

Julgada em 22/01/2000 e aprovada pela Comissão
Fernando José Von Zuben
Orientador

Universidade Estadual de Campinas

Faculdade de Engenharia Elétrica e de Computação

Departamento de Engenharia de Computação e Automação Industrial

INTELIGÊNCIA COMPUTACIONAL NO PROJETO AUTOMÁTICO DE REDES NEURAIS HÍBRIDAS E REDES NEUROFUZZY HETEROGÊNEAS

EDUARDO MASATO IYODA

Orientador: PROF. DR. FERNANDO JOSÉ VON ZUBEN

(DCA – FEEC / UNICAMP)

Tese apresentada à Faculdade de Engenharia Elétrica e de
Computação (FEEC / UNICAMP) como parte dos requisitos
exigidos para obtenção do título de Mestre em Engenharia Elétrica.

Banca Examinadora: Prof. Dr. Márcio Luiz de Andrade Netto (FEEC / UNICAMP)

Prof. Dr. Maurício Fernandes Figueiredo (DIN / UEM - PR)

CAMPINAS

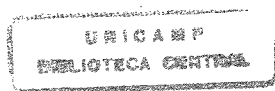
Estado de São Paulo – Brasil

Janeiro de 2000

UNICAMP

BIBLIOTECA CENTRAL

SEÇÃO CIRCULANTE



2000.173

CHAMADA:
UNICAMP
Iy9i
Ex.
DATA DO/ 41121
VAL. 278/00
C ☐ D ☒
REC9 RB 11.00
ATA 16-06-00
* CPD

CM-00142797-9

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

Iy9i Iyoda, Eduardo Masato
Inteligência computacional no projeto automático de
redes neurais híbridas e redes neurofuzzy heterogêneas /
Eduardo Masato Iyoda.--Campinas, SP: [s.n.], 2000.

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

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

RESUMO

Esta tese apresenta um estudo a respeito de possíveis interações entre os principais paradigmas que compõem a área de inteligência computacional: redes neurais artificiais, sistemas *fuzzy* e computação evolutiva. Como principais contribuições, são propostas duas novas estratégias de solução de problemas de engenharia: as redes neurais híbridas e as redes *neurofuzzy* heterogêneas. A rede neural híbrida corresponde a uma extensão dos modelos de aproximação por busca de projeção, onde são consideradas também composições multiplicativas das funções de ativação dos neurônios escondidos. A arquitetura *neurofuzzy* heterogênea, diferentemente das arquiteturas *neurofuzzy* tradicionais, utiliza neurônios lógicos que podem ter pares distintos de normas triangulares. Os resultados de simulações computacionais mostram que os dois novos modelos propostos são bastante promissores, no sentido de que eles são capazes de fornecer soluções de melhor qualidade do que os modelos convencionais.

ABSTRACT

UNICAMP
BIBLIOTECA CENTRAL
SEÇÃO CIRCULANTE

This thesis presents a study on possible combinations of the main paradigms that compose the field of computational intelligence: artificial neural networks, fuzzy systems and evolutionary computation. Among other contributions, two new engineering problem-solving strategies are proposed: hybrid neural networks and heterogeneous neurofuzzy networks. Hybrid neural networks correspond to an extension of project pursuit learning models, where multiplicative compositions of the hidden neurons' activation functions are also considered. Differently from traditional neurofuzzy architectures, heterogeneous neurofuzzy networks employ logical neurons that may have distinct pairs of triangular norms. Simulation results show that these new proposed models are very promising, in the sense that they are capable of providing higher quality solutions than traditional models.

*Dedico aos meus pais,
Akira & Mitsuru.*

Agradecimentos

Em primeiro lugar, agradeço ao meu orientador, o Professor Fernando José Von Zuben. O Professor Von Zuben, com seus profundos conhecimentos na área de inteligência computacional, norteou de forma decisiva este trabalho. Sua dedicação, a atenção dispensada aos seus orientados e sua determinação para o trabalho constituem um exemplo que procurarei seguir durante toda a minha vida.

Agradeço ao Professor Fernando Antonio Campos Gomide pelo grande incentivo e valiosas contribuições dados às idéias e resultados apresentados no Capítulo 7 desta tese.

Sou grato aos colegas Leandro Nunes de Castro, Eurípedes Pinheiro dos Santos e Clodoaldo Aparecido de Moraes Lima, que colaboraram nas mais diversas formas para a realização desta tese: através de ajuda nas simulações computacionais, de profícuas discussões e mesmo através de sua amizade e companheirismo.

Aos amigos Fabrício e Aldvan Figueiredo, Leonardo de Oliveira Garcia e Luís Augusto de Sá Pessoa agradeço o carinho e a amizade, que foram de grande importância durante o decorrer deste trabalho.

Agradeço ao Departamento de Engenharia de Computação e Automação Industrial, Faculdade de Engenharia Elétrica e de Computação, Unicamp, pela oportunidade de fazer o curso de Mestrado.

Por fim, agradeço ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) pelo auxílio financeiro fornecido através do processo 133691/97-7.

Índice

Resumo.....	i
Abstract.....	i
Dedicatória	iii
Agradecimentos.....	v
Índice.....	vii
Capítulo 1. Introdução	1
1.1 Inteligência Computacional	2
1.2 Organização da Tese	4
Capítulo 2. Redes Neurais Multicamadas	7
2.1 Modelo de Neurônio	7
2.1.1 Tipos de Função de Ativação.....	9
2.2 Redes Neurais Multicamadas.....	11
2.3 Aprendizado Supervisionado	12
2.4 O Algoritmo de Retropropagação (<i>Backpropagation</i>)	14
2.4.1 Os Dois Passos de Computação.....	21
2.4.2 Funções de Ativação.....	21
2.4.3 Modos de Treinamento	22
2.4.4 Critério de Parada	23
2.4.5 Inicialização dos Pesos	24
2.4.6 Resumo do Algoritmo de Retropropagação.....	25
2.5 Métodos de Segunda Ordem.....	25
2.6 Aproximação de Funções.....	27
2.7 Conclusão	30
Capítulo 3. Algoritmos Genéticos.....	31
3.1 Evolução Natural	31

3.2	Computação Evolutiva.....	35
3.3	Algoritmos Genéticos	37
3.3.1	Codificação de Indivíduos	39
3.3.2	Definição da População Inicial	40
3.3.3	Operadores Genéticos	40
3.3.4	Seleção de Indivíduos para a Próxima Geração.....	45
3.4	Teoria dos Esquemas	46
3.5	Conclusões	50
Capítulo 4.	Redes Neurais Construtivas.....	53
4.1	Métodos Construtivos	53
4.1.1	Estratégia de Busca no Espaço de Estados	54
4.2	O Algoritmo <i>Cascade-Correlation</i>	57
4.2.1	CASCOR com Funções de Ativação Distintas.....	60
4.3	Aprendizado por Busca de Projeção (Baseado em VON ZUBEN (1996)).....	61
4.3.1	O Problema de Aproximação Resultante.....	63
4.3.2	Busca da Direção Inicial de Projeção (<i>Projection Pursuit</i>).....	65
4.3.3	Determinação da Função de Expansão Ortogonal.....	66
4.3.4	O Processo de Ajuste Retroativo	67
4.3.5	O Tratamento de Múltiplas Saídas	69
4.3.6	O Algoritmo de Aproximação Construtivo	70
4.4	O Algoritmo A*	72
4.4.1	O Algoritmo A* na Otimização de Arquiteturas de Redes Neurais	74
4.5	Comparação entre CASCOR, PPL e A*.....	76
4.6	Conclusão	78
Capítulo 5.	Redes Neurais Evolutivas.....	79
5.1	Redes Neurais e Computação Evolutiva.....	79
5.2	Computação Evolutiva para Ajuste de Pesos	80
5.2.1	Revisão de Métodos Evolutivos para Treinamento de Redes Neurais	81
5.2.2	Comparação entre Algoritmo Genético e Método do Gradiente Conjugado para Treinamento de Redes Neurais	82

5.3	Computação Evolutiva para Definição de Arquiteturas de Redes Neurais	87
5.3.1	Revisão de Algoritmos Evolutivos para Definição de Arquiteturas de Redes Neurais	89
5.4	Conclusões	94
Capítulo 6. Arquiteturas Híbridas de Redes Neurais Artificiais		95
6.1	Introdução	95
6.2	Redes Neurais Híbridas	98
6.3	Treinamento da Rede Neural Híbrida: Busca Local	101
6.4	A Abordagem Evolutiva	105
6.4.1	Codificação	106
6.4.2	Inicialização da População	108
6.4.3	Avaliação dos Indivíduos	108
6.4.4	Operador de Seleção	109
6.4.5	Operador de <i>Crossover</i>	110
6.4.6	Operador de Mutação	110
6.5	Resultados de Simulações	110
6.5.1	Aproximação de Funções do Conjunto de Candidatos	111
6.5.2	Comparação com Outras Arquiteturas	114
6.6	Conclusão	117
Capítulo 7. Redes Neurofuzzy Heterogêneas		119
7.1	Fundamentos de Sistemas <i>Fuzzy</i>	120
7.2	Normas Triangulares	121
7.3	Neurônios Lógicos	123
7.4	A Rede <i>Neurofuzzy</i> AND/OR	125
7.5	Algoritmo de Treinamento da Rede <i>Neurofuzzy</i> AND/OR	127
7.5.1	Geração das Funções de Pertinência	128
7.5.2	Geração das Conexões da Rede e Inicialização dos Pesos w_{ji} e v_j	131
7.5.3	Determinação dos Neurônios AND e OR Ativos	133
7.5.4	<i>Fuzzificação</i>	134
7.5.5	Determinação da Classe Vencedora	135

7.5.6	Atualização dos Pesos.....	135
7.6	Redes <i>Neurofuzzy</i> Heterogêneas.....	136
7.6.1	A Abordagem Evolutiva.....	137
7.6.2	Problemas de Teste.....	139
7.6.3	Resultados de Simulações.....	141
7.7	Conclusões.....	143
Capítulo 8.	Conclusão.....	145
8.1	Novas Estratégias de Solução de Problemas de Engenharia.....	146
8.2	Extensões.....	147
Bibliografia.....		149
Índice de Autores.....		163

Capítulo 1

Introdução

A busca por sistemas artificiais que apresentam algum tipo de comportamento inteligente, similar ao exibido por muitos sistemas biológicos (incluindo seres humanos), sempre fascinou muitos cientistas. Os sistemas biológicos são resultado de um longo processo de evolução natural, e apresentam características como adaptabilidade, tolerância a falhas e robustez a variações ambientais. Tais características são bastante desejáveis em sistemas de engenharia, levando diversos pesquisadores a propor estratégias que procuram emular alguns dos aspectos observados em sistemas biológicos naturais. Dentre as áreas de pesquisa que procuram por sistemas artificiais inteligentes, uma das mais promissoras é a chamada *inteligência computacional*.

A inteligência computacional compreende paradigmas computacionais que procuram desenvolver sistemas que apresentam alguma forma de inteligência similar à exibida por determinados sistemas biológicos. Alguns dos paradigmas que compõem a inteligência computacional foram de fato inspirados em sistemas biológicos (como as redes neurais artificiais e a computação evolutiva), enquanto que outros, apesar de não terem inspiração biológica, tentam gerar sistemas que produzam algum tipo de comportamento próximo ao observado em sistemas naturais (como por exemplo, o raciocínio aproximado dos sistemas *fuzzy*).

Métodos baseados nestes paradigmas invariavelmente demandam uma quantidade grande de recursos computacionais, seja no desenvolvimento ou no uso das ferramentas de solução. Entretanto, nos últimos anos, temos observado um desenvolvimento estrondoso da tecnologia de microprocessadores e memória e, como consequência, temos à nossa disposição cada vez mais recursos computacionais a baixo custo. Este fato tem levado a um interesse crescente da comunidade científica nos métodos baseados em inteligência computacional.

Métodos que, até algum tempo atrás, seriam considerados ineficazes do ponto de vista computacional, puderam ser efetivamente implementados e testados. Como a tecnologia dos computadores continua a se desenvolver rapidamente, temos cada vez mais recursos computacionais à disposição e, como consequência, podemos esperar grandes desenvolvimentos na área de inteligência computacional.

1.1 Inteligência Computacional

BEZDEK (1994) sugere que um sistema é *computacionalmente inteligente* quando: trabalha apenas com dados numéricos (baixo nível), tem um componente de reconhecimento de padrões, e não usa conhecimento no sentido da inteligência artificial clássica (conhecimento simbólico, não-numérico); e, adicionalmente, quando ele exhibe (ou começa a exhibir):

1. adaptabilidade computacional;
2. tolerância computacional a falhas;
3. velocidade de processamento comparável à de processos cognitivos humanos;
4. taxas de erro que se aproximam do desempenho humano.

A área de inteligência computacional engloba diversos paradigmas computacionais diferentes. Os principais paradigmas da inteligência computacional são:

- *Redes Neurais Artificiais*: sistemas de processamento de informação formados pela interconexão maciça entre unidades simples de processamento, denominadas neurônios artificiais. Os neurônios artificiais recebem essa denominação porque foram originados a partir de um modelo matemático de um neurônio biológico. Várias arquiteturas de redes neurais já foram propostas na literatura, sendo a arquitetura multicamadas a mais popular. As mais atraentes propriedades destas arquiteturas são a sua capacidade de aproximação universal e de aprendizado a partir de exemplos.
- *Sistemas Fuzzy*: os sistemas *fuzzy* são baseados no conceito de conjunto *fuzzy*. Os conjuntos *fuzzy* surgiram como uma nova forma de representação de conceitos como imprecisão e incerteza. Os conjuntos *fuzzy* são especialmente adequados na descrição de sistemas de processamento de informação complexos, não-lineares ou não claramente definidos. Observe que, além de trabalhar com dados numéricos, os sistemas *fuzzy*

também são capazes de realizar processamento simbólico, através de uma base de regras *fuzzy*.

- *Computação Evolutiva*: a computação evolutiva é formada por algoritmos inspirados na teoria da evolução natural de Darwin. A computação evolutiva tem sido muito aplicada em problemas de otimização, em especial naqueles em que técnicas tradicionais de otimização não são aplicáveis (ou apresentam desempenho insatisfatório).

O interesse na busca por sistemas computacionais que explorem possíveis combinações entre os paradigmas acima têm crescido de forma expressiva nos últimos tempos. Diversos pesquisadores têm constatado que as técnicas acima são, em muitos aspectos, complementares. Assim, muitas pesquisas têm sido realizadas no sentido de investigar possíveis formas de cooperação entre estes métodos, e mesmo entre métodos de inteligência computacional e métodos tradicionais de engenharia. O principal objetivo dessas pesquisas é desenvolver sistemas que sejam eficientes, robustos, fáceis de operar, e capazes de fornecer soluções de qualidade para problemas complexos.

É importante notar também que os paradigmas acima citados (especialmente sistemas *fuzzy* e redes neurais artificiais) têm sido empregados com sucesso em diversas aplicações práticas, algumas disponíveis comercialmente, principalmente em aplicações de eletrônica de consumo e automação industrial. Tem-se observado, na prática, que tais sistemas de fato apresentam desempenho extraordinário, que não poderia ser obtido se apenas técnicas convencionais de engenharia fossem empregadas (HIROTA & SUGENO, 1995; SIMPSON, 1996).

Nesta tese, procuramos investigar algumas possíveis interações entre os paradigmas da inteligência computacional. Foram estudadas redes neurais construtivas, e o desempenho de alguns algoritmos construtivos foram comparados. Redes neurais evolutivas foram investigadas, e são apresentados resultados de simulações computacionais comparando o desempenho entre algoritmos genéticos e um método de segunda ordem no treinamento de redes neurais artificiais. Além disso, foram desenvolvidas duas arquiteturas baseadas em inteligência computacional: uma *rede neural híbrida* e uma *rede neurofuzzy heterogênea*. Estas arquiteturas correspondem a extensões de arquiteturas já propostas anteriormente. Nosso principal objetivo ao propor estas arquiteturas foi o de gerar sistemas mais adaptados, dedicados a um problema particular. Um sistema dedicado a um determinado problema pode

apresentar diversas vantagens em relação a modelos mais genéricos, como por exemplo maior robustez e uso mais eficaz dos recursos computacionais disponíveis.

1.2 Organização da Tese

O Capítulo 2 apresenta uma introdução às redes neurais artificiais multicamadas. São apresentados conceitos básicos relacionados às redes neurais multicamadas que serão importantes em capítulos subseqüentes. O algoritmo de retropropagação do erro, usado no processo de treinamento de redes neurais multicamadas, é apresentado em detalhes. Redes neurais artificiais são apresentadas como modelos de aproximação de funções e o teorema da aproximação universal é introduzido.

No Capítulo 3 são abordados os conceitos básicos relacionados à computação evolutiva, em especial aos algoritmos genéticos. São apresentados conceitos relativos à evolução natural, que inspiraram o desenvolvimento de algoritmos computacionais evolutivos. São apresentadas, em linhas gerais, as principais abordagens evolutivas já propostas na literatura. O restante do capítulo é dedicado ao estudo dos algoritmos genéticos, e aos seus principais componentes: codificação de indivíduos, operadores genéticos (*crossover* e mutação) e seleção de indivíduos. Por fim, é introduzida a teoria dos esquemas, proposta para explicar o funcionamento de um algoritmo genético.

As redes neurais construtivas são introduzidas no Capítulo 4, e compreendem algoritmos para definição automática de redes neurais artificiais, que começam com alguma arquitetura mínima e vão acrescentando neurônios ou camadas de neurônios até que um determinado nível de erro seja atingido. São apresentados alguns conceitos básicos a respeito de redes neurais construtivas e são descritos três algoritmos construtivos: o *cascade-correlation*, o aprendizado por busca de projeção e o A*. São então apresentados resultados de simulações computacionais envolvendo os três algoritmos mencionados, a fim de comparar o seu desempenho (em termos de arquitetura final obtida).

O Capítulo 5 é dedicado ao estudo das redes neurais evolutivas, que correspondem a métodos que combinam computação evolutiva e redes neurais artificiais. A computação evolutiva pode ser empregada no treinamento ou na definição de arquiteturas de redes neurais artificiais. É apresentada uma revisão bibliográfica de alguns dos principais métodos já

propostos na literatura, relacionando computação evolutiva e redes neurais artificiais. São apresentados resultados de simulações computacionais comparando o desempenho de algoritmos genéticos e um método de segunda ordem no treinamento de redes neurais artificiais.

O Capítulo 6 introduz as redes neurais híbridas, um modelo de rede neural artificial mais genérico, que pode ser visto como uma extensão de modelos de aproximação por busca de projeção. É apresentada uma extensão do algoritmo de retropropagação, que permite realizar o treinamento de redes neurais híbridas. Um algoritmo genético é proposto para seleção adequada de funções de ativação e operadores para a rede neural híbrida. São apresentadas diversas simulações computacionais com o objetivo de testar as potencialidades das redes neurais híbridas. Também são apresentados resultados comparando o desempenho de redes neurais híbridas com o de outras arquiteturas.

Uma arquitetura *neurofuzzy* heterogênea para classificação de padrões é apresentada no Capítulo 7. Esta arquitetura se distingue das arquiteturas *neurofuzzy* tradicionais pela possibilidade do uso de pares de normas triangulares distintos em cada neurônio da rede. Um algoritmo genético é utilizado para escolha de pares de normas triangulares para cada neurônio da rede. São apresentados resultados de simulações comparando o desempenho de redes *neurofuzzy* heterogêneas com o de outras arquiteturas em problemas de classificação de padrões.

Capítulo 2

Redes Neurais Multicamadas

Redes neurais artificiais têm sido aplicadas com sucesso nos mais diversos problemas. Dentre as principais áreas de aplicação de redes neurais artificiais podemos citar: sistemas de controle (WHITE & SOFGE, 1992), reconhecimento de padrões (BISHOP, 1995) e aproximação de funções (VON ZUBEN, 1996). Embora existam inúmeras arquiteturas de redes neurais, a arquitetura multicamadas é, sem dúvida, a mais frequentemente encontrada na literatura. Entre as razões para a sua popularidade podemos citar sua capacidade de *aproximação universal* e sua flexibilidade para formar soluções de qualidade para uma ampla classe de problemas, a partir de um mesmo algoritmo de aprendizado.

Neste capítulo, apresentaremos os principais tópicos relacionados às arquiteturas multicamadas, e que serão importantes no desenvolvimento deste trabalho. Em especial, descreveremos com grande detalhe o algoritmo de *retropropagação de erro* utilizado no treinamento de redes neurais artificiais multicamadas. Este algoritmo foi um dos principais responsáveis pelo ressurgimento do interesse da comunidade científica em redes neurais, após um período de grande ceticismo em relação às suas potencialidades. Apresentaremos também as redes multicamadas como modelos para aproximação de funções e discutiremos a importante propriedade de aproximação universal apresentada por estas arquiteturas.

2.1 Modelo de Neurônio

Um *neurônio* é a unidade fundamental de processamento de informação de uma rede neural (HAYKIN, 1999). A Figura 2.1 mostra o *modelo* de um neurônio artificial. Podemos identificar três elementos básicos no modelo:

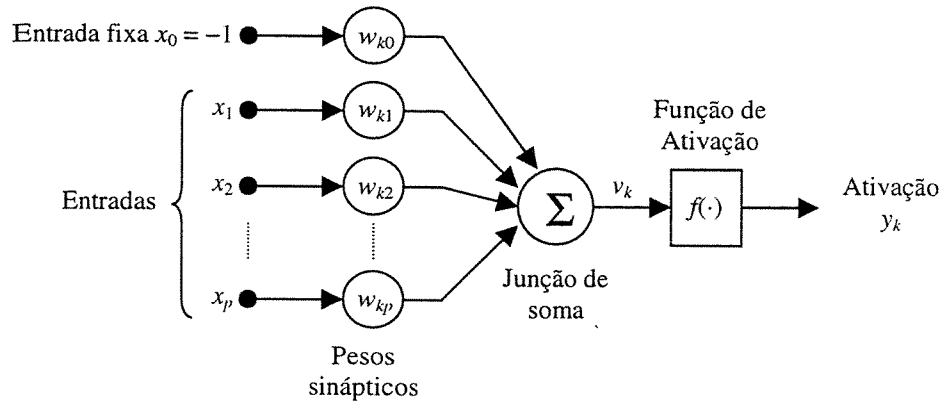


Figura 2.1 Modelo de neurônio.

1. Um conjunto de *sinapses* ou *conexões* de entrada, sendo cada entrada ponderada por um *peso sináptico*. Sendo assim, um sinal x_j na entrada da sinapse j conectada ao neurônio k é multiplicado pelo peso sináptico w_{kj} . Observe a ordem adotada para os índices (subscritos) na notação aqui empregada: o primeiro índice se refere ao neurônio em questão e o segundo índice ao terminal de entrada da sinapse ao qual o peso se refere. Quando uma entrada fixa está presente (entrada x_0 na Figura 2.1), então o peso sináptico correspondente é denominado peso de *polarização*.
2. Uma *junção de soma*, responsável pela combinação aditiva dos sinais de entrada, ponderados pelos respectivos pesos das sinapses do neurônio.
3. Uma *função de ativação* geralmente não-linear e de formato sigmoidal, representando um efeito de saturação na ativação de saída y_k do neurônio. Tipicamente a excursão da ativação do neurônios é confinada ao intervalo $(0, 1)$ ou $(-1, 1)$.

Podemos descrever o modelo de neurônio ilustrado na Figura 2.1 pelo seguinte par de equações:

$$v_k = \sum_{j=0}^p w_{kj} x_j \quad (2.1)$$

e

$$y_k = f(v_k), \quad (2.2)$$

onde $x_0, x_1, \dots, x_p$ são os sinais de entrada, $w_{k0}, w_{k1}, \dots, w_{kp}$ são os pesos sinápticos do neurônio k , v_k é o *nível de ativação interna* ou *potencial de ativação* do neurônio k , $f(\cdot)$ é a função de ativação e y_k é a ativação de saída do neurônio k .

