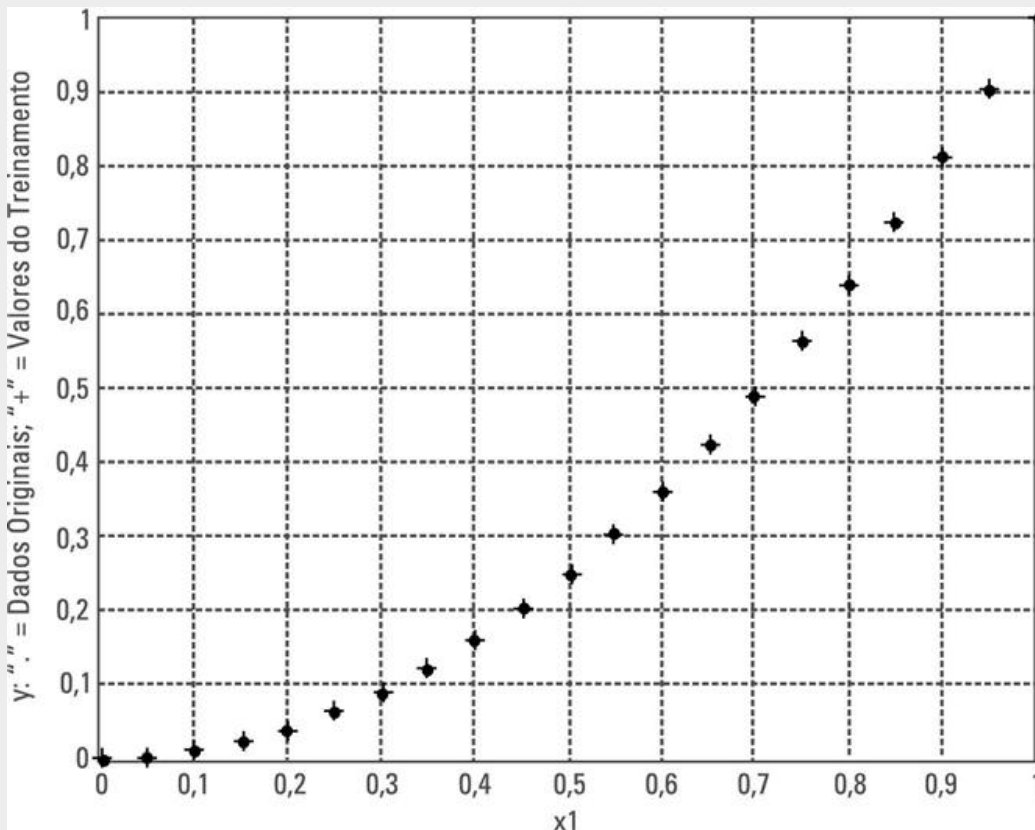


FIGURA 3.29 Resultados da RBF obtida pelo toolbox de redes neurais.

Exemplo 3.6

Aqui será resolvido o problema do [Exemplo 2.5](#), relativo à modelagem de um processo dinâmico não linear, que possui um modelo contínuo dado por (2.26), ou um modelo amostrado representado por (2.27). Dados do processo estão ilustrados nas [Figuras 2.25 e 2.26](#).

A [Figura 3.30](#) demonstra os valores estimados por meio de um modelo ARX linear (cuja modelagem foi obtida pelo método dos mínimos quadrados, a partir dos dados da [Figura 2.25](#), e é expressa por $y(k) = a_1 y(k-1) + b_1 u(k-1)$, onde os coeficientes são $a_1 = 0,9789$ e $b_1 = 0,0285$), e os dados originais do sistema de nível em questão (referentes agora aos dados de teste da [Figura 2.26](#)). O erro (soma de quadrados) da modelagem resultante foi de 168,15 e com um grau de correlação cruzada (*best fits*) em 57,65%. Nota-se no gráfico que a modelagem obtida não reproduz

adequadamente os dados originais do processo, o que é explicado pelo fato de o sistema real possuir característica não linear e o modelo ARX ser linear.

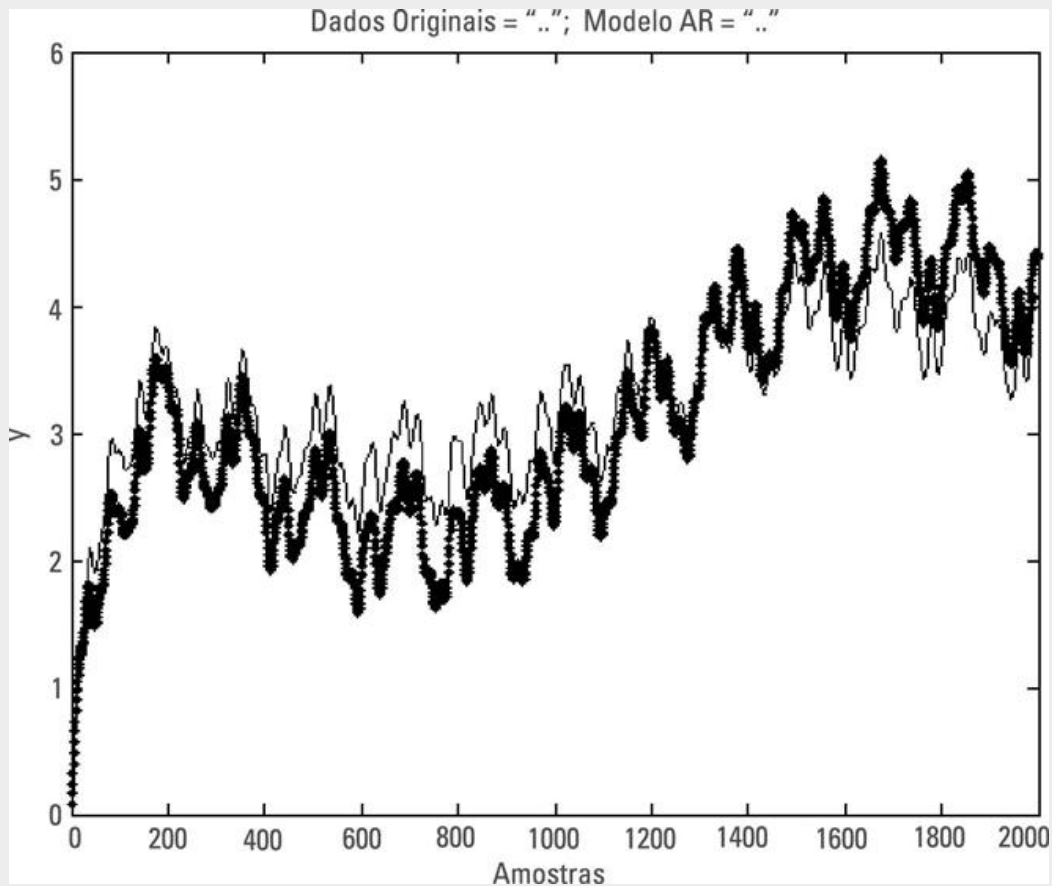


FIGURA 3.30 Dados do processo e do modelo ARX linear.

Será considerada agora uma rede RBF para a modelagem. Ela possui seis nós na camada escondida e utiliza como variáveis de entrada e saída uma estrutura similar a da [Figura 2.27](#), mas agora usando uma rede neural no lugar do modelo *fuzzy*. Os parâmetros das funções gaussianas possuem os seguintes valores de centros (os 12 primeiros valores) e desvios (os outros 12 dados) para as variáveis em questão: 1,2244; 2,0988; 3,5615; 3,3604; 1,6765; 3,0679; 3,7259; 4,4499; 2,0468; 4,2379; 3,1200; 2,2608; 2,1237; 2,0833; 2,0377; 2,1545; 2,2534; 2,2281; 2,1887; 2,1047; 2,1489; 2,0432; 2,2162; 2,8420.

Ajustando um fator multiplicativo para as dispersões em um valor igual a 10, têm-se os seguintes valores dos pesos da camada de saída: $w_0 = -64,2028$; $w_1 = -73,2999$; $w_2 = -454,2324$; $w_3 = -92,9178$; $w_4 = 277,0609$; $w_5 = 174,6446$; $w_6 = 236,6277$.

Detalhes da obtenção dos parâmetros da rede estão disponíveis em [Pinheiro et al. \(2013\)](#). A [Figura 3.31](#) mostra os dados originais do sistema de nível (que foram utilizados para treinamento) e os valores da rede RBF projetada; o erro (soma de quadrados) foi agora de 8,34.

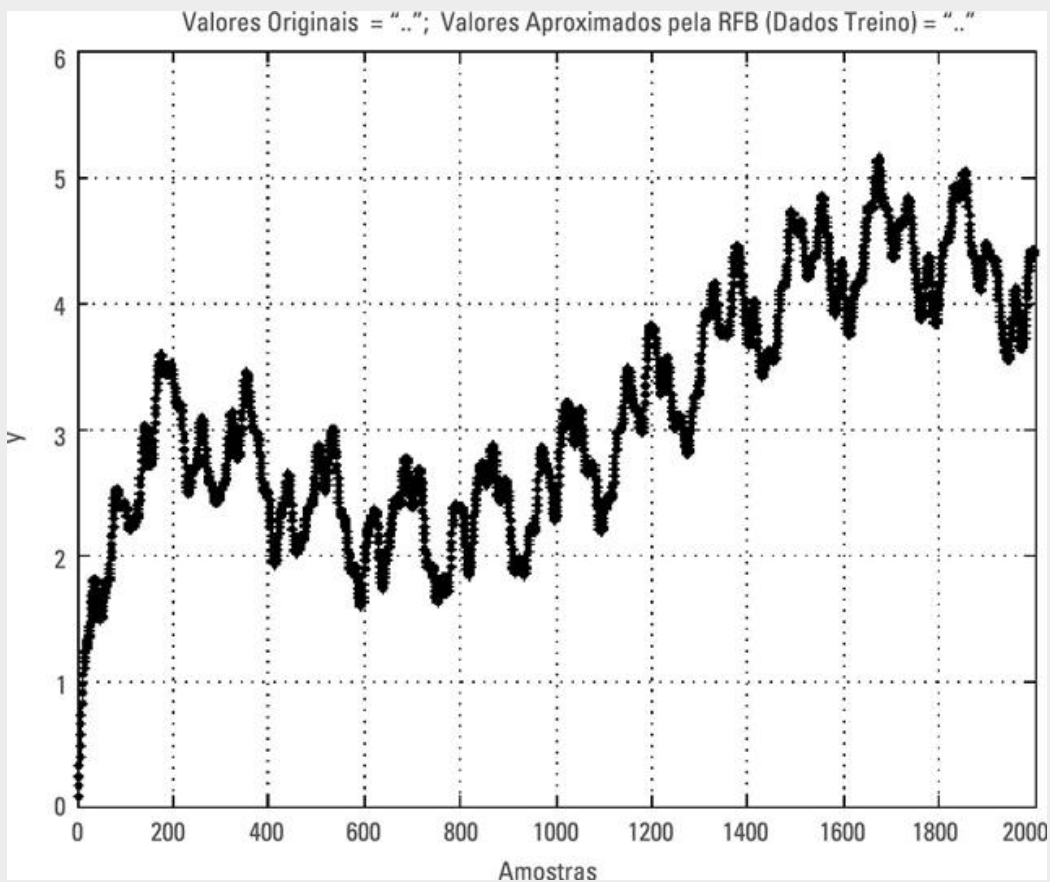


FIGURA 3.31 Dados originais do processo e do modelo RBF.

A [Figura 3.32](#) ilustra os valores estimados pela rede RBF com os dados utilizados para teste ([Figura 2.26](#)) oriundos do sistema de nível. A soma de quadrados do erro foi agora de 4,22. Verifica-se que a rede neural obtida modelou adequadamente o processo não linear considerado.

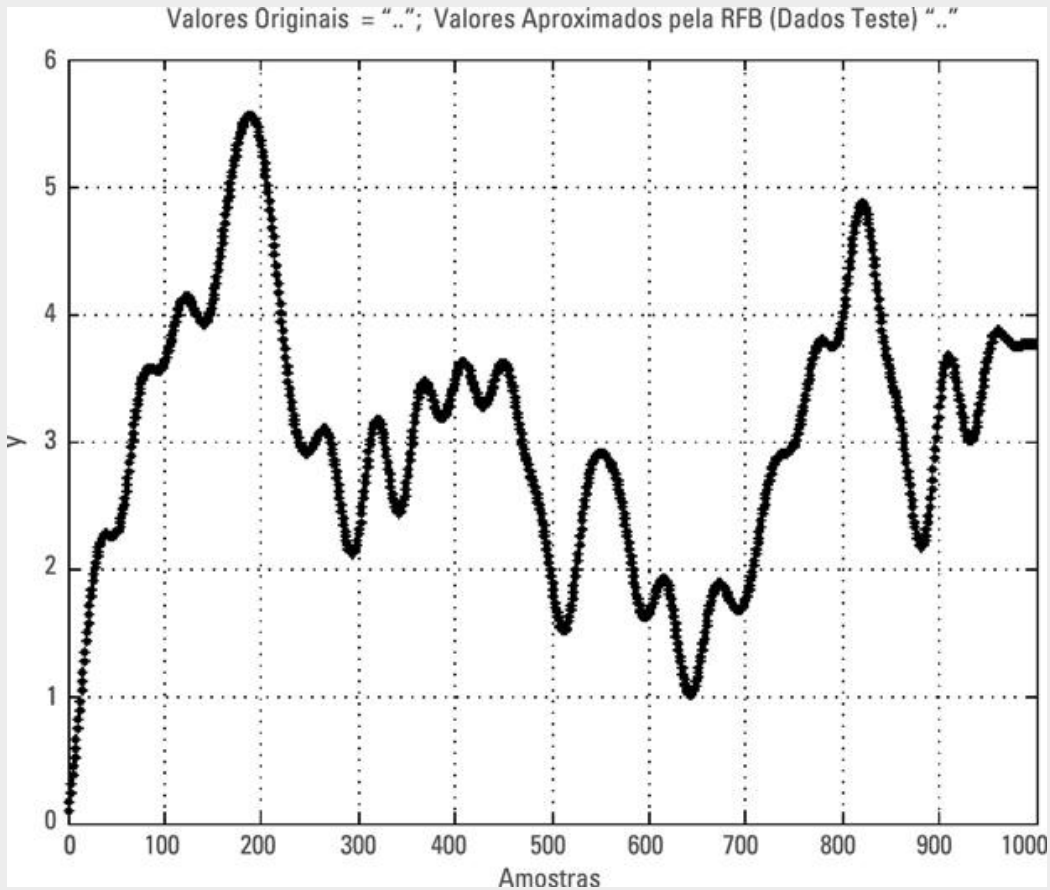


FIGURA 3.32 Dados de teste do processo e estimados pelo modelo RBF.

3.11 Outros tipos de Redes Neurais Artificiais

Existem outros tipos de redes neurais como a de Hopfield, as Recorrentes, a máquina de Boltzmann, as Máquinas de Vetor de Suporte, entre outras (Haykin, 2007).

Uma rede de Hopfield utiliza o conceito de função de energia, em que a rede em questão converge para um conjunto estável de ativações em relação à função considerada (Hopfield, 1984).

As redes Recorrentes diferem das de Hopfield em dois aspectos (Haykin 2007) que as tornam adequadas para aplicações em tempo real: têm camadas escondidas; possuem dinâmicas arbitrárias.

A máquina de Boltzmann foi proposta por Hinton e Sejnowski (1983), e nela os estados das unidades da rede com valores binários possuem transições:

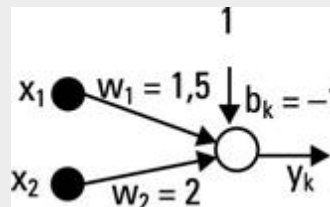
de estados probabilísticas. Em aplicações em problemas de otimização, os pesos desta rede representam as restrições do problema a ser otimizado.

As Máquinas de Vetor de Suporte (SVM – Support Vector Machine) utilizam aprendizagem supervisionada e funções de Kernel na sua composição, sendo aplicadas com sucesso em problemas de classificação de padrões e de regressão linear (Haykin, 2007). Em Carpinteiro et al. (2011) esse tipo de rede foi aplicado na previsão de séries temporais, e os resultados obtidos foram comparados com redes MLP e modelos hierárquicos.

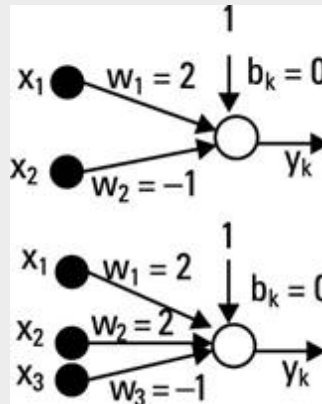
Modelos com redes neurais podem estar associados a outras técnicas de IA como sistemas *fuzzy*, definindo-se estruturas conhecidas como neurofuzzy, e com algoritmos de inspiração evolucionária (como os Algoritmos Genéticos Enxame de Partículas, e outros).

Exercícios

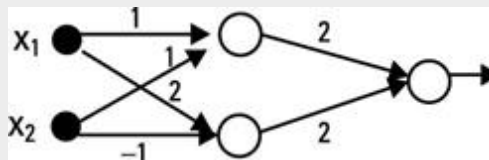
1. Considere a rede a seguir, com função de ativação de McCulloch-Pitts e com limiar θ nulo. Apresente a saída da rede para todas as combinações de entradas binárias (0 e 1) possíveis.



2. Recalcule a mesma rede, agora com entradas bipolares (-1 e 1) e limiar igual a 1 .
3. Considere a rede a seguir, com função de Ativação de McCulloch-Pitts e com limiar θ em 2 . Apresente a saída da rede para todas as combinações de entradas binárias (0 e 1) possíveis.

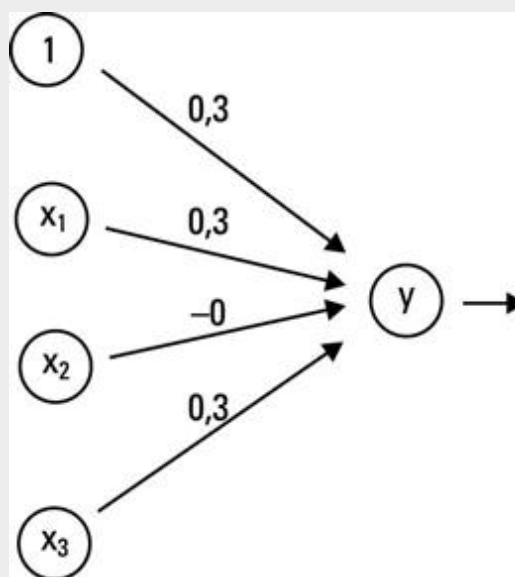


4. Considere a rede a seguir, com função de ativação de McCullock-Pitts e com limiar θ em 4. Apresente a saída da rede para todas as combinações de entradas binárias (0 e 1) possíveis.



5. Assumir uma rede com função de ativação de McCullock-Pitts, com limiar θ em 2 e com polarização $b = 0$ para todos os nós. Apresente a saída da rede para todas as combinações de entradas binárias (0 e 1) possíveis.
6. Elabore uma rede com função de ativação de McCullock-Pitts e com limiar $\theta = 0$, que responda como uma função OR NOT bipolar. Use para treinar a regra de Hebb.
7. Construa uma rede com função de ativação de McCullock-Pitts, com limiar $\theta = 0$, que responda como uma função AND NOT bipolar. Use para treinar a regra de Hebb.
8. Use a regra de Hebb para classificar os vetores $a = (1, 1, 1, 1)$ e $b = (-1, 1, -1, -1)$ em um grupo (saída = 1), e os vetores $c = (1, 1, 1, -1)$ e $d = (1, -1, -1, 1)$ em outro grupo (saída = -1). Use limiar $\theta = 0$. Faça um esquema da rede, construa a tabela de treinamento e preencha a tabela usando a regra de Hebb.
9. Repita o problema anterior usando a regra de treinamento Perceptron, com $\eta = 1$ e $\theta = 0$. Inicie todos os pesos com zero.

10. Usando regra Perceptron, treine uma rede para emular uma porta AND.
11. Construa um Perceptron de duas saídas para classificar os vetores $a = (1, 1, 1)$ e $b = (1, -1, -1)$ em uma classe, e $c = (1, 1, -1)$ e $d = (-1, 1, -1)$ em outra. Faça $\eta = 1$, $\theta = 0$, e inicialize os pesos e viés com zero. Construa a tabela de treinamento, faça um esquema gráfico da rede e teste-a para as entradas dadas e para os vetores $e = (-1, -1, 1)$, $f = (0, -1, 1)$, $g = (-1, 1, 0)$ e $h = (1, 1, 0)$. Discuta os resultados.
12. Estabeleça uma rede Adaline para a função lógica AND, utilizando entradas e saídas bipolares. Inicie todos os pesos com o valor 0,1 e $\eta = 0,25$.
13. Construa uma rede Adaline para a função lógica OR NOT, utilizando entradas e saídas bipolares. Inicie todos os pesos com o valor 0,1 e $\eta = 0,25$.
14. Foi construída uma rede Adaline, com $\eta = 0,25$, com os pesos e as bias iniciados em 0,1, para classificar os vetores $a = (1, -1, 1)$ e $b = (1, 1, -1)$ na classe de saída 1, e $c = (1, -1, -1)$ na classe de saída -1. Imediatamente antes de entrar o vetor b , a rede apresentou a configuração a seguir.
Forneça os valores dos pesos após o processamento do vetor de entrada b .



15. Utilizar como base o programa do [Anexo 3.2](#) para mapear e testar com uma RNA a função lógica OR.
16. Utilizar a ferramenta *nntool* para mapear e testar com uma RNA-MLP a função lógica XOR.

Referências bibliográficas

1. CARPINTEIRO OAS, LEITE JPRR, PINHEIRO CAM, LIMA I. Forecasting models for prediction in time series. *Artificial Intelligence Review* 2011;1–9.
2. CARPENTER GA. Distributed learning, recognition, and prediction by ART and ARTMAP neural networks. *Neural Networks*. 1997;10:1.473–94.
3. CHAUVIN YA. A backpropagation algorithm with optimal use of hidden units. In: San Mateo, CA: Morgan Kaufmann Publishers Inc; 1989:519–526. Touretzky DS, ed. *Advances in neural information processing systems 1*. 1.
4. FAUSETT L. *Fundamentals of Neural Networks: Architectures, Algorithms, and Applications*. Nova Jersey: Prentice Hall; 1994.
5. FUKUSHIMA K. Cognitron: A Self-Organizing Multi-layered Neural Network. *Biological Cybernetics*. 1975;20:121–136.
6. Haykin S. In: Porto Alegre: Artmed Editora S.A; 2007; *Redes Neurais: Princípios e Prática*.
7. HEBB DO. *The Organization of Behavior: A Neuropsychological Theory*. Nova York: Wiley; 1949.
8. HERTZ J, KROGH A, PALMER RG. *Introduction to the Theory of Neural Computation*. Reading, MA: Addison-Wesley; 1991.
9. HINTON, G., SEJNOWSKI, T. (1983). **Optimal Perceptual Inference**. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. Washington DC, 448-53.
10. HOPFIELD JJ. Neurons with graded response have collective computational properties like those of two-state neurons. *Proceedings of the National Academy of Sciences*. 1984;81:3088–3092.
11. JACBS RA. Increasead rates of convergence through learning rate adaptation. *Neural Networks*. 1988;1:295–307.

12. KOHONEN T. Correlation matrix memories. *IEEE Transactions on Computers*. 1972;21:353–359.
13. KOHONEN T. *Self-organization and associative memory*. Berlin: Springer; 1984.
14. KOHONEN T. *Self-organizing maps*. Berlin: Springer; 2001.
15. MINSKY ML, PAPERT SA. *Perceptrons*. Cambridge, MA: MIT Press; 1969.
16. NARENDRA KS, PARTHASARATHY K. Identification and control of dynamical systems using neural networks. *IEEE Trans on Neural Networks*. 1990;1:4–27.
17. NOWLON SJ, HINTON GE. In: San Mateo, CA: Morgan Kaufmann; 1992:993–1000. *Adaptive soft weight tying using Gaussian mixtures Advances in Neural Information Processing Systems*. 4.
18. PINHEIRO C, SANTOS FO, LIMA I. A Systematic Approach to Settling Parameters in RBF Networks. *Submitted to Artificial Intelligence Review* 2013.
19. RITTER H, MARTINETZ T, SCHULTEN K. *Neural Computation and Self-Organizing Maps An Introduction*. Reading, Massachusetts: Addison-Wesley; 1992.
20. ROSENBLATT F. The Perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*. 1958;65:386–408.
21. RUMELHART DE, HINTON GE, WILLIAMS RJ. Learning Representations of Back- Propagation Error. *Nature*. 1986;323:533–536.
22. RUMELHART D, ZIPSER ED. Feature discovery by competitive learning. *Cognitive Science*. 1985;9:75–112.
23. SUN YF, et al. Optimal partition algorithm of the RBF neural network and its application to financial time series forecasting. *Neural Compu & Applic*. 2005;14:36–44.
24. TAFNER M, XEREZ M, FILHO RODRIGUES##I. *Redes Neurais Artificiais: introdução e princípios de neurocomputação*. Editora EKO 1995.
25. WATROUS RL. Learning algorithms for connectionist networks: Applied methods of nonlinear optimization. *Proceedings of the IEEE 1st International Conference on Neural Networks*. 1987;2:619–628.

26. WEIGEND AS, RUMELHART DE, HUBERMAN BA. In: San Mateo, CA: Morgan Kaufmannn; 1991:875–882. *Generalization by weight-elimination with application to forecasting Advances in Neural Information Processing Systems*. 3.
27. WERBOS, P.J. (1974). **Beyond regression: New tools for prediction and analysis in the behavioral sciences. Ph.D.** Thesis, Harvard University, Cambridge, MA.
28. WHITE H. Connectionist nonparametric regression: Multilayer feedforward networks can learn arbitrary mappings. *Neural Networks*. 1990;3:535–549.
29. WINDROW B, HOFF Jr ME. Adaptive switching circuits. *Ire Wescon Convention Record* 1960:96–104.

Anexo 3.1

Código básico do Exemplo 3.1


```

% Exemplo de aproximação de função via RNA-MLP.

clear all;

%Carga dos Dados
load vetx;
load vety;

vX1 = vetx;
vYd = vety;
Np=41;

NN = 3; % Número de neurônios.
err_tol = 0.001; % Valor de parada do erro.

% Inicialização aleatória dos pesos e bias da rede.
V11 = 2 * rand - 1;
V12 = 2 * rand - 1;
V13 = 2 * rand - 1;
B01 = 2 * rand - 1;
B02 = 2 * rand - 1;
B03 = 2 * rand - 1;
W11 = 2 * rand - 1;
W21 = 2 * rand - 1;
W31 = 2 * rand - 1;
B0 = 2 * rand - 1;

% Valores de offset das entradas da rede neural.
C1 = 1; C2 = 1;

contdisp = 0; % Contador de visualização do erro.
flagdisp = 100; % Dado de impressão do erro.
max_epoc = 200000; % Número máximo de épocas.
fa = 0.005; % Fator de aprendizado.

for epoc = 1:max_epoc % Etapa de treinamento.

    for n = 1:Np

        X1 = vX1(n); Yd = vYd(n); % Padrão fornecido à rede.