Observe na Figura 2.1 a presença de uma entrada de polarização fixa $x_0 = -1$. Esta entrada, juntamente com o peso w_{k0} a ela associada, tem o efeito de transladar a função de ativação em torno da origem (transformação afim), fazendo com que a ativação interna v_k do neurônio não seja nula quando todas as demais entradas $\{x_1, x_2, \dots, x_p\}$ forem nulas. Podemos observar mais claramente este efeito de polarização se rescrevermos a Equação (2.2) como

$$y_k = f\left(\sum_{j=1}^p w_{kj}x_j - w_{k0}\right). \quad (2.3)$$

A entrada de polarização x_0 poderia assumir qualquer outro valor fixo diferente de zero. Alguns autores fazem uma distinção entre os termos polarização (quando a entrada fixa assume o valor $x_0 = +1$) e limiar (quando $x_0 = -1$). Neste trabalho usaremos sempre o termo polarização, independente do valor que a entrada fixa x_0 assumir.

Podemos escrever a ativação do neurônio em notação vetorial, como segue:

- seja $\mathbf{x} = [x_0 \ x_1 \ \dots \ x_p]^T$ o vetor de entradas do neurônio k ;
- seja $\mathbf{w} = [w_{k0} \ w_{k1} \ \dots \ w_{kp}]^T$ o vetor de pesos do neurônio k ;

então a ativação y_k é dada por

$$y_k(\mathbf{w}, \mathbf{x}) = f(\mathbf{w}^T \mathbf{x}).$$

2.1.1 Tipos de Função de Ativação

A função de ativação $f(\cdot)$ é responsável por definir a ativação de saída do neurônio em termos do seu nível de ativação interna. Podemos identificar três classes principais de função de ativação:

1. *Função Sinal (Heaviside)*. Para este tipo de função, ilustrada na Figura 2.2(a), temos

$$f(v) = \begin{cases} 1 & \text{se } v > 0 \\ 0 & \text{se } v \leq 0 \end{cases}.$$

Assim, a ativação de um neurônio k que usa este tipo de função de ativação é dada por

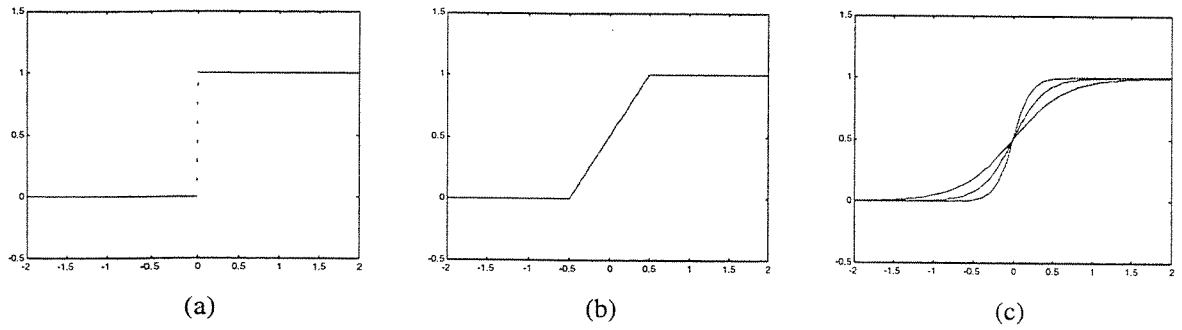


Figura 2.2 Funções de ativação: (a) função sinal, (b) função linear por partes ($a = 0.5$), (c) função sigmoidal.

$$y_k = \begin{cases} 1 & \text{se } v_k > 0 \\ 0 & \text{se } v_k \leq 0 \end{cases},$$

onde v_k é o nível de ativação interno do neurônio, definido na Equação (2.1). Um neurônio com esta função de ativação é conhecido como *modelo de McCulloch-Pitts*, em homenagem ao trabalho pioneiro de MCCULLOCH & PITTS (1943).

1. *Função Linear por Partes*. Para a função linear por partes, ilustrada na Figura 2.2(b), temos

$$f(v) = \begin{cases} 0, & \text{se } v \leq -a \\ v + a, & \text{se } -a < v < a \\ 1, & \text{se } v \geq a \end{cases}$$

No limite, quando a tende a zero, esta função se aproxima assintoticamente da função sinal.

2. *Função Sigmoidal*. A função sigmoidal é a função de ativação mais utilizada em redes neurais artificiais. Ela é definida como uma função monotônica crescente que apresenta propriedades assintóticas e de suavidade. Um exemplo de função sigmoidal é a chamada *função logística*, definida por

$$f(v) = \frac{1}{1 + e^{-av}}, \quad (2.4)$$

onde a é o parâmetro de inclinação da função sigmoidal. Variando o parâmetro a podemos obter funções sigmoidais com diferentes inclinações, como mostrado na Figura 2.2(c).

Aqui também, no limite, quando a tende a infinito, esta função se aproxima da função sinal.

Em muitas situações é desejável termos uma função sigmoidal que varie entre -1 e $+1$. Nestas situações, uma função comumente empregada é a *tangente hiperbólica*, definida por

$$f(v) = \operatorname{tgh}\left(\frac{v}{2}\right) = \frac{1 - e^{-v}}{1 + e^{-v}}.$$

2.2 Redes Neurais Multicamadas

As redes neurais multicamadas são arquiteturas onde os neurônios são organizados em duas ou mais camadas de processamento, já que sempre vai existir pelo menos uma camada de entrada e uma camada de saída. Estas arquiteturas são as mais frequentemente encontradas na literatura referente a redes neurais artificiais.

As redes neurais com apenas duas camadas são constituídas de uma camada de entrada que se conecta a uma camada de neurônios de saída. Os neurônios da camada de entrada são neurônios especiais, cujo papel é exclusivamente distribuir cada uma das entradas da rede (sem modificá-las) a todos os neurônios da camada seguinte. A forma mais simples deste tipo de rede neural, consiste de um único neurônio na camada de saída, sendo conhecido como *perceptron*. A Figura 2.1 é um exemplo de um perceptron, com p neurônios na camada de entrada e um único neurônio na camada de saída, de acordo com a notação adotada nesta seção. Um exemplo de rede neural com apenas duas camadas é apresentado na Figura 2.3(a), com 3 neurônios na camada de entrada e 4 neurônios na camada de saída. O perceptron foi objeto de intensa pesquisa durante os anos 50 e 60, mas em 1969, M. Minsky e S. Papert provaram matematicamente que este tipo de estrutura de processamento apresenta limitações importantes e podem ser aplicadas com sucesso a uma classe muito restrita de problemas (MINSKY & PAPERT, 1988). Mais especificamente foi provado que o perceptron é capaz de resolver apenas problemas *linearmente separáveis*.

No entanto, com a utilização de *redes de múltiplas camadas* com pelo menos uma camada escondida (camada que não é nem entrada, nem saída), ou *perceptron multicamadas* (MLP – *multilayer perceptron*), muitas das limitações apresentadas pelo perceptron deixam

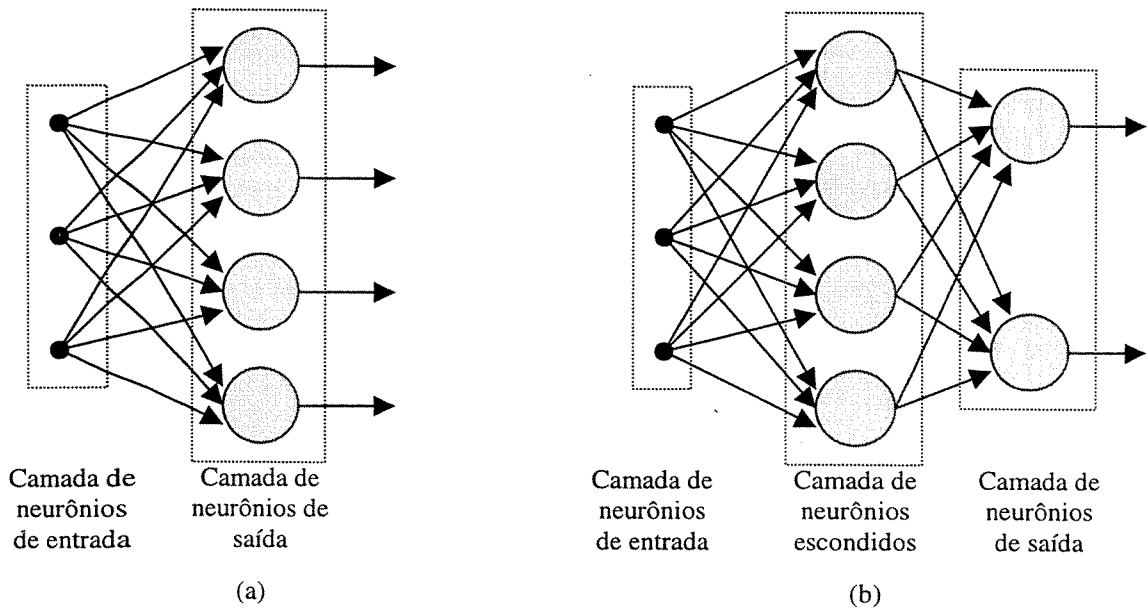


Figura 2.3 Redes neurais em camadas: (a) rede com apenas duas camadas de neurônios, (b) rede com uma camada de entradas, uma camada escondida e uma camada de saída.

de existir. Na Figura 2.3(b) apresentamos uma rede neural com uma camada escondida. Por simplicidade, esta arquitetura é referida como uma rede 3-4-2, isto é, 3 neurônios de entrada, 4 neurônios escondidos e 2 neurônios de saída. Como outro exemplo, uma rede neural com p entradas, h_1 neurônios na primeira camada escondida, h_2 na segunda camada escondida e q neurônios na camada de saída é dita ser uma rede $p-h_1-h_2-q$.

2.3 Aprendizado Supervisionado

A mais importante propriedade de uma rede neural artificial é sua capacidade de aprendizado. Uma rede neural aprende através de um processo iterativo de ajustes aplicados aos seus pesos sinápticos e limiares, o qual pode ser expresso na forma de um algoritmo computacional.

Não existe uma definição precisa universalmente aceita de “aprendizado”. No contexto de redes neurais artificiais, HAYKIN (1999) define aprendizado da seguinte forma: “Aprendizado é um processo pelo qual os parâmetros livres de uma rede neural são adaptados através de um processo de estímulo pelo ambiente no qual a rede está inserida”. Assim, um processo de aprendizagem de uma rede neural implica na seguinte seqüência de eventos:

1. a rede neural é estimulada pelo ambiente de informação;
2. a estrutura interna da rede é alterada como resultado do estímulo;
3. devido às alterações que ocorreram em sua estrutura interna, a rede tem modificada sua resposta aos estímulos do ambiente.

Um tipo particular de aprendizado será de especial interesse para este trabalho: o *aprendizado supervisionado*. Este tipo de aprendizado é caracterizado pela presença de um “professor” externo. A função do “professor” durante o processo de aprendizado é suprir a rede neural com uma *resposta desejada* a um determinado estímulo apresentado pelo ambiente. Definimos um *signal de erro* como a diferença entre resposta desejada e a resposta observada na saída da rede neural. Os parâmetros da rede são então ajustados de acordo com o sinal de erro. A Figura 2.4 apresenta um diagrama de blocos de um sistema com aprendizado supervisionado.

Uma forma de implementar uma estratégia de aprendizado supervisionado em redes neurais é através de procedimentos iterativos de correção de erro. Seja $s_k(n)$ a resposta desejada para um neurônio k no instante n e seja $y_k(n)$ a resposta observada para este neurônio. A resposta $y_k(n)$ é produzida por um estímulo (vetor) $x(n)$ aplicado à entrada da rede da qual o neurônio k faz parte. O sinal de erro para o neurônio k é definido como a diferença entre a resposta desejada e a resposta observada:

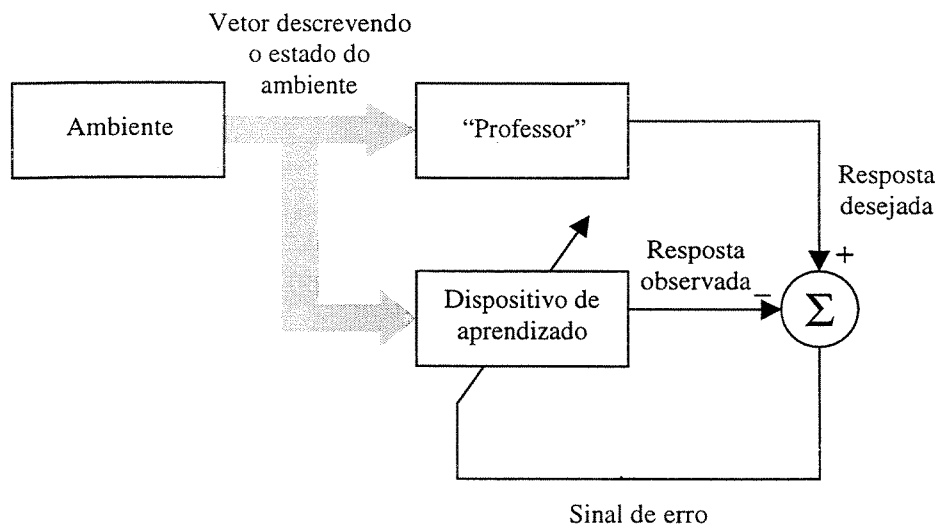


Figura 2.4 Diagrama de blocos de um sistema com aprendizado supervisionado.

$$e_k(n) = s_k(n) - y_k(n).$$

O objetivo do procedimento de aprendizado por correção de erro é minimizar alguma *função de custo* baseada no sinal de erro $e_k(n)$, de modo que a resposta observada de cada neurônio da rede se aproxime da resposta desejada para aquele neurônio, em algum sentido estatístico. De fato, uma vez definida uma função de custo, o problema de aprendizado torna-se um problema de otimização. Uma função de custo comumente empregada é a *soma dos erros quadráticos*:

$$J = \frac{1}{2} \sum_k e_k^2(n),$$

onde o somatório é realizado sobre todos os neurônios da camada de saída da rede neural. A rede neural aprende através da minimização de J em relação aos pesos sinápticos da rede.

Note que J define uma *superfície de erro* sobre o espaço dos pesos. Se P é o número de pesos ajustáveis da rede neural, então $J: \mathcal{R}^P \rightarrow \mathcal{R}$. A superfície de erro é caracterizada pela presença de mínimos locais e um ou mais mínimos globais. Os métodos de otimização utilizados na minimização de J usualmente recorrem à informação de *gradiente do erro* para ajustar os parâmetros da rede. Teoricamente, estes métodos sempre atingem um ponto de mínimo da superfície de erro, mas observe que nada se pode afirmar sobre a natureza (local ou global) do ponto de mínimo obtido a partir de uma condição inicial arbitrária.

2.4 O Algoritmo de Retropropagação (*Backpropagation*)

O algoritmo de *retropropagação de erro*, ou simplesmente *retropropagação*, é um algoritmo utilizado no treinamento de redes neurais multicamadas com uma ou mais camadas escondidas. Basicamente, o algoritmo de retropropagação consiste em dois passos de computação: o processamento direto e o processamento reverso. No processamento direto, uma entrada é aplicada à rede neural e seu efeito é propagado pela rede, camada a camada. Durante o processamento direto, os pesos da rede permanecem fixos. No processamento reverso, um sinal de erro calculado na saída da rede é propagado no sentido reverso, camada a camada, e ao final deste processo os pesos são ajustados de acordo com uma regra de correção de erro. O treinamento de uma rede através de retropropagação é realizado de forma

supervisionada, ou seja, é apresentada à rede uma determinada entrada (exemplo) e também é disponibilizada a resposta desejada para aquela entrada. A seguir, apresentamos em detalhes o algoritmo de retropropagação.

O sinal de erro na saída do neurônio j na iteração n (isto é, apresentação do n -ésimo padrão de treinamento) é definido por

$$e_j(n) = s_j(n) - y_j(n), \quad (2.5)$$

onde $s_j(n)$ é a resposta desejada para o neurônio j da camada de saída. O valor instantâneo do erro quadrático para o neurônio j é definido por $\frac{1}{2}e_j^2(n)$. A soma dos erros quadráticos instantânea da rede é então definida por

$$\varepsilon(n) = \frac{1}{2} \sum_{j \in C} e_j^2(n), \quad (2.6)$$

onde C é o conjunto que contém todos os neurônios da camada de saída da rede. Seja N o número total de padrões de treinamento contido no conjunto de treinamento. O erro quadrático médio é então definido por

$$\varepsilon_{av} = \frac{1}{N} \sum_{n=1}^N \varepsilon(n). \quad (2.7)$$

Observe que a soma dos erros quadráticos instantâneos, e portanto o erro quadrático médio, é uma função de todos os pesos sinápticos da rede, já que são eles que definem a função de transferência ou mapeamento de entrada-saída da rede neural. Para um dado conjunto de treinamento, ε_{av} representa uma *função de custo*, isto é, uma medida de desempenho da rede neural. O objetivo do processo de treinamento é justamente minimizar ε_{av} .

Inicialmente, consideramos um método de treinamento em que os pesos são ajustados *padrão-a-padrão*. Os pesos da rede são ajustados de acordo com o erro computado para *cada* padrão apresentado à rede. Uma outra possibilidade de treinamento é o treinamento em lote ou batelada (*batch*), que será considerado posteriormente.

Considere a Figura 2.5, que mostra o neurônio de saída j , alimentado pelas ativações de todos os neurônios da camada imediatamente anterior. O nível de ativação interno do neurônio j é dado por

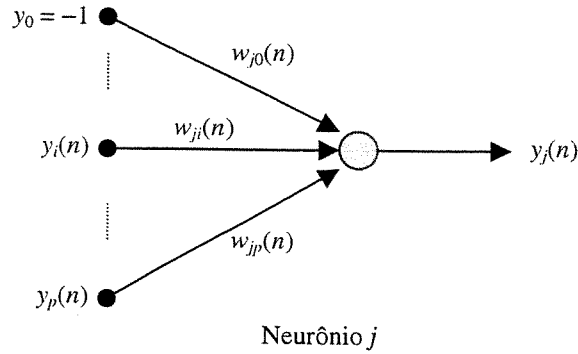


Figura 2.5 Neurônio de saída j , cujas entradas são as ativações de todos os neurônios da camada imediatamente anterior.

$$v_j(n) = \sum_{i=0}^p w_{ji}(n) y_i(n), \quad (2.8)$$

onde p é o número total de entradas (excluindo a polarização) aplicadas ao neurônio j , ou seja, o número de neurônios da camada imediatamente anterior. Portanto a ativação $y_j(n)$ do neurônio j é dada por

$$y_j(n) = f_j(v_j(n)). \quad (2.9)$$

Para minimizar o erro quadrático médio, necessitamos primeiramente determinar o gradiente instantâneo $\partial \mathcal{E}(n)/\partial w_{ji}(n)$. Aplicando a regra da cadeia, podemos expressar este gradiente como

$$\frac{\partial \mathcal{E}(n)}{\partial w_{ji}(n)} = \frac{\partial \mathcal{E}(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial w_{ji}(n)}. \quad (2.10)$$

Derivando a Equação (2.6) em relação a $e_j(n)$, obtemos

$$\frac{\partial \mathcal{E}(n)}{\partial e_j(n)} = e_j(n). \quad (2.11)$$

Derivando a Equação (2.5) em relação a $y_j(n)$, obtemos

$$\frac{\partial e_j(n)}{\partial y_j(n)} = -1. \quad (2.12)$$

Agora, derivando a Equação (2.9) em relação a $v_j(n)$, obtemos

$$\frac{\partial y_j(n)}{\partial v_j(n)} = f'_j(v_j(n)), \quad (2.13)$$

onde o símbolo ' significa derivação em relação ao argumento. Finalmente, derivando a Equação (2.8) em relação a $w_{ji}(n)$ obtemos

$$\frac{\partial v_j(n)}{\partial w_{ji}(n)} = y_i(n). \quad (2.14)$$

Substituindo as Equações (2.11) a (2.14) na Equação (2.10), obtemos

$$\frac{\partial \varepsilon(n)}{\partial w_{ji}(n)} = -e_j(n) f'_j(v_j(n)) y_i(n), \quad (2.15)$$

que é a equação da derivada do erro instantâneo $\varepsilon(n)$ em relação ao peso sináptico $w_{ji}(n)$ do neurônio j da camada de saída. Observe que a Equação (2.15) corresponde a um componente do vetor gradiente do erro, cujos elementos representam a derivada parcial de $\varepsilon(n)$ em relação a todos os pesos da rede neural, arranjados em uma ordem fixa, mas arbitrária. Definindo o *gradiente local* $\delta_j(n)$ como

$$\delta_j(n) = -\frac{\partial \varepsilon(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} = e_j(n) f'_j(v_j(n)), \quad (2.16)$$

podemos rescrever (2.15) na forma

$$\frac{\partial \varepsilon(n)}{\partial w_{ji}(n)} = -\delta_j(n) y_i(n). \quad (2.17)$$

Para minimizar o erro, vamos aplicar aos pesos uma correção proporcional ao oposto do gradiente do erro, pois vamos caminhar no espaço de pesos na direção oposta ao do gradiente. Matematicamente, podemos expressar esta regra de ajuste na forma:

$$w_{ji}(n+1) = w_{ji}(n) - \eta(n) \frac{\partial \varepsilon(n)}{\partial w_{ji}(n)}, \quad (2.18)$$

onde $\eta(n)$ é o *passo* do algoritmo na iteração n . Note que, no algoritmo de retropropagação original, proposto por RUMELHART *et al.* (1986), é utilizado um passo constante durante todas

as iterações do algoritmo, passo este conhecido como *taxa de aprendizagem*. Da teoria de otimização não-linear de funções é sabido que há alternativas melhores a esta abordagem, como o uso de procedimentos de busca unidimensional para determinar $\eta(n)$ a cada iteração do algoritmo (veja, por exemplo, LUENBERGER (1984) ou BAZARAA *et al.* (1992)).

Observe, das Equações (2.16) e (2.17), que um fator fundamental para o cálculo do gradiente do erro é o sinal do erro $e_j(n)$. Devemos então considerar dois casos distintos: no caso I, o neurônio j é um neurônio da camada de saída da rede e no caso II, o neurônio j é um neurônio pertencente a uma camada escondida, sendo então denominado neurônio escondido. O caso I é bastante simples, pois as saídas desejadas para a rede neural são conhecidas. Já no caso II, não sabemos qual a saída desejada para um neurônio escondido, e portanto, não é possível calcular diretamente o sinal de erro. Este problema é resolvido de forma bastante elegante através da retropropagação do erro de saída através da rede.

Consideraremos agora os casos I e II individualmente.

Caso I: Neurônio j é um Neurônio de Saída

Quando o neurônio j é um neurônio de saída, sabemos qual é a saída desejada $d_j(n)$ para o neurônio. Assim, podemos usar a Equação (2.5) para calcular o sinal de erro $e_j(n)$ associado ao neurônio j , com $s_j(n) = d_j(n)$. Com o erro $e_j(n)$ determinado, é direto o cálculo de $\delta_j(n)$ usando a Equação (2.16).