```



```

% Processamento dos dados da camada escondida.
Z1in = B01*C1 + V11*X1;
Z2in = B02*C1 + V12*X1;
Z3in = B03*C1 + V13*X1;

% Funções de ativação da camada escondida.
Z1 = 2/(1 + exp(-2*Z1in)) - 1;
Z2 = 2/(1 + exp(-2*Z2in)) - 1;
Z3 = 2/(1 + exp(-2*Z3in)) - 1;

% Processamento dos dados da camada de saída.
O1 = B0*C2 + W11*Z1 + W21*Z2 + W31*Z3;
Y = O1; % Função de ativação linear.

% Treinamento por retropropagação.

e = Yd - Y; % Erro entre o valor desejado e a saída.
E(n) = e;

% Ajuste dos pesos da camada de saída.
de = e;
dW11 = fa*de*Z1; dW21 = fa*de*Z2; dW31 = fa*de*Z3;
dB0 = fa*de;
W11 = W11 + dW11; W21 = W21 + dW21; W31 = W31 + dW31;
B0 = B0 + dB0;

% Ajuste dos pesos da camada escondida.
d = de*(W11 + W21 + W31);
d1 = d*(1 - (2/(1 + exp(-2*Z1in)) - 1)^2);
d2 = d*(1 - (2/(1 + exp(-2*Z2in)) - 1)^2);
d3 = d*(1 - (2/(1 + exp(-2*Z3in)) - 1)^2);
dV11 = fa*d1*X1; dV12 = fa*d2*X1; dV13 = fa*d3*X1;
dB01 = fa*d1; dB02 = fa*d2; dB03 = fa*d3;
V11 = V11 + dV11; V12 = V12 + dV12; V13 = V13 + dV13;
B01 = B01 + dB01; B02 = B02 + dB02; B03 = B03 + dB03;

end

SSE = (sum(sum(E.*E)))/2; % Somatória do erro.

% Testar se o erro está na faixa de tolerância.
if SSE < err_tol, break, end
contdisp = contdisp + 1;
if contdisp >= flagdisp
    contdisp = 0;
    fprintf('Epoca Treinam. = %.0f      Som.Err.Quad.Treinam. = %g\n\n', epoc, SSE);
end

end % Final do treinamento.

% Imprimir número de treinamento e erro obtido.
fprintf('Epoca Treinam. = %.0f      Som.Err.Quad. = %g\n\n', epoc, SSE);

```



```

for n=1:Np % Teste da rede treinada.

    X1 = vX1(n);

    % Processamento dos dados da camada escondida.
    Z1in = B01*C1 + V11*X1;
    Z2in = B02*C1 + V12*X1;
    Z3in = B03*C1 + V13*X1;

    % Funções de ativação da camada escondida.
    Z1 = 2/(1 + exp(-2*Z1in)) - 1;
    Z2 = 2/(1 + exp(-2*Z2in)) - 1;
    Z3 = 2/(1 + exp(-2*Z3in)) - 1;

    % Processamento dos dados da camada de saída.
    O1 = B0*C2 + W11*Z1 + W21*Z2 + W31*Z3;
    Y = O1; % Função de ativação linear.
    VetYr(n) = Y;

end

% Gráficos dos dados de treinamento e do modelo neural obtido.
plot(vX1,vYd,'k.',vX1,VetYr,'k+'); grid;
xlabel('x1'); ylabel('y : "+" = Dados Originais; "." = Valores do Treinamento');
figure

for n=1:Np % Teste da rede treinada.

    Load vX1;
    X1 = vX1;

    % Processamento dos dados da camada escondida.
    Z1in = B01*C1 + V11*X1; % + V21*X2;
    Z2in = B02*C1 + V12*X1; % + V22*X2;
    Z3in = B03*C1 + V13*X1; % + V23*X2;

    % Funções de ativação da camada escondida.
    Z1 = 2/(1 + exp(-2*Z1in)) - 1;
    Z2 = 2/(1 + exp(-2*Z2in)) - 1;
    Z3 = 2/(1 + exp(-2*Z3in)) - 1;

    % Processamento dos dados da camada de saída.
    O1 = B0*C2 + W11*Z1 + W21*Z2 + W31*Z3;
    Y = O1; % Função de ativação linear.
    VetYr(n) = Y;

end

% Gráficos dos dados originais e de valores estimados pela rede.
plot(vX1,vYd,'k.',vX1,VetYr,'k'); grid;
xlabel('x1'); ylabel('y : "+" = Dados Originais; "." = Valores Estimados pela MLP');

```

Anexo 3.2

Código básico do Exemplo 3.2

```

% Exemplo de mapeamento da função XOR via RNA-MLP.

clear all;

% Dados da função XOR.

vX1 = [0;0;1;1]; vX2 = [0;1;0;1]; vYd = [0;1;1;0];
Np=4;

NN = 3; % Número de neurônios.
err_tol = 0.0001; % Tolerância do erro.

% Inicialização aleatória dos pesos da rede neural.
V11 = 2 * rand - 1;
V12 = 2 * rand - 1;
V13 = 2 * rand - 1;
V21 = 2 * rand - 1;
V22 = 2 * rand - 1;
V23 = 2 * rand - 1;
B01 = 2 * rand - 1;
B02 = 2 * rand - 1;
B03 = 2 * rand - 1;

W11 = 2 * rand - 1;
W21 = 2 * rand - 1;
W31 = 2 * rand - 1;
B0 = 2 * rand - 1;

% Valores de offset das entradas da rede neural.
C1 = 1; C2 = 1;

contdisp = 0; % Contador de visualização do erro.
flagdisp = 100; % Variável de impressão do erro.
max_epoc = 200000; % Número máximo de épocas.
fa = 0.005; % Fator de aprendizado.

for epoc = 1:max_epoc % Etapa de treinamento.

    for n=1:Np

        X1 = vX1(n); X2 = vX2(n); Yd = vYd(n);

```



```

% Camada escondida.
Z1in = B01*C1 + V11*X1 + V21*X2;
Z2in = B02*C1 + V12*X1 + V22*X2;
Z3in = B03*C1 + V13*X1 + V23*X2;

% Funções de ativação.
Z1 = 2/(1 + exp(-2*Z1in)) - 1;
Z2 = 2/(1 + exp(-2*Z2in)) - 1;
Z3 = 2/(1 + exp(-2*Z3in)) - 1;
% Camada de saída.
O1 = B0*C2 + W11*Z1 + W21*Z2 + W31*Z3;
Y = O1; % Função de ativação linear.

% Treinamento por retropropagação.
e = Yd - Y; % Erro entre dado original e a saída.
E(n) = e;

% Ajuste dos pesos da camada de saída.
de = e;
dW11 = fa*de*Z1; dW21 = fa*de*Z2; dW31 = fa*de*Z3;
dB0 = fa*de;
W11 = W11 + dW11; W21 = W21 + dW21; W31 = W31 + dW31;
B0 = B0 + dB0;

% Ajuste dos pesos da camada escondida.
d = de*(W11 + W21 + W31);
d1 = d*(1 - (2/(1 + exp(-2*Z1in)) - 1)^2);
d2 = d*(1 - (2/(1 + exp(-2*Z2in)) - 1)^2);
d3 = d*(1 - (2/(1 + exp(-2*Z3in)) - 1)^2);
dV11 = fa*d1*X1; dV12 = fa*d2*X1; dV13 = fa*d3*X1;
dV21 = fa*d1*X2; dV22 = fa*d2*X2; dV23 = fa*d3*X2;
dB01 = fa*d1; dB02 = fa*d2; dB03 = fa*d3;
V11 = V11 + dV11; V12 = V12 + dV12; V13 = V13 + dV13;
V21 = V21 + dV21; V22 = V22 + dV22; V23 = V23 + dV23;
B01 = B01 + dB01; B02 = B02 + dB02; B03 = B03 + dB03;

end

SSE = (sum(sum(E.*E)))/2; % Somatória do erro.

% Testar se o erro está na faixa de tolerância.
if SSE < err_tol, break, end
contdisp = contdisp + 1;
if contdisp >= flagdisp
    contdisp = 0;
    fprintf('Epoca Treinam. = %.0f    Som.Err.Quad.Treinam. = %g\n\n',
        epoc, SSE);
end

end % Final do treinamento.

```

```

% Teste de valores.
X1 = 0; X2 = 1;

% Camada escondida.
Z1in = B01*C1 + V11*X1 + V21*X2;
Z2in = B02*C1 + V12*X1 + V22*X2;
Z3in = B03*C1 + V13*X1 + V23*X2;

% Funções de ativação.
Z1 = 2/(1 + exp(-2*Z1in)) - 1;
Z2 = 2/(1 + exp(-2*Z2in)) - 1;
Z3 = 2/(1 + exp(-2*Z3in)) - 1;

% Camada de saída.
O1 = B0*C2 + W11*Z1 + W21*Z2 + W31*Z3;
Y = O1 % Função de ativação linear.

```

Anexo 3.3

Código básico do Exemplo 3.4

```

% Exemplo de mapeamento da função XOR via rede RBF.

clear all;

% Parâmetros da RBF.
c1=0.25; d1=1; c2=0.75; d2=1;

% Pesos da Camada de Saída da Rede.
.
w0=8.8375; w1=-7.3206; w2=-7.3206;

% Teste da RBF
x = input('x = '); % Valores de Entrada da Rede.
x1=x(1); x2=x(2);

while (x1 >= 0),

    % Funções de Base Radial.
    f1 = exp(-((x1-c1)^2/(d1^2)+(x2-c1)^2/(d1^2)));
    f2 = exp(-((x1-c2)^2/(d2^2)+(x2-c2)^2/(d2^2)));

    y = w0 + w1*f1 + w2*f2; % Modelo de Saída

    y % Valor Estimado pela Rede.

    x = input('x = ');
    x1=x(1); x2=x(2);

end

```

CAPÍTULO

QUATRO

Desenvolvimento de Aplicações Utilizando Redes Neurais

4.1 Introdução

Como citado no capítulo anterior, as redes neurais artificiais são aplicadas em áreas diversas: análise de séries temporais; processamento de sinais; sistemas de controle; classificações de padrões; aproximações de funções; entre outras. As características que uma RNA deve apresentar no contexto das aplicações citadas são: capacidade de generalização; tolerância a ruídos; adaptação em relação aos procedimentos de treinamento ([Palade et al., 1995](#)).

Uma forma geral de estruturar uma rede neural para uma determinada aplicação está ilustrada na [Figura 4.1](#) ([Caudill, 1991](#)).



FIGURA 4.1 Etapas da aplicação de uma rede neural.

Em termos computacionais as redes neurais são desenvolvidas através de softwares exclusivamente voltados para esta finalidade, ou se desenvolve um código específico para o problema em questão utilizando uma linguagem de programação. Existem também pacotes comerciais de simuladores de redes neurais de domínio público ([Braga et al., 2000](#)). Em aplicações embarcadas é possível hardwares disponíveis no mercado.

Este capítulo tem o objetivo de ilustrar as etapas necessárias para o desenvolvimento de aplicações utilizando Redes Neurais Artificiais.

4.2 Definição do problema

Analizando o problema a ser solucionado, deve-se definir o contexto em questão para encontrar as informações que poderão ser utilizadas no modelo neural. Torna-se necessário verificar se realmente a utilização das RNAs irá permitir atingir o objetivo almejado do problema. Em caso afirmativo, deve-se extrair o maior número de informações possíveis para auxiliar no projeto e aprendizado de uma RNA.

4.3 Coleta de dados

Como as RNAs aprendem em cima de exemplos (amostras) do contexto da aplicação, torna-se necessário realizar uma coleta de dados do processo em que as RNAs irão atuar. Deve-se ressaltar que os dados adquiridos devem conter informações de todo o espaço amostral no qual esta rede neural deverá processar, pois a rede neural não consegue obter um bom resultado final se tiver de lidar com padrões totalmente diferentes dos que foram usados para se treinar.

Para que se possa utilizar uma RNA é fundamental que se disponha de dados em quantidade e qualidade suficientes. Caso a quantidade de dados seja pequena, a rede não conseguirá criar um modelo suficientemente representativo e obter um bom desempenho quando aplicada em situações reais. Os dados devem englobar todos os aspectos do problema em questão de modo que o modelo criado seja genérico. Em geral, tais dados precisam ser convertidos em formatos padronizados (normalizados) para utilização pelas RNAs.

4.4 Pré-Processamento dos dados

Após a construção de um banco de dados com amostras do domínio do problema, há a necessidade de se realizar um pré-processamento destes padrões para que possam ser utilizados pela rede neural, com intuito de reduzir a dimensionalidade dos dados. Uma padronização dos dados amostrados para uma determinada faixa na qual a rede neural irá atuar constitui um procedimento usual, bem como a minimização de eventuais ruídos nos dados, e a redução de informações desnecessárias que podem comprometer o processo de aprendizado. Há diversas técnicas de pré

processamento dos dados, como filtrações especiais, normalizações de valores etc., apropriadas a cada tipo de aplicação.

A etapa de normalização de dados tem o objetivo de evitar que valores de maior amplitude exerçam influência sobre a convergência dos algoritmos utilizados na etapa de treinamento da rede, privilegiando determinada variável em detrimento de outras com menor amplitude (mas não menos importantes). Para a obtenção de um bom desempenho das RNAs, geralmente, trabalham com dados num intervalo definido, na maioria das vezes na faixa de $[0,1]$ ou $[-1,1]$. A equação (4.1) mostra o procedimento de normalização no intervalo $[0,1]$, enquanto a equação (4.2) trata da normalização no intervalo $[-1,1]$.

$$Z'(t) = \frac{Z(t) - \min(Z)}{\max(Z) - \min(Z)} \quad (4.1)$$

$$Z'(t) = \frac{Z(t) - \max(Z) - \min(Z)}{\max(Z) - \min(Z)} \quad (4.2)$$

4.5 Separação em conjunto de dados

Sob a perspectiva de aprendizagem supervisionada, o conjunto de padrões disponíveis para um dado problema pode ser separado em subconjuntos que contribuem ainda mais para o aumento da capacidade de generalização da rede. Assim, o conjunto original de dados é subdividido em três subconjuntos: treinamento; validação; teste. Esta subdivisão do conjunto de dados original em três subconjuntos é muito comum em aplicações de inteligência artificial, principalmente em algoritmos que requerem algum tipo de treinamento supervisionado e aprendizagem. Essa divisão obedece a uma porcentagem para cada subconjunto. O conjunto de treinamento deve reunir o maior número possível de padrões, sem prejudicar, obviamente, os outros subconjuntos, de forma a representar convenientemente o universo das observações e permitir a generalização do aprendizado. O conjunto de validação é utilizado para verificar a eficiência da rede quanto a sua capacidade de generalização durante

o treinamento, e também pode ser empregado como critério de parada da etapa de treinamento.

Após a separação de conjuntos, as informações destes são geralmente organizadas em ordem aleatória, para prevenção de tendências associadas à ordem de apresentação dos dados na etapa de treinamento da rede considerada.

4.6 Configuração da rede

Depois da realização do pré-processamento, os dados estão prontos para serem apresentados à rede neural considerada para a etapa de treinamento. Esse processo pode ser trabalhoso e não segue uma receita fixa. Essa etapa determina o comportamento posterior da rede, conforme os padrões de entrada e de validação apresentados nas etapas anteriores. A definição da configuração da rede pode ser dividida em três passos:

- seleção do paradigma neural (tipo de rede) apropriado à aplicação;
- determinação da topologia da rede a ser utilizada, por exemplo: o número de camadas; o número de unidades em cada camada etc.;
- determinação de parâmetros do algoritmo de treinamento e funções de ativação utilizadas.

Esses passos têm impacto no desempenho do sistema resultante, pois com uma escolha adequada desses parâmetros a rede terá boas chances de convergência no treinamento e capacidade de generalização. Muitas vezes a definição dessas configurações é realizada de forma empírica.

4.7 Treinamento da rede

Depois da configuração de todos os parâmetros da rede neural considerada inicia-se o processo de treinamento, a fim de que ela possa “aprender”. O aprendizado é, em geral, um processo gradual e iterativo, em que os pesos das conexões são modificados várias vezes, gradualmente, seguindo-se uma regra de aprendizado (algoritmo) que estabelece como esses pesos são alterados. O aprendizado é realizado utilizando-se um conjunto de dados de aprendizado disponível (base de exemplos). Para cada iteração desse processo gradativo de adaptação dos pesos de uma rede neural, é apresentado o conjunto de dados de treinamento, em que ela recebe a denominação de época de aprendizado. É

importante considerar, nessa fase, alguns aspectos como a inicialização da rede, o modo de treinamento e o tempo de treinamento. Uma boa escolha dos valores iniciais dos pesos da rede pode diminuir o tempo necessário para o treinamento. Normalmente, os valores iniciais dos pesos da rede são números aleatórios uniformemente distribuídos, em um intervalo definido. A escolha inadequada destes pesos pode levar a uma saturação prematura do processo de treinamento. Quanto ao modo de treinamento, na prática é mais utilizado o modo-padrão sequencial, que exige menor armazenamento de dados nos processamentos correspondentes, além de ser menos suscetível ao problema de mínimos locais no procedimento de minimização do erro do modelo neural. Por outro lado, no modo *batch*, se tem melhor estimativa do vetor gradiente do procedimento de aprendizado supervisionado, o que torna o procedimento computacional da etapa de treinamento mais estável. A eficiência relativa dos dois modos de treinamento depende do problema que está sendo tratado.

Uma rede pode se especializar demasiadamente em relação aos exemplos contidos na base de aprendizado. Esse tipo de comportamento gera um problema de aprendizado conhecido como superaprendizado (*over-training* ou *over-fitting*). Normalmente, o *over-fitting* pode ser detectado e evitado por meio do uso de um teste de generalização por validação cruzada (*cross validation*).

Quanto ao tempo de treinamento, vários fatores podem influenciar a sua duração, porém sempre será necessário utilizar algum critério de parada. O critério de parada do algoritmo *backpropagation* não é único, e geralmente é utilizado um número máximo de ciclos, no qual deve ser considerada a taxa de erro médio por ciclo e a capacidade de generalização da rede. Pode ocorrer que em determinado instante do treinamento a generalização comece a degenerar, causando *over-training*.

O treinamento deve ser interrompido quando a rede apresentar uma boa capacidade de generalização e quando a taxa de erro for suficientemente pequena, ou seja, menor que um erro admissível. Assim, deve-se encontrar um ponto ótimo de parada com erro mínimo e capacidade de generalização máxima.

4.8 Teste da rede

Após o processo de treinamento e aprendizagem da rede, todo o conhecimento está armazenado nas conexões sinápticas do modelo empregado. Há a necessidade de se verificar a capacidade de generalização da rede neural realizando o processo de validação e teste. Nesta etapa são apresentados à rede padrões nunca vistos durante o processo de treinamento. A rede neural terá de classificar corretamente esses padrões para a confirmação de um aprendizado satisfatório. Em caso positivo, avança-se para a próxima etapa: integração da aplicação. Se o desempenho não for adequado, deve-se retornar à etapa anterior e refazer parte do processo, como, por exemplo, alterar a topologia da rede, o algoritmo de aprendizagem, ou, eventualmente, o próprio conjunto de dados. Devem ser considerados ainda outros testes como análise do comportamento da rede, utilizando entradas especiais e análise dos pesos da rede, pois se existirem valores muito pequenos, as conexões associadas podem ser consideradas insignificantes e, assim, serem eliminadas (*prunning*). De modo inverso, valores substancialmente maiores que os outros poderiam indicar que houve *over-training* da rede.

4.9 Integração da Rede ao Sistema e Manutenção

Finalmente, com a rede treinada e validada, ela pode ser integrada ao sistema computacional da aplicação. Para melhor eficiência da solução, este sistema deverá conter facilidades de utilização, como interfaces convenientes, e facilidades de aquisição de dados, através de planilhas eletrônicas, interfaces com unidades de processamento de dados, ou arquivos padronizados. Uma boa documentação do sistema e o treinamento de usuários são necessários para o sucesso do projeto final da aplicação pretendida.

Além disso, o sistema deve periodicamente monitorar seu desempenho e realizar manutenções ou atualizações, quando for necessário. Outras melhorias poderão ainda ser sugeridas, quando os usuários forem se tornando mais familiares com o sistema. Essas sugestões poderão ser muito úteis em novas versões ou em novos produtos relacionados com o projeto inicial do sistema correspondente.

Se as etapas anteriores apresentaram sucesso, o modelo resultante pode ser realizado para uso contínuo, lembrando que todo sistema computacional está sempre sujeito a alterações e melhorias contínuas.

Exemplo de implantação

Para exemplificar o desenvolvimento de aplicação utilizando uma rede neural, tomemos como base um problema de previsão de séries temporais. O desenvolvimento de técnicas da área de IA tem fornecido ferramentas eficientes para modelagem de séries temporais, na qual se destacam as modelagens baseadas em redes neurais artificiais.

Segundo [Guirelli et al. \(2004\)](#), os métodos de modelagem baseados em técnicas de IA apresentam problemas próprios. A codificação de conhecimentos nem sempre é uma tarefa trivial, e questões de como determinar quais dados e variáveis são importantes para um processo de modelagem constituem um procedimento que pode ser tão complexo quanto a utilização de modelos clássicos ou estatísticos.

Neste exemplo relacionado com a previsão de carga elétrica, os dados utilizados foram extraídos da competição organizada pela *European Network of Intelligent Technologist* (EUNITE), referência na área de previsão de séries temporais abordando o tema sobre previsão de cargas elétricas em 2004 ([Rojas, 2004](#)). Nessa competição, dadas as temperaturas e as cargas elétricas nos anos de 1997 e 1998 de uma determinada região, os competidores estimavam as previsões das cargas elétricas para o mês de janeiro de 1999. Os dados da competição foram fornecidos pela Corporação Elétrica Oriental Eslovaca (*Eastern Slovakian Electricity Corporation*). Esta instituição tem grande interesse em aplicações de tecnologias inteligentes e adaptativas em previsões de cargas elétricas.

A série de carga investigada representa a demanda máxima diária da carga consumida por um sistema elétrico. Esses dados compreendem valores de pico de carga no período de 1997 a 1998, cujos os dados estão ilustrados na [Figura 4.2](#). O objetivo é fornecer uma previsão de valores máximos diários de demanda de carga para o mês de janeiro de 1999, com a estimativa de 31 valores diários referentes a este mês.



FIGURA 4.2 Série temporal diária de carga de janeiro de 1997 a dezembro de 1998.

A série coletada contém informações de cargas elétricas medidas durante todo um dia, onde cada observação corresponde a um horário específico. A série contém valores de cargas elétricas diárias dos anos de 1997 e 1998. Os horários utilizados como intervalos entre as observações apresentam-se com variação de meia hora durante todo um dia, iniciando à 00h30 e terminando às 24h. Um pré-processamento de dados foi realizado por intermédio da média dos valores dos horários de pico, gerando-se os valores diários utilizados no estudo. Para estabelecer um modelo baseado em redes neurais realizou-se a normalização dos valores das observações da série temporal, através da expressão (4.3).

$$Norm = \frac{mtz}{\max | \max(mtz) |} \quad (4.3)$$

A partir dos correlogramas extraídos da série pode-se identificar a existência de sazonalidade nos dados, que pode ter sido causada pela influência de fatores climáticos ou devido a hábitos semanais dos consumidores. Portanto, o conjunto de dados deve ser tratado antes de ser apresentado ao treinamento da rede.

A organização desses dados é preparada para a previsão diária. Desta forma, para prever determinado dia, inclui-se nos padrões de entrada da rede o conjunto de valores das observações da série agora estacionária, definindo a janela temporal de análise, ou seja, o número de amostras do passado a ser incluído para a previsão à frente. As redes neurais MLPs utilizadas em previsão de séries temporais têm um conjunto de entrada de valores passados ordenados no tempo e um valor de saída que consiste no próximo valor na sequência de dados da série. As amostras da rede neural terá a estrutura descrita por (4.4).

$$Z^t = f(Z(t-1), \dots, Z(t-p)) \quad (4.4)$$

Onde $Z(t)$ é o ponto da série correspondente ao tempo t , p é o tamanho da janela de tempo e Z_t é a amostra a ser prevista. Para um modelo com n amostras utiliza-se como base os valores das amostras $Z(1)$, $Z(2)$, $Z(3)$, $Z(4)$, $Z(5)$..., $Z(n)$. Durante o treinamento, a ideia é fornecer como entradas valores sucessivos da série, por exemplo, os valores de $Z(1)$, $Z(2)$, $Z(3)$, $Z(4)$ $Z(5)$ $Z(6)$ e $Z(7)$, e definir a saída desejada para $Z(8)$. Na segunda iteração, os valores das entradas seriam $Z(2)$, $Z(3)$, $Z(4)$, $Z(5)$ $Z(6)$ $Z(7)$, $Z(8)$, e a saída desejada $Z(9)$, e assim sucessivamente para o restante dos pontos da série ([Tabela 4.1](#)).

Tabela 4.1

Entradas e saídas para o treinamento de MLPs com janela de sete valores

Entradas	Saída desejada
$Z(1), Z(2), Z(3), Z(4), Z(5), Z(6), Z(7)$	$Z(8)$
$Z(2), Z(3), Z(4), Z(5), Z(6), Z(7), Z(8)$	$Z(9)$
$Z(3), Z(4), Z(5), Z(6), Z(7), Z(8), Z(9)$	$Z(10)$
$Z(4), Z(5), Z(6), Z(7), Z(8), Z(9), Z(10)$	$Z(11)$
...	...
$Z(n-7), Z(n-6), \dots Z(n-1)$	$Z(n)$

O número de valores passados, usados como entrada, é chamado de ordem de linha, atraso ou janela. No exemplo citado usou-se sete valores para prever o próximo, portanto, a janela é de ordem sete (a previsão é realizada considerando sete dias, sendo o oitavo dia previsto). A melhor janela a ser utilizada vem de estudos prévios, como as informações oriundas do correlograma da série original. Além disso, variáveis exógenas, como temperatura, no caso de séries de carga em sistemas elétricos, são consideradas, podendo ser relevantes em sistemas de previsão.

A próxima etapa no desenvolvimento da aplicação de uma rede neural é a divisão de conjuntos de dados em subconjuntos de treinamento, validação e teste (que é efetuada obedecendo a uma porcentagem para cada subconjunto). A [Figura 4.3](#) apresenta a porcentagem utilizada sobre o conjunto original de dados considerado neste exemplo.

Treinamento 70%	Validação 25%	Teste 5%
-----------------	---------------	----------

FIGURA 4.3 Exemplo da divisão de dados na metodologia utilizada.

Em processos com comportamentos não lineares o modelo de rede neural mais indicado é o Perceptron de Múltiplas Camadas ou topologia MLP, pois se trata de um interpolador universal de funções (Haykin, 2007). As topologias MLPs adotadas representam os modelos de redes neurais mais conhecidos e utilizados. Uma MLP consiste de n nós de entrada, h camadas de nós intermediários ou escondidos e m nós de saída, e todas as camadas são interconectadas pelos seus pesos sinápticos. A MLP deve ser treinada com dados históricos adequados, objetivando encontrar os valores apropriados dos pesos. O número de neurônios requeridos na camada oculta pode ser estabelecido empiricamente, ou a partir de estudos prévios sobre a complexidade do problema a ser analisado.

Neste exemplo será considerada uma rede neural MLP com um algoritmo de treinamento SCGM (*Scaled Conjugate Gradient Modified*). A Tabela 4.2 exibe alguns dados utilizados inicialmente na busca dos parâmetros para a topologia da rede.

Tabela 4.2

Parâmetros da rede

Número de Camada	[2-3]
Número de Neurônios	[1 - 10]
Taxa Aprendizagem	[0,0001 - 0,001]
Coeficiente <i>Momentum</i>	[0,1 - 0,9]
Função de Ativação	[Tan Hiperbólica, Logística Sigmoidal]

Para obtenção do modelo de rede neural foram utilizadas duzentas épocas de treinamento. A Figura 4.4 apresenta o gráfico com os dados originais (+) e os valores resultantes (*) do treinamento da rede referente à série de dados de carga da competição Eunite.

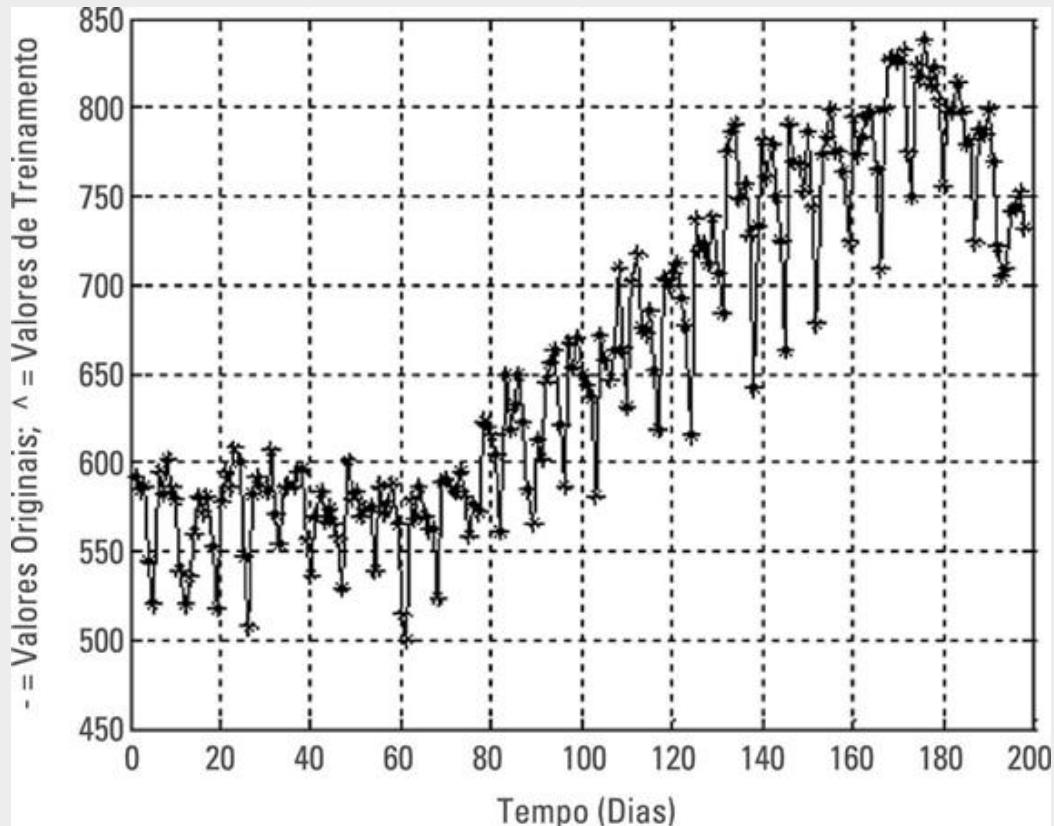


FIGURA 4.4 Resultado do treinamento da rede.

Foi utilizado como critério de comparação dos dados obtidos pelo modelo neural em relação aos dados originais da série o erro percentual absoluto médio (*Mean Absolute Percentage Error* – Mape). O valor do Mape (4.6) foi calculado a partir das previsões de um passo à frente definido por cada modelo gerado (Stok e Watson, 2004).

$$MAPE(\%) = \frac{1}{N} \sum_{i=1}^n \left| \frac{Y'_i - Y_i}{Y_i} \right| \times 100 \quad (4.5)$$

Onde:

Y'_i = representa o conjunto de valores previstos;

Y_i = representa os valores reais, ou seja, o valor atual da série;

n = a quantidade de amostras utilizadas na previsão.

O Mape obtido com os dados do teste da rede foi de 0,09%, e a [Figura 4.5](#) contém os dados originais e os valores resultantes do teste da rede neural em questão. Neste exemplo a rede neural classificou corretamente os padrões apresentados, confirmando um bom aprendizado com os parâmetros escolhidos para a MLP utilizada. O tratamento inicial da série temporal retirando-se tendências e sazonalidades, também constitui um fator determinante para os bons resultados obtidos.



FIGURA 4.5 Teste da capacidade de estimação da rede.

Referências bibliográficas

1. BRAGA AP, CARVALHO APLF, LUDERMIR TB. *Redes Neurais Artificiais: teoria e aplicações*. Rio de Janeiro: Livros Técnicos e Científicos; 2000.
2. CAUDILL M. Expert Network. *Byte*, out 1991:108–116.
3. GUIRELLI CR, JARDINI JA, MAGRINI LC, YASUOKA J, CAMPOS AC, BASTOS MR. In: *Tool for short-term load forecasting in transmission systems based on artificial intelligence techniques*. Latin America IEEE/PES'04. São Paulo: In: Transmission and Distribution Conference and Exposition; 2004:243–248.
4. HAYKIN S. *Redes neurais: princípios e prática*. 2ª ed. Porto Alegre: Artmed Editora S.A; 2007.
5. PALADE V, BOCANIALA CD, JAIN LC. *Computational intelligence in fault diagnosis*. Nova York: Springer; 1995.
6. ROJAS HP. In: *Soft-computing techniques for time series forecasting* Bruges: 12th European Symposium On Artificial Neural Networks

(ESANN); 2004:93–102.

7. STOCK JH, WATSON MW. *Econometria*. Addison-Wesley 2004.

OceanofPDF.com

CAPÍTULO

SEIS

Conjuntos Aproximados (*Rough Sets*)

6.1 Introdução

O processamento de informações na presença de incertezas constitui um procedimento de grande importância em vários campos de aplicações, como em sistemas computacionais que empregam técnicas de Inteligência Artificial e neste contexto entende-se por incerteza, informações incompletas e/ou imprecisas. Na prática, as informações de sistemas reais são frequentemente incertas, imprecisas ou incompletas, e algumas metodologias foram desenvolvidas para tratar essas questões, entre as quais a Teoria de Dempster Shafer (Dempster, 1967; Shafer, 1976) e a Lógica *Fuzzy* (Zadeh, 1973).

A Teoria dos Conjuntos Aproximados (TCA) foi proposta por Pawlak (1982) como uma alternativa para o processamento de informações que contenham incertezas ou imprecisões em seus dados, principalmente em aplicações relacionadas a sistemas de informações. A capacidade de identificar uma determinada quantidade de informação e extrair um conhecimento associado se constitui em uma característica inerente ao ser humano e à sua capacidade natural de aprendizado. A realização automática dessa tarefa por meios computacionais pode constituir um procedimento complexo, principalmente quando as informações estão desorganizadas, incompletas ou apresentam ambiguidades. Neste contexto, a TCA é empregada para facilitar a transformação automática de dados em conhecimento (Pawlak, 1991), em que as representações computacionais correspondentes estão baseadas em conjuntos de atributos que caracterizam regras de decisão em formatos similares aos utilizados em sistemas especialistas.

A TCA é considerada uma extensão da teoria clássica dos conjuntos (Pawlak e Skowron, 2007a), e uma das principais vantagens dessa teoria é

poder representar as similaridades conceituais entre os dados de um determinado sistema de informação, agrupando valores que são conceitualmente similares ou equivalentes. Valores que pertencem a um mesmo grupo são considerados **indiscerníveis** e, assim, é possível levar em consideração eventuais relações nos dados, e não apenas tratar os seus valores de forma isolada. Outra vantagem desta teoria é que não se necessita de qualquer informação adicional ou preliminar a respeito dos dados a serem processados, tais como distribuição de probabilidade, atribuição de valor de crença, ou grau de possibilidade (Pawlak e Skowron, 2007b).

Os conceitos relacionados aos Conjuntos Aproximados (*Rough Sets*) são aplicados em áreas diversas como em aprendizado de máquina, sistemas especialistas, reconhecimento de padrões, sistemas de decisão, mineração de dados, em automação de diagnósticos (Hvidsten, 1999), processamento de imagens e muitas outras (Pawlak e Skowron, 2007a).

Em aplicações de aprendizado de máquina, os sistemas computacionais utilizados devem ser capazes de exibir “**comportamentos inteligentes**”, em que o conhecimento relacionado com o domínio do problema abordado deve ser apropriadamente adquirido e incorporado aos sistemas em questão. A aquisição de conhecimento pode ser realizada de maneira explícita ou implícita. A forma explícita está associada às informações fornecidas por especialistas humanos. A maneira implícita é mais complexa em termos computacionais, pois o conhecimento deve ser extraído a partir de informações contidas em bancos de dados genéricos, não sendo fornecidas diretamente por um especialista no domínio de interesse do problema em questão.

A realização da aquisição de conhecimento implícito é uma tarefa relacionada a uma área da IA conhecida como **Aprendizado de Máquina (AM)**, que emprega métodos computacionais relacionados à obtenção e à organização automática de conhecimentos (Mitchell, 1997). Os procedimentos de aprendizado envolvidos neste contexto são classificados como supervisionados ou não supervisionados. No aprendizado não supervisionado o objetivo é encontrar agrupamentos de padrões que sejam similares, e identificar potenciais classes nos dados processados, as quais estejam relacionadas ao problema ou ao exemplo abordado.

No aprendizado supervisionado, o objetivo consiste em extrair conhecimentos de exemplos previamente rotulados em determinadas classes

com a finalidade de prever a qual classe pertence eventuais novos exemplos. Nesse tipo de aprendizado, os exemplos possuem características denominadas “atributos”, que estão associadas aos dados do sistema em questão. Um *AM* supervisionado pode ser descrito por exemplos compostos por n atributos $x = (x_1, x_2, \dots, x_n)$, e pela classe associada a cada exemplo y , cuja tarefa consiste em obter um mapeamento f , tal que $y = f(x)$. Se os atributos representam as propriedades essenciais dos exemplos, o sistema de *AM* pode representá-los em uma estrutura capaz de generalizar o conhecimento implicitamente associado aos exemplos.

Em geral, os sistemas de *AM* computacionalmente factíveis não operam bem na presença de grande quantidade de atributos, e, neste contexto, a aplicação da *TCA* é particularmente apropriada na obtenção de atributos em bancos de dados, fornecendo uma abordagem sistemática na descoberta de conhecimentos e na sua representação de forma compacta e eficiente. Em [Pili \(2001\)](#) são encontrados detalhes relacionados à seleção de atributos relevante para o aprendizado de máquina, utilizando a abordagem dos Conjuntos Aproximados. O artigo de [Shen e Chouchoulas \(2001\)](#) aborda a questão da redução de dimensionalidade em processos de aprendizagem (supervisionado ou não), utilizando procedimentos baseados na *TCA*.

Os próximos itens deste capítulo contêm as definições básicas sobre Conjuntos Aproximados, as relações com sistemas de informações e exemplos de aplicações.

6.2 Definições básicas

Um espaço aproximado é definido por $S = (U, A)$, onde U é um conjunto de exemplos, objetos ou observações (o_i) chamado de universo, e A é um conjunto de atributos de condições (a_j). Para um Sistema de Informação (*SI*) representado por uma tabela de atributos-valores do gênero “objeto = condições + decisão”, a representação típica é dada por $SI = [U, A, V, f]$, na qual se determinam classificações $f: U \times A \rightarrow V$, onde V constitui um conjunto de valores de atributos. A representação tabular genérica de um *SI* está ilustrada na [Tabela 6.1](#), onde os valores dos atributos de decisão estão incorporados na coluna d da tabela.

Tabela 6.1

Representação tabular genérica de um SI

	a_1	...	a_j	...	a_n	d
o_1	$f(o_1, a_1)$		$f(o_1, a_j)$		$f(o_1, a_n)$	$f(o_1, d_1)$
$\vdots$	$\vdots$		$\vdots$		$\vdots$	$\vdots$
o_i	$f(o_i, a_1)$		$f(o_i, a_j)$		$f(o_i, a_n)$	$f(o_i, d_1)$
$\vdots$	$\vdots$		$\vdots$		$\vdots$	$\vdots$
o_m	$f(o_m, a_1)$		$f(o_m, a_j)$		$f(o_m, a_n)$	$f(o_m, d_m)$

Tabela 6.2**Sistema de informação relativo ao Exemplo 6.1**

MBA GRAD EXP CONTRAT				
o_1	N	A	S	Maior
o_2	N	A	S	Maior
o_3	S	A	S	Não
o_4	N	C	N	Baixa
o_5	N	C	N	Média

Em conjuntos aproximados trabalha-se geralmente com valores discretos. Para atributos numéricos é necessário aplicar algum processo de discretização para torná-los nominais (Risvik, 1999). Algumas abordagens foram propostas para minimizar eventuais efeitos da quantização nos dados de um SI (Nguyen e Skowron, 1995; Carvalho, 2010).

Um conceito importante na TCA é o conceito da relação de não discernimento ou indiscernibilidade. Se essa relação existe entre dois objetos então, tem-se um conjunto de atributos associados, e se os valores desses atributos são idênticos entre si, não se distingue um objeto do outro. Para cada subconjunto de atributos $B \subseteq A$, uma relação de equivalência $IND(B)$ é associada e recebe o nome de relação de não discernimento, sendo definida pela expressão (6.1).

$$IND(B) = \{(o_i, o_j) \in U^2 \mid \forall a_k \in B, f(o_i, a_k) = f(o_j, a_k)\} \quad (6.1)$$

Um atributo a_k é chamado dispensável em B se $IND(B) = IND(B - \{a_k\})$ caso contrário, é indispensável. O conjunto de todas as classes de equivalência determinadas por $IND(B)$ é representado pela notação $U/IND(B)$. Das classes de equivalência emergem dois outros conceitos importantes: a aproximação inferior e a aproximação superior. Sendo O um conjunto de observações tal que $O \subseteq U$, define-se a aproximação inferior B_* dada pela equação (6.2), que representa o conjunto de possíveis exemplos ou objetos que podem ser classificados com certeza como membros de O para o correspondente conjunto de atributos B .

$$B_*(O) = \{o \in U \mid IND(B) \subseteq O\} \quad (6.2)$$

A aproximação superior B^* é definida pela equação (6.3), que indica a possibilidade dos elementos de O fazerem parte da classificação em questão.

$$B^*(O) = \{o \in U \mid IND(B) \cap O \neq \Phi\} \quad (6.3)$$

Um conjunto O de objetos ou exemplos é denominado preciso (*crisp*) se $B_*(O) = B^*(O)$, caso contrário, ele é definido como impreciso, grosseiro (*rough* ou aproximado, lembrando que todos os elementos do conjunto de aproximação inferior fazem parte do conjunto de aproximação superior. Os objetos ou observações que não pertencem ao conjunto dos elementos que caracteriza a aproximação superior (que engloba também a aproximação inferior) são denominados fora da região, sendo essa definida por (6.4). A região conhecida como de borda ou de fronteira é dada pela diferença da aproximação superior pela inferior.

$$FR(O) = U - B^*(O) \quad (6.4)$$

As aproximações citadas têm métricas definidas em termos dos próprios elementos que as definem. O coeficiente de imprecisão $\alpha_B(O)$ representa a qualidade da aproximação em relação aos elementos de O , e é expresso por

(6.5), onde $|B_*(O)|$ e $|B^*(O)|$ denotam a cardinalidade da aproximação inferior e superior, respectivamente. O valor de $\alpha_B(O)$ varia no intervalo $[0, 1]$, e quanto mais próximo de um é este valor, mais preciso é O em relação ao conjunto de atributos de B . Por outro lado, quanto mais próximo de zero é o valor, mais impreciso (*rough*) é O em relação ao conjunto de atributos de B .

$$\alpha_B(O) = \frac{|B_*(O)|}{|B^*(O)|} \quad (6.5)$$

O coeficiente de qualidade da aproximação superior $\alpha_B(B^*(O))$, definido por (6.6), é interpretado como sendo o percentual de todos os objetos possivelmente classificados como pertencentes a O , sendo $|U|$ a cardinalidade do conjunto de objetos do sistema de informação em questão.

$$\alpha_B(B^*(O)) = \frac{|B^*(O)|}{|U|} \quad (6.6)$$

O coeficiente de qualidade da aproximação inferior $\alpha_B(B_*(O))$, definido por (6.7), é interpretado como sendo o percentual de todos os objetos certamente classificados como pertencentes ao conjunto O .

$$\alpha_B(B_*(O)) = \frac{|B_*(O)|}{|U|} \quad (6.7)$$

O próximo item traz os procedimentos necessários para processar os dados de um sistema de informação, objetivando extrair as informações mínimas necessárias para representá-lo de forma concisa, mesmo que na presença de imprecisões e ambiguidades.

6.3 Matriz e função de discernibilidade, redutos e regras de decisão

Apresentados os conceitos básicos relacionados aos Conjuntos Aproximados este item apresenta os procedimentos necessários para a extração de conhecimentos (ou de regras de decisão) de um conjunto de dados relacionados a um sistema de informação genérico. Para este propósito definiu-se a chamada matriz de discernibilidade (6.8), denotada por $M_D(B)$. Essa matriz é simétrica e de dimensão $n \times n$, cujos elementos são dados por (6.9). Os elementos $m_D(i,j)$ da matriz de discernibilidade constituem o conjunto de atributos condicionais de B , que diferenciam os objetos das classes de um SI com relação aos seus valores nominais.

$$M_D(B) = [m_D(i, j)]_{n \times n}, i \geq 1, j \leq \text{card}(U/IND(B)) \quad (6.8)$$

$$m_D(i, j) = \{a_k \in B \mid f(o_i, a_k) \neq f(o_j, a_k)\} \quad (6.9)$$

Com as informações de (6.8) e (6.9) é composta a função de discernibilidade $F(B)$ dada por (6.10), cujos elementos são definidos por (6.11). Essa função booleana determina o conjunto mínimo de atributos necessários para diferenciar qualquer classe de equivalência para o sistema de informação considerado. A função $F(B)$ é obtida da seguinte forma: entre os atributos contidos dentro de cada célula da matriz de discernibilidade aplica-se o operador lógico “V” ou “OR”; e entre os conteúdos das células dessa matriz utiliza-se o operador “^” ou “AND”. A expressão booleana resultante deve ser simplificada com a utilização de teoremas, propriedades e postulados da Álgebra Booleana, obtendo-se assim uma expressão na forma de “produto de soma”, que contém as informações dos chamados redutos do sistema.

$$F(B) = \wedge \{ \vee \bar{m}_D(i, j) \} \quad (6.10)$$

$$\bar{m}_D(i, j) = \{\bar{a}_k \mid a_k \in m_D(i, j)\} \quad (6.11)$$

O conjunto formado pelo termo mínimo de $F(B)$ determina os chamados redutos de B . Reduto é um conjunto de atributos mínimos necessários para manter as mesmas propriedades de um SI que utiliza todos os atributos originais do sistema considerado. Pode existir mais de um reduto para um mesmo conjunto de atributos. A obtenção dos redutos mínimos de um SI de dimensão elevada, geralmente consiste de um problema de complexidade computacional crescente com o volume de dados do sistema. Algumas abordagens são utilizadas para tratar esse tipo de problema no processamento de redutos, por exemplo, por intermédio de relações de similaridade (Huang *et al.*, 2007), ou por outras técnicas de IA como Colônia de Formigas (Quana e Biao, 2008).

Para transformar um reduto em regra de decisão, basta agregar os valores dos atributos condicionais da classe de objetos da qual foi originado o reduto com os seus atributos correspondentes, e depois completar a regra com os atributos de decisão. Para um determinado reduto, um exemplo de regras de decisão pode ser expresso por (6.12). A utilização da TCA possibilita, de modo sistemático, que as regras de decisão resultantes apresentem informações concisas em relação a um determinado SI , tratando adequadamente eventuais redundâncias, incertezas, ou imprecisões presentes nos dados do sistema de informação considerado.

$$\begin{aligned} &IF\ a_1 = f(o_1, a_1) AND \dots AND\ a_k = f(o_m, a_k) \\ &THEN\ d_1 = f(o_1, d_1) OR \dots OR\ f(o_i, d_i) \end{aligned} \quad (6.12)$$

Regras que têm os mesmos atributos de condição e diferentes atributos de decisão são chamadas de inconsistentes ou conflitantes, caso contrário, são denominadas de consistentes. Regras consistentes são conhecidas como regras certas, pois determinam decisões únicas a respeito de seus atributos de condição. Regras inconsistentes são chamadas de regras possíveis, pois algumas delas podem levar a decisões certas.

Existem ferramentas computacionais disponíveis, muitas de acesso livre para processamento de sistemas de informações por intermédio da TCA , entre

as quais: RSL (Rough Sets Library); Rough Enough; CI (Column Importance facility); Rosetta.

O próximo item deste capítulo traz alguns exemplos de aplicação da Teoria dos Conjuntos Aproximados.

6.4 Exemplos de aplicação

Apresentada a conceituação básica sobre Conjuntos Aproximados, os exemplos a seguir mostrarão a aplicação da *TCA* em alguns contextos relacionados com a extração de conhecimentos em sistemas de diagnóstico ou decisão, modelagem de funções e sistemas de controle.

Exemplo 6.1

Este exemplo básico ilustra um sistema de informação simples representado pela [Tabela 6.2](#). Essa tabela contém observações que podem eventualmente definir possíveis informações que indiquem condições mais favoráveis para contratações de profissionais por uma determinada empresa. Os seguintes dados e atributos foram considerados no contexto em questão: que os candidatos tenham (S) ou não (N) alguma experiência profissional (EXP); possuam ou não alguma especialização (MBA, por exemplo); graduaram-se (GRAD) em universidades com mais (A) ou com menos (C) prestígio; e as contratações (Contrat) correspondentes realizadas (Maior, Média, Baixa e Não). O problema consiste em determinar quais são os atributos relevantes para uma eventual contratação.

Considerando o conjunto $B = \{\text{MBA, GRA, EXP}\}$ de todos os possíveis atributos do *SI* em questão, o conjunto de todas as classes de equivalência determinadas por $IND(B)$ é:

$$U/IND(B) = \{\{o_1, o_2\}, \{o_3\}, \{o_4, o_5\}\}.$$

Para gerar a matriz de discernibilidade do sistema, é montada a tabela a seguir, contendo apenas os exemplos distintos (e_1, e_2, e_3) referentes aos atributos de condição do *SI* em questão.

A partir da [Tabela 6.3](#), é definida a matriz de discernibilidade, cujo conteúdo de cada célula dessa matriz está ilustrado na [Tabela 6.4](#).

Tabela 6.3

Exemplos distintos relativos à [Tabela 6.2](#)

	MBA	GRAD	EXP
e_1	N	A	S
e_2	S	A	S
e_3	N	C	N

Tabela 6.4

Matriz de discernibilidade associada ao [Exemplo 6.1](#)

	e_1	e_2	e_3
e_1	-		
e_2	MBA	-	
e_3	GRAD, EXP	MBA, GRAD, EXP	-

Tabela 6.5**Sistema de informação associado ao Exemplo 6.2**

	Ex1	Ex2	Ex3	Ex4	Diagnósticos
o_1	1	P	2	> 32	D1
o_2	2	P	3	12-23	D2
o_3	4	N	3	24-32	D3
o_4	1	N	2	< 12	D1
o_5	3	N	1	24-32	D2
o_6	1	N	4	< 12	D1

A função de discernibilidade resultante é dada por:

$$F(B) = MBA \wedge (GRAD \vee EXP) \wedge (MBA \vee GRAD \vee EXP)$$

Simplificando a expressão anterior, tem-se:

$$F(B) = MBA \vee (GRAD \vee EXP)$$

Expressando $F(B)$ na forma “soma do produto”, tem-se:

$$F(B) = (MBA \wedge GRAD) \vee (MBA \wedge EXP)$$

Assim, têm-se os redutos correspondentes $\{\{MBA, GRAD\}, \{MBA, EXP\}\}$ do *SI* considerado. Para o reduto $\{MBA, GRAD\}$ têm-se as seguintes regras de decisão:

IF MBA = N AND GRAD = A THEN CONTRAT = Maior;
IF MBA = S AND GRAD = A THEN CONTRAT = Não;
IF MBA = N AND GRAD = C THEN CONTRAT = Baixa OR Média.

Para o reduto $\{MBA, EXP\}$, seguem as regras de decisão correspondentes:

IF MBA = N AND EXP = S THEN CONTRAT = Maior;
IF MBA = S AND EXP = S THEN CONTRAT = Não;
IF MBA = N AND EXP = N THEN CONTRAT = Baixa OR Média.

Conclusão referente aos conhecimentos relativos às regras de decisão geradas: As melhores chances de contratação estão associadas aos candidatos formados em universidades de mais prestígio, ou então com experiência profissional. No [Anexo 6.1](#) encontra-se a resolução desse exemplo, através da ferramenta de acesso livre Rosetta (Ohrn e Komorowski, 1997).

Exemplo 6.2

Este exemplo está relacionado com a determinação da menor combinação de exames clínicos necessários para classificar os diagnósticos que estão sendo avaliados em um experimento. A [Tabela 6.5](#) contém observações de certos exames clínicos (Ex1, Ex2, Ex3 e Ex4) que podem classificar três tipos de diagnósticos (D1, D2 e D3). As informações dos exames são classificadas como positiva (P) ou negativa (N), e em classes de valores como 1, 2, 3 e 4, ou < 12, 12-23, 24-32 e > 32.

Em Conjuntos Aproximados geralmente trabalha-se com valores discretos e não contínuos, assim, as informações das faixas referentes ao atributo “Exame4” podem ser representadas por etiquetas no lugar de faixas de valores, por exemplo: F1 para os valores < 12; F2 para a faixa 12-23; F3 para 24-32; F4 para os dados > 32. A [Tabela 6.6](#) a seguir ilustra a matriz de discernibilidade do sistema de informação correspondente.