Caso II: Neurônio j é um Neurônio Escondido

Quando o neurônio j é um neurônio escondido, não há nenhuma saída desejada pré-especificada para o neurônio. Assim, o sinal de erro de um neurônio escondido deve ser calculado em termos dos sinais de erro de todos os neurônios aos quais o neurônio escondido está diretamente conectado. Considere a Figura 2.6, que mostra o neurônio j como um neurônio escondido da rede. Da Equação (2.16), podemos redefinir o gradiente local $\delta_j(n)$ para o neurônio escondido j como

$$\delta_j(n) = -\frac{\partial \varepsilon(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} = -\frac{\partial \varepsilon(n)}{\partial y_j(n)} f'_j(v_j(n)), \quad (2.19)$$

na qual fizemos uso da Equação (2.13). Para calcular a derivada parcial $\partial \varepsilon(n) / \partial y_j(n)$, podemos proceder como segue: da Figura 2.6, podemos ver que, para o neurônio k pertencente à camada de saída,

$$\varepsilon(n) = \frac{1}{2} \sum_{k \in C} e_k^2(n). \quad (2.20)$$

Derivando a Equação (2.20) em relação a $y_j(n)$, obtemos

$$\frac{\partial \varepsilon(n)}{\partial y_j(n)} = \sum_k e_k \frac{\partial e_k(n)}{\partial y_j(n)}. \quad (2.21)$$

Usando a regra da cadeia para calcular $\partial e_k(n) / \partial y_j(n)$, podemos rescrever a Equação (2.21) na forma

$$\frac{\partial \varepsilon(n)}{\partial y_j(n)} = \sum_k e_k(n) \frac{\partial e_k(n)}{\partial v_k(n)} \frac{\partial v_k(n)}{\partial y_j(n)}. \quad (2.22)$$

O erro na saída do neurônio k é dado por

$$e_k(n) = d_k(n) - y_k(n) = d_k(n) - f_k(v_k(n)), \quad (2.23)$$

onde $v_k(n)$ é o nível de ativação interna do neurônio k . Portanto,

$$\frac{\partial e_k(n)}{\partial v_k(n)} = -f'_k(v_k(n)). \quad (2.24)$$

Da Figura 2.6, podemos ver que o nível de ativação interna do neurônio k é dado por

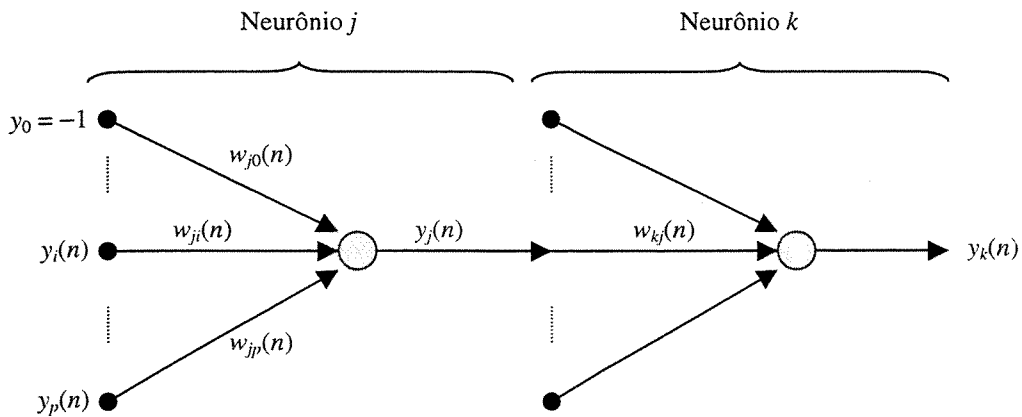


Figura 2.6 Neurônio escondido j conectado a um neurônio de saída k .

$$v_k(n) = \sum_{j=0}^q w_{kj}(n) y_j(n), \quad (2.25)$$

onde q é o número total de entradas (excluindo a polarização) aplicadas ao neurônio k . Derivando a Equação (2.25) em relação a $y_j(n)$ obtemos

$$\frac{\partial v_k(n)}{\partial y_j(n)} = w_{kj}(n). \quad (2.26)$$

Assim, substituindo as Equações (2.24) e (2.26) na Equação (2.22) obtemos a derivada parcial desejada:

$$\begin{aligned} \frac{\partial \mathcal{E}(n)}{\partial y_j(n)} &= -\sum_k e_k(n) f'_k(v_k(n)) w_{kj}(n) = \\ &= -\sum_k \delta_k(n) w_{kj}(n) \end{aligned} \quad (2.27)$$

onde, na segunda linha, fizemos uso da definição de gradiente local $\delta_k(n)$ dada na Equação (2.16), com o índice j substituído por k .

Finalmente, substituindo a Equação (2.27) na Equação (2.19), obtemos a expressão para o gradiente local $\delta_j(n)$ para o neurônio escondido j :

$$\delta_j(n) = f'_j(v_j(n)) \sum_k \delta_k(n) w_{kj}(n). \quad (2.28)$$

O termo $f'_j(v_j(n))$ na Equação (2.28) depende apenas da função de ativação associada ao neurônio j . O termo restante, isto é, o somatório sobre k , depende de dois conjuntos de termos. O primeiro conjunto de termos, os $\delta_k(n)$, exigem o conhecimento dos sinais de erro $e_k(n)$, para todos os neurônios localizados na camada imediatamente posterior à camada onde se encontra o neurônio j , e que estão diretamente conectados ao neurônio j ; veja a Figura 2.6. O segundo conjunto de termos, os $w_{kj}(n)$, consiste nos pesos sinápticos associados a estas conexões.

Podemos agora resumir os resultados obtidos para o algoritmo de retropropagação. Em primeiro lugar, a derivada parcial da soma dos erros instantâneos (Equação (2.6)) em relação ao peso $w_{ji}(n)$ que conecta o neurônio i ao neurônio j é definida por

$$\begin{bmatrix} \text{Derivada do} \\ \text{Erro em} \\ \text{Relação a } w_{ji}(n) \end{bmatrix} = - \begin{bmatrix} \text{Gradiente} \\ \text{local } \delta_j(n) \end{bmatrix} \begin{bmatrix} \text{Sinal de entrada} \\ \text{do neurônio } j \\ y_i(n) \end{bmatrix}$$

Em segundo lugar, o cálculo do gradiente local $\delta_j(n)$ depende do neurônio j ser um neurônio de saída ou um neurônio escondido:

1. Se o neurônio j é um neurônio de saída, $\delta_j(n)$ é igual ao produto da derivada $f'_j(v_j(n))$ pelo sinal de erro $e_j(n)$, ambos associados ao neurônio j ; veja Equação (2.16).
2. Se o neurônio j é um neurônio escondido, $\delta_j(n)$ é igual ao produto da derivada $f'_j(v_j(n))$ associada ao neurônio j e a soma ponderada dos δ 's calculados para os neurônios da camada posterior que estão conectados ao neurônio j ; veja Equação (2.28).

2.4.1 Os Dois Passos de Computação

O algoritmo de retropropagação envolve a execução de dois passos distintos de computação: o *processamento direto* (*forward*) e o *processamento reverso* (*backward*).

O procedimento direto é executado no sentido entrada $\rightarrow$ saída da rede. Neste caso, um exemplo de treinamento é apresentado à rede e as saídas de todos os neurônios são computadas, usando as Equações (2.8) e (2.9). Comparamos então as saídas da rede com as saídas desejadas e calculamos o erro, usando as Equações (2.5) e (2.6).

O processamento reverso é executado no sentido saída $\rightarrow$ entrada. Neste caso, os sinais de erro são propagados da saída para a entrada, camada a camada, através dos cálculos dos δ 's (gradientes locais) para cada neurônio da rede. Para os neurônios de saída, δ é calculado usando a Equação (2.16) e os respectivos pesos são ajustados usando a Equação (2.18). Com os δ 's da camada de saída calculados, usamos a Equação (2.28) para calcular os δ 's dos neurônios localizados na camada imediatamente anterior à camada de saída; os pesos destes neurônios são então atualizados segundo a Equação (2.18). Continuamos este procedimento camada a camada, até que a camada de entrada seja alcançada.

2.4.2 Funções de Ativação

Observando as Equações (2.16) e (2.28), vemos que precisamos conhecer a derivada da função de ativação $f(\cdot)$ para calcular os gradientes locais δ . Assim, vemos que

diferenciabilidade é uma propriedade fundamental que deve ser apresentada pela função de ativação. Uma das funções mais utilizadas é a função logística, definida na Equação (2.4). Assim, para um neurônio j , teremos

$$y_j(n) = f_j(v_j(n)) = \frac{1}{1 + e^{-v_j(n)}}$$

onde $v_j(n)$ é o nível de ativação interna do neurônio j . Uma característica interessante da função logística é que sua derivada pode ser expressa em termos da ativação $y_j(n)$, a saber:

$$f'_j(v_j(n)) = y_j(n)[1 - y_j(n)].$$

A função logística é limitada, assumindo valores no intervalo $(0, 1)$. Em muitas situações, é desejável termos ativações dos neurônios no intervalo $(-1, 1)$. Uma função muito utilizada nestes casos é a tangente hiperbólica:

$$f_j(v_j(n)) = a \cdot \text{tgh}(b \cdot v_j(n)) = a \cdot \left(\frac{1 - e^{-b \cdot v_j(n)}}{1 + e^{-b \cdot v_j(n)}} \right),$$

onde a e b são constantes de escalamento. Valores típicos para a e b são $a = 1,716$ e $b = \frac{2}{3}$ (HAYKIN, 1999).

2.4.3 Modos de Treinamento

Uma apresentação completa do conjunto de treinamento durante o processo de aprendizagem é denominada *época*. Dado um conjunto de treinamento, o algoritmo de retropropagação pode ser executado em dois modos distintos:

1. *Modo padrão-a-padrão ou instantâneo (pattern mode)*: os pesos são atualizados após a apresentação de cada padrão à rede neural. Mais especificamente, considere uma época consistindo da apresentação de N exemplos (padrões) de treinamento organizados na ordem $[\mathbf{x}(1) \mathbf{d}(1)], \dots, [\mathbf{x}(N) \mathbf{d}(N)]$. O primeiro exemplo da época $[\mathbf{x}(1) \mathbf{d}(1)]$ é apresentado à rede, e a sequência de processamentos direto e reverso é realizada, resultando no ajuste de pesos da rede, incluindo os pesos de polarização. Então o segundo exemplo da época $[\mathbf{x}(2) \mathbf{d}(2)]$ é apresentado, e a sequência de processamentos direto e reverso é repetida, e os pesos são novamente ajustados. Este processo continua até que o

último padrão $[\mathbf{x}(N) \mathbf{d}(N)]$ seja apresentado à rede, finalizando a época. Neste modo, é importante apresentar os padrões em ordem aleatória a cada época, já que a ordem de apresentação dos padrões de treinamento não representa nenhuma informação que deva condicionar o processo de ajuste de pesos.

2. *Modo por lote ou batelada (batch mode)*: os pesos são ajustados somente após a apresentação de todos os exemplos que constituem uma época. Para uma determinada época, definimos a função de custo como o erro quadrático médio, definido na Equação (2.7), e que reproduzimos aqui em forma expandida:

$$\varepsilon_{av} = \frac{1}{2N} \sum_{n=1}^N \sum_{j \in C} e_j^2(n).$$

A correção a ser aplicada a cada peso w_{ji} será proporcional à derivada parcial de ε_{av} em relação a w_{ji} . Assim,

$$\frac{\partial \varepsilon_{av}}{\partial w_{ji}} = \frac{1}{N} \sum_{n=1}^N \frac{\partial \varepsilon(n)}{\partial w_{ji}},$$

onde, para calcular $\partial \varepsilon(n) / \partial w_{ji}$ procedemos da mesma forma descrita anteriormente.

O desempenho de um modo ou outro de treinamento depende do problema em questão. MØLLER (1993) argumenta que o modo instantâneo parece apresentar melhor desempenho em problemas caracterizados por um conjunto de treinamento grande contendo um grande número de informação redundante e por superfícies de erro não muito complexas. O modo por lote, entretanto, é superior em problemas que não apresentam estas características. MØLLER (1993) também argumenta que o modo instantâneo não é consistente com a teoria clássica de otimização, embora se assemelhe a certos tipos de processos estocásticos.

2.4.4 Critério de Parada

Seja o vetor de pesos $\mathbf{w}^* \in \mathcal{R}^P$, onde P é o número de pesos ajustáveis da rede neural, um mínimo local ou global da superfície de erro definida no espaço de pesos, $\varepsilon: \mathcal{R}^P \rightarrow \mathcal{R}$. Uma condição necessária para $\mathbf{w}^*$ ser um mínimo é que o vetor gradiente $\nabla \varepsilon$ da superfície de

erro em relação a $\mathbf{w}$ se anule em $\mathbf{w} = \mathbf{w}^*$. HAYKIN (1999) sugere três possíveis critérios de parada para o algoritmo de retropropagação:

1. O algoritmo de retropropagação converge quando a norma euclidiana do gradiente do erro $\|\nabla \varepsilon\|$ for inferior a um limiar pré-especificado e arbitrariamente pequeno.
2. O algoritmo de retropropagação converge quando a taxa de variação absoluta no erro quadrático médio por época for suficientemente pequena.
3. O algoritmo de retropropagação converge quando $\|\nabla \varepsilon\| \leq \xi$, onde ξ é um limiar suficientemente pequeno, ou quando $\varepsilon_{av}(\mathbf{w}) \leq \tau$, onde τ é um limiar suficientemente pequeno.

2.4.5 Inicialização dos Pesos

O primeiro passo do algoritmo de retropropagação é, obviamente, a inicialização dos pesos da rede, o que corresponde à definição de um ponto inicial da superfície de erro. Uma boa escolha inicial dos pesos é fundamental para um bom desempenho do algoritmo de retropropagação. Uma inicialização inadequada (ponto inicial mal localizado) pode fazer com que o algoritmo de treinamento fique preso em um mínimo local ou apresente problemas numéricos que de outra forma poderiam ser evitados. Como usualmente não temos nenhuma informação que possa ser diretamente empregada na inicialização dos pesos da rede, um método comumente utilizado é inicializar os pesos aleatoriamente, com distribuição uniforme sobre um pequeno intervalo em torno do zero.

Diversos métodos de inicialização de pesos já foram propostos na literatura (por exemplo, KIM & RA (1991), BOERS & KUIPER (1992), NGUYEN & WIDROW (1990)). DE CASTRO & VON ZUBEN (1998) propuseram um método de inicialização no qual os dados de treinamento são utilizados para inicializar os pesos da rede de forma que os neurônios estejam operando inicialmente nas proximidades de sua região mais linear, tendo obtido resultados superiores, em média, a todos os demais resultados que já haviam sido apresentados na literatura e que foram considerados para comparação. DE CASTRO *et al.* (1998) utilizaram algoritmos genéticos para extrair características importantes de um bom conjunto de pesos iniciais para treinamento. Basicamente, o processo evolutivo mostrou que os melhores e piores conjuntos de pesos iniciais correspondiam a casos

caracterizados por apresentarem valores absolutos médios elevados, enquanto que conjuntos de pesos iniciais caracterizados por apresentarem valores absolutos médios reduzidos apresentavam um desempenho intermediário. Sendo assim, como não se sabe a priori como deve ser a distribuição inicial de pesos, é grande a possibilidade de produzir um conjunto inicial ruim caso se adotem valores absolutos médios elevados.

2.4.6 Resumo do Algoritmo de Retropropagação

A seguir, apresentamos resumidamente os passos do algoritmo de retropropagação. Consideramos, aqui, a versão instantânea do algoritmo; para a versão por lote, apenas algumas pequenas modificações precisam ser incorporadas, tendo sido apresentadas na Seção 2.4.3.

1. *Inicialização.* Inicialize os pesos da rede aleatoriamente em um intervalo pequeno ou segundo algum método mais refinado proposto na literatura.
2. *Processamento direto.* Apresente um padrão à rede. Compute as ativações de todos os neurônios da rede e então compute o erro quadrático instantâneo.
3. *Passo reverso.* Calcule os gradientes locais δ s para cada neurônio da rede, no sentido retroativo (isto é, da saída para a entrada), camada a camada. Se o neurônio em questão for um neurônio de saída da rede, use a Equação (2.16), senão use a Equação (2.28).
4. *Teste de parada.* Teste o critério de parada adotado. Se satisfeito, termine o algoritmo; senão volte ao passo 2.

2.5 Métodos de Segunda Ordem

Como discutimos anteriormente, o treinamento de redes neurais multicamadas é um problema de otimização não-linear de uma função de custo, que mede o erro quadrático médio produzido pela saída da rede neural frente a uma saída desejada. Este fato traz grandes vantagens devido à vasta literatura existente sobre métodos de otimização não-linear, onde podemos encontrar uma grande variedade de métodos poderosos para otimização que podem ser prontamente aplicados ao problema de treinamento de redes neurais, caso se disponha de informações suficientes acerca da função de erro a ser minimizada.

O algoritmo de retropropagação apresentado na Seção 2.4 é uma implementação de um método conhecido como *método do gradiente*. No método do gradiente, o vetor de parâmetros (pesos) são ajustados na direção oposta ao do vetor gradiente. O método do gradiente pode ser classificado como um método de *primeira ordem*, já que utiliza apenas a informação do gradiente da função de erro para ajustar os pesos da rede. Os métodos de primeira ordem são conhecidos por serem ineficientes no tratamento de problemas de larga escala, pois apresentam taxas de convergência muito pobres, especialmente em regiões próximas a mínimos locais (LUENBERGER, 1984; BAZARAA *et al.*, 1992).

Em vista disso, e do fato de que os dados para treinamento geralmente apresentam grande dimensionalidade, é justificável a utilização de um método de otimização não-linear de segunda ordem. Nos métodos de segunda ordem, além do vetor gradiente da função objetivo, fazemos uso também da matriz *hessiana* (matriz de derivadas de segunda ordem) da função erro. Na literatura referente à otimização não-linear, uma classe de algoritmos de segunda ordem denominados algoritmos de *gradiente conjugado* é apontada como apropriada para problemas de larga escala. Apesar de notadamente superiores aos métodos de primeira ordem, os métodos de segunda ordem também apresentam desvantagens, sendo a principal delas o alto custo computacional associado ao cálculo e armazenamento da matriz hessiana.

DE CASTRO (1998) realizou comparações efetivas entre diversos métodos de primeira e segunda ordem aplicados no treinamento de redes neurais artificiais. Dentre os algoritmos testados, o algoritmo do *gradiente conjugado escalonado* proposto por MØLLER (1993) se destacou por apresentar melhor desempenho na maioria dos problemas considerados. Seu desempenho pode ser ainda melhor se for adotado o método de cálculo exato da informação de segunda ordem, proposto por PEARLMUTTER (1994). Este algoritmo apresenta as seguintes vantagens em relação a outros algoritmos de segunda ordem:

- não possui nenhum parâmetro crítico dependente de definição por parte do usuário;
- nenhum procedimento de busca unidimensional é necessário. Estes procedimentos, extremamente custosos computacionalmente, são dispensáveis no algoritmo de Møller.

2.6 Aproximação de Funções

Nesta seção apresentaremos uma importante propriedade das redes neurais multicamada com uma única camada de neurônios escondidos. Iniciaremos apresentando o *problema geral de aproximação* (VON ZUBEN, 1996):

Seja Ω uma região compacta do $\mathcal{R}^m$ e seja $g: \Omega \subset \mathcal{R}^m \rightarrow \mathcal{R}^r$ uma função desconhecida que associa a cada vetor de variáveis de entrada $\mathbf{x} \in \Omega$ um vetor de saídas $\mathbf{s} \in \mathcal{R}^r$. Se g é uma função limitada, então a imagem de Ω sob g é também uma região compacta $g[\Omega] \subset \mathcal{R}^r$ definida na forma:

$$g[\Omega] = \{g(\mathbf{x}) : \mathbf{x} \in \Omega\}.$$

Elementos de um conjunto multidimensional de pares de vetores de entrada-saída $(\mathbf{x}_l, \mathbf{s}_l)$, $l = 1, \dots, N$, são gerados considerando-se que os vetores de entrada $\mathbf{x}_l$ estão distribuídos na região compacta $\Omega \subset \mathcal{R}^m$ de acordo com uma função densidade de probabilidade fixa $d_p: \Omega \subset \mathcal{R}^m \rightarrow [0, 1]$, e que os vetores de saída $\mathbf{s}_l$ assumem valores finitos produzidos pelo mapeamento definido pela função g na forma:

$$\mathbf{s}_l = g(\mathbf{x}_l) + \boldsymbol{\varepsilon}_l, \quad l = 1, \dots, N,$$

onde $\boldsymbol{\varepsilon}_l \in \mathcal{R}^r$ é um vetor de erro aditivo, composto por variáveis aleatórias de média zero e variância fixa, geralmente refletindo a dependência de $\mathbf{s}_l$, $l = 1, \dots, N$, em relação a quantidades não-controláveis (por exemplo, ruído no processo de amostragem) e não-observáveis (por exemplo, variáveis não-mensuráveis).

Dado o conjunto de dados amostrados $\{(\mathbf{x}_l, \mathbf{s}_l)\}_{l=1}^N$ definidos no espaço $\Omega \times \mathcal{R}^r$, o problema de aproximação envolve a determinação de um modelo de aproximação $\hat{g}: \Omega \rightarrow \mathcal{R}^r$ que corresponda à melhor aproximação para o mapeamento realizado pela função $g: \Omega \rightarrow \mathcal{R}^r$.

Considere agora uma rede neural multicamada com uma única camada escondida, como a ilustrada na Figura 2.7. Nesta figura temos uma rede com m entradas, n neurônios na camada escondida e r neurônios na camada de saída. Consideraremos aqui que todos os neurônios da camada intermediária possuem a mesma função de ativação $f(\cdot)$. Assumimos

também que $f(\cdot)$ é uma função contínua, limitada e monotônica crescente. Os neurônios da camada de saída possuem como função de ativação a função identidade. Os pesos da rede podem ser arranjados na forma de matrizes, como segue:

$$\mathbf{V} = \begin{bmatrix} v_{10} & v_{11} & \cdots & v_{1m} \\ v_{20} & v_{21} & \cdots & v_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ v_{n0} & v_{n1} & \cdots & v_{nm} \end{bmatrix} \text{ e } \mathbf{W} = \begin{bmatrix} w_{10} & w_{11} & \cdots & w_{1n} \\ w_{20} & w_{21} & \cdots & w_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{r0} & w_{r1} & \cdots & w_{rn} \end{bmatrix}.$$

Redes neurais com uma única camada escondida são consideradas estruturas poderosas para aproximação de funções. Note que o aprendizado de um mapeamento entrada-saída a partir do conjunto de dados amostrados $\{(\mathbf{x}_l, \mathbf{s}_l)\}_{l=1}^N$ pode ser considerado como a síntese de um modelo de aproximação $\hat{g}(\cdot)$ para a função desconhecida $g(\cdot)$. Podemos então usar uma rede neural com uma camada intermediária e treinamento supervisionado para obter estimativas $\hat{\mathbf{s}}$ para todo $\mathbf{x} \in \Omega$, tal que $\hat{\mathbf{s}} = \hat{g}(\mathbf{x})$. Da Figura 2.7, vemos que a ativação do k -ésimo neurônio de saída ($k = 1, \dots, r$) é dada por

$$\hat{s}_k = w_{k0} + \sum_{j=1}^n \left[w_{kj} f \left(\sum_{i=1}^m v_{ji} x_i + v_{j0} \right) \right], \quad (2.29)$$

onde:

- v_{ji} , $i \neq 0$, representa o peso sináptico que conecta a i -ésima entrada ao j -ésimo neurônio escondido;
- v_{j0} é a polarização do j -ésimo neurônio escondido;
- w_{kj} , $j \neq 0$, representa o peso que conecta o j -ésimo neurônio escondido ao k -ésimo neurônio de saída;
- w_{k0} representa a polarização do k -ésimo neurônio de saída.