Tabela 6.6

Matriz de discernibilidade do Exemplo 6.2

	o_1	o_2	o_3	o_4	o_5	o_6
o_1	-					
o_2	Ex1,Ex3,Ex4	-				
o_3	Ex1,Ex2,Ex3,Ex4	Ex1,Ex2,Ex4	-			
o_4	Ex2,Ex4	Ex1,Ex2,Ex3,Ex4	Ex1,Ex3,Ex4	-		
o_5	Ex1,Ex2,Ex3,Ex4	Ex1,Ex2,Ex3,Ex4	Ex1,Ex3	Ex1,Ex3,Ex4	-	
o_6	Ex2,Ex3,Ex4	Ex1,Ex2,Ex3,Ex4	Ex1,Ex3,Ex4	Ex3	Ex1,Ex3,Ex4	-

Com as informações da matriz de discernibilidade obtém-se a função de discernibilidade associada:

$$\begin{aligned}
 F(B) = & (Ex1 \vee Ex3 \vee Ex4) \wedge (Ex1 \vee Ex2 \vee Ex3 \vee Ex4) \wedge (Ex2 \vee Ex4) \\
 & \wedge (Ex1 \vee Ex2 \vee Ex3 \vee Ex4) \wedge (Ex2 \vee Ex3 \vee Ex4) \\
 & \wedge (Ex1 \vee Ex2 \vee Ex3) \wedge (Ex1 \vee Ex2 \vee Ex3 \vee Ex4) \\
 & \wedge (Ex1 \vee Ex2 \vee Ex3 \vee Ex4) \wedge (Ex1 \vee Ex2 \vee Ex3 \vee Ex4) \\
 & \wedge (Ex1 \vee Ex3 \vee Ex4) \wedge (Ex1 \vee Ex3) \wedge (Ex1 \vee Ex3 \vee Ex4) \\
 & \wedge (Ex1 \vee Ex3 \vee Ex4) \wedge (Ex3) \wedge (Ex1 \vee Ex3 \vee Ex4).
 \end{aligned}$$

Simplificando e colocando na forma da “soma do produto”, tem-se:

$$F(B) = (Ex1) \vee (Ex2 \wedge Ex3) \vee (Ex3 \wedge Ex4).$$

Assim, obtêm-se os redutos $\{\{Exame1\}, \{Exame2, Exame3\}, \{Exame3, Exame4\}\}$. Escolhendo o reduto com menor número de atributos, ou seja, $\{Exame1\}$, têm-se as regras de decisão correspondentes:

IF Ex1 = 1 THEN Diag = D1;
IF Ex1 = 2 THEN Diag = D2;
IF Ex1 = 4 THEN Diag = D3;
IF Ex1 = 3 THEN Diag = D2.

Logo, para a classificação dos diagnósticos (D1, D2 e D3), a informação do primeiro exame (Ex1) é suficiente. O conhecimento extraído possibilita reduzir o número de exames necessários para se classificar os diagnósticos almejados.

Tabela 6.7**Sistema de informação associado ao Exemplo 6.3**

	Cor	Dimensão	Densidade	Acabamento	Material	Vendas
o_1	azul	maior	alta	áspero	plástico	baixa
o_2	vermelha	média	média	liso	madeira	média
o_3	amarela	menor	baixa	padrão	alumínio	alta
o_4	azul	média	média	padrão	plástico	baixa
o_5	amarela	menor	baixa	áspero	plástico	média
o_6	verde	maior	alta	liso	madeira	alta
o_7	amarela	menor	alta	áspero	aço	alta
o_8	amarela	menor	alta	áspero	plástico	alta
o_9	verde	maior	alta	liso	madeira	média
o_{10}	verde	média	média	liso	plástico	média

Exemplo 6.3

O exemplo em questão está relacionado em determinar qual é a melhor combinação de características de um produto que possui algumas versões ofertadas. O objetivo consiste em reduzir os itens em estoque de forma a maximizar o lucro associado ao volume das versões vendidas. A tabela a seguir mostra observações de vendas anteriores com as descrições das opções do produto (essas foram baseadas em [Patrício et al., 2005](#)). Os atributos de condição considerados foram Cor, Dimensão, Densidade, Acabamento e Material. O atributo de decisão está relacionado com as Vendas.

Para os atributos de condição $B = \{\text{Cor, Dimensão, Densidade, Acabamento, Material}\}$, vem: $U/IND(B) = \{\{o_1\}, \{o_2\}, \{o_3\}, \{o_4\}, \{o_5\}, \{o_6, o_9\}, \{o_7\}, \{o_8\}, \{o_{10}\}\}$.

Para o conjunto $O = \{o_3, o_6, o_7, o_8\}$ **relativo à condição de venda alta**, têm-se as aproximações:

$$B_*(O) = \{\{o_3\}, \{o_7\}, \{o_8\}\};$$

$$B^*(O) = \{\{o_3\}, \{o_7\}, \{o_6, o_9\}, \{o_8\}\}.$$

As respectivas métricas são:

→ $\alpha(O) = \frac{3}{5} = 0,6$, ou seja, 60% dos elementos de O são precisos em relação a B ;

→ $\alpha(B^*(O)) = \frac{5}{10} = 0,5$, ou 50% das observações de U possivelmente pertencem a B ;

→ $\alpha(B_*(O)) = \frac{3}{10} = 0,3$, ou seja, 30% dos elementos de U certamente pertencem a B .

Graficamente as aproximações correspondentes estão ilustradas na [Figura 6.1](#).

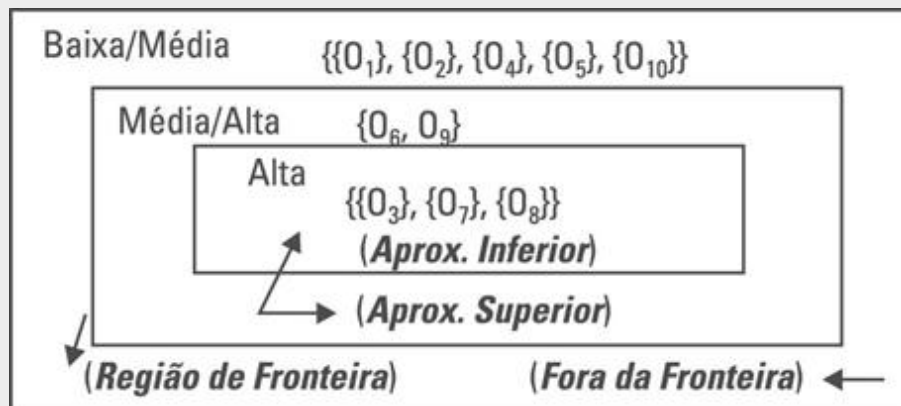


FIGURA 6.1 Ilustração gráfica das aproximações associadas ao Exemplo 6.3.

Para a matriz de discernibilidade do sistema montou-se a [Tabela 6.8](#), que contém apenas os exemplos distintos ($e_1, e_2, \dots, e_9$) associados aos atributos da condição do *SI*.

Tabela 6.8

Exemplos distintos relativos à [Tabela 6.7](#)

	Cor	Dimensão	Densidade	Acabamento	Material
e_1	azul	maior	alta	áspero	Plástico
e_2	vermelha	média	média	liso	Madeira
e_3	amarela	menor	baixa	padrão	Alumínio
e_4	azul	média	média	padrão	Plástico
e_5	amarela	menor	baixa	áspero	Plástico
e_6	verde	maior	alta	liso	Madeira
e_7	amarela	menor	alta	áspero	Aço
e_8	amarela	menor	alta	áspero	Plástico
e_9	verde	média	média	liso	Plástico

Considerando as abreviações Co, Di, De, Ac e Ma para os respectivos atributos, a [Tabela 6.9](#) constitui a matriz de discernibilidade do sistema correspondente.

Tabela 6.9**Matriz de discernibilidade do Exemplo 6.3**

	e_1	e_2	...	e_7	e_8	e_9
e_1	-					
e_2	Co,Di,De, Ac,Ma	-				
e_3	Co,Di,De, Ac,Ma	Co,Di,De, Ac,Ma	...			
e_4	Di,De,AC	Co,Ac,Ma	...			
...	...	...	...	...		
e_9	Co,Di,De, Ac	Co,Ma	...	Co,Di,De, Ac,Ma	Co,Di,De, Ac	-

A função de discernibilidade associada às informações da matriz correspondente está indicada abaixo.

$$F(B) = (Co \vee Di \vee De \vee Ac \vee Ma) \wedge (Co \vee Di \vee De \vee Ac \vee Ma) \wedge (Di \vee De \vee Ac) \wedge \dots \wedge (Co \vee Di \vee De \vee Ac)$$

Simplificando a função, tem-se:

$$F(B) = (Co \wedge De \wedge Ma) \vee (Co \wedge De \wedge Ac) \vee (Di \wedge De \wedge Ac).$$

Logo, os redutos resultantes são $\{\{Cor, Densidade, Material\}, \{Cor, Densidade, Acabamento\}, \{Dimensão, Densidade, Acabamento\}\}$. Para os

respectivos redutos têm-se as regras de decisão relativas à condição de venda alta:

```

IF Cor = amarela AND Densidade = baixa AND Material = alumínio THEN Venda = alta;
IF "    "    "    "    alta    "    "    aço    "    "    alta;
IF "    "    "    "    alta    "    "    plástico    "    "    alta;
IF Cor = amarela AND Densidade = baixa AND Acabamento = padrão THEN Venda = alta;
IF "    "    "    "    alta    "    "    áspero    "    "    alta;
IF "    "    "    "    alta    "    "    áspero    "    "    alta;
IF Dimensão = menor AND Densidade = baixa AND Acabamento = alumínio THEN Venda = alta;
IF "    "    "    "    alta    "    "    aço    "    "    alta;
IF "    "    "    "    alta    "    "    plástico    "    "    alta.

```

Conclusões referentes às regras de decisão geradas: As melhores aquisições futuras do produto, objetivando um estoque com menores combinações de características e um lucro maior (venda alta), estão relacionadas com as regras de decisão geradas. As novas aquisições do produto podem ser escolhidas por uma das três combinações de atributos (ou de suas características construtivas) definidas pelos redutos obtidos, os quais definem os conhecimentos extraídos dos dados do sistema de informação em questão.

Exemplo 6.4

Este problema é similar ao do [Exemplo 2.1 \(Capítulo 2\)](#) referente à aproximação da função associada aos dados da [Figura 2.8](#). Os domínios das variáveis x (ou x_1) e y estão compreendidos nas faixas $[0, 10]$ e $[2, 11]$, respectivamente. Conceitos relacionados com a *TCA* podem ser aplicados na aproximação de funções e na modelagem de sistemas dinâmicos com comportamentos não lineares. Mais detalhes podem ser encontrados em [Pinheiro et al., \(2010\)](#), onde se mostra também como podem ser obtidas regras difusas a partir de conceitos da *TCA*.

A variável x constitui o atributo de condição do sistema de informação associado, e a variável y define o atributo de decisão. A variável x constitui o reduto intrínseco desse *SI*. Como os dados associados à variável x

possuem valores fracionários, é necessário aplicar um procedimento de discretização nesses valores. Realizando o procedimento em três níveis, têm-se: [0, 3,375]; [3,375, 6,875]; [6,875, 10]. Assim, as regras de decisão correspondentes são:

```
IF x=[0,3,375] THEN y = 2 OR y = 2,2197) OR y = 2,3811 OR y = 2,5136 OR y = 2,731
OR y = 2,7827 OR y = 2,8327 OR y = 3,0351 OR y = 2,9551 OR y = 3,3973
OR y = 3,5117 OR y = 3,5909 OR y = 3,7345 OR y = 3,8419;
IF x=[3,375,6,875] THEN y = 4,0952) OR y = 4,2879) OR y = 4,4000) OR y = 4,8764
OR y = 5,2843 OR y = 5,9241 OR y = 6,3302 OR y = 6,9608 OR y = 7,3044 OR y = 7,6791
OR y = 8,2819 OR y = 9,0139 OR y = 9,3387 OR y = 10,042;
IF x=[6,875,10] THEN y = 10,4 OR y = 10,6437 OR y = 10,4786 OR y = 10,4928 OR y = 10,7082
OR y = 10,6233 OR y = 10,8862 OR y = 10,683 OR y = 10,8393 OR y = 10,9186 OR y = 10,8814
OR y = 10,9779 OR y = 11.
```

(6.13)

No [Anexo 6.2\(a\)](#) encontra-se a obtenção das regras de decisão desse exemplo, através da ferramenta Rosetta, e no [Anexo 6.2\(b\)](#), o programa de simulação dele.

As regras de decisão obtidas são classificadas como inconsistentes ou regras possíveis. Em [Pinheiro et al. \(2010\)](#) foi proposta a utilização da equação (6.14) de interpolação linear para a estimação de valores de modelos denominados aproximados, que constituem uma classe de modelos baseados em regras possíveis de aproximar funções genéricas ou modelar sistemas dinâmicos não lineares.

$$f(x_1, x_2, \dots, x_N) = \gamma(x_n, x_n^{(i)}, \gamma^{(i)})_{i=k,m}^{n=1,N} = \gamma^{(k)} + \frac{(\gamma^{(m)} - \gamma^{(k)})}{N} \sum_{n=1}^N \frac{(x_n - x_n^{(k)})}{(x_n^{(m)} - x_n^{(k)})}$$

(6.14)

Para o exemplo da aproximação da função em questão, considerando $N = 1$ (uma única variável de condição) na equação (6.14), tem-se (6.15) que constitui a expressão de interpolação de Lagrange.

$$f(x_1) = \gamma(x_1, x_1^{(k)}, x_1^{(m)}, \gamma^{(k)}, \gamma^{(m)}) = \gamma^{(k)} + \frac{(\gamma^{(m)} - \gamma^{(k)})}{1} \frac{(x_1 - x_1^{(k)})}{(x_1^{(m)} - x_1^{(k)})} \quad (6.15)$$

O procedimento descrito em [Pinheiro et al. \(2010\)](#) consiste em utilizar os valores extremos das variáveis associadas a cada regra obtida, de modo que podemos reescrever (6.13) na forma (6.16).

$$\begin{aligned} r_1: IF x_1 = [0, 3, 375] THEN \gamma &= [2, 3, 8419]; \\ r_2: IF x_1 = [3, 375, 6, 875] THEN \gamma &= [4, 0952, 10, 042]; \\ r_3: IF x_1 = [6, 875, 10] THEN \gamma &= [10, 4, 11] \end{aligned} \quad (6.16)$$

Utilizando (6.15), as regras (6.16) podem ser escritas na forma (6.17), que **representam o conhecimento inerente ao processo**. A obtenção dos coeficientes da primeira regra é dada por $y = 2 + (3,8419 - 2) \cdot (x_1 - 0) / (3,375 - 0) = 0,5457x_1 + 2$, e de forma similar para os coeficientes das outras duas regras deste exemplo.

$$\begin{aligned} r_1: IF x_1 = [0, 3, 375] THEN \gamma &= 0,5457x_1 + 2; \\ r_2: IF x_1 = [3, 375, 6, 875] THEN \gamma &= 1,6991x_1 - 1,6392; \\ r_3: IF x_1 = [6, 875, 10] THEN \gamma &= 0,192x_1 + 9,08. \end{aligned} \quad (6.17)$$

Para $x_1 = 2$, tem-se $y = 0,5457 \cdot 2 + 2 = 3,0914$. Para valores extremos da variável de condição (x_1), se mais de uma regra fornecer um valor estimado y para a função, o dado resultante é fornecido pela média aritmética dos valores inferidos pelas regras em questão. A [Figura 6.2](#) ilustra a aproximação obtida para a função correspondente.

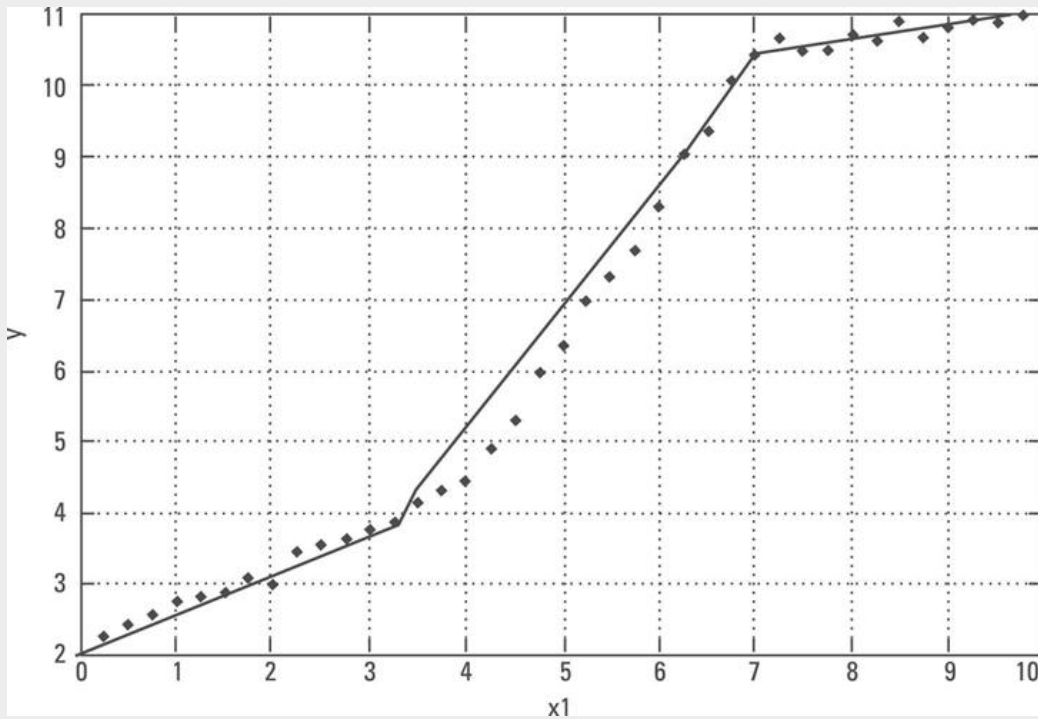


FIGURA 6.2 Resultado da modelagem aproximada associada a (6.17).

Exemplo 6.5

Esta exemplificação refere-se ao sistema de suspensão mostrado no [Exemplo 2.4 \(Capítulo 2\)](#), no qual foram empregadas regras *fuzzy* para controlar o sistema, e que agora serão utilizadas regras de controle obtidas via conceitos relacionados com a *TCA*. Informações gerais dos procedimentos envolvidos estão detalhadas em [Pinheiro et al., 2012](#), onde também é mostrado como obter regras nebulosas via *TCA*. Os gráficos da [Figura 6.3](#) mostram informações do sistema de suspensão como velocidade absoluta (*Absolute speed*), velocidade relativa (*Relative speed*), informação de comando (*Command input*) e força de compensação (*Compensation force*), todas associadas com a estratégia de controle continuamente variável (2.22), citada anteriormente.

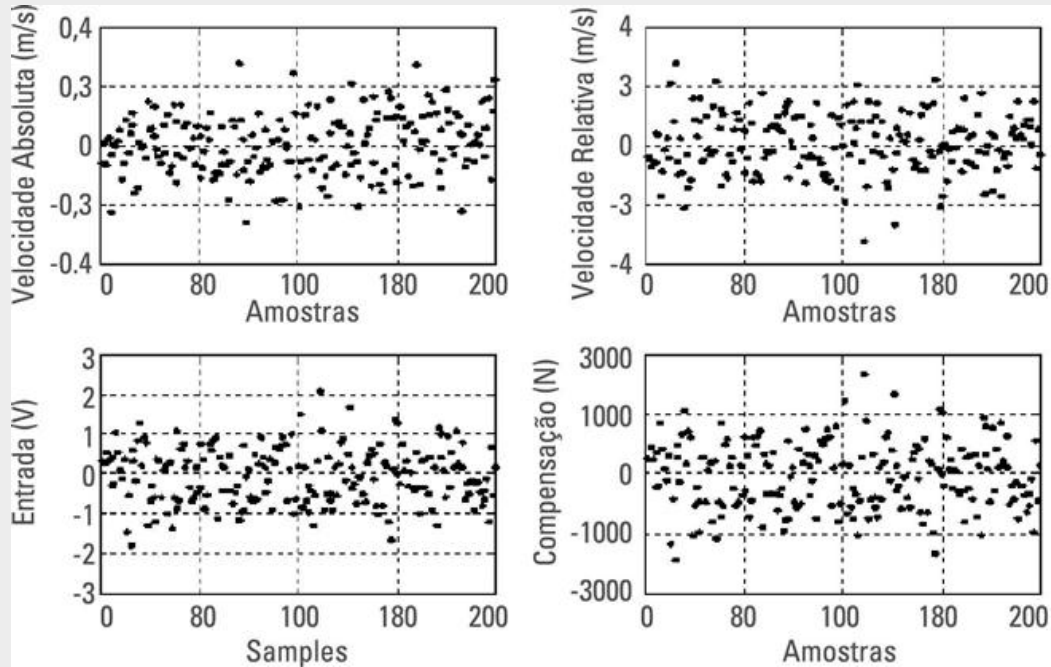


FIGURA 6.3 Informações relacionadas à [Figura 2.18](#) e às equações (2.21) e (2.22).

As variáveis x_1 = “velocidade absoluta” e x_2 = “velocidade relativa” constituem os atributos de condição do sistema de informação correspondente, e a variável y define o atributo de decisão relacionado à força de compensação ou a informação de comando associada ao processo de controle. As variáveis x_1 e x_2 constituem os redutos intrínsecos deste SI. Utilizando o software Rosetta (de forma similar à exemplificada no [Anexo 6.2](#)), com cinco níveis de discretização no processamento dos dados do SI em questão, obtêm-se as 25 regras de decisão (6.18) cujos valores dos intervalos das variáveis de condição são: $x_1^{(a)} = -2,385$; $x_1^{(b)} = -0,681$; $x_1^{(c)} = -0,184$; $x_1^{(d)} = 0,383$; $x_1^{(e)} = 0,9$; $x_1^{(f)} = 2,731$; $x_2^{(a)} = -0,3153$; $x_2^{(b)} = -0,078$; $x_2^{(c)} = -0,008$; $x_2^{(d)} = 0,04$; $x_2^{(e)} = 0,1$; $x_2^{(f)} = 0,368$. Essas **regras representam o conhecimento associado à estratégia de controle considerada**.

(6.18)

- r_1 : IF $x_1 = [x_1^{(e)}, x_1^{(f)}]$ AND $x_2 = [x_2^{(e)}, x_2^{(f)}]$ THEN $y = [-3,709,1,562]$;
 r_2 : IF $x_1 = [x_1^{(e)}, x_1^{(f)}]$ AND $x_2 = [x_2^{(d)}, x_2^{(e)}]$ THEN $y = [-3,593,1,379]$;
 r_3 : IF $x_1 = [x_1^{(c)}, x_1^{(d)}]$ AND $x_2 = [x_2^{(d)}, x_2^{(e)}]$ THEN $y = [0,621,0,226]$;
 r_4 : IF $x_1 = [x_1^{(e)}, x_1^{(f)}]$ AND $x_2 = [x_2^{(b)}, x_2^{(c)}]$ THEN $y = [-2,385,-1,23]$;
 r_5 : IF $x_1 = [x_1^{(e)}, x_1^{(f)}]$ AND $x_2 = [x_2^{(c)}, x_2^{(d)}]$ THEN $y = [-2,279,-1,218]$;
 r_6 : IF $x_1 = [x_1^{(d)}, x_1^{(e)}]$ AND $x_2 = [x_2^{(c)}, x_2^{(d)}]$ THEN $y = [-1,092,-0,597]$;
 r_7 : IF $x_1 = [x_1^{(a)}, x_1^{(b)}]$ AND $x_2 = [x_2^{(a)}, x_2^{(b)}]$ THEN $y = [1,513,3,387]$;
 r_8 : IF $x_1 = [x_1^{(e)}, x_1^{(f)}]$ AND $x_2 = [x_2^{(a)}, x_2^{(b)}]$ THEN $y = [-2,251,-1,128]$;
 r_9 : IF $x_1 = [x_1^{(a)}, x_1^{(b)}]$ AND $x_2 = [x_2^{(d)}, x_2^{(e)}]$ THEN $y = [0,967;3,443]$;
 r_{10} : IF $x_1 = [x_1^{(d)}, x_1^{(e)}]$ AND $x_2 = [x_2^{(b)}, x_2^{(c)}]$ THEN $y = [-0,513;-1,062]$;
 r_{11} : IF $x_1 = [x_1^{(a)}, x_1^{(b)}]$ AND $x_2 = [x_2^{(c)}, x_2^{(d)}]$ THEN $y = [0,923;3,174]$;
 r_{12} : IF $x_1 = [x_1^{(c)}, x_1^{(d)}]$ AND $x_2 = [x_2^{(a)}, x_2^{(b)}]$ THEN $y = [-0,437;-0,074]$;
 r_{13} : IF $x_1 = [x_1^{(b)}, x_1^{(c)}]$ AND $x_2 = [x_2^{(a)}, x_2^{(b)}]$ THEN $y = [0,783,1,547]$;
 r_{14} : IF $x_1 = [x_1^{(c)}, x_1^{(d)}]$ AND $x_2 = [x_2^{(c)}, x_2^{(d)}]$ THEN $y = [-0,555,0,188]$;
 r_{15} : IF $x_1 = [x_1^{(d)}, x_1^{(e)}]$ AND $x_2 = [x_2^{(a)}, x_2^{(b)}]$ THEN $y = [-1,088,-0,58]$;
 r_{16} : IF $x_1 = [x_1^{(c)}, x_1^{(d)}]$ AND $x_2 = [x_2^{(e)}, x_2^{(f)}]$ THEN $y = [-1,048,0,116]$;
 r_{17} : IF $x_1 = [x_1^{(d)}, x_1^{(e)}]$ AND $x_2 = [x_2^{(d)}, x_2^{(e)}]$ THEN $y = [-1,361,-0,773]$;
 r_{18} : IF $x_1 = [x_1^{(b)}, x_1^{(c)}]$ AND $x_2 = [x_2^{(d)}, x_2^{(e)}]$ THEN $y = [0,282,0,8]$;
 r_{19} : IF $x_1 = [x_1^{(d)}, x_1^{(e)}]$ AND $x_2 = [x_2^{(c)}, x_2^{(d)}]$ THEN $y = [0,300,0,81]$;
 r_{20} : IF $x_1 = [x_1^{(c)}, x_1^{(d)}]$ AND $x_2 = [x_2^{(b)}, x_2^{(c)}]$ THEN $y = [-0,384,0,3]$;
 r_{21} : IF $x_1 = [x_1^{(a)}, x_1^{(b)}]$ AND $x_2 = [x_2^{(e)}, x_2^{(f)}]$ THEN $y = [0,854,2,688]$;
 r_{22} : IF $x_1 = [x_1^{(d)}, x_1^{(e)}]$ AND $x_2 = [x_2^{(e)}, x_2^{(f)}]$ THEN $y = [-2,169,-0,992]$;
 r_{23} : IF $x_1 = [x_1^{(b)}, x_1^{(c)}]$ AND $x_2 = [x_2^{(e)}, x_2^{(f)}]$ THEN $y = [0,235,0,848]$;
 r_{24} : IF $x_1 = [x_1^{(a)}, x_1^{(b)}]$ AND $x_2 = [x_2^{(b)}, x_2^{(c)}]$ THEN $y = [1,073,2,998]$;
 r_{25} : IF $x_1 = [x_1^{(b)}, x_1^{(c)}]$ AND $x_2 = [x_2^{(b)}, x_2^{(c)}]$ THEN $y = [0,408,0,99]$.

Utilizando a equação (6.14) de interpolação linear e considerando duas variáveis de entrada (logo, $N = 2$), tem-se (6.19) para o valor da informação de atuação do processo de controle considerado.

$$f(x_1, x_2) = y^{(k)} + \frac{(y^{(m)} - y^{(k)})}{2} \left(\frac{(x_1 - x_1^{(k)})}{(x_1^{(m)} - x_1^{(k)})} + \frac{(x_2 - x_2^{(k)})}{(x_2^{(m)} - x_2^{(k)})} \right) \quad (6.19)$$

A Figura 6.4 ilustra o resultado da simulação da aceleração (*Acceleration*) do processo de suspensão controlado via estratégia de controle clássica (*Classical Control*) definida por (2.22), e utilizando as regras de controle (6.18) com a informação de comando fornecida com a interpolação dada por (6.19), em uma forma resultante denominada controle aproximado (*Rough Control*). Similarmente ao Exemplo 2.4, o resultado da estratégia definida pelas regras de comando foi melhor que com o controle clássico, pois essas regras mapearam adequadamente as variações possíveis no contexto do sistema de controle em questão.

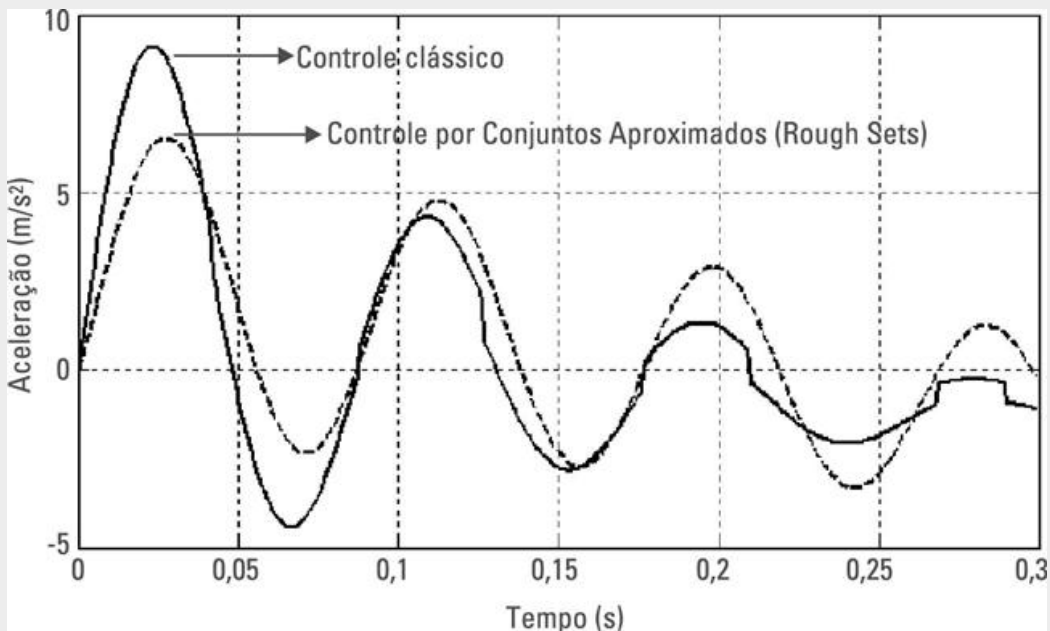


FIGURA 6.4 Resposta do sistema de suspensão veicular via estratégia com as regras (6.18).

A vantagem da utilização das regras aproximadas (6.18) em relação às regras *fuzzy* (2.23) relativas à Tabela 2.5 e funções de pertinência definidas nas Figuras 2.21 e 2.22 do Capítulo 2, é que as regras aproximadas não utilizam procedimentos de fuzificação e defuzificação de dados no processamento correspondente ao controle *fuzzy* considerado, o que resulta em tempos de processamentos menores. Esse contexto constitui uma característica interessante em aplicações de tempo real.

Na referência Machado e Pinheiro (2013) há informações sobre uma técnica de controle baseada na TCA associada com métodos de retroação de estados. Em Oliveira (2013) têm-se detalhes sobre a modelagem de sistemas não lineares com múltiplas entradas e saídas baseada em conceitos da TCA. Em Guaracy (2013) e Rodor (2012) se encontram detalhes sobre técnicas de controle SMC (*Sliding Mode Control*) e IMC (*Internal Model Control*) associadas com conceitos da TCA. Nas referências Faustino (2011) e Faustino et al. (2011) têm-se aplicações de conceitos de *Rough Sets* em previsões de séries temporais.

No item a seguir tem-se uma breve introdução de alguns conceitos relacionados com outras abordagens de IA.

6.5 Outras abordagens de IA

Entre outras abordagens de IA frequentemente utilizadas em vários contextos estão os Algoritmos Evolutivos, ou a área de IA conhecida como “Computação Evolucionária”.

Entre os “Algoritmos Evolutivos” mais conhecidos estão os Algoritmos Genéticos (*Genetic Algorithms* – GA), os “Enxames de Partículas” (*Particle Swarm* ou *Particle Swarm Optimization* – PSO) e os “Sistemas Imunológicos”

Os Algoritmos Genéticos (Holland, 1992) são baseados em conceitos associados com difusão dos genes (que definem características) entre gerações de uma determinada espécie, e em mutações ocorridas nesses genes e que alteram algumas características da espécie em questão, que sendo vantajosa se propagam para as gerações futuras. Os algoritmos que utilizam esses conceitos possuem capacidades de adaptações aos contextos em que são aplicados, fornecendo resultados otimizados em relação aos algoritmos convencionais elaborados para as mesmas tarefas correspondentes.

Os algoritmos de Enxame de Partículas ([Eberhart et al., 2001](#)) possuem diferentes classificações conforme seus conceitos de inspiração biológica: Colônia de Formigas ou *Ant Colony* ([Bonabeau et al., 1999](#)); Enxame de Abelhas (*Bees Algorithm*); Bando de Aves (*Bird Algorithm*); Cardume de Peixes (*Fish Swarm*). Estes algoritmos são baseados no comportamento coletivo de espécies como aves, peixes, abelhas, formigas e outras que podem ser modeladas como um conjunto de partículas (indivíduos) que agem coletivamente na procura de alimentos, abrigo, rotas etc. Similarmente aos GA, os algoritmos de Enxame de Partículas fornecem resultados otimizados em relação aos algoritmos convencionais.

Os algoritmos com inspiração em sistemas imunológicos ([Dasgupta, 1999](#)) utilizam conceitos baseados na ação dos mecanismos de defesa dos organismos biológicos, como os glóbulos brancos dos seres humanos. Similarmente aos GA e os algoritmos de Enxame de Partículas, os algoritmos imunológicos fornecem resultados otimizados em relação aos algoritmos convencionais projetados para resolver tarefas correspondentes.

Os algoritmos evolutivos têm aplicações típicas relacionadas com otimização de sistemas em geral. Os itens seguintes trazem uma breve conceituação sobre os Algoritmos Genéticos e os de Enxame de Partículas assim como exemplos ilustrativos.

6.5.1 Conceituação básica sobre GA

Computacionalmente (e conceitualmente), um Algoritmo Genético típico pode ser descrito nos passos seguintes ([Fogel, 1995](#)):

1. O problema a ser resolvido é definido por uma função-objetivo que indica a aptidão (*fitness*) de alguma solução em potencial.
2. Uma população de soluções candidatas é inicializada. Cada tentativa de solução é codificada como um vetor $\mathbf{x}$ denominado cromossomo, com elementos simulando genes que variam valores de posições específicas em sua estrutura, cuja representação básica consiste de uma *string* de valores binários.
3. Cada cromossomo $\mathbf{x}_i$ em uma população é codificado de forma apropriada para avaliação via um índice de aptidão $\mu(\mathbf{x}_i)$, também conhecido como *fitness*.

4. Para cada cromossomo é associada uma probabilidade de reprodução relativa aos outros cromossomos da população.
5. De acordo com as probabilidades de reprodução atribuídas, uma nova população de cromossomos é gerada aleatoriamente selecionando-se características da população corrente. Para esta tarefa são aplicados “operadores genéticos” denominados cruzamento (*crossover*) e mutação. O *crossover* é aplicado a dois cromossomos parentes, que gera um novo cromossomo descendente via seleção randômica da representação codificada por ele. Um operador de mutação de bit promove a mudança de bits do código genético correspondente, codificando uma nova solução do sistema.
6. O procedimento é interrompido se uma solução viável é encontrada ou se o número máximo especificado de repetições do algoritmo ocorre. Caso contrário, o processo retorna ao terceiro passo, em que um novo cromossomo é selecionado e o procedimento continua.

Um GA apresenta um comportamento semelhante à atuação da genética na natureza. Em uma população de indivíduos, cada qual representando uma solução do problema em questão, para cada indivíduo associa-se um grau de aptidão que determina a sua capacidade de competir com os demais membros da população. Quanto maior for sua aptidão (*fitness*), maior a sua probabilidade de ser selecionado para se reproduzir, cruzando (*crossover*) seu material genético com outro indivíduo selecionado. Esse cruzamento produzirá novos indivíduos com características de seus “pais”. Este processo chamado de “reprodução”, é repetido até que uma solução satisfatória seja encontrada. Sobre esses novos indivíduos deverá atuar ainda o operador de mutação, que é necessário para introduzir e manter a diversidade genética da “população”.

Como exemplo, seja o problema de encontrar o valor da variável x que minimiza a função $f(x) = x^2$. A [Tabela 6.10](#) resume os procedimentos básicos realizados pelo algoritmo na busca de uma solução para o problema.

Tabela 6.10**Dados do GA para $f(x) = x^2$**

Iteração i	String x_i	Fitness $\mu(x_i)$	Aptidão Relativa
1	01010	100	0,12
2	01110	196	0,23
3	10110	484	0,56
4	01001	81	0,09
Total		861	1,00

Para a função simples deste exemplo, a solução analítica exata do problema é obtida derivando $f(x)$ e igualando-a a zero, obtendo-se $x = 0$ e $f(x) = 0$. Para funções complexas, soluções analíticas não são triviais, e algoritmos de otimização (convencionais ou via GA ou PSO) são empregados. A listagem abaixo ilustra a solução do problema via comandos específicos para GA do MatLab, que também possui um *toolbox* genérico para otimização de sistemas (possuindo também a opção de GA).

Listagem 6.1

% Definição da função a ser otimizada por intermédio de um arquivo.m:

```
function y = Exemplo_fitness(x)
    y = x(1) ^ 2;
```

% Utilização de comandos específicos para GA do MatLab:

```
FitnessFunction = @Exemplo_fitness;  
numberOfVariables = 1;  
opts = gaoptimset('PlotFcns', {@gaplotbestf, @gaplotstopping});  
[x, Fval, exitFlag, Output] = ga(FitnessFunction, numberOfVariables, opts);
```

Os resultados obtidos pelo GA foram $x = 7,11 \cdot 10^{-4}$ e $f(x) = Fval = 5,05 \cdot 10^{-7}$ constituindo valores numéricos bem próximos da solução exata do problema. Os gráficos da [Figura 6.5](#) ilustram o número de gerações e os valores das aptidões (*fitness*) processadas pelo algoritmo.

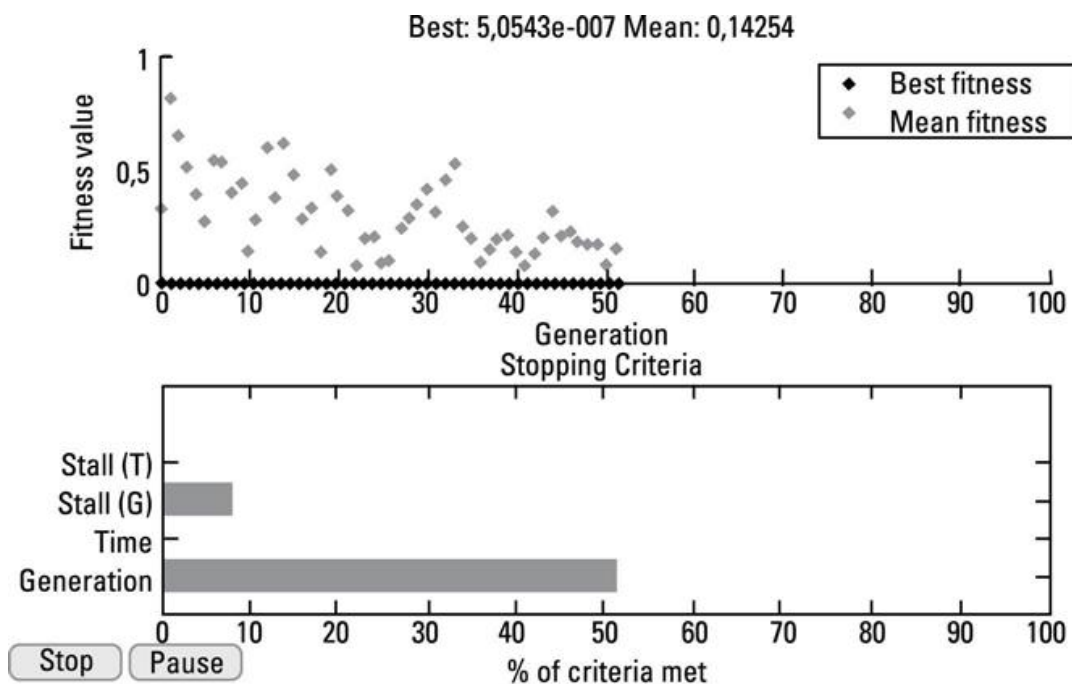


FIGURA 6.5 Número de gerações e valores das aptidões processadas pelo algoritmo.

A utilização do *toolbox* de otimização citado é realizada digitando na área de trabalho `optimtool('ga')`, e na tela que se abre colocar no campo “Fitness function” o nome da função-objetivo em questão (`@Exemplo_fitness`, neste

exemplo), em “Number of variables” digitar “1” e pressionar o botão *Start* (existem outras seleções de parâmetros que não foram consideradas neste exemplo). O resultado do processamento do algoritmo GA (ou de outros eventualmente selecionados) aparece na janela inferior esquerda do aplicativo

6.5.2 Conceituação básica sobre PSO

Um algoritmo PSO tem algumas características em comum com técnicas de GA. O processo é inicializado com uma população de soluções aleatórias, e procura-se por um resultado ótimo do problema abordado. Diferentemente do GA, um algoritmo PSO não tem operadores como *crossover* e mutação. As soluções potenciais (chamadas de partículas) se deslocam através do espaço de estados do problema, seguindo as partículas que possuem os melhores valores de avaliação (*fitness*) no momento. Um algoritmo PSO é computacionalmente fácil de realizar e possui poucos parâmetros para se ajustar.

Um PSO simula o comportamento de uma revoada de pássaros, por exemplo, em que um grupo de aves está procurando aleatoriamente por comida. Inicialmente nenhum pássaro sabe exatamente onde se encontra o alimento, mas eles sabem estimar a cada nova informação de posição e de velocidade dos indivíduos do bando a melhor estratégia para encontrar a comida, e a maioria do bando tende a seguir o pássaro que está mais perto do alimento.

O algoritmo é inicializado com um grupo de partículas aleatórias (soluções que procura pela solução ótima a cada iteração. A cada momento as partículas tendem a melhorar as suas posições em relação ao objetivo almejado (“alimento” ou uma função de custo em aplicações de otimização), conforme dois indicadores de avaliação. O primeiro deles é o melhor resultado local que uma partícula considerada no momento encontrou, sendo chamado de *Pbest* (ou *LocalBest*). O outro indicador é o melhor valor obtido pelo conjunto de outras partículas da população, e é conhecido por *Gbest* (ou *GlobalBest*). Quando um determinado problema apresenta restrições, essas podem ser modeladas na própria função de custo do algoritmo.

As equações básicas que modelam as posições (p) e as velocidades (v) de cada partícula de um PSO, em determinado instante de tempo (t), estão definidas abaixo:

$$p(t) = p(t-1) + v(t-1);$$

$$v(t) = v(t-1) + c_1 \cdot \text{rand.}(\text{Pbest}(t-1) - p(t-1)) + c_2 \cdot \text{rand.}(\text{Gbest}(t-1) - p(t-1)).$$

Onde *rand* constitui um número aleatório na faixa [0, 1], c_1 e c_2 são coeficientes (usualmente com valores iguais a dois) de convergência do algoritmo, conhecidos como parâmetros cognitivo e social, respectivamente. Um valor de velocidade máxima (v_max) é utilizado para limitar a soma das velocidades de todas as partículas em uma determinada direção do espaço de estados do problema.

No [Anexo 6.3](#) deste capítulo encontra-se um algoritmo básico de PSO que será utilizado no exemplo a seguir. Ele consiste no mesmo problema utilizado na exemplificação de GA, em que será empregada a função de custo $f(x) = x^2$. A questão é encontrar o valor da variável x que minimiza a função $f(x)$. A função em questão é definida por intermédio de um arquivo.m, como:

$$\text{function } \gamma = \text{Custo}(x)$$

$$\gamma = x^2;$$

Executando o algoritmo PSO ([Anexo 6.3](#)), tem-se $x = -3,51 \cdot 10^{-6}$ e $f(x) = 1,23 \cdot 10^{-11}$ (valores numéricos muito próximos da solução exata do problema). Os gráficos da [Figura 6.6](#) ilustram o número de iterações e os valores de *fitness* resultantes do processamento do algoritmo (os dados dos *fitness* não estão normalizados entre [0, 1]).

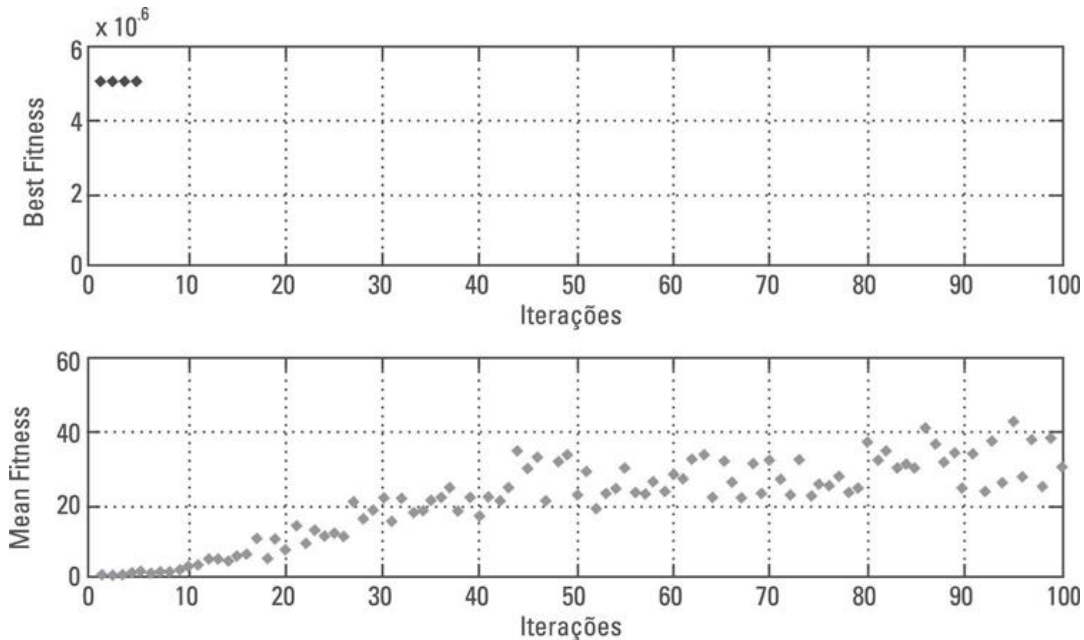


FIGURA 6.6 Número de passos e valores das aptidões obtidas pelo algoritmo.

Outra lógica com várias implicações teóricas e práticas em contextos de IA é a Lógica Paraconsistente proposta pelo acadêmico brasileiro Newton [Da Costa \(1963\)](#). Essa lógica lida com inconsistências da lógica clássica ([Da Costa et al., 1991](#)) e possui várias aplicações práticas ([Da Costa et al., 1999](#)). Atualmente dispõem-se de várias publicações de renome que possuem artigos relativos à Lógica Paraconsistente.

As diversas técnicas de IA apresentadas neste livro podem ser combinadas entre si, objetivando a obtenção de melhores desempenhos nas aplicações realizadas. Atualmente têm-se vários artigos, textos e teses disponíveis sobre este assunto.

A seguir, alguns exercícios propostos para treinar os conceitos vistos neste capítulo.

6.6 Exercícios

Exercício 6.1

Seja o sistema de informação ilustrado abaixo contendo oito registros relacionados à indicação de consumo de vinhos que estão associados a

quatro atributos de condição: origem do vinho; tipo de uva; safra; temperatura de armazenamento.

	Origem	Tipo de Uva	Safra	Temperatura de Armazenamento	Indicação de Consumo
o_1	Bordeaux	Cabernet	1992	12 - 15	Atual
o_2	Rhone	Syrah	1992	< 12	Futuro
o_3	Chile	Cabernet	1995	12 - 15	Atual
o_4	Bordeaux	Merlot	1995	> 15	Atual
o_5	Chile	Cabernet	1995	12 - 15	Futuro
o_6	Rhone	Merlot	1992	12 - 15	Futuro
o_7	Bordeaux	Merlot	1995	> 15	Atual
o_8	Chile	Merlot	1992	< 12	Futuro

- Considerando o conjunto $B = \{\text{Origem, Tipo de Uva, Safra, Temperatura de Armazenamento}\}$ de todos os possíveis atributos do SI em questão, obter as classes de equivalência correspondentes a $IND(B)$.
- Obter o valor da aproximação inferior relativa à indicação de consumo “Atual”. Esboçar um desenho similar ao da [Figura 6.1](#), contendo a representação da aproximação inferior, superior, região de borda e fora de fronteira.
- Obter os redutos desse SI referentes à indicação de consumo “Futuro”. Para o reduto $\{\text{Origem, Tipo de Uva}\}$ indicar a(s) regra(s) de decisão correspondente(s).

Exercício 6.2

Obter todos os redutos e as correspondentes regras de decisão relacionadas ao contexto do item (c) do exercício anterior, através do software Rosetta.

Exercício 6.3

A tabela abaixo ([Sienkiewicz, 1995](#)) constitui um *SI* representando um mapa lógico de um sistema binário (p. ex., a síntese lógica de um circuito digital), onde as variáveis “a”, “b”, “c” constituem as entradas do sistema e “s” a variável de saída. Utilizando conceitos da *TCA*, obter as regras de decisão correspondentes ao *SI*. Analisando as regras de decisão obtidas, qual é a função lógica correspondente?

a	b	c	s
0	0	0	0
0	1	0	0
0	0	1	1
1	1	0	1

Exercício 6.4

Obter as regras de decisão relativas ao exercício anterior com o software Rosetta.

Exercício 6.5

Seja o [Exemplo 3.5](#) ([Capítulo 3](#)) referente ao problema de aproximação da função não linear $y = x_1^2$ (para $x_1 \in [0, 1]$), via modelo neural. Resolver esse problema, via conceitos da *TCA*, utilizando o [Exemplo 6.4](#) como referência. Considerar três níveis de discretização com intervalos igualmente espaçados para a variável x_1 . Apresentar as três regras aproximadas de modelagem e esboçar o gráfico da aproximação resultante.

Exercício 6.6

Obter as regras de decisão da modelagem associada ao exercício anterior, via software Rosetta.

Exercício 6.7

Obter as regras de decisão associadas ao [Exemplo 6.2](#), via software Rosetta.

Exercício 6.8

Obter as regras de decisão associadas ao [Exemplo 6.3](#), com a ferramenta Rosetta.

Exercício 6.9

Baseando-se na [Listagem 6.1](#), associada à ferramenta de GA utilizada no exemplo relativo a $f(x) = x^2$, obter os valores ótimos da função de Rosenbrock:

$$f(x_1, x_2) = 100(x_1^2 - x_2)^2 + (1 - x_1)^2.$$

Exercício 6.10

Baseado-se no programa de PSO listado no [Anexo 6.3](#), utilizado no exemplo relativo a $f(x) = x^2$, obter os valores ótimos da função de Rosenbrock:

$$f(x_1, x_2) = 100(x_1^2 - x_2)^2 + (1 - x_1)^2.$$

Referências bibliográficas

1. BONABEAU E, DORIGO M, THERAULAZ G. *Swarm Intelligence Natural to Artificial Systems*. Nova York: Oxford University Press; 1999.

2. CARVALHO, M. A. (2010). **Discretização de atributos contínuos em sistemas de informação utilizando algoritmos genéticos para a aplicação da Teoria de Conjuntos Aproximados**. Dissertação de Mestrado, Universidade Federal de Itajubá. Disponível em: <http://juno.unifei.edu.br/bim/0037080.pdf>.
3. DA COSTA NCA. *Sistemas Formais Inconsistentes*. Curitiba: Tese (cátedra), Universidade do Paraná; 1963.
4. DA COSTA, N. C. A.; SUBRAHMANIAN, V. S.; VAGO, C. (1991). **The Paraconsistent Logic Pt. Zeitschrift fur Mathematische Logik und Grundlagen der Mathematik**, v.37, p. 139-48.
5. DA COSTA NCA, ABE JM, MUROLO AC. *Lógica paraconsistente aplicada*. São Paulo: Atlas; 1999.
6. DASGUPTA D. *Artificial Immune Systems and Their Applications*. Berlim: Springer Verlag; 1999.
7. DEMPSTER AP. Upper and lower probabilities induced by a multivalued mapping. *The Annals of Mathematical Statistics* 1967.
8. EBERHART R, SHI Y, KENNEDY J. *Swarm Intelligence*. São Francisco: Morgan Kaufmann; 2001.
9. FAUSTINO, C. P. (2011) **Previsão de séries temporais via modelos baseados em regras**. Dissertação de Mestrado, Universidade Federal de Itajubá. Disponível em: <http://juno.unifei.edu.br/bim/0038068.pdf>.
10. FAUSTINO C, PINHEIRO C, CARPINTEIRO O, LIMA I. Time series forecasting through rule-based models obtained via rough sets. *Artificial Intelligence Review*. 2011;1:1–12.
11. Fogel DB. *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*. Nova York: IEEE Press; 1995.
12. GUARACY, F. H. D. (2013). **Contribuições a aplicações práticas de sistemas de controle por modos deslizantes. Dissertação de Mestrado**. Universidade Federal de Itajubá. Disponível em: <http://juno.unifei.edu.br/bim/0039492.pdf>.
13. HOLLAND JH. *Adaptation in natural and artificial systems*. MIT Press 1992.
14. HUANG B, GUO L, ZHOU X. Approximation Reduction Based on Similarity Relation. *EEE Fourth International Conference on Fuzzy Systems and Knowledge Discovery* 2007:124–128.