Redes neurais com uma camada intermediária, como as descritas aqui, apresentam uma propriedade que as tornam bastante atraentes para uso em problemas de aproximação de funções: a capacidade de *aproximação universal*. Apresentamos então o *teorema da aproximação universal* (CYBENKO, 1989; FUNAHASHI, 1989; HORNIK *et al.*, 1989):

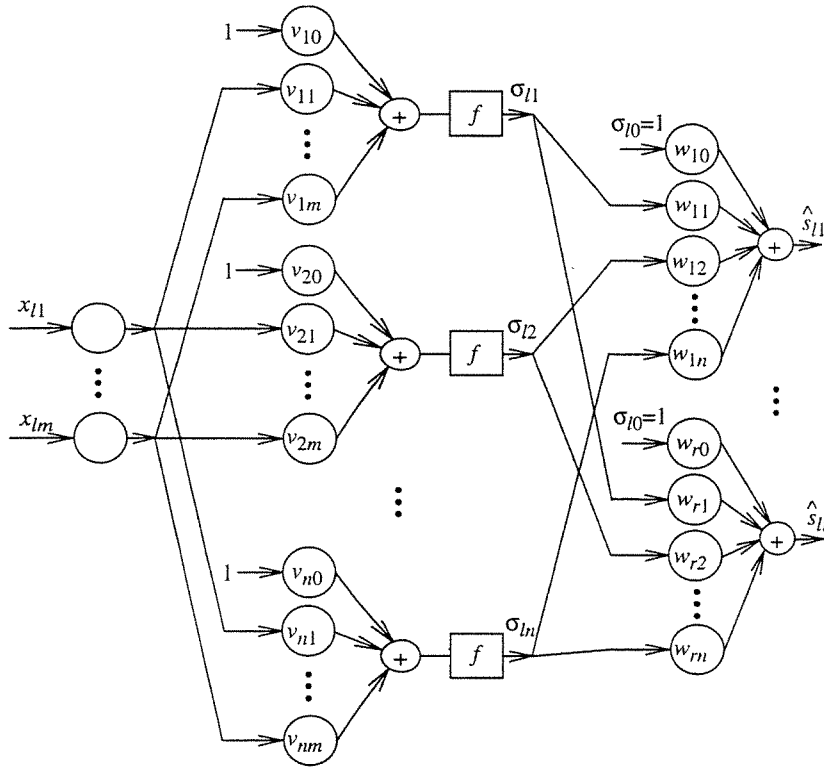


Figura 2.7 Rede neural com uma única camada escondida.

Teorema 2.1 Seja $C[\Omega]$ o espaço de funções contínuas em Ω , onde Ω é uma região compacta. Se $g : \Omega \subset \mathbb{R}^m \rightarrow \mathbb{R}^r$ é tal que $g \in C[\Omega]$, então, para qualquer $\xi > 0$, existem n , $\mathbf{V}$ e $\mathbf{W}$ tal que

$$\text{dist}(g(\mathbf{x}), RN^{[n]}(\mathbf{V}, \mathbf{W}, \mathbf{x})) < \xi \quad \forall \mathbf{x} \in \Omega,$$

onde o operador $\text{dist}(\cdot, \cdot)$ mede a distância entre a função g a ser aproximada e a função $RN^{[n]}(\mathbf{V}, \mathbf{W}, \mathbf{x})$ produzida pela rede neural. Note que n é o número de neurônios na camada intermediária.

O Teorema 2.1 afirma que existe uma rede neural com uma única camada escondida, e com um número adequado de neurônios na camada escondida, que seja capaz de aproximar com precisão ξ uma função contínua desconhecida, dado um conjunto de amostras para treinamento $\{(\mathbf{x}_l, \mathbf{s}_l)\}_{l=1}^N$. Observe entretanto que o teorema não afirma que uma única camada escondida é ótima no sentido de tempo de aprendizado ou facilidade de implementação e também não indica como obter n , $\mathbf{V}$ e $\mathbf{W}$, sendo apenas um resultado existencial.

2.7 Conclusão

Neste capítulo apresentamos uma introdução às redes neurais multicamadas. Os principais pontos abordados foram o modelo de neurônio, o algoritmo de retropropagação, os métodos de segundo ordem e a visão de redes neurais como modelos de aproximação de funções. Os tópicos apresentados serão importantes nos capítulos subsequentes deste trabalho.

Capítulo 3

Algoritmos Genéticos

Métodos baseados em computação evolutiva constituem uma classe de algoritmos de busca e otimização estocástica inspirados na teoria da evolução natural de Darwin. Estes algoritmos têm recebido especial atenção nos últimos tempos por se tratarem de métodos robustos, capazes de fornecer soluções de alta qualidade para problemas considerados intratáveis por métodos tradicionais de otimização, os quais foram concebidos para problemas lineares, contínuos e diferenciáveis. Mas, como observa SCHWEFEL (1994), o mundo real é não-linear e dinâmico, cheio de fenômenos como descontinuidade, instabilidade estrutural e formas geométricas fractais. Em problemas em que precisamos levar em conta tais fenômenos, os métodos tradicionais certamente não apresentarão desempenho satisfatório. Métodos evolutivos são uma alternativa para tentar superar as limitações apresentadas por métodos tradicionais, embora não garantam a obtenção da solução exata.

Neste capítulo apresentaremos os principais aspectos relacionados à computação evolutiva, mais especificamente algoritmos genéticos. Este capítulo encontra-se estruturado da seguinte forma: na Seção 3.1, apresentamos algumas idéias referentes à teoria da evolução natural; na Seção 3.2, apresentamos a computação evolutiva e descrevemos rapidamente as principais abordagens presentes na literatura; na Seção 3.3 apresentamos mais detalhadamente os algoritmos genéticos; na Seção 3.4 apresentamos a teoria dos esquemas e na Seção 3.5 apresentamos as conclusões e algumas perspectivas na área de computação evolutiva.

3.1 Evolução Natural

A evolução natural é o processo que guia o surgimento de estruturas orgânicas complexas e adaptadas ao ambiente (BÄCK *et al.*, 1997). De forma sucinta, e com grau

elevado de simplificação, BÄCK *et al.* (1997) descreve a evolução como o resultado da influência mútua entre a criação de informação genética nova e sua avaliação e seleção. Um indivíduo de uma população é afetado por outros indivíduos da população (por exemplo, através de competição por alimentação, predadores, acasalamento, etc.) e também pelo ambiente em que vive (por exemplo, através da oferta de comida, clima, etc.). Quanto maior a adaptação de um indivíduo a tais condições, maior a chance do indivíduo sobreviver por mais tempo e gerar uma prole, que por sua vez herda a informação genética dos pais (possivelmente com algum erro de cópia e sujeita à recombinação das informações fornecidas pelos pais). Ao longo do processo de evolução, vai ocorrendo na população uma penetração majoritária de informação genética oriunda de indivíduos com adaptação acima da média. Além disso, o caráter não-determinístico da reprodução leva a uma produção permanente de informação genética nova e, portanto, à criação de descendentes diferenciados. Para maiores detalhes, veja ATMAR (1994) e FOGEL (1999).

Indivíduos e espécies podem ser vistos como uma dualidade entre seu código genético (*genótipo*) e suas características comportamentais, fisiológicas e morfológicas (*fenótipo*) (FOGEL, 1994). Em sistemas evoluídos naturalmente, não existe uma relação biunívoca entre um gene (elemento do genótipo) e uma característica (elemento do fenótipo): um único gene pode afetar diversos traços fenotípicos simultaneamente (*pleiotropia*) e uma única característica fenotípica pode ser determinada pela interação de vários genes (*poligenia*). Os efeitos de pleiotropia e poligenia geralmente tornam os resultados de variações genéticas imprevisíveis. Sistemas naturais em evolução são fortemente pleiotrópicos e altamente poligênicos (HARTL & CLARK, 1989). O mesmo não ocorre em sistemas artificiais, onde uma das principais preocupações é com o custo computacional do sistema. Assim, em sistemas artificiais, existe uma relação de um-para-um entre genótipo e fenótipo.

O processo de evolução pode ser formalizado como segue (ATMAR, 1994; FOGEL, 1999): considere dois espaços de estados distintos – um espaço de estados genotípico (de codificação) **G** e um espaço fenotípico (comportamental) **F**. Considere também um alfabeto de entrada composto de símbolos provenientes do ambiente **I**.

O processo de evolução de uma população em uma geração encontra-se esquematizado na Figura 3.1. Existem 4 mapeamentos atuando neste processo:

$$f_1 : \mathbf{I} \times \mathbf{G} \rightarrow \mathbf{F},$$

$$f_2 : \mathbf{F} \rightarrow \mathbf{F},$$

$$f_3 : \mathbf{F} \rightarrow \mathbf{G},$$

$$f_4 : \mathbf{G} \rightarrow \mathbf{G}.$$

O mapeamento f_1 , denominada *epigênese*, mapeia elementos $g_1 \in \mathbf{G}$ em uma coleção particular de fenótipos p_1 do espaço fenotípico $\mathbf{F}$, cujo desenvolvimento é modificado por seu ambiente, um conjunto de símbolos $\{i_1, \dots, i_k\} \in \mathbf{I}$. Este mapeamento é inerentemente de muitos-para-um, pois existe uma infinidade de genótipos que podem resultar num mesmo fenótipo; elementos de um conjunto infinito de códigos não-expressos (não-participantes na produção do fenótipo) podem existir em g_1 (ATMAR, 1994).

O mapeamento f_2 , *seleção*, mapeia fenótipos p_1 em p_2 . Este mapeamento descreve os processos de seleção, imigração e emigração de indivíduos dentro da população local. Como a seleção natural opera apenas nas expressões fenotípicas do genótipo, o código g_1 não está envolvido no mapeamento f_2 . ATMAR (1994) enfatiza que a seleção atua apenas no sentido de eliminar as variantes comportamentais menos apropriadas do inevitável excesso da população, já que assume-se aqui que os recursos provenientes do ambiente são limitados,

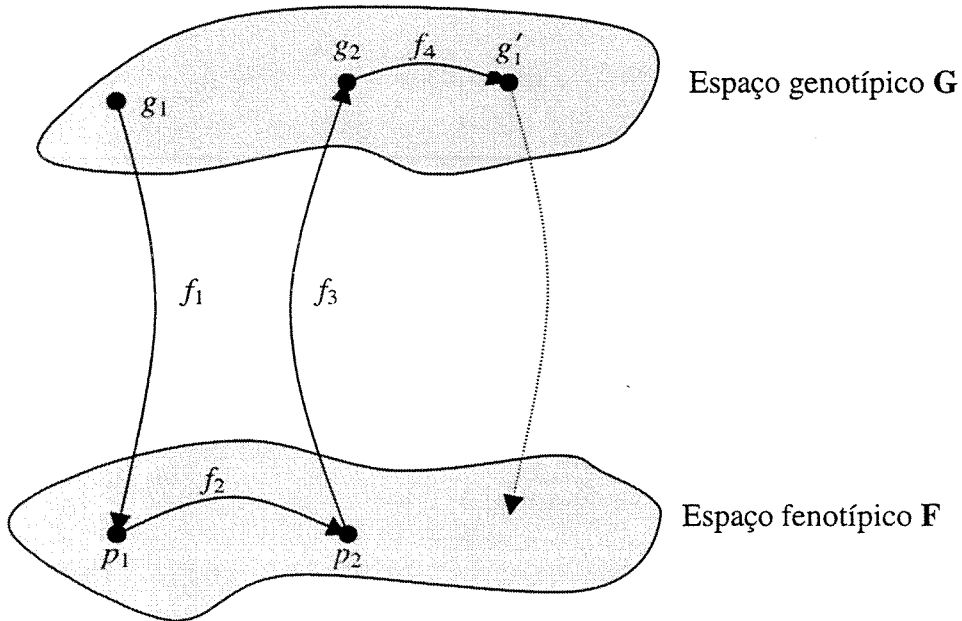


Figura 3.1 Evolução de uma população durante uma geração.

exigindo a competição pela sobrevivência. Neste processo de competição, a seleção nunca opera sobre uma característica simples, isoladamente do conjunto comportamental.

O mapeamento f_3 , *representação* (ATMAR, 1994) ou *sobrevivência genotípica* (FOGEL, 1999), descreve os efeitos dos processos de seleção e migração em G .

O mapeamento f_4 , *mutação e recombinação*, mapeia códigos $g_2 \in G$ em $g'_1 \in G$. Este mapeamento descreve as “regras” de mutação e recombinação, e abrange todas as alterações genéticas. A mutação é um erro de cópia no processo de transmissão do código genético dos pais para a sua prole. Em um universo com diferencial de entropia positivo, erros de replicação são inevitáveis e a otimização evolutiva torna-se inevitável em qualquer população que se reproduz em uma arena limitada (ATMAR, 1994).

Com a criação da nova população de genótipos g'_1 , uma geração está completa. A adaptação evolutiva ocorre em sucessivas iterações destes mapeamentos.

O biólogo Sewell Wright propôs, em 1931, o conceito de superfície de adaptação para descrever nível de adaptação de indivíduos e espécies (FOGEL, 1999). Uma população de genótipos é mapeada em seus respectivos fenótipos que por sua vez são mapeados na superfície de adaptação. Cada pico (máximo local) da superfície de adaptação corresponde a uma coleção de fenótipos otimizada, e portanto a um ou mais conjuntos de genótipos otimizados. A evolução é um processo que conduz, de forma probabilística, populações em direção a picos da superfície, enquanto que a seleção elimina variantes fenotípicas menos apropriadas. Outros pesquisadores propõem uma visão invertida da superfície de adaptação: populações avançam descendo picos da superfície de adaptação até que um ponto de mínimo seja encontrado.

Qualquer que seja o ponto de vista, a evolução é inerentemente um processo de otimização. A seleção produz fenótipos tão próximos do ótimo quanto possível, dadas condições iniciais, restrições ambientais e parâmetros evolutivos. Observe, no entanto, que em sistemas biológicos reais, não existem superfícies de adaptação estáticas. O ambiente está em constante mudança, fazendo com que populações estejam em constante evolução em direção a novos pontos de ótimo. Neste caso, assumindo que as mudanças ambientais são significativas, a taxa evolutiva deve ser suficientemente elevada para acompanhar as mudanças ambientais.

3.2 Computação Evolutiva

A computação evolutiva é formada por algoritmos inspirados na teoria da evolução natural de Darwin. A computação evolutiva pode desempenhar os seguintes papéis básicos:

- ferramenta adaptativa para solução de problemas;
- modelo computacional de processos evolutivos naturais.

Os sistemas baseados em computação evolutiva mantêm uma população de soluções potenciais, aplicam processos de seleção baseados na adaptação de um indivíduo e também empregam outros operadores “genéticos”. Diversas abordagens para sistemas baseados em evolução foram propostas, sendo que as principais diferenças entre elas dizem respeito aos operadores genéticos empregados (uma discussão sobre operadores genéticos será apresentada mais adiante neste capítulo). As principais abordagens propostas na literatura são:

- algoritmos genéticos;
- estratégias evolutivas;
- programação evolutiva.

Os *algoritmos genéticos* foram introduzidos por J. Holland em 1975 (HOLLAND, 1992) com o objetivo de formalizar matematicamente e explicar rigorosamente processos de adaptação em sistemas naturais e desenvolver sistemas artificiais (simulados em computador) que retenham os mecanismos originais encontrados em sistemas naturais. Os algoritmos genéticos empregam os operadores de *crossover* e mutação (a serem apresentados mais adiante neste capítulo). Os algoritmos genéticos têm sido intensamente aplicados em problemas de otimização, apesar de não ter sido este o propósito original que levou ao seu desenvolvimento.

Uma extensão dos algoritmos genéticos, denominada *programação genética*, foi introduzida por KOZA (1992)¹. A programação genética teve por objetivo inicial evoluir programas de computador usando os princípios da evolução natural. Atualmente a programação genética tem sido aplicada a uma grande variedade de problemas, como na síntese de circuitos elétricos analógicos (KOZA *et al.*, 1997) e na definição de arquiteturas de redes neurais artificiais (GRUAU, 1994).

¹ J. R. Koza detém uma patente sobre programação genética.

Estratégias evolutivas (RECHENBERG, 1973; SCHWEFEL, 1995) foram inicialmente propostas com o objetivo de solucionar problemas de otimização de parâmetros, tanto discretos como contínuos. As estratégias evolutivas empregam apenas o operador de mutação.

A *programação evolutiva*, introduzida por FOGEL *et al.* (1966), foi originalmente proposta como uma técnica para criar inteligência artificial através da evolução de máquinas de estado finito. A programação evolutiva também emprega apenas mutação. Recentemente, a programação evolutiva tem sido aplicada a problemas de otimização e é, neste caso, virtualmente equivalente às estratégias evolutivas; apenas pequenas diferenças no que diz respeito aos procedimentos de seleção e codificação de indivíduos estão presentes nas duas abordagens atualmente (FOGEL, 1994).

Apesar das abordagens acima citadas terem sido desenvolvidas de forma independente, seus algoritmos possuem uma estrutura comum. Usaremos o termo *algoritmo evolutivo* para denominar todos os sistemas baseados em evolução. A estrutura de um algoritmo evolutivo está ilustrada na Figura 3.2 (MICHALEWICZ, 1996).

Um algoritmo evolutivo é um algoritmo probabilístico que mantém uma população de indivíduos $P(t) = \{x'_1, \dots, x'_n\}$ na iteração (geração) t . Cada indivíduo representa um candidato à solução do problema em questão e, em qualquer implementação computacional, assume a forma de alguma estrutura de dados S . Cada solução x'_i é avaliada e produz alguma medida de adaptação, ou *fitness*. Então, uma nova população (iteração $t + 1$) é formada pela seleção dos indivíduos mais adaptados (passo de seleção). Alguns indivíduos da população são submetidos a transformações (passo de alteração) por meio de operadores genéticos para formar novas soluções. Existem transformações unárias m_i (mutação) que criam novos indivíduos através de pequenas mudanças em um indivíduo ($m_i : S \rightarrow S$), e transformações de ordem superior c_j (*crossover*), que criam novos indivíduos através da combinação de dois ou mais indivíduos ($c_j : S \times \dots \times S \rightarrow S$). Após um número de gerações, a condição de parada deve ser atendida, a qual geralmente indica a existência, na população, de um indivíduo que represente uma solução quase-ótima (razoável) para o problema.

As abordagens evolutivas apresentadas nesta seção diferem em diversos aspectos: nas estruturas de dados utilizadas para codificar um indivíduo, nos operadores genéticos empregados, nos métodos para criar a população inicial, nos métodos para selecionar

```

procedimento programa evolutivo
início
     $t \leftarrow 0$ 
    inicialize  $P(t)$ 
    avalie  $P(t)$ 
    enquanto (não condição de parada) faça
        início
             $t \leftarrow t + 1$ 
            selecione  $P(t)$  a partir de  $P(t - 1)$ 
            altere  $P(t)$ 
            avalie  $P(t)$ 
        fim
    fim

```

Figura 3.2 Estrutura de um algoritmo evolutivo.

indivíduos para a geração seguinte, etc. Entretanto, elas compartilham o mesmo princípio comum: uma população de indivíduos sofre algumas transformações e durante a evolução os indivíduos competem pela sobrevivência.

3.3 Algoritmos Genéticos

Os algoritmos genéticos empregam uma terminologia originada da teoria da evolução natural e da genética. Um indivíduo da população é representado por um único *cromossomo*, o qual contém a *codificação* (genótipo) de uma possível solução do problema (fenótipo). Cromossomos são usualmente implementados na forma de vetores, onde cada componente do vetor é conhecido como *gene*. Os possíveis valores que um determinado gene pode assumir são denominados *alelos*.

O processo de evolução executado por um algoritmo genético corresponde a um processo de busca em um espaço de soluções potenciais para o problema. Como enfatiza MICHALEWICZ (1996), esta busca requer um equilíbrio entre dois objetivos aparentemente conflitantes: o aproveitamento das melhores soluções e a exploração do espaço de busca (*exploitation* $\times$ *exploration*). Métodos de otimização do tipo *hillclimbing* são exemplos de métodos que aproveitam a melhor solução na busca de possíveis aprimoramentos; em compensação, estes métodos ignoram a exploração do espaço de busca. Métodos de busca aleatória são exemplos típicos de métodos que exploram o espaço de busca ignorando o

aproveitamento de regiões promissoras do espaço. Algoritmos genéticos constituem uma classe de métodos de busca de propósito geral que apresentam um balanço notável entre aproveitamento de melhores soluções e exploração do espaço de busca.

Os algoritmos genéticos pertencem à classe dos algoritmos probabilísticos, mas eles não são métodos de busca puramente aleatórios, pois eles combinam elementos de métodos de busca diretos e estocásticos. Outra propriedade importante dos algoritmos genéticos (assim como de todos os algoritmos evolutivos) é que eles mantêm uma população de soluções candidatas enquanto que os métodos alternativos, como *simulated annealing* (AARTS & KORST, 1989), processam um único ponto no espaço de busca a cada instante.

O processo de busca realizado pelos algoritmos genéticos é multi-direcional, através da manutenção de soluções candidatas, e encoraja a troca de informação entre as direções. A cada geração, soluções relativamente “boas” se reproduzem, enquanto que soluções relativamente “ruins” são eliminadas. Para fazer a distinção entre diferentes soluções é empregada uma função objetivo (de avaliação ou de adaptabilidade) que simula o papel da pressão exercida pelo ambiente sobre o indivíduo.

A estrutura de um algoritmo genético é a mesma do algoritmo evolutivo apresentado na Figura 3.2. Podemos descrever um algoritmo genético como segue (MICHALEWICZ, 1996). Durante a iteração t , um algoritmo genético mantém uma população de soluções potenciais (cromossomos, vetores), $P(t) = \{\mathbf{x}'_1, \dots, \mathbf{x}'_n\}$. Cada solução $\mathbf{x}'_i$ é avaliada e produz uma medida de sua adaptação, ou *fitness*. Uma nova população (iteração $t+1$) é então formada privilegiando a participação dos indivíduos mais adaptados. Alguns membros da nova população passam por alterações por meio de *crossover* e mutação para formar novas soluções potenciais. Este processo se repete até que um número pré-determinado de iterações seja atingido, ou até que o nível de adaptação esperado seja alcançado.

Um algoritmo genético para um problema particular deve ter os seguintes componentes:

- uma representação genética para soluções candidatas ou potenciais (processo de codificação);
- uma maneira de criar uma população inicial de soluções candidatas ou potenciais;

- uma função de avaliação que faz o papel da pressão ambiental, classificando as soluções em termos de sua adaptação ao ambiente (ou seja, sua capacidade de resolver o problema);
- operadores genéticos;
- valores para os diversos parâmetros usados pelo algoritmo genético (tamanho da população, probabilidades de aplicação dos operadores genéticos, etc.).

3.3.1 Codificação de Indivíduos

Cada indivíduo de uma população representa um candidato em potencial à solução do problema em questão. No algoritmo genético clássico, proposto por HOLLAND (1992), as soluções candidatas são codificadas em arranjos binários de tamanho fixo. A motivação para o uso de codificação binária vem da teoria dos esquemas (*schemata theory*), que será apresentada mais adiante. HOLLAND (1992) argumenta que seria benéfico para o desempenho do algoritmo maximizar o paralelismo implícito inerente ao algoritmo genético (veja Seção 3.4), e prova que um alfabeto binário maximiza o paralelismo implícito.

Entretanto, em diversas aplicações práticas a utilização de codificação binária leva a um desempenho insatisfatório. Em problemas de otimização numérica com parâmetros reais, algoritmos genéticos com representação em ponto flutuante frequentemente apresentam desempenho superior à codificação binária. MICHALEWICZ (1996) argumenta que a representação binária apresenta desempenho pobre quando aplicada a problemas numéricos com alta dimensionalidade e onde alta precisão é requerida. Suponha por exemplo, que temos um problema com 100 variáveis com domínio no intervalo $[-500, 500]$ e que precisamos de 6 dígitos de precisão após a casa decimal. Neste caso precisaríamos de um cromossomo de comprimento 3000, e teríamos um espaço de busca de dimensão aproximadamente 10^{1000} . Neste tipo de problema o algoritmo genético clássico apresenta desempenho pobre. MICHALEWICZ (1996) apresenta também simulações computacionais comparando o desempenho de algoritmos genéticos com codificação binária e com ponto flutuante, aplicados a um problema de controle. Os resultados apresentados mostram uma clara superioridade da codificação em ponto flutuante.

A argumentação de MICHALEWICZ (1996), de que o desempenho de um algoritmo genético com codificação binária é pobre quando o espaço de busca é de dimensão elevada,

não é universalmente aceita na literatura referente a algoritmos genéticos. FOGEL (1994) argumenta que o espaço de busca por si só (sem levar em conta a escolha da representação) não determina a eficiência do algoritmo genético. Espaços de busca de dimensão elevada podem às vezes ser explorados eficientemente, enquanto que espaços de busca de dimensão reduzida podem apresentar dificuldades significativas. FOGEL (1994), entretanto, concorda que a maximização do paralelismo implícito nem sempre produz um desempenho ótimo.

Fica claro, portanto, que a codificação é uma das etapas mais críticas na definição de um algoritmo genético. A definição inadequada da codificação pode levar a problemas de convergência prematura do algoritmo genético. A estrutura de um cromossomo deve representar uma solução como um todo, e deve ser a mais simples possível. Em problemas de otimização restrita, a codificação adotada pode fazer com que indivíduos modificados por *crossover*/mutação sejam inválidos. Nestes casos, cuidados especiais devem ser tomados na definição da codificação e/ou dos operadores.

3.3.2 Definição da População Inicial

O método mais comum utilizado na criação da população é a inicialização aleatória dos indivíduos. Se algum conhecimento inicial a respeito do problema estiver disponível, pode ser utilizado na inicialização da população. Por exemplo, se é sabido que a solução final (assumindo codificação binária) vai apresentar mais 0's do 1's, então esta informação pode ser utilizada, mesmo que não se saiba exatamente a proporção. Já em problemas com restrição, deve-se tomar cuidado para não gerar indivíduos inválidos na etapa de inicialização.

3.3.3 Operadores Genéticos

Os operadores genéticos mais freqüentemente utilizados em algoritmos genéticos são o *crossover* e a mutação. Nesta seção apresentamos os principais aspectos relacionados a estes operadores.

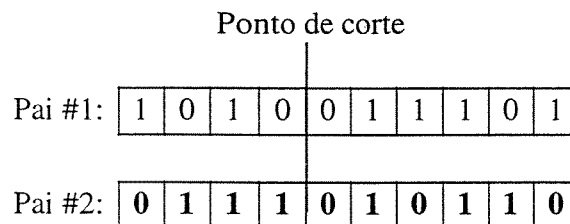
O Operador de *Crossover*

O operador de *crossover* ou recombinação cria novos indivíduos através da combinação de dois ou mais indivíduos. A idéia intuitiva por trás do operador de *crossover* é

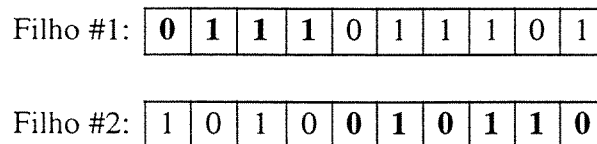
a troca de informação entre diferentes soluções candidatas. No algoritmo genético clássico é atribuída uma probabilidade de *crossover* fixa aos indivíduos da população.

O operador de *crossover* mais comumente empregado é o *crossover* de um ponto. Para a aplicação deste operador, são selecionados dois indivíduos (pais) e a partir de seus cromossomos são gerados dois novos indivíduos (filhos). Para gerar os filhos, seleciona-se um ponto de corte aleatoriamente nos cromossomos dos pais, e os segmentos de cromossomo criados a partir do ponto de corte são trocados. Ilustraremos este operador com um exemplo.

Exemplo 3.1 Considere dois indivíduos selecionados como pais a partir da população inicial de um algoritmo genético e suponhamos que o ponto de corte escolhido (aleatoriamente) encontra-se entre as posições 4 e 5 dos cromossomos dos pais:



Após o *crossover*, teremos os seguintes indivíduos filho:



Muitos outros tipos de *crossover* têm sido propostos na literatura. Uma extensão simples do *crossover* de um ponto é o *crossover* de dois pontos, onde dois pontos de corte são escolhidos e material genético são trocados entre eles. Ilustremos este operador com um exemplo:

Exemplo 3.2 Considere os mesmos indivíduos pais do Exemplo 3.1. Suponha que os pontos de *crossover* escolhidos estão localizados entre as posições 3 e 4 e entre as posições 7 e 8. Então os novos indivíduos produzidos serão:

Filho #1:

1	0	1	1	0	1	0	1	0	1
---	---	---	---	---	---	---	---	---	---

Filho #2:

0	1	1	0	0	1	1	1	1	0
---	---	---	---	---	---	---	---	---	---

□

Outro tipo de *crossover* muito comum é o *crossover uniforme* (SYSWERDA, 1989): para cada bit no primeiro filho é decidido (com alguma probabilidade fixa p) qual pai vai contribuir com seu valor para aquela posição. Consideremos mais um exemplo.

Exemplo 3.3 Consideremos os mesmos indivíduos-pai dos exemplos anteriores e seja $p = 0,5$. Poderíamos então obter os seguintes indivíduos com *crossover* uniforme:

Pai #1: <table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td></tr></table>	1	0	1	0	0	1	1	1	0	1	→	Filho #1: <table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td></tr></table>	1	1	1	1	0	1	1	1	1	1
1	0	1	0	0	1	1	1	0	1													
1	1	1	1	0	1	1	1	1	1													
Pai #2: <table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td><td>0</td></tr></table>	0	1	1	1	0	1	0	1	1	0		Filho #2: <table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td></tr></table>	0	0	1	0	0	1	0	1	0	0
0	1	1	1	0	1	0	1	1	0													
0	0	1	0	0	1	0	1	0	0													

□

Como o *crossover* uniforme troca bits ao invés de segmentos de bits (que aqui fazem o papel dos genes), ele pode combinar características independentemente da sua posição relativa no cromossomo.

ESHELMAN *et al.* (1989) relata diversos experimentos com vários operadores de *crossover*. Os resultados indicam que o operador com pior desempenho é o *crossover* de um ponto; entretanto, não há nenhum operador de *crossover* que claramente apresente um desempenho superior aos demais. Uma conclusão a que se pode chegar a partir destes resultados é que cada operador de *crossover* é particularmente eficiente para uma determinada classe de problemas e extremamente ineficiente para outras.

Os operadores de *crossover* descritos até aqui também podem ser utilizados em cromossomos com codificação em ponto flutuante. Entretanto existem operadores de *crossover* especialmente desenvolvidos para uso com codificação em ponto flutuante. Um exemplo é o chamado *crossover aritmético* (MICHALEWICZ, 1996). Este operador é definido como uma combinação linear de dois vetores (cromossomos): sejam $\mathbf{x}_1$ e $\mathbf{x}_2$ dois indivíduos selecionados para *crossover*, então os dois filhos resultantes serão $\mathbf{x}'_1 = a\mathbf{x}_1 + (1-a)\mathbf{x}_2$ e

$\mathbf{x}'_2 = (1 - a)\mathbf{x}_1 + a\mathbf{x}_2$ onde a é um número aleatório pertencente ao intervalo $[0, 1]$. Este operador é particularmente apropriado para problemas de otimização numérica com restrições, onde a região factível é convexa. Isto porque, se $\mathbf{x}_1$ e $\mathbf{x}_2$ pertencem à região factível, combinações convexas de $\mathbf{x}_1$ e $\mathbf{x}_2$ serão também factíveis. Assim, garante-se que o *crossover* não gera indivíduos inválidos para o problema em questão. Outros exemplos de *crossover* especialmente desenvolvidos para utilização em problemas de otimização numérica restritos e codificação em ponto flutuante são o *crossover geométrico* e o *crossover esférico*, descritos em MICHALEWICZ & SCHOENAUER (1996).

Um aspecto importante em um algoritmo genético diz respeito a como escolher os indivíduos que serão submetidos a *crossover*. Aqui também temos diversas alternativas possíveis, sendo que entre as mais comuns podemos citar:

- *Crossover* entre indivíduos aleatórios: são escolhidos indivíduos da população atual aleatoriamente ou por meio de *roulette wheel* (veja Seção 3.3.4) e efetua-se o *crossover*.
- *Crossover* entre indivíduo aleatório e melhor indivíduo: é escolhido um indivíduo da população atual aleatoriamente ou por meio de *roulette wheel*, sendo o outro indivíduo o melhor indivíduo da população, e efetua-se o *crossover*.

O Operador de Mutação

O operador de mutação modifica aleatoriamente um ou mais genes de um cromossomo. A probabilidade de ocorrência de mutação em um gene é denominada *taxa de mutação*. Usualmente, são atribuídos valores pequenos para a taxa de mutação. A ideia intuitiva por trás do operador de mutação é criar uma variabilidade extra na população, mas sem destruir o progresso já obtido com a busca.

Considerando codificação binária, o operador de mutação padrão simplesmente troca o valor de um gene em um cromossomo (HOLLAND, 1992). Assim, se um gene selecionado para mutação tem valor 1, o seu valor passará a ser 0 após a aplicação da mutação, e vice-versa.

No caso de problemas com codificação em ponto flutuante, o operador de mutação mais popular é a *mutação gaussiana* (MICHALEWICZ & SCHOENAUER, 1996), que modifica todos os componentes de um cromossomo $\mathbf{x} = [x_1 \dots x_n]$ na forma:

$$\mathbf{x}' = \mathbf{x} + N(0, \sigma),$$

onde $N(0, \sigma)$ é um vetor de variáveis aleatórias gaussianas independentes, com média zero e desvio padrão σ .

Um operador importante para problemas em que os indivíduos empregam codificação em ponto flutuante é a *mutação uniforme* (MICHALEWICZ, 1996). Este operador seleciona aleatoriamente um componente $k \in \{1, 2, \dots, n\}$ do cromossomo $\mathbf{x} = [x_1 \dots x_k \dots x_n]$ e gera um indivíduo $\mathbf{x}' = [x_1 \dots x'_k \dots x_n]$, onde x'_k é um número aleatório (com distribuição de probabilidade uniforme) amostrado no intervalo $[LB, UB]$ e LB e UB são, respectivamente, os limites inferior e superior da variável x_k .

Outro operador de mutação, especialmente desenvolvido para problemas de otimização com restrição e codificação em ponto flutuante, é a chamada *mutação não-uniforme* (MICHALEWICZ, 1996; MICHALEWICZ & SCHÖENAUER, 1996). A mutação não-uniforme é um operador dinâmico, destinado a melhorar a sintonia fina de um elemento simples. Podemos defini-lo da seguinte forma: seja $\mathbf{x}' = [x_1 \dots x_n]$ um cromossomo e suponha que o elemento x_k foi selecionado para mutação; o cromossomo resultante será $\mathbf{x}^{t+1} = [x_1 \dots x'_k \dots x_n]$, onde

$$x'_k = \begin{cases} x_k + \Delta(t, UB - x_k), & \text{com 50\% de probabilidade} \\ x_k - \Delta(t, x_k - LB), & \text{com 50\% de probabilidade} \end{cases}$$

onde LB e UB são os limites inferiores e superiores da variável x_k . A função $\Delta(t, y)$ retorna um valor no intervalo $[0, y]$ tal que a probabilidade de $\Delta(t, y)$ ser próximo de zero aumenta à medida que t aumenta. Esta propriedade faz com que este operador inicialmente explore o espaço globalmente (quando t é pequeno) e localmente em gerações avançadas (quando t é grande); MICHALEWICZ (1996) propõe a seguinte função

$$\Delta(t, y) = y \cdot \left(1 - r^{(1-t/T)^b}\right),$$

onde r é um número aleatório no intervalo $[0, 1]$, T é o número máximo de gerações e b é um parâmetro que determina o grau de dependência do número de iterações (valor proposto por MICHALEWICZ (1996): $b = 5$). Este operador foi usado com sucesso por DE CASTRO *et al.* (1998) na evolução de condições iniciais para o treinamento de redes neurais artificiais (veja Seção 2.4.5).

Outros exemplos de operadores de mutação para problemas de otimização numérica e com codificação em ponto flutuante podem ser encontrados em MICHALEWICZ & SCHOENAUER (1996).

3.3.4 Seleção de Indivíduos para a Próxima Geração

O algoritmo genético clássico utiliza um esquema de seleção de indivíduos para a próxima geração chamado *roulette wheel* (MICHALEWICZ, 1996). O *roulette wheel* atribui a cada indivíduo de uma população uma probabilidade de passar para a próxima geração proporcional ao seu *fitness* medido, em relação à somatória do *fitness* de todos os indivíduos da população. Assim, quanto maior o *fitness* de um indivíduo, maior a probabilidade dele passar para a próxima geração. Para um exemplo, veja a Figura 3.3.

Observe que a seleção de indivíduos por *roulette wheel* pode fazer com que o melhor indivíduo da população seja perdido, ou seja, não passe para a próxima geração. Uma alternativa é escolher como solução o melhor indivíduo encontrado em todas as gerações do algoritmo. Outra opção é simplesmente manter sempre o melhor indivíduo da geração atual na geração seguinte, estratégia essa conhecida como *seleção elitista* (FOGEL, 1994; MICHALEWICZ, 1996).

Outro exemplo de mecanismo de seleção é a *seleção baseada em rank* (BÄCK *et al.*, 1997). Esta estratégia utiliza as posições dos indivíduos quando ordenados de acordo com o *fitness* para determinar a probabilidade de seleção. Podem ser usados mapeamentos lineares ou não-lineares para determinar a probabilidade de seleção. Para um

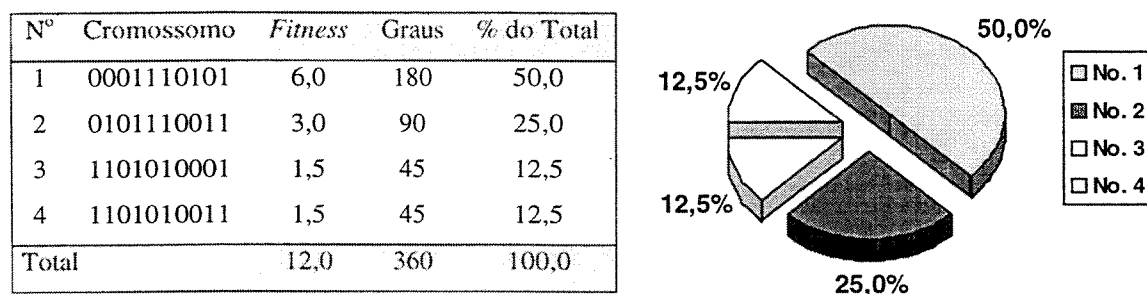


Figura 3.3 Exemplo de aplicação do *roulette wheel*: cada indivíduo em uma determinada geração recebe uma probabilidade de passar à próxima geração proporcional ao seu *fitness*, medido em relação ao *fitness* total da população.

exemplo de mapeamento não-linear, veja (MICHALEWICZ, 1996). Uma variação deste mecanismo é simplesmente passar os N melhores indivíduos para a próxima geração.

A seguir, citamos alguns outros possíveis mecanismos de seleção:

- Seleção por diversidade: são selecionados os indivíduos mais diversos da população.
- Seleção bi-classista: são selecionados os $P\%$ melhores indivíduos e os $(100 - P)\%$ piores indivíduos.
- Seleção aleatória: são selecionados aleatoriamente N indivíduos da população. Podemos subdividir este mecanismo de seleção em:
 - Salvacionista: seleciona-se o melhor indivíduo e os outros aleatoriamente.
 - Não-salvacionista: seleciona-se aleatoriamente todos os indivíduos.

3.4 Teoria dos Esquemas

A teoria dos esquemas foi proposta por HOLLAND (1992) para tentar explicar por que os algoritmos genéticos funcionam. Nesta seção, apresentaremos os principais resultados da teoria dos esquemas: o teorema de crescimento dos esquemas e a hipótese dos blocos construtivos.

Um *esquema* é uma representação capaz de descrever diversos cromossomos simultaneamente. Um esquema é construído inserindo um caractere *don't care* (*) no alfabeto dos genes, indicando que aquele gene representa qualquer alelo. Por exemplo, o esquema $[1 * 0 1 0 0 1]$ representa os cromossomos $[1 0 0 1 0 0 1]$ e $[1 1 0 1 0 0 1]$. O esquema $[1 * 0 * 1 1 0]$ representa quatro cromossomos: $[1 0 0 0 1 1 0]$, $[1 0 0 1 1 1 0]$, $[1 1 0 0 1 1 0]$ e $[1 1 0 1 1 1 0]$. Obviamente, o esquema $[1 1 1 0 0 1 0]$ representa apenas um cromossomo, enquanto que o esquema $[* * * * * *]$ representa todos os cromossomos de comprimento 7. Observe que cada esquema representa 2^r cromossomos, onde r é o número de caracteres *don't care* “*” presentes no esquema. Por outro lado, cada cromossomo de comprimento m é representado por 2^m esquemas. Por exemplo, considere o cromossomo $[0 1 0 0 1 0 0]$. Este cromossomo é representado pelos seguintes 2^7 esquemas:

$[0 1 0 0 1 0 0]$

$[* 1 0 0 1 0 0]$

$[0 * 0 0 1 0 0]$

$$\begin{array}{c}
\vdots \\
[0\ 1\ 0\ 0\ 1\ 0\ *] \\
[*\ * \ 0\ 0\ 1\ 0\ 0] \\
[*\ 1\ * \ 0\ 1\ 0\ 0] \\
\vdots \\
[0\ 1\ 0\ 0\ 1\ * \ *] \\
[*\ * \ * \ 0\ 1\ 0\ 0] \\
\vdots \\
[*\ * \ * \ * \ * \ * \ *]
\end{array}$$

Considerando cromossomos de comprimento m , há um total de 3^m possíveis esquemas. Numa população de tamanho n , entre 2^m e $n \cdot 2^n$ diferentes esquemas podem ser representados.

A *ordem* de um esquema S , $o(S)$, é definida como o número de 0's e 1's presentes no esquema, isto é, o número de *posições fixas* (caracteres diferentes de *don't care*) presentes no esquema. A ordem de um esquema define sua especificidade, de modo que quanto maior a ordem, mais específico é o esquema.

O *comprimento definitório* de um esquema S , denotado por $\delta(S)$, é a maior distância entre posições fixas de um cromossomo. O comprimento definitório define o nível de compactação da informação contida no esquema.

O *fitness* de um esquema S na geração t , $\text{eval}(S, t)$, é definido como a média dos *fitness* de todos os cromossomos na população representados pelo esquema S . Assuma que há p cromossomos $\{\mathbf{x}_{i_1}^t, \dots, \mathbf{x}_{i_p}^t\}$ representados pelo esquema S_i na geração t . Então

$$\text{eval}(S_i, t) = \frac{1}{p} \sum_{j=1}^p \text{eval}(\mathbf{x}_{i_j}^t),$$

onde $\text{eval}(\mathbf{x}_{i_j}^t)$ é o *fitness* do indivíduo $\mathbf{x}_{i_j}^t$.

Seja tam_pop o tamanho da população. O *fitness* médio da população na geração t , $\bar{F}(t)$ é dado por

$$\bar{F}(t) = \frac{1}{tam_pop} \sum_{i=1}^{tam_pop} eval(\mathbf{x}_i').$$

Sejam p_c e p_m as probabilidades de *crossover* e mutação, respectivamente, e m o comprimento dos cromossomos. Seja $\xi(S_i, t)$ o número de cromossomos representados pelo esquema S_i na geração t . Pode-se mostrar que (MICHALEWICZ, 1996):

$$\xi(S_i, t+1) \geq \frac{\xi(S_i, t) eval(S_i, t)}{\bar{F}(t)} \left[1 - p_c \frac{\delta(S_i)}{m-1} - o(S_i) p_m \right], \quad (3.1)$$

A Equação (3.1) é conhecida como *equação de crescimento reprodutivo do esquema*. Esta equação é deduzida assumindo que a função de *fitness* $f(\cdot)$ assume apenas valores positivos. Se a função a ser otimizada assume valores negativos, um mapeamento entre as funções de otimização e de *fitness* é necessário.

A equação de crescimento (3.1) mostra que a seleção aumenta a amostragem de esquemas cujo *fitness* está acima da média da população, e este aumento é exponencial (MICHALEWICZ, 1996). A seleção, por si só, não introduz nenhum novo esquema (não representado na geração inicial em $t=0$). Esta é a razão da introdução do operador de *crossover*: possibilitar a troca de informação estruturada, ainda que aleatória. Além disso, o operador de mutação introduz uma variabilidade maior na população. O efeito (destrutivo) combinado destes operadores não é significativo se o esquema é curto e de ordem baixa. O resultado final da equação de crescimento (3.1) pode ser formulado como segue:

Teorema 3.1 (Teorema dos Esquemas) Esquemas com comprimento definitório curto, de ordem baixa, e com *fitness* acima da média, têm um aumento exponencial de sua participação em gerações consecutivas de um algoritmo genético.

Prova: Veja HOLLAND (1992).

Uma consequência imediata do Teorema 3.1 é que os algoritmos genéticos tendem a explorar o espaço por meio de esquemas curtos e de baixa ordem que, subsequente, são usados para troca de informação durante o *crossover*.

Hipótese dos Blocos Construtivos Um algoritmo genético busca desempenho quase-ótimo através da justaposição de esquemas curtos, de baixa ordem e alto desempenho, chamados de *blocos construtivos*.

Em uma população de tamanho tam_pop , indivíduos de comprimento m processam pelo menos 2^m e no máximo 2^{tam_pop} esquemas. Alguns deles são processados de forma útil: são amostrados a uma taxa crescente exponencial (desejável); e outros são quebrados por meio de *crossover* e mutação.

HOLLAND (1992) mostrou que, em uma população de tamanho tam_pop , pelo menos tam_pop^3 são processados de forma útil. Esta propriedade foi denominada *paralelismo implícito*, pois é obtida sem nenhuma exigência extra de memória e processamento. Entretanto, BERTONI & DORIGO (1993) mostraram que a estimativa tam_pop^3 é válida apenas para o caso particular em que tam_pop é proporcional a 2^l , onde $l = \frac{1}{2}m\varepsilon$ e ε é a probabilidade de um esquema ser rompido por *crossover*.

Note entretanto que, em alguns problemas, alguns blocos construtivos (esquemas curtos, de ordem baixa) podem direcionar erroneamente o algoritmo, levando-o a convergir a pontos sub-ótimos. Este fenômeno é conhecido como *decepção*. Assim, a hipótese dos blocos construtivos não fornece uma explicação definitiva do porquê os algoritmos genéticos funcionam. Ela é apenas uma indicação do porquê os algoritmos genéticos funcionam para uma certa classe de problemas.

Algumas alternativas foram propostas para combater o problema da decepção (GOLDBERG, 1989b; MICHALEWICZ, 1996). A primeira assume que há algum conhecimento a priori da função objetivo para que seja possível codificá-la de forma apropriada (que forme blocos construtivos “coesos”). A segunda é a utilização de um operador de *inversão*: selecionam-se 2 pontos em um cromossomo e inverte-se a ordem dos bits entre os pontos selecionados (alterando-se a codificação em conjunto com a operação). A terceira opção é utilizar algoritmos genéticos *messy*, que diferem do algoritmo genético clássico de várias maneiras: codificação, operadores, presença de cromossomos de tamanho distinto e fases evolutivas.

3.5 Conclusões

Neste capítulo, apresentamos os principais conceitos relacionados à computação evolutiva, mais especificamente aos algoritmos genéticos. Foram introduzidos conceitos relativos à evolução natural, computação evolutiva, algoritmos genéticos e à teoria dos esquemas.