15. HVIDSTEN, T. R. (1999). **Fault Diagnosis in Rotating Machinery Using Rough Set Theory and Rosetta**. Research Report, Norwegian University of Science and Technology.
citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.33.9525.
16. MACHADO K, PINHEIRO C. Rough Controllers With State Feedback. *Engineering Applications of Artificial Intelligence*. 2013;26:2.467–2.479.
17. MITCHELL TM. *Machine Learning*. Nova York: MacGraw-Hill; 1997.
18. NGUYEN S, SKOWRON A. Quantization of real value attributes. *Proc of the Second Joint Annual Conference on Information Sciences* 1995:37–40.
19. OHRN, A., KOMOROWSKI, J. (1997). **Rosetta – a rough sets toolkit for analyses of data**. Third International Joint Conference on Information Sciences, p. 403-7. Disponível em:
<http://www.lcb.uu.se/tools/rosetta/>.
20. OLIVEIRA, R. H. (2013). **Modelagem de Sistemas Mimo Baseada em Conjuntos Aproximados**. Tese de Doutorado. Universidade Federal de Itajubá. Disponível em: www.unifei.edu.br/biblioteca.
21. PATRÍCIO, C. M. M. M.; PINTO, J. O. P.; SOUZA, C. C. (2005). **Rough Sets – Técnica de Redução de Atributos e Geração de Regras para Classificação de Dados**. XXVIII Congresso Nacional de Matemática Aplicada e Computacional – CNMAC, São Paulo. Disponível em:
http://www.sbmec.org.br/eventos/cnmac/cd_xxviii_cnmac/resumos%20estendidos/cristian_patricio_ST18.pdf.
22. PAWLAK Z. Rough Sets. *International Journal of Computer and Information Sciences* 1982.
23. PAWLAK Z. *Rough Sets: Theoretical aspects of reasoning about data* Londres, Netherlands: Kluwer Academic Publishers; 1991.
24. PAWLAK Z, SKOWRON A. Rudiments of Rough Sets. *Information Sciences*. 2007a;117:3–27.
25. Rough sets: Some extensions. *Information Sciences*. 2007b;117:28–40.
26. PILA, A. D. (2001). **Seleção de Atributos Relevantes para Aprendizado de Máquina Utilizando a Abordagem de Rough**

- Sets.** Dissertação de Mestrado. USP-São Carlos. Disponível em: www.teses.usp.br/teses/disponiveis/55/55134/tde-13022002-153921/.
27. PINHEIRO, C.; GOMIDE F.; CARPINTEIRO, O.; LOPEZ, B. (2010). **Granular Synthesis of Rule-Based Models and Function Approximation using Rough Sets**. Chapter in the book *Novel Developments in Granular Computing*, ed. JingTaoYao, Information Science Publishing.
 28. PINHEIRO, C.; CAMATTA, U.; REZEK, A. (2012). **Rough Controller Synthesis**. Chapter in the book *Fuzzy Logic: Controls, Concepts, Theories and Applications*, ed. Elmer P. Dadios, INTECH.
 29. QUAN XL, BIAO MH. Self-Adaptive Ant Colony Algorithm for Attributes Reduction. *IEEE International Conference on Granular Computing* 2008:686–689.
 30. RISVIK, K. M. (1999). **Discretization of Numeric Attributes. Research Report, Norwegian University of Science and Techonology**. Disponível em: citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.47.3158&rep=rep1&type=pdf.
 31. RODOR, F. F. (2012). **Aplicação de Conceitos de Conjuntos Aproximados na Adaptação da Constante de Filtro de Controladores IMC**. Dissertação de Mestrado, Universidade Federal de Itajubá. <http://juno.unifei.edu.br/bim/0039510.pdf>.
 32. SHAFER G. *A Mathematical Theory of Evidence*. Princeton University Press 1976.
 33. SHEN Q, CHOUCHOULAS A. Rough Set-Based Dimensionality Reduction for Supervised and Unsupervised Learning. *Int J Appl Math Comput Sci*. 2001.
 34. SIENKIEWICZ, J. (1995). **Rough Sets for Boolean Functions Minimization**. Research Report, Warsaw Institute of Technology. Disponível em: citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.56.5052.
 35. ZADEH LA. Outline of a New Approach To the Analysis of Complex Systems And Decision Processes. *IEEE Trans on Systems, Man and Cybernetics*. 1973;3:28–44.

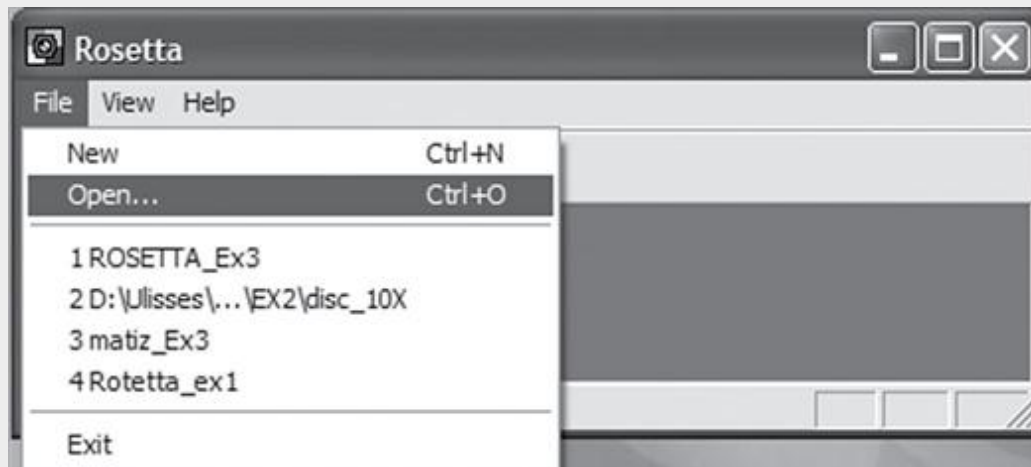
Anexo 6.1

Resolução do Exemplo 6.1 com a ferramenta Rosetta

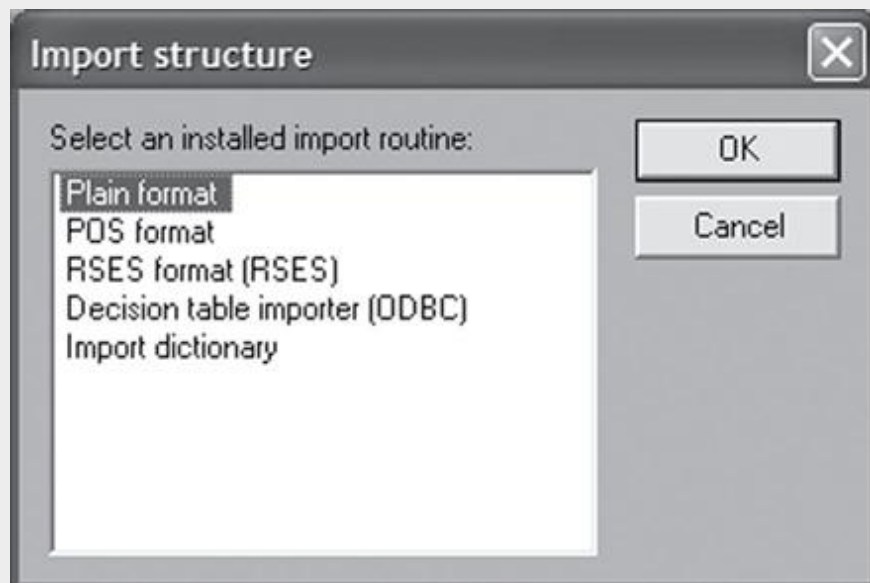
Os dados do problema devem estar associados a arquivos ou planilhas correspondentes. Por exemplo, um arquivo tipo texto (.txt). A planilha exemplificada abaixo ilustra a representação do arquivo de dados do *SI* em questão, com as variáveis do sistema declaradas como *string* neste exemplo. Outros tipos de variáveis do aplicativo são *integer* e *float*.

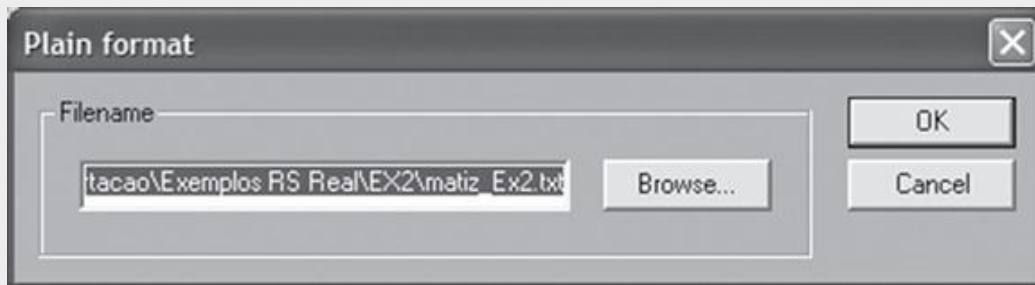
MBA	GRAD	EXP	CONTRAT
String	String	String	String
N	A	S	Maior
N	A	S	Maior
S	A	S	Não
N	C	N	Baixa
N	C	N	Média

Executando o software tem-se a tela principal do aplicativo, como ilustrado abaixo.

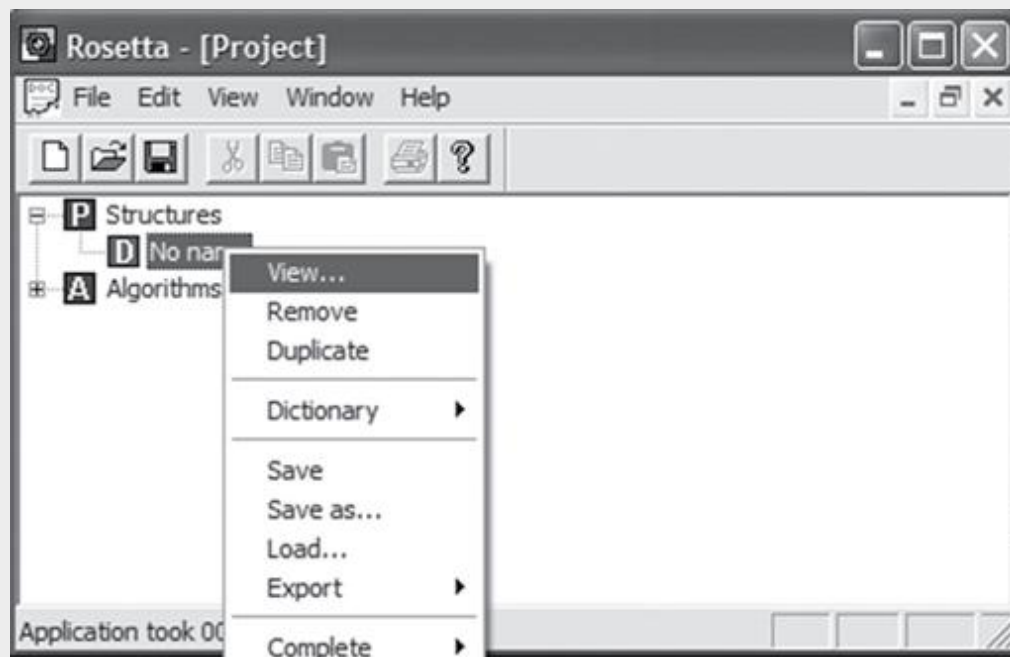


Na opção “File => New” carrega-se a planilha do sistema de informação em questão, onde aparece a janela abaixo, e selecciona-se a opção “Plain format”.

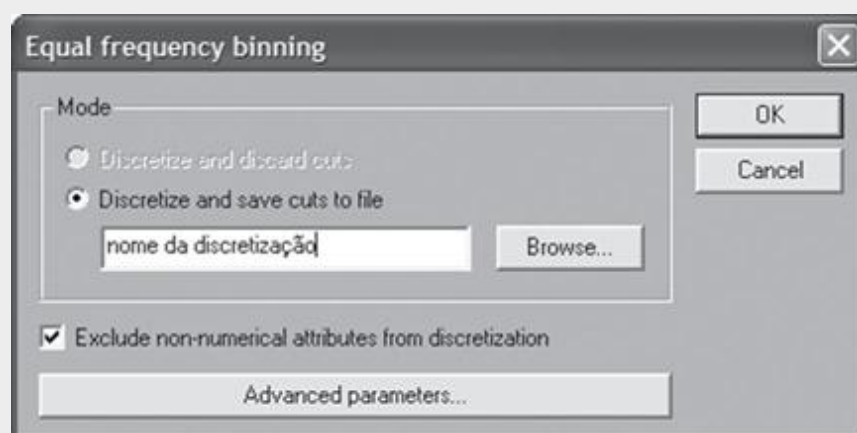
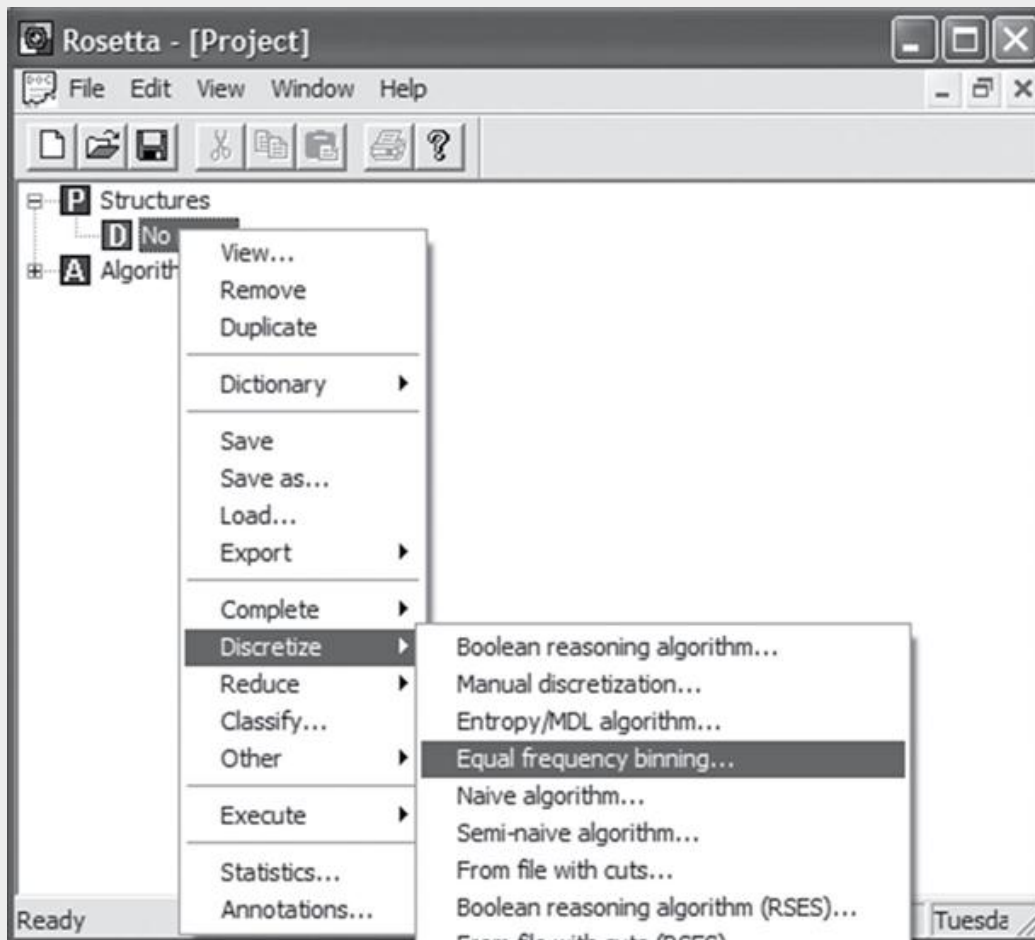




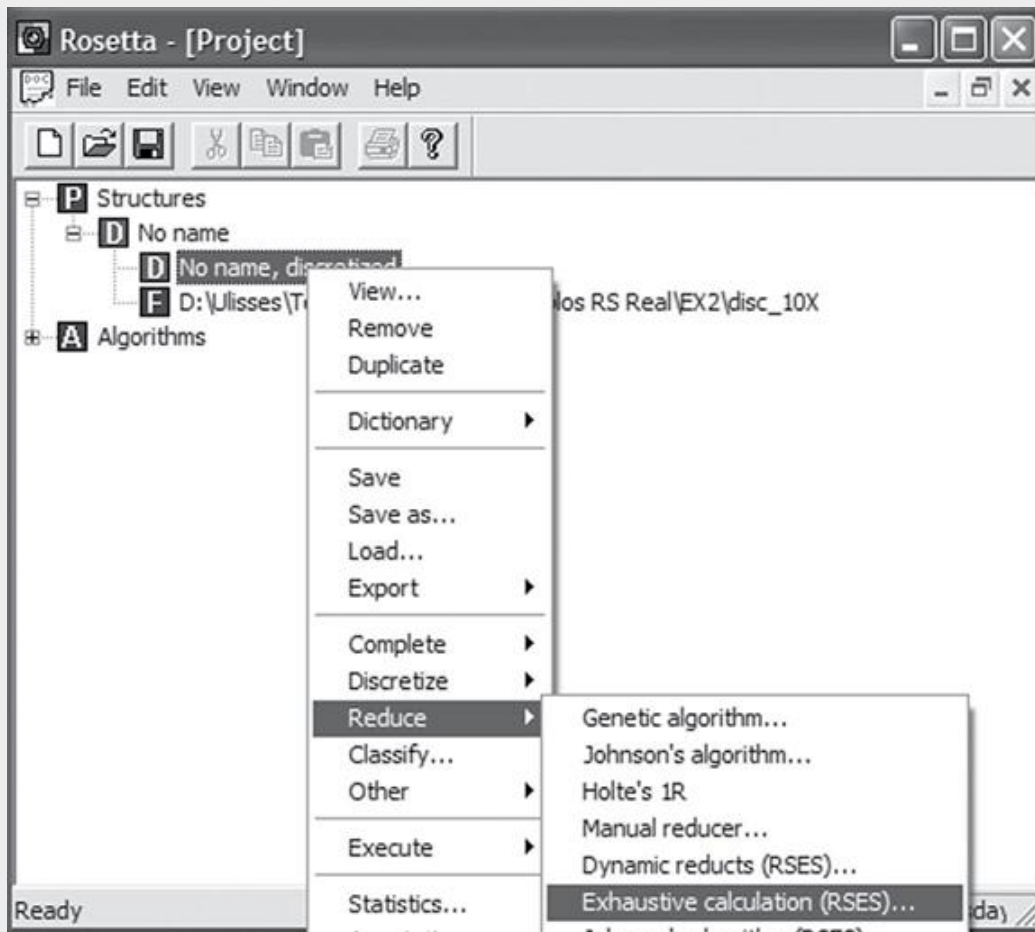
Para visualizar o arquivo com os dados carregados do *SI*, basta selecionar a opção “View”, como ilustrado na janela a seguir.



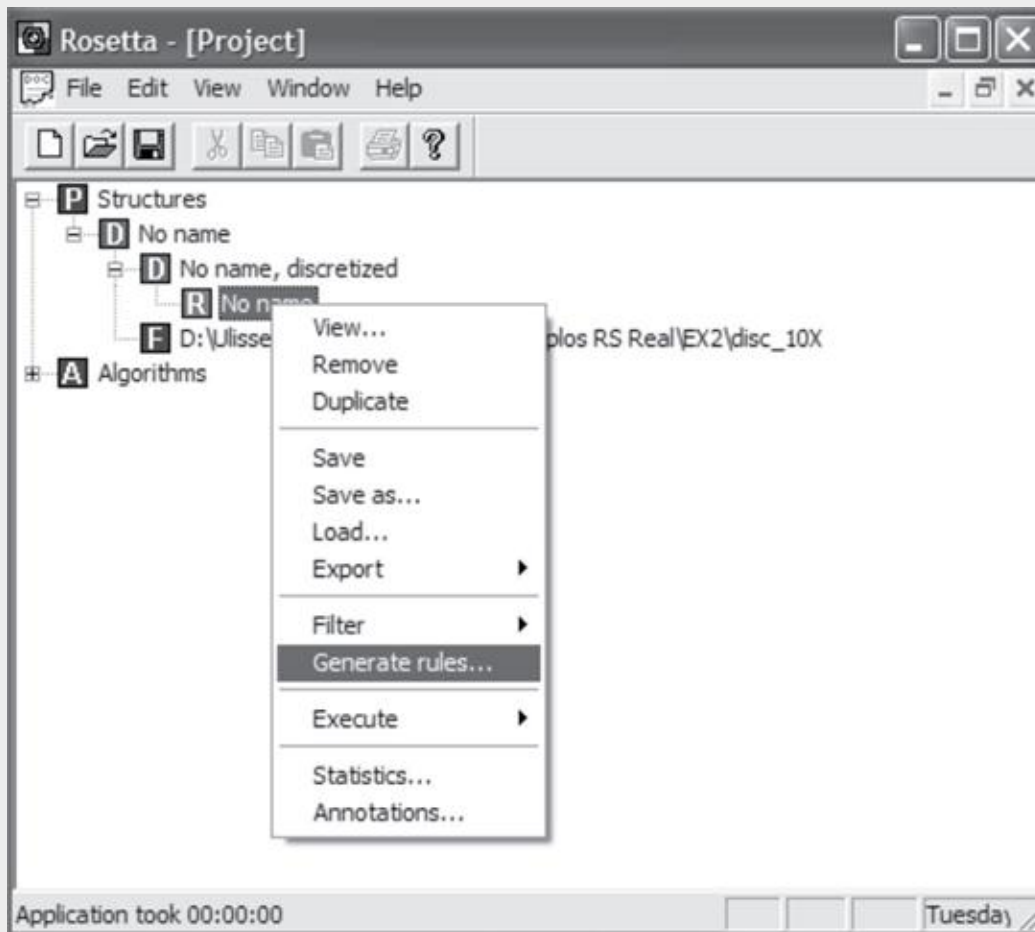
Se fosse necessário “discretizar” os dados do *SI* (*não é o caso deste exemplo*), os procedimentos estão ilustrados nas duas telas a seguir.



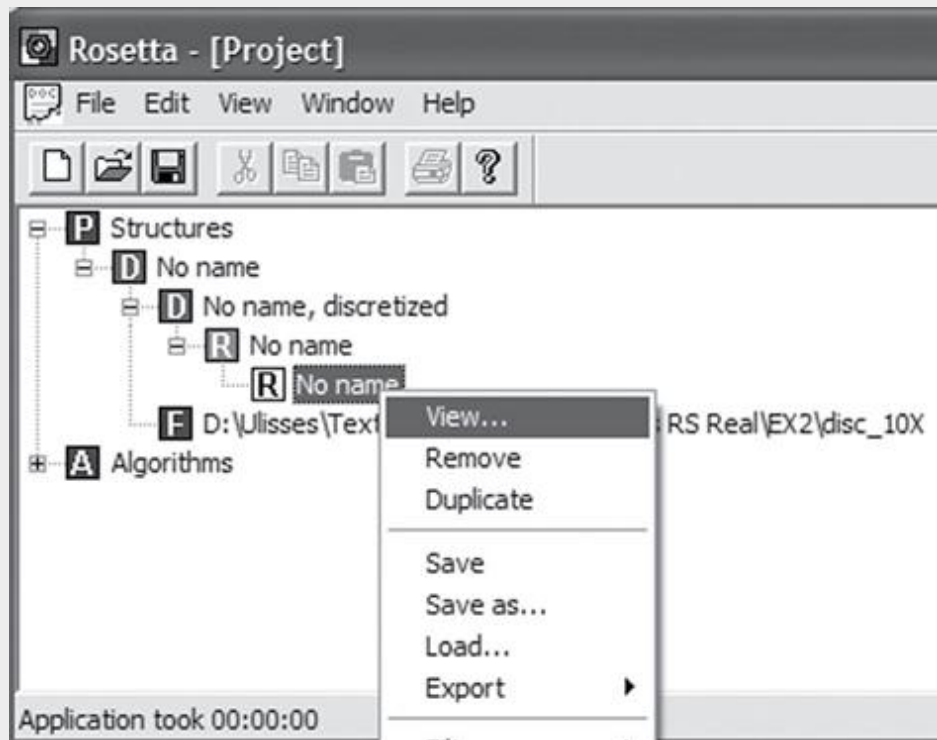
A próxima etapa consiste na obtenção dos redutos, que pode ser realizada conforme ilustrado na tela abaixo.



Os redutos obtidos {MBA, GRAD} e {MBA, EXP} podem ser visualizados na aba “R” com a opção “View”. As regras de decisão podem ser geradas na forma ilustrada na janela a seguir.



As regras geradas podem ser visualizadas na aba “R”, em cor amarela, conforme ilustrada na janela abaixo.



A listagem a seguir exemplifica as regras de decisão obtidas pelo aplicativo. O termo *IF* das regras está implícito, o símbolo “=” é representado por “()” e “= >” representa o termo *THEN* na representação das regras.

	LHS Support	RHS Accuracy	LHS Coverage
MBA(N) AND GRAD(A) => CONTRAT(Maior)	2	1,0	0,4
MBA(S) AND GRAD(A) => CONTRAT(Não)	1	1,0	0,2
MBA(N) AND GRAD(C) => CONTRAT (Baixa) OR CONTRAT(Média)	2	0,5, 0,5	0,4
MBA(N) AND EXP(S) => CONTRAT(Maior)	1,0	2	0,4
MBA(S) AND EXP(S) => CONTRAT(Não)	1,0	1	0,2
MBA(N) AND EXP(N) => CONTRAT (Baixa) OR CONTRAT(Média)	2	0,5, 0,5	0,4

As informações contidas nas colunas à direita da listagem são as seguintes: as informações “LHS Support”, “RHS Accuracy” e “LHS Coverage” são medidas referentes às regras de decisão no formato $\alpha \rightarrow \beta$; LHS Support indica o número de regras que possuem as propriedades α e β ; RHS Accuracy relaciona o quão fidedigna é a regra de decisão β em base da evidência α ; LHS Coverage mede quão bem a evidência α descreve a

decisão referente à β ; geralmente, quanto maior a quantidade de antecedentes das regras, menor o índice LHS Coverage e maior a medida RHS Accuracy; em geral as regras de maior importância são as que apresentam maiores valores de suporte e de precisão (*accuracy*).