Apesar da área de computação evolutiva ter experimentado um crescimento vertiginoso nos últimos anos, ainda há muitas questões em aberto. MICHALEWICZ (1996) aponta direções na área de computação evolutiva nas quais podemos esperar grande atividade e resultados significativos, algumas das quais apresentamos a seguir:

- **Fundamentação teórica.** Apesar de haver alguns resultados teóricos importantes, é quase que um consenso que a atual teoria de algoritmos evolutivos não é vista como uma base útil para praticantes de computação evolutiva (EIBEN *et al.*, 1999). Além da complexidade inerente aos processos evolutivos, muitas modificações incorporadas ao algoritmo genético clássico (como por exemplo o uso de codificação em ponto flutuante ao invés de codificação binária) impedem a aplicação dos conceitos teóricos já obtidos para o caso clássico. Esta é uma das tarefas mais desafiadoras apresentada aos pesquisadores de computação evolutiva.
- **Sistemas Auto-Adaptativos.** Os algoritmos evolutivos clássicos exigem definição por parte do usuário de diversos parâmetros, como tamanho da população e probabilidades de *crossover* e mutação, e estes parâmetros permanecem fixos durante toda a execução do algoritmo ou são alterados arbitrariamente em pontos específicos do processo evolutivo. Estes parâmetros em grande parte determinam se o algoritmo será capaz de encontrar uma solução de qualidade e também a eficiência na busca desta solução. Entretanto, encontrar valores apropriados para estes parâmetros é uma tarefa computacionalmente muito custosa, pois não há uma metodologia eficiente que ajude na definição dos mesmos. Assim, muitas abordagens têm sido propostas no sentido de ajustar estes parâmetros durante a execução do algoritmo. Em EIBEN *et al.* (1999) podemos encontrar diversas técnicas para o controle dos parâmetros durante a execução do algoritmo. Esta parece ser uma das áreas de pesquisa mais promissoras em computação evolutiva.

- **Sistemas Co-Evolutivos.** Em sistemas co-evolutivos, mais de um processo evolutivo ocorre simultaneamente. Usualmente considera-se mais de uma população (por exemplo, uma população de “presas” e outra de “predadores”) que interagem entre si. Em sistemas desse tipo, a função de *fitness* de uma população pode depender do estado da outra população. Sistemas co-evolutivos podem ser abordagens interessantes para problemas de larga escala, de modo que um problema complexo é decomposto em sub-problemas menores. Haveria então um processo evolutivo para cada sub-problema, e os processos estariam todos inter-relacionados.
- **Algoritmos de Implementação Paralela.** Algoritmos evolutivos são adequados para implementação paralela. Como observado por GOLDBERG (1989a), não deixa de ser irônico que, num mundo onde algoritmos seriais são paralelizados através de inúmeros truques e contorcionismos, algoritmos genéticos (algoritmos inerentemente paralelos) sejam implementados serialmente através de truques igualmente anti-naturais. Entretanto, não há atualmente uma metodologia padrão para paralelização de algoritmos evolutivos. Para uma análise das principais abordagens já propostas para a paralelização de algoritmos genéticos, veja CANTÚ-PAZ (1998).

Apesar de todas as questões em aberto, a área de computação evolutiva caminha rapidamente para o amadurecimento, e tudo leva a crer que os algoritmos evolutivos se tornarão parte permanente do conjunto de ferramentas de engenharia (GOLDBERG, 1998).

Capítulo 4

Redes Neurais Construtivas

Como vimos no Capítulo 2, o teorema da aproximação universal mostra que redes neurais artificiais possuem a capacidade de aproximar qualquer mapeamento contínuo com precisão arbitrária. Entretanto, este teorema não fornece nenhuma indicação a respeito da arquitetura de rede neural necessária para atingir a precisão desejada, sendo apenas um resultado existencial. As redes neurais construtivas são algoritmos que, a partir de uma arquitetura mínima, procuram encontrar de forma automática uma arquitetura de rede neural ótima para um determinado problema. Neste capítulo, apresentaremos brevemente três algoritmos construtivos para o projeto de redes neurais: o *cascade-correlation*, o aprendizado por busca de projeção e o A*. Apresentaremos também uma comparação de desempenho dos três algoritmos.

4.1 Métodos Construtivos

Os métodos construtivos para projeto de redes neurais são algoritmos que começam com uma arquitetura mínima de rede e adicionam neurônios até que uma solução adequada seja encontrada.

Outra forma muito estudada de determinação automática de arquiteturas de redes neurais são os chamados métodos de poda (REED, 1993), cuja idéia é iniciar com uma arquitetura de dimensão elevada e ir retirando unidades ou conexões até que se obtenha uma arquitetura adequada.

Entretanto, os métodos de poda apresentam invariavelmente os seguintes problemas (VON ZUBEN, 1996):

- Não existe um método prático para determinação direta de uma arquitetura inicial que contenha garantidamente tanta estrutura quanto a necessária para realizar a tarefa de

aproximação. Com isso, para aumentar a probabilidade de se escolher uma arquitetura com tal característica, geralmente adotam-se arquiteturas iniciais fortemente sobredimensionadas.

- Já que a maior parte do processo de aproximação é realizado considerando-se redes neurais sobredimensionadas, a demanda por recursos computacionais é grande e parte dos recursos computacionais utilizados acaba sendo desperdiçada toda vez que a poda elimina estruturas que já passaram por alguma fase de processamento.
- Como geralmente inúmeras redes neurais de diferentes dimensões são capazes de representar soluções aceitáveis para o problema, a aplicação de métodos de poda não favorece a escolha da solução de menor dimensão, ou seja, aquela com menor número de componentes e operadores.
- Para que métodos de poda sejam computacionalmente factíveis, eles devem estimar o efeito da eliminação de cada recurso individualmente, mas devem eliminar múltiplos recursos simultaneamente. Com isso, não é possível obter uma estimativa confiável do efeito que cada operação de poda possa vir a causar junto ao erro de aproximação.

Por operarem no sentido oposto ao dos métodos de poda, os métodos construtivos podem evitar a ocorrência de problemas como os mencionados acima. No entanto, pelo fato de não ser possível garantir que toda inclusão de estrutura por parte do algoritmo construtivo venha contribuir para a solução do problema, métodos de poda representam um procedimento complementar importante no sentido de promover a eliminação de estruturas desnecessariamente incluídas.

Portanto, pode-se concluir que um método híbrido contendo etapas construtivas seguidas de etapas de poda é o mais adequado. Entretanto, em razão de procedimentos de poda só entrarem em operação quando o método construtivo falha na definição de estruturas adicionais, o método híbrido é predominantemente construtivo.

4.1.1 Estratégia de Busca no Espaço de Estados

O problema de construção de uma rede neural pode ser visto como uma busca no espaço de *estados* da rede neural (KWOK & YEUNG, 1997). Os componentes principais em qualquer problema de busca são o espaço de estados, o estado inicial, a estratégia de busca e o critério de terminação da busca.

Espaço de Estados

Dado um espaço de funções G , para implementar um elemento deste espaço usando uma rede neural artificial, é necessário especificar:

- l : a quantidade de unidades intermediárias da rede;
- C : o grafo de conectividade, especificando como as entradas, neurônios escondidos e as saídas estão interconectadas;
- Φ : a forma das funções de ativação dos neurônios escondidos;
- W : os parâmetros de toda a rede, incluindo os pesos e os outros parâmetros associados a Φ .

Assim, uma quádrupla (l, C, Φ, W) pode ser unicamente utilizada para especificar um elemento $\hat{g} \in G$. Note que os elementos da quádrupla não são independentes: o grafo de conectividade pode ser dado por $C(V, E)$, onde V é o conjunto de neurônios e E é o conjunto de conexões (pesos). O *espaço de estados* P corresponde à coleção de funções que podem ser implementadas por uma dada classe de redes neurais. Cada estado $p \in P$ pode ser representado por $p = C(V, E)$.

Critério de Terminação da Busca

Crítérios muito utilizados para a interrupção do procedimento de busca são: aguardar até que o erro de treinamento atinja um determinado limiar, ou até que o erro não sofra decrescimento significativo após adicionado um determinado número de neurônios escondidos. As vantagens destes critérios são a simplicidade e a facilidade de observação do erro de treinamento. Por outro lado, eles requerem a definição de vários parâmetros por parte do usuário.

Estratégia de Busca

Esta é a parte mais crítica dos métodos construtivos (KWOK & YEUNG, 1997). A estratégia de busca determina como mover-se de um estado para outro no espaço de estados, até que a busca seja finalizada. De forma equivalente, determina como o grafo de conectividade C evolui durante a busca. Seja $p_1 = C(V_1, E_1)$ o *estado atual* e $p_2 = C(V_2, E_2)$ o *próximo estado*. Nos métodos construtivos, as seguintes propriedades são quase sempre satisfeitas:

I. $V_1 \subseteq V_2$;

II. $E_1 \subset E_2$.

Ou seja, neurônios e conexões existentes no estado atual devem ser preservados no próximo estado, e deve haver algumas conexões adicionais no estado seguinte. Em princípio, é possível que haja mudança apenas em E (com $V_1 = V_2$) para alguns estados sucessivos. Isto pode levar a várias arquiteturas distintas com desempenho similares, e portanto este esquema não é muito utilizado na prática. Portanto, geralmente tem-se na prática $V_1 \subset V_2$, com $|V_2| = |V_1| + 1$, onde $|\cdot|$ denota a cardinalidade de um conjunto.

Transição de Estados

O mapeamento de transição de estados $\Delta: S \rightarrow S$, é um componente chave para os algoritmos construtivos. Os algoritmos construtivos, guiados pelas propriedades I e II acima, devem especificar as transições de estado possíveis, definindo um mapeamento adequado.

Δ pode ser mono ou multivariável. No caso mais simples em que ele é monovariável, existe apenas um estado seguinte a ser explorado. A passagem de estados é representada por uma cadeia simples de estados (veja Figura 4.1(a)). Um possível problema é a perda de flexibilidade.

Em um mapeamento Δ multivariável, geralmente existem vários próximos estados possíveis para um determinado estado atual (veja Figura 4.1(b)). Cada um deles é chamado de candidato; sendo que candidatos diferentes devem possuir diferentes grafos de conectividade, gerando arquiteturas distintas. A existência de múltiplos candidatos permite assim analisar

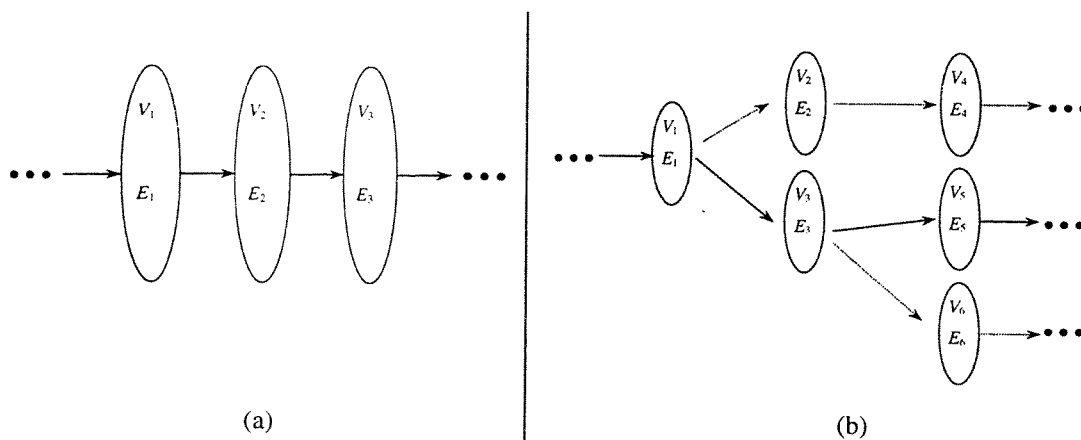


Figura 4.1 Mapeamentos de transição de estados: (a) monovariável e (b) multivariável.

várias arquiteturas de rede distintas, de forma que a escolhida seja a mais adaptada ao problema.

4.2 O Algoritmo *Cascade-Correlation*

O algoritmo *cascade-correlation* (CASCOR), proposto por FAHLMAN & LEBIERE (1990), tem capacidade de construir redes com múltiplas camadas intermediárias. O CASCOR combina duas idéias-chave:

- a *arquitetura em cascata*, onde neurônios escondidos são incorporados à rede um de cada vez e não alteram os neurônios já incorporados;
- o algoritmo de treinamento, que cria e incorpora novos neurônios escondidos. Para cada novo neurônio escondido, tenta-se maximizar a magnitude da *correlação* entre a saída da nova unidade escondida e o erro residual, o qual deve ser minimizado.

O CASCOR começa sem nenhum neurônio escondido. Todas as entradas e a polarização se conectam às unidades de saída através de pesos ajustáveis, como ilustrado na Figura 4.2(a).

Os neurônios escondidos são adicionados à rede um por um. Cada novo neurônio escondido se conecta a cada uma das entradas originais da rede e também à saída de cada um dos neurônios escondidos pré-existentes (Figuras 4.2(b) e (c)). Apenas os pesos da entrada do novo neurônio escondido são otimizados no momento que o mesmo é adicionado à rede; em seguida, estes pesos são mantidos fixos e apenas as conexões de saída de todos os neurônios são treinadas repetidamente. Note que cada novo neurônio adicionado à rede corresponde a uma nova camada da rede.

O algoritmo de treinamento começa sem nenhum neurônio escondido. Nesta etapa pode ser usado qualquer algoritmo de treinamento para ajustar os pesos que conectam as entradas às saídas. Na proposta original de FAHLMAN & LEBIERE (1990) foi utilizado um algoritmo denominado *quickprop* (FAHLMAN, 1988).

Quando o critério de parada do algoritmo de treinamento é satisfeito, é avaliado o desempenho (erro) da rede junto ao conjunto de treinamento. Se o desempenho da rede é satisfatório, o CASCOR é terminado; se não, haverá algum erro residual que desejamos reduzir. Tenta-se reduzir o erro residual através da adição de um novo neurônio escondido à

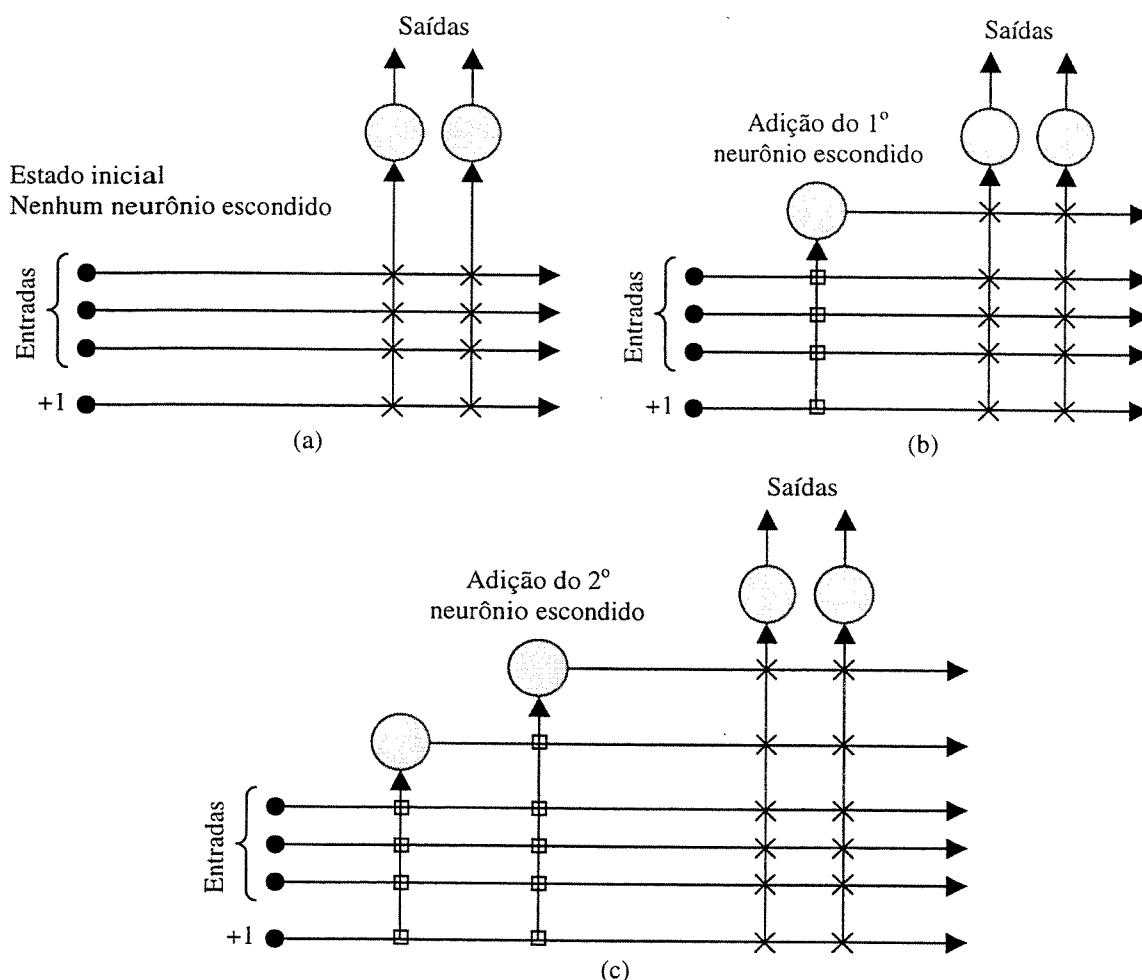


Figura 4.2 A arquitetura em cascata gerada pelo CASCOR: (a) estado inicial, (b) após a adição do primeiro neurônio escondido, (c) após a adição do segundo neurônio escondido. Conexões representadas por quadrados correspondem a pesos congelados imediatamente após serem ajustados durante uma única sequência de épocas, conexões representadas por cruzes são pesos ajustados repetidamente após a introdução de novos neurônios.

rede, usando o algoritmo de criação de neurônios descrito a seguir. O novo neurônio é incorporado à rede, seus pesos de entrada são ajustados até a convergência e depois congelados, e todos os pesos de saída são treinados novamente usando o *quickprop*. Este ciclo é repetido até que o erro seja suficientemente pequeno.

Para criar um novo neurônio escondido, começamos com uma *unidade candidata* que se conecta a todas as entradas da rede e a todos os neurônios escondidos pré-existent através de pesos ajustáveis. A saída desta unidade candidata ainda não é conectada à rede. Apresentamos algumas épocas do conjunto de treinamento para ajustar os pesos de entrada

desta unidade candidata. O objetivo deste treinamento é maximizar S , a soma sobre todas as unidades de saída o da magnitude da correlação entre V , a ativação da unidade candidata e E_o , o erro residual de saída observado na unidade o . Definimos S como

$$S = \sum_o \left| \sum_p (V_p - \bar{V})(E_{p,o} - \bar{E}_o) \right|, \quad (4.1)$$

onde o é a saída da rede na qual o erro é medido e p é o padrão de treinamento. $\bar{V}$ e $\bar{E}_o$ são os valores médios de V e E_o sobre todos os padrões de treinamento.

Para maximizar S , devemos computar $\partial S / \partial w_i$, a derivada parcial de S em relação aos pesos de entrada w_i da unidade candidata. De forma similar ao algoritmo de retropropagação, podemos expandir e derivar a Equação (4.1) para obter

$$\frac{\partial S}{\partial w_i} = \sum_{p,o} \sigma_o (E_{p,o} - \bar{E}_o) f'_p I_{i,p},$$

onde σ_o é o sinal da correlação entre a ativação do candidato e a saída o ; f'_p é a derivada para o padrão p da função de ativação da unidade candidata em relação à soma de suas entradas; e $I_{i,p}$ é entrada que a unidade candidata recebe da unidade i para o padrão p ¹.

Após computar $\partial S / \partial w_i$ para cada conexão de entrada, podemos executar uma subida pelo gradiente para maximizar S . Observe que novamente estamos ajustando pesos de entrada de um único neurônio. Nesta etapa, FAHLMAN & LEBIERE (1990) novamente sugerem o uso do algoritmo *quickprop* para maximizar S . Após a convergência do algoritmo de maximização de S , os pesos de entrada são congelados e o ciclo continua como descrito acima. A Figura 4.3 apresenta a estrutura básica do algoritmo CASCOR.

Ao invés de uma única unidade candidata, é possível usar um conjunto de candidatos, onde cada unidade tem um conjunto de pesos iniciais diferentes. Todos os candidatos recebem os mesmos sinais de entrada e o mesmo erro residual para cada padrão de treinamento. Como os candidatos não interagem uns com os outros nem afetam a rede atual durante o treinamento, todos os candidatos podem ser treinados em paralelo; incorporamos então à rede

¹ Estritamente falando, a Equação (4.1) não é diferenciável nos pontos onde $S = 0$.

procedimento CASCOR

início

conecte os nós de entrada aos nós de saída;

ajuste os pesos das conexões entre entradas e saídas;

enquanto (não condição de parada) faça

início

adicione um neurônio escondido;

ajuste as conexões da entrada do novo neurônio, com o objetivo de maximizar S (Equação (4.1));

congele os pesos da entrada do novo neurônio;

conecte a saída do novo neurônio às saídas da rede;

ajuste os pesos das conexões entre entradas e saídas da rede e entre as saídas dos neurônios escondidos e as saídas da rede;

fim

fim

Figura 4.3 A estrutura básica do algoritmo CASCOR.

o candidato com a maior correlação. O uso de um conjunto de candidatos é benéfico em dois aspectos:

- um conjunto de candidatos reduz consideravelmente a probabilidade de existência de uma unidade com participação insignificante, ou seja, que um candidato que convergiu para um mínimo local ruim seja permanentemente conectado à rede;
- em um computador com processamento paralelo, pode-se acelerar o treinamento, pois várias regiões diferentes do espaço de pesos podem ser exploradas simultaneamente.

4.2.1 CASCOR com Funções de Ativação Distintas

FAHLMAN & LEBIERE (1990) já haviam sugerido a possibilidade de usar neurônios com diferentes funções de ativação no conjunto de unidades candidatas, embora não tenham implementado esta opção em suas simulações. ENSLEY & NELSON (1992) implementaram o CASCOR com a possibilidade de construção da rede neural usando candidatos com funções de ativação distintas. Assim, unidades candidatas com funções de ativação distintas podem competir simultaneamente. O CASCOR então seleciona o melhor candidato, permitindo definir qual função de ativação, a partir de um conjunto finito, é mais adequada considerando-se o estágio atual do problema. A Tabela 4.1 mostra as funções de ativação candidatas usadas na implementação do CASCOR desenvolvida neste trabalho.

Tabela 4.1 Funções de ativação candidatas utilizadas pelo CASCOR.

Ativação	Expressão	Intervalo
Seno	$f(x) = \text{sen}(x)$	$[-1,0; 1,0]$
Cosseno	$f(x) = \cos(x)$	$[-1,0; 1,0]$
Gaussiana	$f(x) = e^{-(x^2/2)}$	$[0,0; 1,0]$
Inverso da gaussiana	$f(x) = -e^{-(x^2/2)}$	$[-1,0; 0,0]$
Tangente hiperbólica	$f(x) = \text{tgh}(x)$	$(-1,0; 1,0)$

4.3 Aprendizado por Busca de Projeção (Baseado em VON ZUBEN (1996))

O Aprendizado por Busca de Projeção (*Projection Pursuit Learning* – PPL), compreende uma classe de algoritmos inspirados em técnicas estatísticas baseadas em modelos de regressão por busca de projeção (*Projection Pursuit Regression* – PPR).

Modelos de aproximação de funções por busca de projeção assumem a forma

$$\hat{g}_n = \sum_{j=1}^n f_j(\mathbf{v}_j^T \mathbf{x}) \quad (4.2)$$

onde $\mathbf{x} \in \mathcal{R}^m$ é o vetor de variáveis de entrada, $\mathbf{v}_j \in \mathcal{R}^n$ é a direção de projeção ($j = 1, \dots, n$) e o produto escalar $\mathbf{v}_j^T \mathbf{x}$ pode ser tomado, a menos de um fator de escala, como uma projeção de $\mathbf{x}$ na direção $\mathbf{v}_j$. Observe que o j -ésimo termo $f_j(\cdot)$ do somatório é constante para valores de $\mathbf{x}$ em hiperplanos do $\mathcal{R}^m$ definidos na forma $\mathbf{v}_j^T \mathbf{x} = c$, para $c \in \mathcal{R}$ constante.

A utilização de modelos na forma da Equação (4.2) conduz a processos de aproximação por expansão ortogonal aditiva que apresentam inúmeras vantagens estatísticas e computacionais (FRIEDMAN & STUETZLE, 1981; VON ZUBEN, 1996). Nestes modelos, os termos da composição aditiva correspondem a funções escalares de expansão ortogonal a projeções unidirecionais, sendo denominado modelo de aproximação por busca de projeção.