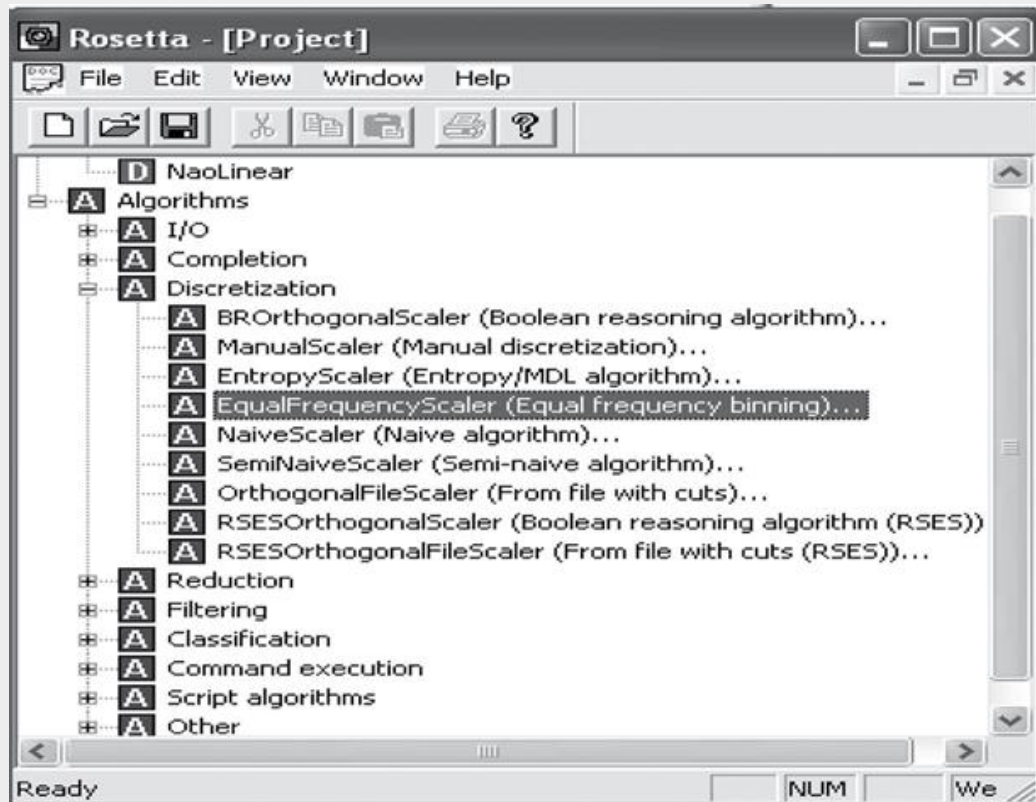
Anexo 6.2(a)

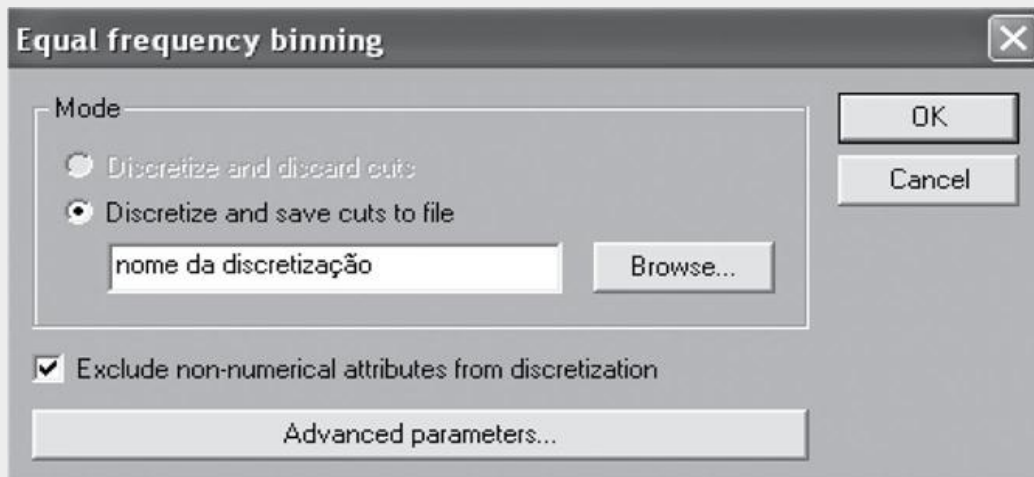
Resolução do Exemplo 6.4 com a ferramenta Rosetta

Os dados do problema devem estar associados a arquivos ou planilhas correspondentes. Por exemplo, um arquivo tipo texto (.txt) com as variáveis do sistema declaradas como *float* neste modelo. A planilha abaixo ilustra os dados do exemplo.

x	y
Float(4)	Float(4)
0.0000	2.0000
0.2500	2.2197
	:
	:
3.5000	4.0952
3.7500	4.2879
	:
	:
9.7500	10.9779
10.0000	11.0000

Na utilização do aplicativo são empregados os mesmos procedimentos (e as mesmas janelas e notações) exemplificados no [Anexo 6.1](#). No processo de discretização foi utilizada a opção *Equal frequency binning*, conforme ilustrado nas figuras seguintes.





Na tela anterior (na aba *Advanced parameters*) foram selecionados três (3) níveis de discretização para o problema em questão.

A figura a seguir ilustra a janela de visualização das regras de decisão geradas, onde o símbolo “*” nos intervalos “[)” da variável x denota os extremos dos valores dessa. Neste exemplo os dados são 0 e 10, respectivamente.

	Rule	LHS Support
1	$x([*, 3.3750)) \Rightarrow y(2.0000) \text{ OR } y(2.2197) \text{ OR } y(2.3811) \text{ OR } y(2.5136) \text{ OR } y(2.7310) \text{ OR } y(2.9484)$	14
2	$x([3.3750, 6.8750)) \Rightarrow y(4.0952) \text{ OR } y(4.2879) \text{ OR } y(4.4000) \text{ OR } y(4.8764) \text{ OR } y(5.2843) \text{ OR } y(5.6922)$	14
3	$x([6.8750, *]) \Rightarrow y(10.4000) \text{ OR } y(10.6437) \text{ OR } y(10.4786) \text{ OR } y(10.4928) \text{ OR } y(10.7082) \text{ OR } y(10.9224)$	13

Anexo 6.2(b)

Programa de simulação referente ao Exemplo 6.4

```

% Simulação do exemplo de aproximação de função via regras aproximadas
clear all;

%Carga dos vetores de dados da função a ser aproximada.
load vetx; load vety;

%Faixas dos atributos das regras.
XA=0; XB=3.375; XC=6.875; XD=10;
Ymi1=2;Ymi2=4.0952;Ymi3=10.4;
Yma1=3.8419;Yma2=10.042;Yma3=11;

ns = size(vetx); nv = ns(1,1);

for nn = 1:nv, %Rotina de processamento.
    x = vetx(nn); y = vety(nn); % Valores das variáveis.

    % Regras do modelo Rough e interpolações.
    ya1=0;ya2=0;ya3=0;

    if ((x >= XA) & (x <= XB))
        ya1 = Ymi1+(Yma1-Ymi1)*(x-XA)/(XB-XA);
    end

    if ((x >= XB) & (x <= XC))
        ya2 = Ymi2+(Yma2-Ymi2)*(x-XB)/(XC-XB);
    end

    if ((x >= XC) & (x <= XD))
        ya3 = Ymi3 +(Yma3-Ymi3)*(x-XC)/(XD-XC);
    end

    % Média dos valores das regras ativadas nos extremos das faixas.
    ya=0;na=0;
    if (ya1 ~= 0)
        ya = ya + ya1;
        na = na + 1;
    end
    if (ya2 ~= 0)
        ya = ya + ya2;
        na = na + 1;
    end
    if (ya3 ~= 0)
        ya = ya + ya3;
        na = na + 1;
    end

    if (na == 0)
        na = 1;
    end
end

```

```
end
ya = ya / na;
vetya(nn) = ya;
end

%Gráfico da função original e do modelo rough associado às regras.
plot(vetx,vety,'k.',vetx,vetya,'k');grid; xlabel('x1');ylabel('y');
```

Anexo 6.3

Programa de algoritmo básico de PSO

```

% Exemplo de algoritmo de PSO
% (incluir a Função de Custo em uma function)
clear all;

% Parâmetros do Algoritmo
numero_particulas = 40; % Número de partículas
numero_passos = 100;    % Número de passos do algoritmo
d=1;                    % Número de variáveis
c1=2;                   %Parâmetro cognitivo
c2=2;                   %Parâmetro social
v_max=10; % Velocidade máxima

% Inicialização das partículas
for i=1:numero_particulas
    p(i,:) = rand(1,d);

    % Função de Custo
    fitness(i)= Custo(p(i,:));
    v(i,d)= 0;

    melhor_fitness(i)= fitness(i);
    melhor_local(i,:)= p(i,:);
end

[fitmin,I]= min(fitness);
melhor_global= p(I,:);

% Deslocamentos e velocidades das partículas
for i=1:numero_passos

    for j=1:numero_particulas
        v(j,:)= v(j,:)+rand*c1*(melhor_local(j,:)-
            p(j,:))+rand*c2*(melhor_global-p(j,:));
        v(j,:)= max(min(v_max,v(j,:)), -v_max);
        p(j,:)= p(j,:)+v(j,:);

        % Avaliando a Função de Custo
        fitness(j)= Custo(p(j,:));
        if fitness(j)<=melhor_fitness(j)
            melhor_fitness(j)= fitness(j);
            melhor_local(j,:)= p(j,:);
        end
    end
end

```

```

for j=1:numero_particulas
    if fitness(j)<=fitmin
        fitmin=fitness(j);
        melhor_global=p(j,:);
    end
end

mean_fitness(i)=mean(fitness);
best_fitness(i)=fitmin;

end

% Gráficos dos valores das iterações e dos fitness.
iter=1:i;
subplot(211);
plot(iter,best_fitness,'k. '); grid;
xlabel('Iterações'); ylabel('Best Fitness');
subplot(212);
plot(iter,mean_fitness,'b. '); grid;
xlabel('Iterações'); ylabel('Mean Fitness');

% Valores Resultantes
x = melhor_global
fval = fitmin

```

CAPÍTULO

CINCO

Sistemas Neuro-Fuzzy

5.1 Introdução

Neste capítulo será apresentada a conceituação básica sobre sistemas híbridos especificamente os sistemas Neuro-Fuzzy. Sabe-se que os sistemas *fuzzy* são apropriados para modelagens a partir de conhecimentos explícitos fornecidos por especialistas humanos, possibilitando interpretações linguísticas dos conhecimentos associados. Já as redes neurais são adequadas à criação de modelos baseados em conhecimentos implícitos, embutidos nos conjuntos de dados utilizados nas modelagens correspondentes. Várias pesquisas foram realizadas no sentido de integrar essas duas técnicas de IA, cujo objetivo consiste em gerar modelos híbridos que possam aproveitar as vantagens de cada uma delas e minimizar suas limitações individuais. Nos últimos anos diversas estruturas de sistemas híbridos foram desenvolvidas, geralmente pela integração das técnicas como Algoritmos Genéticos (GA), Redes Neurais Artificiais (RNA) e Lógica Fuzzy.

Nas redes neurais o conhecimento é representado pelas interconexões das unidades estruturais utilizadas. Uma RNA é um modelo não linear cujos parâmetros são determinados a partir de um conjunto de dados de suporte e por meio de algoritmos de auto-organização ou de treinamento. Embora apresente bons resultados numéricos na solução de vários problemas, uma RNA geralmente não permite a incorporação do conhecimento de especialistas numa determinada aplicação, podendo fazer com que o modelo resultante seja centrado nos dados ([Evsukoff e Almeida, 2005](#)).

Por outro lado, em um modelo *fuzzy*, representações de conhecimento podem ser associadas a regras ou tabelas de forma explícita, como em um sistema de apoio a decisões, por exemplo. Em sistemas *fuzzy*, os dados ou as informações utilizadas podem conter incertezas que são tratadas adequadamente por meio de funções de pertinência.

Assim, levando-se em conta as características da lógica difusa e os modelos conexionistas, diversos autores (Nauck *et al.*, 1997; Azevedo *et al.*, 2000; Kasabov e Song, 2001) mostraram combinações que procuram aproveitar suas potencialidades e contornar suas limitações, proporcionando abordagens que conseqüentemente promovem mais robustez nas modelagens resultantes.

Para se referir aos modelos que integram os paradigmas dos sistemas *fuzzy* e neurais, cunhou-se o termo Neuro-Fuzzy. Nauck *et al.* (2007) apresentaram uma taxonomia para as diferentes combinações de redes neurais e sistemas *fuzzy*:

- Redes neurais *fuzzy* – nesta combinação métodos *fuzzy* são usados para aperfeiçoar as capacidades de aprendizado ou de desempenho de uma RNA. Por exemplo, podem-se usar regras *fuzzy* para mudar a taxa de aprendizado do procedimento de treinamento da rede em questão.
- Sistemas concorrentemente neural-*fuzzy* – nesses sistemas uma RNA e um sistema *fuzzy* trabalham concorrentemente na mesma tarefa, mas sem a influência de um sobre o outro, ou seja, nenhum dos dois sistemas é usado para determinar os parâmetros do outro. A finalidade é a obtenção de resultados concorrentes relativos a uma determinada aplicação.
- Sistemas Neuro-Fuzzy cooperativos – os parâmetros do sistema *fuzzy* (regras, ponderações das regras ou conjuntos nebulosos) são determinados por uma RNA. Os sistemas trabalham independentes e com tarefas distintas.
- Sistemas Neuro-Fuzzy híbridos – uma RNA e um sistema *fuzzy* são combinados em uma arquitetura homogênea. O sistema pode ser interpretado como uma RNA especial ou como um sistema *fuzzy* parametrizado por uma RNA. A maioria dos modelos propostos na literatura pertence a essa categoria.

Wang e Mendel (1992) introduziram um algoritmo que permite a geração de modelos *fuzzy* em processos de modelagem genéricas. Na primeira etapa do algoritmo, os dados do processo são divididos em partições para definição dos conjuntos difusos. Na segunda etapa, as regras *fuzzy* são inicialmente geradas. Na terceira etapa define-se um grau de representação para cada regra gerada. Na quarta etapa, a base de conhecimento é refinada. Na quinta parte ocorre a inferência do valor do modelo para as entradas correspondentes, definindo-se o modelo em questão. Apesar de simples, o processo de determinação de modelos *fuzzy* pelo método descrito pode ser bastante tedioso e demorado.

Com a finalidade de diminuir esforços computacionais, gerar modelos mais eficientes e de forma automática, Jang *et al.* (1997) apresentaram uma estrutura Neuro-Fuzzy com esse objetivo. A ideia básica consiste em realizar sistemas de inferência *fuzzy* (denominados FIS) sobre arquiteturas de redes neurais artificiais, caracterizando um sistema híbrido. O conceito consiste em utilizar paradigmas de aprendizado comuns às redes neurais para a geração de modelos difusos. Dessa forma, aproveitam-se os benefícios das redes neurais que são apropriadas para a criação de modelos a partir de conhecimento implícito embutido em um conjunto de dados, e dos sistemas *fuzzy*, que são adequados para a criação de modelos a partir de um conhecimento explícito.

Dentre os vários sistemas Neuro-Fuzzy propostos, pode-se destacar o Anfis (*Adaptive Network Based Fuzzy Inference System*), o FSOM (*Fuzzy Self Organization Map*), o NEFClass (*Neuro Fuzzy Classification*), o Nefcon (*Neuro Fuzzy Control*), o Nefprox (*Neuro Fuzzy Function Approximation*) e o Denfis (*Dynamic Evolving Neural-Fuzzy Inference System*). O FSOM proposto por Mitra e Pal (1994), é um sistema Neuro-Fuzzy baseado no modelo de mapas auto-organizáveis de Kohonen, capaz de realizar classificações por meio da aplicação de técnicas de lógica nebulosa. A arquitetura de uma FSOM é composta por um conjunto de cinco camadas cujo funcionamento é semelhante à de um Anfis (que foi descrito no Capítulo 2 deste livro). O NEFClass, proposto por Nauck e Kruse (1995), é um sistema Neuro-Fuzzy cuja aplicação básica é em sistemas de classificação. A diferença principal entre o NEFClass e o FSOM é que o primeiro apresenta valores *crisp* na camada de saída que generaliza o formato das regras *fuzzy* que compõem os modelos gerados pelo NEFClass. Nefcon é dedicado a aplicações em sistemas de controle utilizando regras tipo Mandami. Kasabov e Song (2001) propuseram o Denfis, que consiste em um sistema Neuro-Fuzzy com a capacidade de adaptação dinâmica, adicionando novas regras ao FIS antes ou durante o processo de treinamento, e removendo regras desnecessárias durante ou após o processo de treinamento.

As referências citadas a seguir mostram algumas das aplicações realizadas com sucesso, via sistemas Neuro-Fuzzy. Fahimifard *et al.* (2009) realizaram um estudo de desempenho entre métodos lineares e não lineares quando aplicados em processos de previsão. O modelo estatístico considerado foi o Arima, e o não linear, o Anfis, em que uma série temporal associada à produção agrícola foi considerada. O processo levou em conta a previsão em

três horizontes distintos com uma, duas e quatro semanas, respectivamente. A análise dos resultados permitiu concluir que o Anfis obteve um desempenho consideravelmente maior em todos os experimentos realizados em relação ao modelo estatístico convencional.

Liao e Tsao (2007) propuseram um sistema híbrido baseado em lógica nebulosa e algoritmos genéticos para a previsão de demanda de carga elétrica de curto prazo. A finalidade da aplicação do GA foi de estimar os parâmetros do modelo *fuzzy* de previsão. Como forma de avaliação do modelo obtido uma comparação foi realizada com modelos obtidos via aplicação de redes neurais. Os modelos baseados em lógica nebulosa obtiveram resultados mais precisos. De forma similar, Syed-Ahmad *et al.* (2007) e Ying e Pan (2008) utilizaram-se de modelos *fuzzy* para prever demanda de carga elétrica. O primeiro a curto prazo e o segundo para longo prazo. Syed-Ahmad *et al.* (2007) consideraram observações horárias de consumo de energia elétrica para compor o conjunto de treinamento usado para treinamento do Anfis. O modelo *fuzzy* obtido foi aplicado na previsão de demanda vinte e quatro horas à frente. Ying e Pan (2008) utilizaram observações anuais de 1981 até 1996 para previsão de carga de 1997 até 2000, em diferentes regiões de Taiwan. Em ambos os casos os modelos obtidos via Anfis obtiveram resultados compatíveis com os modelos neurais usados para fins de comparação. Chen Ying e Pan (2010), a partir do uso do Anfis, estimaram um modelo *fuzzy* capaz de prever a quantidade de turistas que chegavam a Taiwan via Estados Unidos, Japão e Hong Kong. Para construção do modelo, um conjunto de treinamento que levava em consideração observações mensais, que iam de 1989 a 2000, foi construído. O objetivo consistia em estimar o número de turistas para os anos de 2001 a 2003, os resultados obtidos apresentaram uma boa consistência com os dados reais.

En-Anfis foi o nome dado ao sistema híbrido proposto por Chen e Zhang (2005) para previsão de séries temporais. Este modelo utiliza o paradigma de composição *Ensemble*, que consiste em treinar vários componentes para a realização da mesma tarefa. O treinamento de cada componente pode ser realizado por meio de conjuntos construídos de forma aleatória ou por meio do método *bootstrap sampling*, com ou sem repetições. O valor de previsão é o resultado da combinação do resultado de previsão de cada componente isolado, sendo que essa combinação pode ser baseada em ponderações ou não. Caso a opção por pesos seja a considerada, atribuem-se valores maiores a

resultados de previsão de componentes que apresentaram menores erros durante o processo de treinamento.

Previsões da condição de luminosidade em ambientes como salas ou cômodos de apartamentos foram realizadas com modelos Neuro-Fuzzy, que foram utilizados no projeto de controladores de iluminação automático (Kurian *et al.*, 2006), no qual desenvolveu-se modelos *fuzzy* via Anfis para previsão de um, seis e dez passos à frente. O conjunto de treinamento foi construído com o auxílio de um software de simulação que emulava as condições de iluminação de um ambiente segundo parâmetros como dimensão da sala, obstruções, posicionamento de móveis, condições de luminosidade externa, dentre outros. Os controladores obtidos apresentaram boa capacidade para se ajustar adequadamente às condições de luminosidade. Ming-Bao e Xin-Ping (2008) aplicaram um sistema híbrido composto por Anfis e GA para prever o fluxo de tráfego de automóveis. Para obtenção dos dados de treinamento, um software de simulação de propósito específico foi utilizado para gerar novecentas observações de fluxo diversas. Das novecentas observações, oitocentas foram usadas para criação do conjunto de treinamento e cem foram usadas para compor o conjunto de testes. Os dados de treinamento foram agrupados por meio do algoritmo “Subtractive Clustering” e, com base nesse agrupamento, o processo de treinamento foi inicializado. A definição de um raio de agrupamento ótimo foi realizada por meio da aplicação de um algoritmo genético. Os resultados foram comparados aos resultados obtidos via aplicação de modelos gerados por rede neural ou lógica difusa distintamente. O sistema híbrido apresentou melhores resultados. Faustino (2011) e Faustino *et al.* (2011, 2012) realizaram previsões de diferentes tipos de séries temporais utilizando modelos obtidos via Anfis e por regras aproximadas (*rough rules*), fazendo comparações com modelos neurais e estatísticos. Na maioria dos casos testados os melhores resultados foram obtidos com modelos difusos.

5.2 Características dos sistemas Neuro-Fuzzy

Redes neurais e sistemas *fuzzy* têm sido aplicados a muitos problemas incluindo identificação de sistemas, previsões de séries temporais, classificações de padrões, sistemas de controle etc.

Um sistema Neuro-Fuzzy é um sistema que usa um algoritmo de aprendizado derivado ou inspirado nas redes neurais artificiais ou em algoritmos genéticos para determinar seus parâmetros, como funções de pertinência, ponderações de regras etc.

Os sistemas Neuro-Fuzzy são normalmente modelados como redes neurais multicamadas, nos quais os procedimentos de *fuzificação* de dados por arquiteturas de redes neurais são também considerados (Oliveira *et al.*, 2011). Em redes Neuro-Fuzzy os pesos das ponderações, propagação de erros e outros detalhes diferem das redes neurais convencionais. Existem abordagens diferentes, e o termo sistema Neuro-Fuzzy é usado para abordagens que apresentam as seguintes propriedades:

- Um sistema Neuro-Fuzzy está baseado em um sistema fuzzy, que é treinado por um algoritmo de aprendizagem, derivado de procedimentos de treinamento das redes neurais. O procedimento opera com informações locais que causam apenas modificações nos parâmetros locais do modelo fuzzy correspondente.
- Um sistema Neuro-Fuzzy pode ser visto como uma rede neural de três camadas. A primeira camada representa as variáveis de entrada, a camada do meio (escondida) representa as regras *fuzzy* e a terceira camada representa as variáveis de saída do sistema, na qual os conjuntos nebulosos são codificados como pesos das conexões associadas. Algumas vezes uma arquitetura de cinco camadas é usada, na qual os conjuntos nebulosos são representados em unidades da segunda e quarta camadas.
- Um sistema Neuro-Fuzzy pode ser sempre interpretado como um sistema de regras *fuzzy*. Mas nem todos os modelos neuro-fuzzy especificam procedimentos de aprendizagem para criação de regras *fuzzy*.
- O procedimento de aprendizagem de um sistema Neuro-Fuzzy torna as propriedades semânticas de um sistema fuzzy em uma descrição que resulta em restrições em possíveis modificações aplicáveis aos parâmetros do sistema. Mas nem todas as abordagens Neuro-Fuzzy possuem essa propriedade;
- Um sistema Neuro-Fuzzy aproxima uma função n -dimensional parcialmente definida pelos dados de treinamento, em que as regras *fuzzy* codificadas pelo sistema representam exemplos que podem ser vistos como protótipos dos dados de treinamento.

5.3 Vantagens e desvantagens dos sistemas Neuro-Fuzzy

Como vantagens dos sistemas Neuro-Fuzzy pode-se citar:

- tratamento de vários tipos de informações (numérica, linguística etc.);
- capacidade de lidar com incertezas;
- colaboração e agregação das características dos modelos *fuzzy* e neurais;
- capacidade de auto-aprendizado e de auto-organização;
- não necessidade de conhecimentos prévios sobre os dados de treinamento.

Como desvantagem dos sistemas Neuro-Fuzzy pode-se citar:

- mais complexidade nas implantações dos modelos correspondentes.

5.4 Arquitetura Neuro-Fuzzy

Um sistema Neuro-Fuzzy ([Figura 5.1](#)) consiste de componentes convencionais de um sistema *fuzzy*, em que a computação de cada estágio é realizada por uma camada de neurônios associada ([Oliveira et al., 2011](#)).



FIGURA 5.1 Diagrama de um sistema Neuro-Fuzzy. Fonte: [Oliveira et al., 2011](#).

Um sistema Neuro-Fuzzy típico pode ser dividido basicamente em três camadas: camada de *fuzificação*; camada de inferência; camada de *defuzificação*. Na camada de *fuzificação*, cada neurônio representa uma função de pertinência (*membership*) associada a uma informação de entrada do sistema. Na camada de inferência, as regras são ativadas, e os valores ao final de cada regra representam o peso inicial da regra que será ajustado via treinamento. A camada de regras contém um nó, onde para cada regra são aplicadas operações *fuzzy* como AND/MIN e OR/MAX. Os operadores AND e OR podem ser implementados em camadas diferentes ([Lee et al., 1996](#); [Liu e Lu, 1995](#)), ou na mesma camada ([Higgins e Goodman, 1994](#)). Na camada de *defuzificação*, cada neurônio representa uma proposição consequente

associada à regra correspondente do modelo, e suas funções de pertinência podem ser modeladas combinando uma ou duas funções de ativação lineares ou não. A camada de saída transforma a informação nebulosa correspondente conforme a estratégia de *defuzzificação* utilizada. O peso de cada saída representa o centro de gravidade de cada função de pertinência ativada. A Figura 5.2 exemplifica um processamento realizado em um determinado sistema Neuro-Fuzzy.

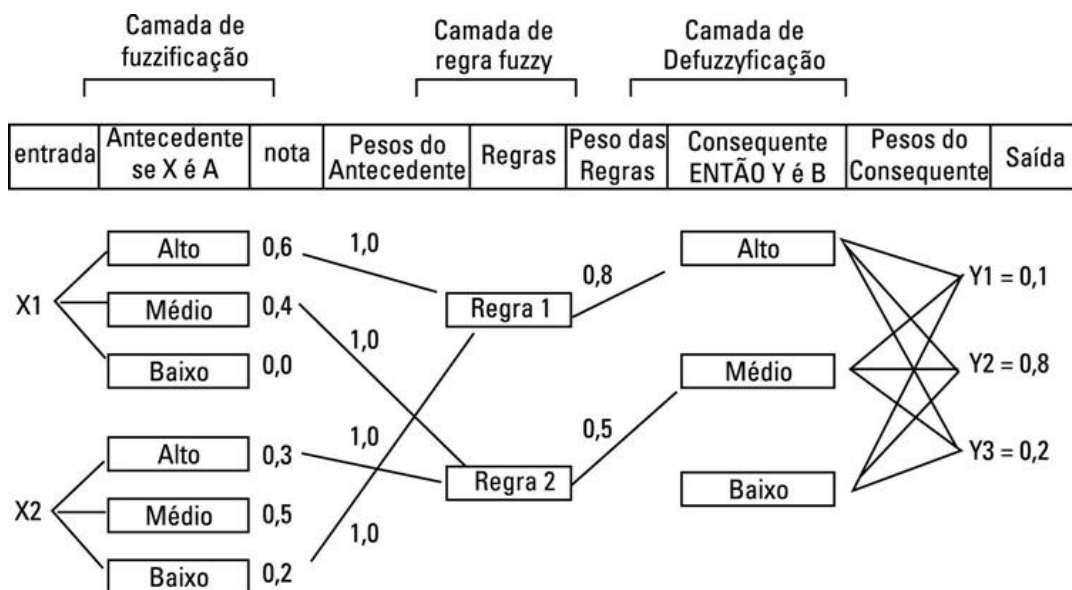


FIGURA 5.2 Exemplificação de processamento de um sistema neuro-fuzzy. Fonte: Oliveira et al., 2011.

5.5 O algoritmo Fuzzy C-Means

Um modelo Anfis pode utilizar alguma técnica de agrupamento como um processamento prévio para a obtenção da modelagem Neuro-Fuzzy correspondente. Entre os algoritmos de agrupamento utilizados tem-se o algoritmo Fuzzy C-Means, que é utilizado para determinar agrupamentos segundo a norma euclidiana existente entre um determinado dado e os centros dos agrupamentos resultantes. O conceito básico entre a relação de dados e um determinado agrupamento correspondente pode ser descrito como: quanto mais próximo do centro de um agrupamento o dado estiver, maior será seu grau de pertinência a esse agrupamento. Considerando um grupo de dados:

correspondente às informações de treinamento $X = x_1, \dots, x_n$, onde cada ponto x_k ($k = 1, \dots, n$) é um vetor em R^p , e considerando U_{cn} como um grupo de matrizes reais $c \times n$, sendo c um inteiro que obedece à relação $2 \leq c < n$, define-se a partição *fuzzy* para o conjunto X segundo a equação (5.1).

$$M_{fuc} = U \in U_{cn} : u_{ik} \in [0,1], \quad \sum_{i=1}^c u_{ik} = 1 < \sum_{k=1}^n u_{ik} < n \quad (5.1)$$

Onde U_{ik} é o grau de pertinência de x_k em um agrupamento i ($i = 1, \dots, c$). A equação (5.2) pode ser utilizada para determinar se o algoritmo Fuzzy C Means encontrou uma partição ótima.

$$J_m(U, V, X) = \sum_{k=1}^n \sum_{i=1}^c (u_{ik})^m \|x_k - v_i\|^2 \quad (5.2)$$

Da equação (5.2) tem-se que $V = (v_1, \dots, v_c)$ é a matriz que contém os centros dos agrupamentos, onde todo $v_i \in R^p$ é um peso constante entre $[1, \infty)$. O centro $v_i^{(t)}$ de cada agrupamento i ($i = 1, \dots, c$) para uma interação pode ser calculado segundo a equação (5.3).

$$v_i^{(t)} = \frac{\sum_{k=1}^n (u_{ik}^t)^m x_k}{\sum_{k=1}^n (u_{ik}^t)^m} \quad (5.3)$$

De posse dos novos centros $v_i^{(t)}$, os valores de pertinência podem ser obtidos pela equação (5.4).

$$u_{(ik)}^{(t+1)} = \frac{1}{\sum_{j=1}^c \left(\frac{\|x_k - v_i^{(t)}\|^2}{\|x_k - v_j^{(t)}\|^2} \right)^{\frac{2}{m-1}}} \quad (5.4)$$

De forma simplificada, um algoritmo Fuzzy C-Means pode ser descrito conforme segue:

1. Escolher valores para c , m e ε .
2. Gerar a partição fuzzy U^0 de forma aleatória, obedecendo às restrições impostas.
3. Atribuir um valor nulo ao contador de interações.
4. Fazer $J_m^{(0)} = 0$.
5. Calcular os centros $v_i^{(t)}$ segundo a equação correspondente.
6. Avaliar a função objetivo $J_m^{(t+1)}$.
7. Determinar os graus de pertinência $U_{ik}^{(t+1)}$.
8. Calcular $\delta = J_m^{(t+1)} - J_m^{(t)}$.
9. Incrementar o contador de interações.
10. Se a condição de saída for falsa, então se retorna ao quinto passo do algoritmo. Senão, finaliza-se o procedimento. A condição de parada do algoritmo é atingida quando $\delta \leq \varepsilon$, ou quando um determinado número de interações é alcançado.

5.6 Aplicações dos sistemas Neuro-Fuzzy

Os exemplos a seguir ilustram algumas aplicações de sistemas Neuro-Fuzzy.

Exemplo 5.1

A utilização de sistemas de IA híbridos na previsão de séries temporais tem se mostrado bastante eficaz devido a sua capacidade de mapeamento não linear eficiente entre as grandezas envolvidas, permitindo a extração de características complexas, o que possibilita muitas vezes mais confiabilidade nas previsões. A base de dados da série utilizada neste

exemplo foi extraída de uma competição organizada pela *European Network of Intelligent Technologist* (Eunite). A série coletada contém informações de cargas elétricas medidas durante todo um dia, na qual cada observação corresponde a um horário específico. A série contém valores de cargas elétricas diárias dos anos de 1997 e 1998. Os horários utilizados como intervalos entre as observações apresentam-se com variação de meia hora durante todo um dia, iniciando às 00h30 e terminando às 24h. Um pré-processamento de dados foi realizado por intermédio da média dos valores dos horários de pico, gerando-se os valores diários utilizados no exemplo.

Neste exemplo será utilizado um modelo Anfis com funções de pertinência tipo *gbellmf* e regras *fuzzy* Takagi-Sugeno. A janela temporal usada no modelo foi de sete amostras. A modelagem Anfis foi obtida via agrupamento Fuzzy C-Means. A [Figura 5.3](#) apresenta o gráfico com valores resultantes do treinamento do modelo Anfis referente à série de dados de carga da competição Eunite. A [Figura 5.4](#) exemplifica os valores utilizados na validação do modelo obtido.



FIGURA 5.3 Gráfico dos valores referentes ao treinamento do modelo Anfis da série.



FIGURA 5.4 Gráfico de validação de modelo Anfis da série Eunite.

A [Figura 5.5](#) ilustra o resultado do modelo de Anfis com os dados do subconjunto de teste, com a previsão de um passo à frente. O valor do erro percentual absoluto médio (Mape) foi de 2,27%.



FIGURA 5.5 Gráfico de previsão de um passo a frente do modelo Anfis da série Eunite.

Exemplo 5.2

Neste segundo exemplo será apresentada uma abordagem para previsão de séries temporais baseadas em técnica de *Ensemble*, por meio de técnicas de IA, particularmente com sistemas híbridos Neuro-Fuzzy. O objetivo consiste na construção de modelos que sejam capazes de descrever o comportamento de determinadas séries temporais e prever seus valores futuros. As técnicas de *Ensembles* são essencialmente combinações de vários componentes de previsores distintos, com a finalidade de se obter melhores previsões que as obtidas pelos componentes individuais em questão.

A aplicação da técnica *ensemble* tem se mostrado uma maneira de melhorar a precisão de séries temporais. As combinações de métodos de previsão individuais, como Redes Neurais Artificiais com topologias distintas (MLP-*Multi Layer Perceptron*, RBF- *Radial Base Function* etc.), e até mesmo com modelos *fuzzy*, podem ser consideradas com o objetivo de melhorar as precisões nas previsões almejadas.

A palavra *Ensembles* origina-se do inglês e significa “agrupamentos”. Essa técnica resulta do trabalho de [Hansen e Salamon \(1990\)](#), que demonstrou que a habilidade de generalização pode ser significativamente melhorada por meio da composição de vários modelos distintos, ou seja, a partir de modelos independentes é realizada a composição das saídas individuais. Ela consiste na combinação dos resultados de vários previsores (ou componentes) distintos para a obtenção de uma saída que seja melhor (ou mais estável) que as obtidas pelos componentes individuais. A capacidade de generalização de um modelo neural pode ser melhorada combinando várias redes neurais, como, por exemplo, treinando as redes individualmente e combinando os seus resultados de alguma forma. Em geral, um *Ensemble* de redes neurais é construído em duas etapas. Na primeira, um determinado número de redes neurais é treinado e, na segunda, os resultados das redes são combinados.

A modelagem *Ensemble* proposta neste exemplo utiliza modelos híbridos Neuro-Fuzzy. A série temporal a ser exemplificada refere-se ao preço de arroba de boi cujos valores estão ilustrados na [Figura 5.6](#), onde há registros de valores (em reais) relativos a 73 meses.



FIGURA 5.6 Dados da série de preço de arroba de boi.

A definição do modelo *Ensemble* Anfis utilizada neste exemplo é composta por três modelos difusos, e cada modelo utiliza funções de pertinências distintas (trapezoidal, gaussiana e triangular), gerando submodelos difusos via *toolbox Anfis* do Matlab, que depois são carregados e gravados com o *toolbox fuzzy* (essas duas ferramentas foram descritas no [Capítulo 2](#)). Na segunda etapa é gerado um modelo difuso que agrega (pondera) as saídas de dados dos submodelos em relação às entradas de treinamento. A função de pertinência parametrizada nessa composição será aquela que apresentar menor erro percentual absoluto médio (Mape). Utilizando o *toolbox fuzzy*, o modelo da composição do *Ensemble* é carregado e gravado. Finalmente, a última etapa testa a modelagem do *Ensemble* Neuro-Fuzzy resultante a partir do modelo gravado na etapa composição, na qual são utilizados os dados de teste para validação da estimação. A [Figura 5.7](#) apresenta a estrutura do modelo *Ensemble* Neuro-Fuzzy proposto neste exemplo.



FIGURA 5.7 Estrutura do Ensemble via modelos Anfis.

Os dados utilizados na etapa de treinamento do *Ensemble* em questão não sofreram qualquer tipo de normalização. Para esse modelo foram consideradas informações de preço com uma janela temporal de cinco atrasos. Foi empregado o algoritmo de agrupamento Fuzzy C-Means. Os três submodelos Anfis que compuseram o *Ensemble* utilizaram as funções de pertinências distintas (trapezoidal, gaussiana e triangular) para cada componente individual, e obtiveram os valores de Mape de 3,7%, 3,5% e 3,6%, respectivamente ([Tabela 5.1](#)). A tabela indica os erros da previsão dos modelos considerados nesse exemplo. O *Ensemble* com arquiteturas Anfis considerado apresentou um erro de modelagem (Mape = 3,3 %) compatível com a literatura, em que a composição resultante forneceu um erro inferior aos das componentes individuais do modelo.

Tabela 5.1

Valores de Mape dos modelos Anfis de previsões para um à frente

Topologias	Mape%
Submodelo Anfis – Trapezoidal	3,7
Submodelo Anfis – Gaussiana	3,5
Submodelo Anfis – Triangular	3,6
Composição – Gaussiana	3,4
Ensemble Anfis – Teste	3,3

Referências bibliográficas

1. AZEVEDO FM, BRASIL LM, OLIVEIRA RCL. Redes neurais com aplicações em controle e em sistemas especialistas. *Visual Books Editora* 2000.
2. CHEN, D.W.; ZHANG, J.P. **Time series prediction based on ensemble Anfis**. Proceedings of the Fourth International Conference on Machine Learning and Cybernetics, 2005, p. 18-21.
3. CHEN, M.S.; YING, L.C.; PAN, M.C. **Forecasting tourist arrivals by using the adaptive network-based fuzzy inference system**. Experts Systems with Applications, v. 37, 2010, p. 1.185-91.
4. EVSUKOFF A, ALMEIDA PEM. *In Sistemas inteligentes: fundamentos e aplicações*. Manole Ltda 2005; p. 203- 24.
5. FAHIMIFARD S. Application of anfis to agricultural economic variables forecasting case study: Poultry retail price. *Journal of Artificial Intelligence*. 2009;2(2):65–72.
6. FAUSTINO, C.P. (2011). **Previsão de séries temporais via modelos baseados em regras**. Dissertação de Mestrado, Universidade Federal de Itajubá. Disponível em: <http://juno.unifei.edu.br/bim/0038068.pdf>.
7. FAUSTINO CP, PINHEIRO C, CARPINTEIRO O, LIMA I. Time series forecasting through rule-based models obtained via rough sets. *Artificial Intelligence Review*. 2011;1:1–12.

8. FAUSTINO CP, NOVAES CP, PINHEIRO CAM, CARPINTEIRO OA. Improving The Performance of Fuzzy Rules-Based Forecasters Through Application of Fcm Algorithm. *Artificial Intelligence Review*. 2012;1:10–24.
9. HANSEN LK, SALAMON P. Neural Networks Ensembles. *IEEE Transactions On Patternanalysis and Machine Intelligence*. 1990;12(10):993–1001.
10. HIGGINS CM, GOODMAN RM. Fuzzy Rule-Based Networks For Control. *IEEE Trans On Fuzzy Systems*. 1994;2(1):82–88.
11. JANG JSR. Anfis: Adaptive-Network-Based Fuzzy Inference System. *IEEE Transaction on Systems, Man and Cybernetics*. 1993;23(3):665–685.
12. JANG JSR, SUN CT, MIZUTANI E. *Neuro-Fuzzy and Soft Computing: A Computational Approach to Learning and Machine Intelligence*. 1^a ed. Prentice Hall 1997.
13. KASABOV N, SONG Q. Denfis: Dynamic evolving neural-fuzzy inference system and its application for time-series prediction. *IEEE Transactions on Fuzzy Systems* 2001.
14. KURIAN CP, et al. Anfis model for the time series prediction of interior daylight illuminance. *AIML Journal*. 2006;6.
15. LEE, K. M.; KWAK, D. H.; KWANG, H. L. **Fuzzy Inference Neural Network for Fuzzy Model Tuning**. *IEEE Trans.on Systems, Man and Cybernetics-Part B: Cybernetics*, v. 26, n. 4, August, 1996, p. 637-45.
16. LIAO GC, TSAO TP. Application of fuzzy neural networks and artificial intelligence for load forecasting. *Electric Power Systems Research*. 2007;70:237–244.
17. LIN FC, LEE FCSG. *Neural Fuzzy Systems A Neuro-Fuzzy Synergism To Inteligente Systems*. Nova Jersey: USA, Prentice-hall Simon & Schuster Company, Upper Saddle River; 1995.
18. MING-BAO P, XIN-PING Z. Traffic flow prediction of chaos time series by using subtractive clustering for fuzzy neural network modeling. *Second International Symposium on Intelligent Information Technology Application* 2008.
19. MITRA S, PAL SK. Self-organizing neural network as a fuzzv classifier. *IEEE Transactions on Systems, Man, and Cybernetics*.

- 1994;24(3):385–399.
20. NAUCK D, KRUSE R. Nefclass – a neuro-fuzzy approach for the classification of data. *Proceedings of the 1995 ACM Symposium on Applied Computing* 1995:461–465.
 21. NAUCK D, KRUSE R, KLAWON F. *Foundations of Neuro-Fuzzy Systems*. John Wiley and Sons. Inc 1997.
 22. OLIVEIRA, A. S; OLIVEIRA, E. A; RABELO, E. Neuro-Fuzzy. 2011. Disponível em: <http://www.din.uem.br/~ia/intelige/neurofuzzy>. Acesso em 10 jan 2014.
 23. SYED-AHMAD MN, et al. Short-term load forecasting using adaptive neuro-fuzzy inference system (anfis). *Application to aleppo load demand* 2007.
 24. WANG LX, MENDEL JM. Generating Fuzzy Rules By Learning From Example. *IEEE*. 1992;22.
 25. YING LC, PAN MC. Using adaptive network based fuzzy inference system to forecast regional electricity loads. *Energy Conversion and Management*. 2008;49:205–211.

Índice

A

Adaline, [57](#), [61](#), [63](#)

agrupamento, [15](#), [85](#), [95](#), [126](#), [127](#)

ajustes de parâmetros, [15](#)

algoritmo

de poda, [79](#), [80](#), [81](#)

de treinamento, [47](#), [51](#), [59](#), [60](#), [62](#), [64](#), [67](#), [68](#), [87](#)

evolutivo, [150](#), [151](#)

genético, [5](#), [100](#), [121](#), [123](#), [124](#), [150](#), [151](#)

amostras, [4](#), [112](#), [117](#)

análise

qualitativa, [14](#), [15](#), [20](#), [21](#), [23](#)

quantitativa, [15](#), [20](#), [21](#)

Anfis, [34](#), [35](#), [36](#), [122](#), [123](#), [124](#), [126](#), [128](#), [129](#), [130](#), [131](#)

aplicações, [4](#), [19](#), [100](#), [111-119](#)

aprendizado

competitivo, [83](#)

aproximação

de função, [72](#), [75](#), [76](#), [104](#), [166](#)

inferior, [137](#), [138](#), [156](#)

superior, [137](#), [138](#)
aproximadores universais, [15](#)
Artificial, Inteligência, [1](#), [2](#), [4](#), [6](#), [113](#), [135](#)
atributos
 de condição, [140](#), [141](#), [143](#), [144](#), [148](#)
 de decisão, [137](#), [140](#)
auto-organizado, [83](#)
autoadaptação, [54](#)
autoadaptativos, [53](#)
autoadaptáveis, [53](#)

B

back-propagation, [55](#)
Bando de Aves (*Bird Algorithm*), [151](#)
base radial, [93](#), [94](#), [95](#), [96](#)
bias, [58](#), [59](#), [61](#), [64](#), [65](#), [73](#), [74](#), [76](#), [94](#), [102](#)

C

camada
 de entrada, [51](#), [52](#), [65](#), [67](#), [91](#)
 de saída, [52](#), [54](#), [61](#), [65](#), [67](#), [68](#), [70](#), [71](#), [73](#), [74](#), [75](#), [91](#), [93](#), [94](#), [95](#), [96](#), [123](#),
 [126](#)
 escondida, [51](#), [73](#), [74](#), [75](#), [76](#), [93](#), [95](#), [96](#), [98](#)
Cardume de Peixes (*Fish Swarm*), [151](#)
Carro, [20](#), [40](#)
classes de equivalência, [137](#), [141](#), [156](#)

cluster, clusters, [83](#), [84](#), [85](#), [88](#), [91](#), [96](#)
coeficiente de imprecisão, [138](#)
coeficiente de qualidade da aproximação
 inferior, [138](#)
 superior, [138](#)
Colônia de Formigas (*Ant Colony*), [139](#), [151](#)
Computação, [1](#), [4](#)
 Evolucionária, [150](#)
Computacional, [47](#)
Configuração, [80](#), [113](#)
conjunto crisp, [8](#), [10](#)
Conjuntos Aproximados (Rough Sets), [4](#), [5](#), [135-155](#)
Conjuntos Difusos/Fuzzy/Nebulosos (Fuzzy Sets), [5](#), [7](#)
cross validation, [114](#)

D

dados, coleta de, [112](#)
de Máquina, [4](#), [15](#), [18](#), [54](#), [135](#), [136](#)
defuzificação, [14](#), [17](#), [20](#), [35](#), [39](#), [126](#)

E

Ensemble, [123](#), [130](#), [131](#), [132](#)
Enxame de Abelhas (*Bees Algorithm*), [151](#)
Enxames de Partículas (*Particle Swarm Optimization* – PSO), [150](#)
Erro Quadrático, [46](#), [73](#), [74](#), [96](#), [98](#)
espaço aproximado, [137](#)

estratégia

de controle, [24](#), [25](#), [148](#), [149](#)

de aprendizado, [47](#)

etiquetas, [9](#), [15](#), [19](#), [21](#)

exemplos, [15](#), [88](#), [95](#), [140](#), [141](#), [144](#)

exercícios, [32](#), [155](#)

F

ferramenta, [36](#), [97](#), [159](#), [164](#)

FIS, [34](#), [122](#), [123](#)

função

de ativação, [50](#), [58](#), [60](#), [66](#), [72](#), [76](#)

de base radial, [93](#), [94](#)

de custo, [81](#), [95](#), [154](#)

de discernibilidade, [139](#), [141](#), [143](#), [145](#)

de erro, [62](#), [78](#), [80](#)

de pertinência, [8](#), [9](#), [10](#), [35](#), [126](#), [131](#)

não linear, [96](#)

funções polinomiais, [17](#)

fuzificação, [9](#), [17](#), [20](#), [35](#), [39](#), [125](#), [126](#), [150](#)

Fuzzy C-Means, [126](#), [127](#), [128](#), [131](#)

G

Gaussiana, [10](#), [131](#), [132](#)

generalização, [53](#), [54](#), [55](#), [56](#), [57](#), [77](#), [79](#), [81](#), [111](#), [113](#), [114](#)

gradiente descendente, [15](#), [35](#), [95](#)

grau de pertinência, [7](#), [8](#), [9](#), [10](#), [35](#), [127](#)

I

indiscernibilidade, [137](#)

informações

imprecisas, [7](#), [9](#)

linguísticas, [5](#), [9](#), [121](#)

vagas, [7](#), [9](#)

inteligência artificial, [1](#), [2](#), [4](#), [6](#), [113](#), [135](#)

J

Janelas, [163](#)

L

lógica

fuzzy, [2](#), [4](#), [7-32](#), [121](#), [135](#)

paraconsistente, [155](#)

M

Madaline, [61](#), [63](#), [64](#), [65](#)

Mapeamento, [4](#), [19](#), [20](#), [35](#), [55](#), [95](#), [96](#), [107](#), [110](#), [128](#), [136](#)

máquina de Boltzmann, [100](#)

Máquinas de Vetor de Suporte, [100](#)

MatLab, [17](#), [23](#), [29](#), [35](#), [36](#), [39](#), [75](#), [76](#), [97](#), [131](#), [152](#)

matriz de discernibilidade, [139](#), [141](#), [142](#), [143](#), [144](#), [145](#)

Membership, [126](#)

método dos mínimos quadrados, [14](#), [15](#), [17](#), [35](#), [61](#), [95](#), [96](#), [97](#)

mínimos locais, [57](#), [78](#), [80](#), [114](#)

MLP, [72](#), [73](#), [74](#), [93](#), [100](#), [117](#), [118](#), [119](#), [130](#)

modelagem, [9](#), [15](#), [16](#), [17](#), [18](#), [27](#), [29](#), [30](#), [31](#), [32](#), [128](#), [131](#)

modelo

aproximado, [146](#)

Arima, [123](#)

ARX, [97](#), [98](#)

Fuzzy, [14](#), [17](#), [19](#), [27](#), [29](#), [30](#), [31](#), [32](#), [33](#), [39](#), [98](#), [121](#), [123](#), [125](#)

NARX, [29](#)

Narmax, [29](#)

Neurônio, [2](#)

Modelos Conexionistas, [47](#), [121](#)

N

Neuro-Fuzzy, [121-132](#)

neurônio, [2](#), [5](#), [34](#), [35](#), [47](#), [49](#), [50](#), [51](#), [52](#), [53](#), [54](#), [56](#), [57](#), [58](#), [59](#), [61](#), [66](#), [68](#), [71](#),
[73](#), [74](#), [75](#), [76](#), [79](#), [86](#), [87](#), [91](#), [92](#), [93](#)

normalização, [35](#), [78](#), [112](#), [116](#), [131](#)

O

objetos, [137](#), [138](#), [139](#)

observações, [4](#), [113](#), [116](#), [117](#), [123](#), [124](#), [128](#), [137](#), [140](#), [142](#), [143](#)

offset, [58](#), [74](#), [94](#)

operações básicas

min, [11](#), [12](#), [13](#)

max, [11](#), [12](#), [13](#), [14](#)

produto algébrico, [11](#), [12](#), [13](#)

S-Norma, [11](#)

T-Norma, [11](#), [35](#)

otimização, [5](#), [6](#), [14](#), [15](#), [95](#), [100](#), [151](#), [152](#), [153](#), [154](#)

over-fitting, [114](#)

over-training, [114](#), [115](#)

P

Perceptron, [47](#), [50](#), [60](#), [61](#), [66](#), [101](#), [117](#)

pesos, [52](#), [54](#), [56](#), [57](#)

polarizador, [58](#)

pré-processamento, [112](#), [113](#), [116](#), [128](#)

R

Radial Basis Function, [93](#)

Rede, Redes

de Hopfield, [100](#)

de Kohonen, [86](#)

Feedforward, [79](#)

MLP, [100](#)

mono-camadas, [52](#)

multicamadas, [52](#)

neurais artificiais, [4](#), [5](#), [47-103](#), [121](#), [122](#), [124](#)

RBF, [93](#), [94](#), [95](#), [96](#), [98](#), [110](#)

Recorrentes, [52](#), [53](#), [100](#)

Redutos, [139](#), [142](#), [143](#), [148](#)

Regra(s)

Aproximadas, [124](#), [149](#), [150](#), [157](#), [166](#)

Consistentes, [140](#)

da cadeia, [62](#), [69](#), [70](#)

de decisão, [135](#), [139](#), [140](#)

de modelagem, [157](#)

Delta, [61](#), [62](#), [69](#)

fuzzy, [14](#), [15](#), [18](#), [19](#), [20](#), [22](#), [23](#), [25](#)

Hebb, [55](#), [59](#), [60](#), [101](#)

inconsistentes ou conflitantes, [140](#)

Mamdani, [7](#), [14](#), [19](#), [22](#)

obtenção, [14](#), [20](#)

rough, [124](#)

Takagi-Sugeno, [14](#), [35](#), [128](#)

Retropropagação, [29](#), [47](#), [55](#), [72](#), [77](#), [79](#), [105](#)

RNA, [47](#), [52](#), [57](#), [73](#), [75](#), [102](#), [111](#), [112](#), [121](#), [122](#)

Rosetta, [140](#), [142](#), [146](#), [148](#), [156](#), [159](#), [164](#)

S

Série Temporal, [30](#), [31](#), [116](#), [119](#), [123](#), [131](#)

sinapses, [49](#), [54](#)

sistema de nível, [28](#), [45](#), [98](#)

sistema de suspensão, [23](#), [24](#), [25](#), [26](#), [148](#), [149](#)

sistemas

de informações, [135](#), [136](#), [140](#)

especialistas, [2](#), [5](#), [6](#), [15](#), [135](#)
fuzzy, [14](#), [15](#), [100](#), [121](#), [122](#), [124](#)
híbridos, [121](#), [130](#)
imunológicos, [150](#), [151](#)
superaprendizado, [114](#)

T

taxa de treinamento, [62](#), [88](#), [90](#)
Teoria dos Conjuntos Aproximados, [4](#), [135](#), [140](#)
Teoria dos Conjuntos Difusos/Nebulosos, [5](#), [7](#)
Termo de Momento, [77](#)
Teste da rede, [106](#), [119](#)
toolbox, [17](#), [18](#), [23](#), [29](#), [36](#), [75](#), [97](#), [131](#), [152](#)
trapezoidal, [10](#), [131](#), [132](#)
treinamento
 auto-organizado, [83](#)
 não supervisionado, [82](#)
triangular, [8](#), [9](#)

U

universo de discurso, [7](#), [8](#), [17](#), [35](#)

V

validação, [56](#), [57](#), [77](#)
 cruzada, [114](#)
valores modais, [8](#), [21](#), [22](#), [29](#), [35](#)

veículos, [20](#), [91](#)

OceanofPDF.com