A projeção consiste de operações lineares em que um mapeamento de uma determinada dimensão tem suprimidas algumas de suas estruturas de modo a tornar possível a

sua representação em espaços de menor dimensão. Neste caso, a estrutura projetada pode ser considerada como uma “sombra” da estrutura original, fazendo com que as projeções mais interessantes sejam aquelas que preservam parcelas representativas da estrutura original.

A busca destas direções de projeção envolve uma série de manipulações do conjunto de dados de aproximação disponíveis. Baseadas nos dados de entrada-saída, as direções de projeção devem enfatizar as relações, possivelmente não-lineares, existentes entre as variáveis do problema de aproximação.

Surge então o problema de determinação automática das direções de projeção. FRIEDMAN & TUKEY (1974) apresentam um método em que a direção de projeção corresponde à solução que maximiza um determinado índice numérico de projeção. A partir daí uma série de índices foram propostos na literatura, cada qual evidenciando um conjunto particular de características a serem atendidas pelos dados projetados.

Em virtude da inexistência de um índice de projeção que se aplique a todos os casos, e como geralmente não existe um conhecimento prévio das características presentes nos dados de aproximação, a utilização de um índice de desempenho na determinação da direção de projeção, ao invés de determiná-la arbitrariamente, permite apenas assegurar um aumento da probabilidade de se encontrar direções de projeção interessantes.

Uma vez definida uma direção de projeção, funções monovariáveis são então determinadas de tal forma que sua expansão em direções ortogonais à direção de projeção forneça a melhor aproximação possível com base nos dados disponíveis. No caso de projeções unidirecionais, a expansão ortogonal ocorre em hiperplanos do espaço de aproximação.

Após a definição de uma direção de projeção e da correspondente função de expansão ortogonal, uma transformação deve ser aplicada ao conjunto de dados para que a informação já representada seja removida, permitindo o reinício do processo a partir de uma nova direção de projeção. A busca sequencial de direções de projeção pode ser implementada na forma (FRIEDMAN *et al.*, 1984):

1. encontra-se uma direção de projeção ótima;
2. remove-se do conjunto de dados a estrutura resultante da projeção dos dados nesta direção;

3. reinicia-se o processo até que nenhuma outra projeção revele qualquer estrutura, ou seja, até que o modelo de aproximação concorde com os dados amostrados em todas as projeções.

Este procedimento iterativo e construtivo, em que cada novo sub-problema de aproximação deve representar apenas informações não representadas pelos sub-problemas de aproximação anteriores, produz funções de aproximação multivariável utilizando composição aditiva de funções monovariáveis na forma da Equação (4.2).

Como as funções $f_j(\cdot)$, $j = 1, \dots, n$, são constantes para valores de $\mathbf{x}$ em hiperplanos do $\mathcal{R}^m$, elas são denominadas funções de expansão ortogonal a uma determinada direção – *ridge functions* (DAHMEN & MICCHELLI, 1987). Tomando $m = 2$ e $f_j(\cdot)$ arbitrário, as Figuras 4.4(a) e (b) permitem verificar esta propriedade (VON ZUBEN, 1996). Observe que a expansão é ortogonal à direção de projeção $\mathbf{v} = [1 \ 0]^T$.

4.3.1 O Problema de Aproximação Resultante

O problema de aproximação por expansão ortogonal aditiva pode ser completamente descrito na forma (VON ZUBEN, 1996; FRIEDMAN & STUETZLE, 1981; HUBER, 1985):

- Seja Ω uma região compacta do $\mathcal{R}^m$ e seja $g: \Omega \subset \mathcal{R}^m \rightarrow \mathcal{R}$ a função a ser aproximada.
- O conjunto de dados de aproximação $\{(\mathbf{x}_i, s_i) \in \mathcal{R}^m \times \mathcal{R}\}_{i=1}^N$ é gerado considerando-se que os vetores de entrada $\mathbf{x}_i$ estão distribuídos em $\Omega \subset \mathcal{R}^m$ de acordo com uma função densidade de probabilidade fixa $d_P: \Omega \subset \mathcal{R}^m \rightarrow [0, 1]$ e que os vetores de saída s_i são

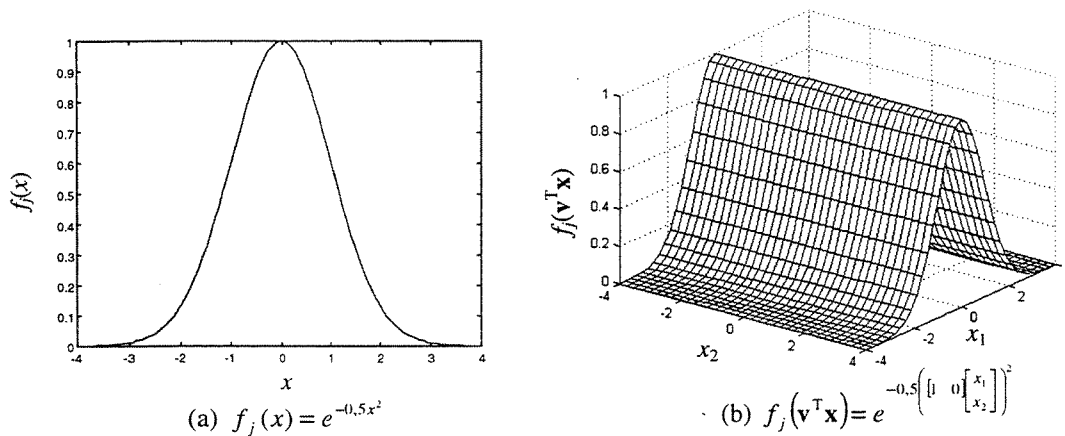


Figura 4.4 Função de expansão ortogonal em que $\mathbf{v} = [1 \ 0]^T$ e $\mathbf{x} = [x_1 \ x_2]^T$.

produzidos pelo mapeamento definido pela função g na forma:

$$s_l = g(\mathbf{x}_l) + \varepsilon_l, \quad l = 1, \dots, N,$$

onde $\varepsilon_l \in \mathfrak{R}$ é uma variável aleatória de média zero e variância fixa.

- A função g que associa a cada vetor de entrada $\mathbf{x} \in \Omega$ uma saída escalar $s \in \mathfrak{R}$ pode ser aproximada com base no conjunto de dados de aproximação $\{(\mathbf{x}_l, s_l) \in \mathfrak{R}^m \times \mathfrak{R}\}_{l=1}^N$ por uma composição aditiva de funções de expansão ortogonal na forma:

$$g(\mathbf{x}) \approx \hat{g}_n(\mathbf{x}) = \sum_{j=1}^n f_j(\mathbf{v}_j^T \mathbf{x}),$$

sendo que as funções de expansão ortogonal $f_j(\cdot)$, por serem constantes em hiperplanos do espaço de aproximação, são consideradas como uma generalização de funções lineares. Por motivações de ordem prática, é interessante, sempre que possível, tomar direções de projeção de norma unitária tal que $\mathbf{v}_j^T \mathbf{v}_j = 1, j = 1, \dots, n$.

- Assuma que os primeiros $n - 1$ termos já foram determinados, ou seja, os vetores $\mathbf{v}_j$ e as funções $f_j(\cdot), j = 1, \dots, n - 1$. Sejam:

$$d_l = s_l - \sum_{j=1}^{n-1} f_j(\mathbf{v}_j^T \mathbf{x}_l), \quad l = 1, \dots, N,$$

os resíduos de aproximação. Obtenha a direção de projeção $\mathbf{v}_n$ e a função $f_n(\cdot)$, soluções do seguinte problema variacional com restrição de suavidade:

$$\min_{\mathbf{v}_n, f_n} \frac{1}{N} \sum_{l=1}^N (d_l - f_n(\mathbf{v}_n^T \mathbf{x}_l))^2 + \lambda_n \phi(f_n), \quad (4.3)$$

onde λ_n é um número positivo denominado parâmetro de regularização e $\phi(f_n)$ é um funcional que assume valores tanto menores quanto mais suave for a função f_n . Note que o segundo termo da Equação (4.3) representa o grau de suavidade imposto ao problema. Para uma discussão mais aprofundada a respeito da determinação do parâmetro λ_n e da função ϕ veja VON ZUBEN (1996).

- Faça $n = n + 1$ e repita o processo a partir do cálculo dos novos resíduos até que o nível de aproximação desejado seja atingido.

4.3.2 Busca da Direção Inicial de Projeção (*Projection Pursuit*)

Um problema de aproximação multidimensional por composição aditiva de funções monovariáveis de expansão ortogonal pode ser formulado como uma composição de problemas mais simples na forma da Equação (4.3), onde devem ser definidas funções monovariáveis de forma a aproximar projeções de g em planos do espaço de aproximação.

Uma projeção linear do $\mathcal{R}^m$ para o $\mathcal{R}^k$ ($k \leq m$) pode ser definida como uma transformação linear realizada por qualquer matriz $\mathbf{V}$, de dimensão $m \times k$ e posto k , na forma:

$$\mathbf{z} = \mathbf{V}^T \mathbf{x}, \mathbf{x} \in \mathcal{R}^m \text{ e } \mathbf{z} \in \mathcal{R}^k.$$

A projeção é ortogonal se as colunas da matriz $\mathbf{V}$ forem ortogonais e de norma unitária.

Para uma única direção de projeção, a matriz $\mathbf{V}$ se reduz a um vetor $\mathbf{s} \in \mathcal{R}^{m \times 1}$. Neste caso, o método de aproximação por busca de projeção tem por objetivo encontrar a direção de projeção que resolve o seguinte problema:

$$\max_{\mathbf{v}} I_P(\mathbf{v}, \mathbf{X}, \mathbf{S}),$$

onde $I_P(\mathbf{v}, \mathbf{X}, \mathbf{S})$ é um índice de projeção, $\mathbf{X} \in \mathcal{R}^{m \times N}$ é a matriz de dados de entrada e $\mathbf{S} \in \mathcal{R}^{N \times r}$ é a matriz de dados de saída. Com $r = 1$, $\mathbf{X}$ e $\mathbf{S}$ são dados respectivamente por:

$$\mathbf{X} = [\mathbf{x}_1 \quad \mathbf{x}_2 \quad \cdots \quad \mathbf{x}_N] \text{ e } \mathbf{S} = [s_1 \quad s_2 \quad \cdots \quad s_N]^T.$$

Observe que as direções obtidas por índices de projeção podem ser utilizadas como condições iniciais para algoritmos iterativos de construção de modelos de aproximação por busca de projeção, a serem apresentados mais adiante. Portanto, os índices de projeção são geralmente aplicados na busca da direção *inicial* de projeção. $I_P(\mathbf{v}, \mathbf{X}, \mathbf{S})$ pode também ser usado como critério de parada, pois é capaz de identificar quando todas as estruturas presentes nos dados já foram identificadas.

Como já enfatizado anteriormente, a busca de projeções através de índices numéricos fornece uma direção interessante. Entretanto, é difícil evitar que diferentes conceitos do que venha a ser uma direção interessante se confundam na definição de um determinado índice. Em vista da dificuldade de modelagem de um único índice que satisfaça a todas as expectativas, recorre-se a diversos índices de projeção, cada qual fornecendo uma solução

específica. Em seguida, as soluções obtidas são comparadas e é selecionada aquela que conduz ao melhor resultado.

Os diversos índices de projeção já propostos na literatura podem ser classificados como segue:

- Índices de projeção baseados apenas nos dados de entrada: baseiam-se em conceitos heurísticos e consideram apenas os dados de entrada representados pela matriz $\mathbf{X} \in \Re^{m \times N}$, ou seja,

$$I_P(\mathbf{v}, \mathbf{X}, \mathbf{S}) = I_P(\mathbf{v}, \mathbf{X}).$$

- Índices de projeção completos: utilizam também a matriz de saída como informação necessária para obter a direção de projeção mais interessante, ou seja, são funções do tipo $I_P(\mathbf{v}, \mathbf{X}, \mathbf{S})$.
- Casos particulares: em alguns casos não é necessário realizar a busca de uma única direção de projeção a cada iteração, enquanto que em outros casos é preciso realizar buscas de múltiplas direções em partições do espaço de aproximação. Métodos como tomografia computadorizada e partição do vetor de entrada foram desenvolvidos para lidar com situações como essas.

Para uma descrição detalhada de diversos índices de projeção já propostos, veja VON ZUBEN (1996).

4.3.3 Determinação da Função de Expansão Ortogonal

Uma vez definida a direção de projeção $\mathbf{v}_n \in \Re^m$, o problema de aproximação regularizado, apresentado na Equação (4.3), tem por objetivo aproximar uma versão suave da projeção dos resíduos da função desconhecida $g: \Re^m \rightarrow \Re$ na direção $\mathbf{v}_n$.

Para $\mathbf{v}_n$ ($n \geq 1$) fixo, é possível renomear as projeções unidirecionais dos dados de entrada na forma

$$z_l = \mathbf{v}_n^T \mathbf{x}_l, \quad l = 1, \dots, N.$$

Com isso, a função monovariável $f_n(\cdot)$ deve resolver o seguinte problema de aproximação regularizado:

$$\min_{f_n} \frac{1}{N} \sum_{l=1}^N [d_l - f_n(z_l)]^2 + \lambda_n \phi(f_n),$$

onde d_l são os resíduos do processo de aproximação, dados por:

$$d_l = s_l - \sum_{j=1}^{n-1} f_j(\mathbf{v}_j^T \mathbf{x}_l), \quad l = 1, \dots, N.$$

Duas técnicas distintas podem ser empregadas na determinação da função de expansão ortogonal:

- Solução paramétrica: utiliza uma base de funções ortonormais, como por exemplo, polinômios de Hermite (HWANG *et al.*, 1994).
- Solução não-paramétrica: uso de splines polinomiais suavizantes (VON ZUBEN, 1996).

4.3.4 O Processo de Ajuste Retroativo

O processo de aproximação por expansão ortogonal segue então os seguintes passos básicos:

- dado o conjunto de dados de aproximação $\{(\mathbf{x}_l, s_l) \in \Omega \times \mathfrak{R}\}_{l=1}^N$, com $\Omega \subset \mathfrak{R}^m$, e seja $g: \Omega \subset \mathfrak{R}^m \rightarrow \mathfrak{R}$ a função a ser aproximada.
- partindo de $n = 1$, e construindo o vetor $\mathbf{d} = [d_1 \dots d_N]^T$ na forma:

$$d_l = s_l - \sum_{j=1}^{n-1} f_j(\mathbf{v}_j^T \mathbf{x}_l), \quad l = 1, \dots, N,$$

resolva o seguinte problema de otimização:

$$\min_{\mathbf{v}_n, f_n} \frac{1}{N} \sum_{l=1}^N (d_l - f_n(\mathbf{v}_n^T \mathbf{x}_l))^2 + \lambda_n \phi(f_n)$$

pela obtenção sucessiva de valores ótimos para f_n e $\mathbf{v}_n$ na forma:

1. defina um valor inicial para $\mathbf{v}_n$ empregando algum índice de projeção;
2. para $\mathbf{v}_n$ fixo, resolva o seguinte problema de otimização (empregando um método paramétrico ou não paramétrico):

$$\min_{f_n} \frac{1}{N} \sum_{l=1}^N (d_l - f_n(\mathbf{v}_n^T \mathbf{x}_l))^2 + \lambda_n \phi(f_n);$$

3. para f_n fixo, partindo do valor atual de $\mathbf{v}_n$, resolva iterativamente o seguinte problema de otimização:

$$\min_{\mathbf{v}_n} \frac{1}{N} \sum_{l=1}^N (d_l - f_n(\mathbf{v}_n^T \mathbf{x}))^2 ;$$

- Faça $n = n + 1$ e repita o processo a partir do cálculo dos novos valores para o vetor de resíduos $\mathbf{d}$ enquanto o nível de aproximação desejado ainda não foi atingido (medido por algum critério de parada).

Deste processo de aproximação resulta, então, um modelo de aproximação por composição aditiva de funções de expansão ortogonal, na forma:

$$g(\mathbf{x}) \approx \hat{g}(\mathbf{x}) = \sum_{j=1}^n f_j(\mathbf{v}_j^T \mathbf{x}).$$

No entanto, o processo de construção deste modelo de aproximação – descrito acima – apresenta uma limitação advinda da estratégia de aproximação empregada, a qual é descrita a seguir:

- 1) com base nos dados de aproximação referentes ao problema de aproximação original;
- 2) encontre uma única direção de projeção e uma única função de expansão ortogonal a esta direção (sujeita a restrições de suavidade) que melhor aproxima os dados;
- 3) remova do conjunto de dados a informação representada no item (2);
- 4) enquanto o nível de aproximação desejado ainda não foi atingido, retorne ao item (2).

Observe que cada um dos n termos da composição aditiva resultou de um processo de aproximação que tinha por objetivo representar *toda* a informação presente nos dados de aproximação e que *ainda não tinham sido representadas* pelos termos *anteriores*. Isto implica que cada novo termo da composição aditiva não leva em conta a possibilidade de que, posteriormente, *novos* termos possam vir a compor o processo de aproximação.

Tomando qualquer termo da composição aditiva, com exceção do n -ésimo termo, e denominando-o k ($1 \leq k < n$), surge a seguinte questão: o que ocorreria com f_k e $\mathbf{v}_k$ se, na solução do problema:

$$\min_{\mathbf{v}_k, f_k} \frac{1}{N} \sum_{l=1}^N (d_l - f_k(\mathbf{v}_k^T \mathbf{x}))^2 + \lambda_k \phi(f_k)$$

em lugar de $\mathbf{d} = [d_1 \dots d_N]^T$ tal que

$$d_l = s_l - \sum_{j=1}^{k-1} f_j(\mathbf{v}_j^T \mathbf{x}_l), \quad l = 1, \dots, N,$$

se tomasse

$$d_l = s_l - \sum_{\substack{j=1 \\ j \neq k}}^n f_j(\mathbf{v}_j^T \mathbf{x}_l), \quad l = 1, \dots, N?$$

Se as duas escolhas para o vetor de resíduos $\mathbf{d}$ produzirem dados distintos, então f_k e $\mathbf{v}_k$ podem ser (e geralmente são) diferentes em cada caso. Conclui-se, portanto, que a solução produzida pelo processo de aproximação descrito acima pode deixar de ser ótima para os termos já calculados sempre que um termo adicional for incorporado. Sendo assim, é recomendável a aplicação de um processo de ajuste retroativo (*backfitting*):

- para cada j ($1 \leq j \leq n$), omite-se $f_j(\mathbf{v}_j^T \mathbf{x})$ do somatório e determina-se novos valores ótimos para f_j e $\mathbf{v}_j$. Repita este processo de ajuste retroativo até a convergência, medida por algum critério de parada.

4.3.5 O Tratamento de Múltiplas Saídas

Nesta seção consideraremos que a função a ser aproximada possui mais de uma saída, ou seja, é do tipo $g: \Omega \subset \mathbb{R}^m \rightarrow \mathbb{R}^r$, com $r > 1$. Duas possíveis extensões para o caso de múltiplas saídas são possíveis: múltiplas aproximações monovariáveis e aproximação multivariável.

A forma mais simples de se obter aproximações considerando múltiplas saídas é utilizar modelos de aproximação independentes, cada um desenvolvido para tratar uma única variável de saída. Sendo r o número de variáveis de saída, determinam-se r modelos de aproximação como segue:

$$\hat{g}_{n_k, k}(\mathbf{x}) = \sum_{j=1}^{n_k} f_{kj}(\mathbf{v}_{kj}^T \mathbf{x}), \quad k = 1, \dots, r,$$

um para cada variável de saída, produzindo um modelo de aproximação na forma:

$$\hat{\mathbf{g}}(\mathbf{x}) = \begin{bmatrix} \hat{g}_{n_1,1}(\mathbf{x}) \\ \vdots \\ \hat{g}_{n_r,r}(\mathbf{x}) \end{bmatrix}. \quad (4.4)$$

Por outro lado, a possível existência de associações entre as variáveis de saída pode ser explorada na obtenção de modelos de aproximação que demandem menos recursos computacionais ao aproximarem múltiplas variáveis de saída simultaneamente. Além disso, modelos computacionais parcimoniosos podem auxiliar na obtenção de melhores resultados em termos de generalização e também conduzir a modelos mais facilmente interpretáveis. Assim, cada variável de saída é aproximada como segue:

$$\hat{g}_{n,k} = \sum_{j=1}^n w_{kj} f_j(\mathbf{v}_j^T \mathbf{x}), \quad k=1, \dots, r, \quad (4.5)$$

produzindo um modelo de aproximação na forma:

$$\hat{\mathbf{g}}(\mathbf{x}) = \begin{bmatrix} \hat{g}_{n,1}(\mathbf{x}) \\ \vdots \\ \hat{g}_{n,r}(\mathbf{x}) \end{bmatrix}. \quad (4.6)$$

Observe que o modelo representado na Equação (4.4) é um caso particular do modelo representado na Equação (4.6), bastando assumir:

- $n = \sum_{k=1}^r n_k$;
- $w_{kj} = 0$, para $j \leq \sum_{i=1}^{k-1} n_i$ e $j > \sum_{i=1}^k n_i$;
- $f_{kj} = w_{kj} f_l$, com $l = \sum_{i=1}^{k-1} n_i + j$.

4.3.6 O Algoritmo de Aproximação Construtivo

Na Seção 2.6 analisamos redes neurais artificiais como um modelo de aproximação de funções. Reproduzindo o modelo de aproximação de uma rede neural com uma única camada intermediária (veja Equação (2.29)) na forma:

$$RN_k^{[n]}(\mathbf{V}, \mathbf{w}_k, \mathbf{x}) = \sum_{j=1}^n w_{kj} f_j(\mathbf{v}_j^T \mathbf{x} - v_{j0}) + w_{k0}, \quad k = 1, \dots, r, \quad (4.7)$$

e comparando com a Equação (4.5), verificamos que a única diferença entre os dois modelos é a presença de termos adicionais na Equação (4.7), representando a polarização dos neurônios.

Com isso, pode-se concluir que uma rede neural não-polarizada com uma camada intermediária pode ser considerada o resultado da implementação de um método estatístico especialmente desenvolvido para a geração de modelos de aproximação na forma da Equação (4.6), denominado SMART (“Smooth Multiple Additive Regression Technique”) (FRIEDMAN, 1984). Quando necessário, a polarização pode ser prontamente adicionada, produzindo um modelo de aproximação $\hat{g}$ formado por uma composição aditiva de funções f_j ($j = 1, \dots, n$) adequadamente transladadas (por v_{j0}), rotacionadas (por $\mathbf{v}_j$) e escalonadas (por $\|\mathbf{v}_j\|$). Para cada saída k , as avaliações de f_j em projeções de $\mathbf{x}$ na direção $\mathbf{v}_j$ são, por sua vez, adequadamente transladadas (por w_{k0}) e escalonadas (por w_{kj}).

Baseando-se no procedimento de ajuste retroativo apresentado na Seção 4.3.4, HWANG *et al.* (1994) propuseram um algoritmo construtivo para redes neurais com uma camada escondida, m entradas e r saídas. No entanto, VON ZUBEN & NETTO (1995) introduziram uma série de aperfeiçoamentos ao algoritmo original, produzindo um processo mais eficiente e menos custoso computacionalmente. Apresentamos a seguir a versão final do algoritmo de aproximação construtivo para múltiplas saídas (VON ZUBEN, 1996):

1. Dados $\mathbf{X} \in \mathcal{R}^{m \times N}$ e $\mathbf{S} \in \mathcal{R}^{r \times N}$, tome $j = 0$ e $\mathbf{D} = \mathbf{S}$.
2. Utilizando $\mathbf{X}$ e $\mathbf{D}$, faça $j = j + 1$ e atribua:
 - ♦ um valor inicial para $\mathbf{v}_j \in \mathcal{R}^m$, empregando índices de projeção;
 - ♦ uma forma inicial para f_j , empregando um método paramétrico ou não-paramétrico;
3. Utilizando $\mathbf{X}$ e $\mathbf{D}$, resolva os seguintes problemas em sequência até a convergência (medida por algum critério de parada):
 - 3.1 fixe f_j e obtenha um valor ótimo para $\mathbf{v}_j$;
 - 3.2 fixe $\mathbf{v}_j$, obtenha um f_j ótimo via técnicas de regularização e retorne ao passo 3.1.

4. Obtenha um valor ótimo para $\mathbf{w}_j$ empregando o método de otimização da condição de solvabilidade (VON ZUBEN & NETTO, 1995; VON ZUBEN, 1996).
5. Para cada b tal que $1 \leq b < j$, calcule:

$$\mathbf{D} = \mathbf{S} - \sum_{\substack{k=1 \\ k \neq b}}^j \mathbf{w}_k f_k(\mathbf{v}_k^T \mathbf{X}),$$

e repita os passos 3 e 4, com $j = b$;

6. Por avaliação da participação de cada neurônio $\mathbf{w}_k f_k(\mathbf{v}_k^T \mathbf{X})$, $k = 1, \dots, j$, na representação da matriz $\mathbf{S}$, aplique um procedimento de poda de neurônios que não apresente um nível de participação mínima.
7. Enquanto um determinado nível de aproximação não for atingido (medido por algum critério de parada), retorne ao passo 2.

A implicação prática do método de retroajuste é promover algum tipo de adaptação por parte de toda a rede neural sempre que um novo neurônio for acrescentado. Com isso, enquanto que no algoritmo de retropropagação o ajuste era mais custoso (se aplicava a todos os neurônios ao mesmo tempo) mas feito uma única vez, aqui o ajuste é menos custoso (se aplica a um neurônio de cada vez) mas deve ser feito várias vezes e de forma cíclica. Outra característica importante é o fato de que o ajuste de cada neurônio, mantendo os outros fixos, é feito por camada e também de forma fixa:

- ajusta-se a direção de projeção ($\mathbf{v}_j$);
- ajusta-se a função de ativação ou função de expansão ortogonal (f_j);
- repete-se o processo até a convergência.

Note que os parâmetros correspondentes à camada de saída ($\mathbf{w}_j$) são obtidos de forma fechada através da otimização da condição de solvabilidade (VON ZUBEN & NETTO, 1995), não requerendo nenhum processo iterativo de ajuste.

4.4 O Algoritmo A*

O A* é um algoritmo de busca heurística, que trabalha em grafos cujos nós são possíveis soluções para o problema e os arcos são denominados custos. O conjunto alvo G é

um subconjunto de nós cujos elementos satisfazem um determinado critério de sucesso. Os objetivos da busca realizada pelo A^* são:

- se uma solução existir, encontrar um elemento do conjunto alvo (admissibilidade);
- encontrar a solução com um esforço de busca mínimo, em termos do número de nós expandidos (otimalidade).

Seja $A = \{n\}$ o conjunto de todos os possíveis nós (soluções) do problema. Observe que A pode conter infinitos elementos. Para aplicar um algoritmo de busca junto a este conjunto, precisamos definir relações entre seus elementos. Isto é equivalente a definir um operador de expansão $\Gamma(n): A \rightarrow 2^A$, onde 2^A é o conjunto de todos os subconjuntos de A (*power set* de A), que mapeia qualquer nó $n \in A$ em um conjunto de sucessores (que é um subconjunto de A).

A idéia básica do algoritmo A^* é que, dado qualquer nó n_i do grafo, o custo do caminho ótimo a partir deste nó para o nó alvo “mais próximo” n_G seja descrito por uma função $h^*(n_i)$. Obviamente, nem o caminho ótimo nem o custo associado são conhecidos. Entretanto, se o custo puder ser estimado por uma função $h(n_i)$, que avalia alguma heurística conhecida acerca do problema de busca, então podemos quantificar o quão “promissora” é a expansão do nó n_i . A função $f(n_i)$ que avalia o quão promissora é a expansão do nó n_i é então definida por

$$f(n_i) = g(n_i) + h(n_i),$$

onde:

- $g(n_i)$ é o custo do melhor caminho conhecido entre o nó inicial n_0 ao nó n_i ;
- $h(n_i)$, denominada *função heurística*, é uma estimativa do custo do melhor caminho a partir do nó n_i ao elemento mais próximo do conjunto alvo n_G .

O algoritmo A^* simplesmente segue a estratégia de expandir o nó mais promissor. Duas condições são essenciais para o sucesso do algoritmo A^* :

- 1) $\forall n: h(n) \leq h^*(n)$ (condição de admissibilidade);
- 2) $\forall n: \forall n' \in \{\Gamma(n)\}: h(n) \leq C(n, n') + h(n')$ (condição de monotonicidade);

onde $C(n, n')$ denota o custo do caminho ótimo entre os nós n e n' . O item (1) acima assegura que o algoritmo A^* encontrará um caminho ao nó alvo com custo mínimo. Se o item (2) é

válido, então o A^* domina amplamente qualquer algoritmo admissível que use a mesma informação heurística (isto é, o A^* expande o menor número possível de nós). A Figura 4.5 apresenta o algoritmo A^* . Para maiores detalhes a respeito do A^* , veja NILSSON (1986).

4.4.1 O Algoritmo A^* na Otimização de Arquiteturas de Redes Neurais

A aplicação do A^* na otimização de arquiteturas de redes neurais artificiais foi proposta por DOERING *et al.* (1997). Neste caso, cada nó corresponde a uma arquitetura de rede neural $C(V, E)$. O operador de expansão $\Gamma(C)$ gera sucessores da seguinte forma:

1. Variando o número de neurônios em uma camada escondida: assuma que C é uma arquitetura com h camadas escondidas ($h \geq 1$). Então todas as arquiteturas C' com um neurônio adicionado à j -ésima camada escondida, $j = 1, \dots, h$, são sucessores de C .
2. Variando o número de camadas escondidas: assuma que C é uma arquitetura com r neurônios de saída. Então a arquitetura C'' com uma camada escondida com r neurônios inserida imediatamente antes da camada de saída é um sucessor de C .

Assim, $\Gamma(C)$ mapeia uma arquitetura com h camadas escondidas em $h + 1$ sucessores. Note que qualquer arquitetura C é acessível a partir de uma arquitetura inicial C_0 sem nenhuma camada escondida. Como exemplo, considere uma arquitetura 5-3-3-3 (5 entradas, 3

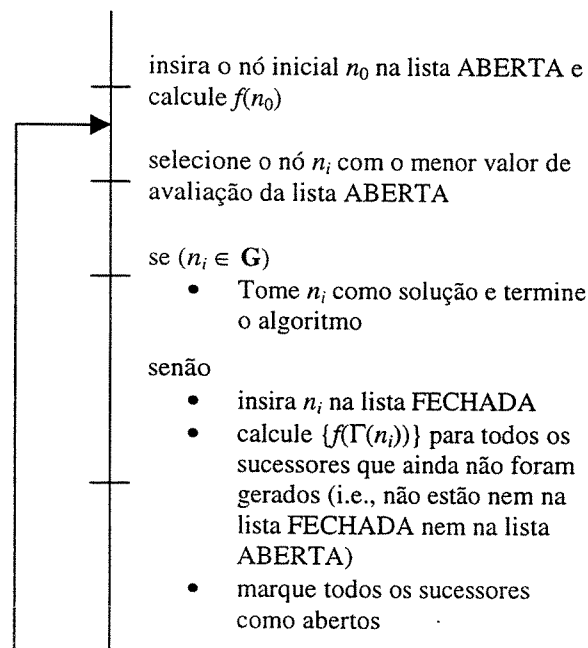


Figura 4.5 O algoritmo A^* .

neurônios na primeira camada escondida, 3 neurônios na segunda camada escondida e 3 neurônios de saída). A aplicação do operador Γ a esta arquitetura produz as seguintes arquiteturas sucessoras: 5-4-3-3, 5-3-4-3 e 5-3-3-3-3. Veja também a Figura 4.6.

Um critério de avaliação que determina o conjunto alvo G é definido como:

$$G = \{C \mid \eta(C) \leq \eta_0\},$$

onde $\eta(C)$ é o erro de generalização estimado aplicado a C e η_0 o critério de parada escolhido. Assim, o conjunto alvo G é composto por todas as arquiteturas que satisfazem o critério de avaliação.

Para aplicar o A^* , devemos ainda definir uma função de custo e uma função heurística:

- *Função de custo.* A cada operação de expansão $C' \in \Gamma(C)$ é atribuída uma função vetorial de custo definida por:

$$g(C, C') = \begin{bmatrix} \vartheta_h(C') - \vartheta_h(C) \\ \vartheta_L(C') - \vartheta_L(C) \end{bmatrix},$$

onde $\vartheta_h(C)$ é o número de neurônios escondidos em C e $\vartheta_L(C)$ é o número de camadas escondidas em C .

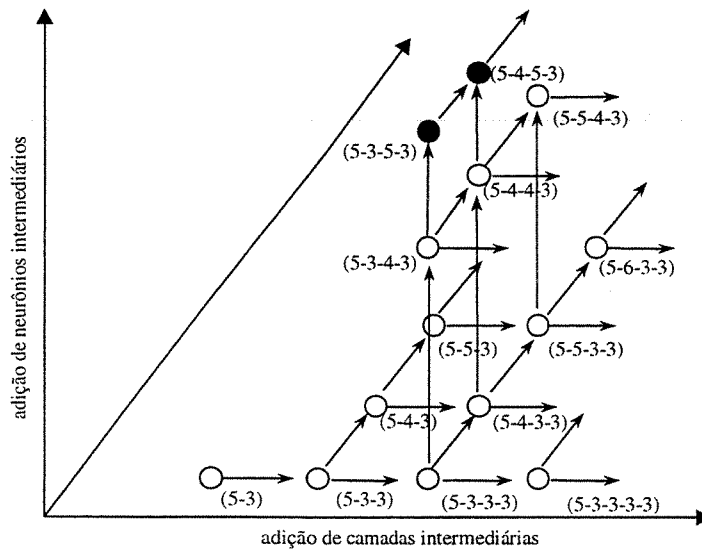


Figura 4.6 Aplicação sucessiva do operador de expansão $\Gamma(C)$ a uma arquitetura inicial com 5 entradas, 3 saídas e nenhum neurônio escondido.

- *Função heurística.* Seja $\varepsilon_L(C|L)$ o erro quadrático médio obtido pela arquitetura C sobre o conjunto de dados de treinamento L , e seja $\varepsilon_T(C|L, T)$ o erro quadrático médio obtido por C sobre o conjunto de dados para teste T após o treinamento utilizando L . Então, a combinação linear

$$h(C) = \frac{1}{h_0} \begin{bmatrix} \alpha \varepsilon_L(C|L) + \beta \varepsilon_T(C|L, T) \\ 0 \end{bmatrix},$$

onde:

$$h_0 = \alpha \varepsilon_L(C_0|L) + \beta \varepsilon_T(C_0|L, T), \text{ e} \\ \alpha + \beta = 1, \quad \alpha, \beta \geq 0,$$

pode ser usada como função heurística.

Por fim, de forma a comparar diferentes valores para a função de avaliação $f(C) = g(C) + h(C)$, precisamos definir a relação $\leq$ no $\mathfrak{R}^2$:

- *Relação $\leq$:* dados dois vetores $\mathbf{a}$ e $\mathbf{b}$ pertencentes ao $\mathfrak{R}^2$, a relação linearmente ordenada $\leq$ é dada por

$$(\mathbf{a} \leq \mathbf{b}) \Leftrightarrow (a_1 \leq b_1) \vee ((a_1 = b_1) \wedge (a_2 \leq b_2)).$$

4.5 Comparação entre CASCOR, PPL e A*

Nesta seção, apresentaremos extensões dos resultados obtidos por DE CASTRO *et al.* (1999), que realizaram uma comparação entre o desempenho dos três algoritmos construtivos estudados neste capítulo. Note que o desempenho dos algoritmos foi medido em termos da parcimônia das arquiteturas resultantes.

Para comparar o desempenho dos algoritmos, foram abordados seis problemas distintos, onde N é o número de amostras para treinamento. As funções de ativação dos neurônios no algoritmo PPL foram determinadas de forma paramétrica, utilizando polinômios de Hermite de ordem 5. A seguir, apresentamos os problemas abordados:

- Paridade 2 (XOR): $N = 4$.
- $\sin(x)\cos(x)$: $N = 25$.

- ESP (Estabilizador de Sistemas de Potência): problema de mundo real, $N = 75$.
- IRIS: problema de mundo real (veja Capítulo 7 para uma breve descrição deste problema), $N = 150$.
- GLASS: problema de mundo real, $N = 214$.
- SOJA: problema de mundo real, $N = 116$.
- FIC: aproximação de uma função de interação complicada, $N = 225$:

$$f(x_1, x_2) = 1,9 \left\{ 1,35 + e^{x_1 - x_2} \sin[13(x_1 - 0,6)^2] \sin[7x_2] \right\}.$$

Em todos os problemas acima, com exceção do problema FIC, utilizamos como critério de parada erro quadrático médio (EQM) inferior a 0,01. Para o problema FIC, por se tratar de um problema reconhecidamente difícil e também devido ao intervalo diferente de excursão da saída desejada, utilizamos como critério de parada $EQM < 0,1$.

O principal resultado apresentado diz respeito ao número de neurônios e camadas intermediárias geradas pelos algoritmos para solucionar os problemas. A Tabela 4.2 mostra os resultados obtidos pelos métodos construtivos.

A Tabela 4.2 mostra que os algoritmos A* e CASCOR nem sempre precisam adicionar neurônios escondidos para resolverem os problemas propostos, dados os critérios de parada escolhidos.

Tabela 4.2 Funções de ativação candidatas utilizadas pelo CASCOR.

	Arquitetura Resultante		
	CASCOR	PPL	A*
XOR	2-1-1	2-1-1	2-2-1
$\sin(x)\cos(x)$	1-2-1	1-3-1	1-6-1
ESP	2-0-5	2-3-5	2-0-5
IRIS	4-9-3	4-30-3	4-4-3
GLASS	9-28-3	9-28-3	9-24-3
SOJA	36-3-1	36-4-1	36-3-1
FIC	2-26-1	2-5-1	2-3-4-1

Verificou-se que, embora o A* permita a criação de redes com múltiplas camadas escondidas, sempre uma rede com uma única camada escondida foi obtida para a maioria dos problemas. Para o problema FIC foi necessário o uso de mais de uma camada escondida.

Outro aspecto interessante é o fato de que o CASCOR sempre escolheu (para os problemas abordados) como função de ativação as funções seno e cosseno, embora outras funções estivessem disponíveis.

O algoritmo PPL apresentou bons resultados para problemas que consideram uma única saída e poucas entradas, e resultado pobre para um dos problemas cuja rede possui mais de uma saída (IRIS). Esta é uma das dificuldades do método, e vários estudos têm sido desenvolvidos no sentido de amenizar os seus efeitos (BREIMAN & FRIEDMAN, 1997). Observe entretanto que, para o problema FIC, também utilizado em HWANG *et al.* (1994), o PPL apresentou um desempenho superior aos demais algoritmos. Isto mostra a adequação da estrutura conexionista com uma única camada intermediária cujos neurônios apresentam funções de ativação distintas, particularmente quando aplicada a problemas que exigem maior flexibilidade do modelo de aproximação.

4.6 Conclusão

Neste capítulo apresentamos três dos principais algoritmos construtivos para redes neurais artificiais já propostos na literatura: o CASCOR, o PPL e o A*. Descrevemos brevemente cada um desses algoritmos e apresentamos resultados de simulações computacionais comparando o desempenho desses algoritmos.

Capítulo 5

Redes Neurais Evolutivas

Métodos baseados em computação evolutiva têm sido propostos como uma alternativa eficiente e robusta para o projeto de redes neurais artificiais. Estes métodos podem ser usados para otimizar diversos aspectos de uma rede neural artificial: pesos, número de camadas escondidas, número de neurônios em cada camada escondida, conexões entre os neurônios, etc. Os algoritmos já apresentados na literatura se propõem a definir um ou mais desses aspectos simultaneamente. Neste capítulo iremos apresentar uma revisão de alguns dos principais métodos baseados em computação evolutiva já propostos na literatura para o projeto de redes neurais artificiais. Embora haja uma grande diversidade de abordagens, o potencial de aplicação de computação evolutiva no projeto de redes neurais artificiais ainda não foi totalmente explorado, conforme mostram os resultados a serem apresentados no Capítulo 6, quando algoritmos evolutivos serão apresentados no projeto de redes neurais híbridas.

5.1 Redes Neurais e Computação Evolutiva

Como já enfatizamos no capítulo anterior, projetar uma rede neural artificial para um problema específico não é uma tarefa fácil. Há uma grande quantidade de parâmetros a serem definidos, e alguns poucos princípios rigorosos de projeto estão à disposição para definição da rede neural. Assim, em muitos casos o projetista depende de sua experiência pessoal anterior ou de procedimentos de tentativa e erro. Os métodos construtivos apresentados no capítulo anterior são uma alternativa para o projeto de redes neurais artificiais, mas o custo computacional requerido pode ser proibitivo para certas aplicações, além da dificuldade apresentada no tratamento de espaços de entrada de elevada dimensão. Portanto, o objetivo

aqui é considerar técnicas avançadas de computação evolutiva, as quais também promovem a obtenção de arquiteturas de redes neurais dedicadas, ou seja, específicas para cada problema.

Podemos dividir as propostas de combinação de computação evolutiva e redes neurais em dois grupos principais: computação evolutiva para treinamento (ajuste de pesos) de redes neurais e computação evolutiva para definição de arquiteturas de redes neurais. A arquitetura de uma rede neural compreende os diversos parâmetros envolvidos no projeto de uma rede neural, como por exemplo número de camadas escondidas, número de neurônios em cada camada escondida, tipo de função de ativação, etc.

Este capítulo encontra-se dividido da seguinte forma: na próxima seção, apresentaremos alguns métodos para treinamento de redes neurais usando computação evolutiva. Apresentaremos também algumas simulações comparando o desempenho de um algoritmo genético e de um método de segunda ordem no treinamento de redes neurais. A seguir, discutiremos algumas propostas para definição de arquiteturas de redes neurais usando computação evolutiva e então apresentaremos as conclusões.

5.2 Computação Evolutiva para Ajuste de Pesos

Algoritmos evolutivos têm sido propostos como uma alternativa viável para o treinamento de redes neurais artificiais. A motivação para o uso de computação evolutiva para treinamento de redes neurais vem das limitações apresentadas pelo algoritmo de retropropagação original. Como todo algoritmo de otimização de primeira ordem, o algoritmo de retropropagação converge para mínimos locais (que podem ser insatisfatórios para o problema em questão) e apresenta desempenho pobre (convergência lenta) em problemas de larga escala (redes neurais podem ter centenas de pesos a serem ajustados). Algoritmos evolutivos têm capacidade de busca global, portanto têm sido estudados como um método alternativo para obtenção de soluções de alta qualidade no treinamento de redes neurais artificiais. Note que os métodos que serão apresentados nesta seção consideram que a arquitetura da rede já foi previamente definida (provavelmente de forma arbitrária e, portanto, não otimizada) e se preocupam apenas com o ajuste de pesos da rede.

A seguir, apresentamos um ciclo típico de um algoritmo para evolução de pesos de redes neurais (YAO, 1993), assumindo que já foi definida uma maneira de codificar geneticamente cada rede neural, que aqui será denominada um indivíduo da população:

1. Decodifique cada indivíduo da geração atual num conjunto de pesos e construa a rede neural correspondente.
2. Calcule o erro quadrático médio entre as saídas da rede e as saídas desejadas para cada rede neural e defina o negativo do erro como o *fitness* do indivíduo que gerou a rede (outras funções de *fitness* também são possíveis).
3. Reproduza um determinado número de indivíduos a partir da população atual com probabilidade proporcional ao *fitness* ou outro critério qualquer de seleção (veja Seção 3.3.4).
4. Aplique operadores genéticos, tais como *crossover* e mutação, e obtenha a nova geração.

Nesta seção, apresentaremos alguns dos principais métodos já propostos na literatura para o treinamento de redes neurais. Apresentaremos também alguns resultados comparando o desempenho de um algoritmo genético e de um método de treinamento de segunda ordem.

5.2.1 Revisão de Métodos Evolutivos para Treinamento de Redes Neurais

WHITLEY *et al.* (1990) realizaram experimentos com algoritmos genéticos usando codificação binária e em ponto flutuante. Para o algoritmo genético com codificação binária foram testados problemas de aproximação de funções booleanas (ou-exclusivo, codificador e dois tipos de somador binário). Também é proposto um esquema de mutação adaptativa com o objetivo de manter a diversidade da população. O algoritmo genético foi capaz de encontrar soluções em todos os problemas, embora tenha apresentado dificuldades em um dos problemas de somador binário. Observe que foi considerado um número extremamente elevado de indivíduos na população – 3000 indivíduos para testes com taxa de mutação fixa e 5000 indivíduos para testes com taxa de mutação adaptativa. Foram relatados também bons resultados para o algoritmo genético com codificação real num problema real de detecção de sinais.

Em FOGEL *et al.* (1990), é proposto o uso de programação evolutiva para o treinamento de redes neurais. Neste método, os indivíduos são vetores reais, sendo que cada elemento do vetor representa um peso da rede. Note que a programação evolutiva não

emprega o operador de *crossover*, baseando-se apenas na mutação para realizar a busca no espaço de parâmetros. Foram testados dois problemas: ou-exclusivo e um problema de predição de octanagem em uma mistura de gasolina. Em ambos os problemas o método evolutivo foi capaz de encontrar soluções em um número de iterações menor do que o algoritmo de retropropagação clássica.

KORNING (1994) conduziu diversos experimentos com o objetivo de mostrar que é possível usar algoritmos genéticos para treinar redes neurais mesmo quando a codificação leva a cromossomos muito longos. Para isso devem ser cuidadosamente selecionados o método de inicialização, a função de *fitness* e os operadores genéticos. O método foi testado em um problema de classificação de grãos de cereais. Um experimento interessante relatado pelo autor diz respeito à escolha da função de *fitness*: em um dos experimentos foi usada como função de *fitness* o percentual de classificação correta de um indivíduo (rede neural) e no outro experimento foi usado o erro quadrático médio obtido pelo indivíduo. Para o problema considerado, o uso do percentual de classificação como função de *fitness* apresentou desempenho superior. Apesar de bons resultados terem sido observados, o melhor resultado (em termos de classificações corretas) obtido por algoritmo genético foi inferior ao melhor resultado obtido por retropropagação.

CHEN (1998) propõe o uso de algoritmo genético com codificação binária para o ajuste de pesos e também para o ajuste de parâmetros relacionados à função de ativação (ganho e inclinação de uma sigmóide). São apresentados resultados comparando o desempenho do algoritmo genético e do algoritmo de retropropagação padrão no problema da íris (veja Capítulo 7). Os resultados mostraram uma superioridade do algoritmo genético sobre o algoritmo de retropropagação.

5.2.2 Comparação entre Algoritmo Genético e Método do Gradiente Conjugado para Treinamento de Redes Neurais

Nesta seção apresentaremos alguns resultados obtidos para o treinamento de redes neurais utilizando algoritmo genético e o método do gradiente conjugado escalonado (MØLLER, 1993). Foram testados dois problemas: o primeiro problema consistia no tradicional ou-exclusivo, e o segundo problema consistia em tentar aproximar a função

$$y = 1,9 \left[1,35 + e^{x_1 - x_2} \sin[13(x_1 - 0,6)^2] \sin[7x_2] \right], \quad (5.1)$$

onde foram utilizados 225 padrões de treinamento uniformemente amostrados no intervalo $[0, 1] \times [0, 1]$ (veja na Figura 5.1 um gráfico desta função).

Foram testados dois algoritmos genéticos: o primeiro com os pesos com representação binária e o segundo com codificação em ponto flutuante. Para o algoritmo com codificação binária, cada peso foi codificado com 19 bits, que corresponde a um número no intervalo $[-2, 2]$ com 5 casas decimais. O operador de *crossover* utilizado foi o *crossover* uniforme, o operador mutação foi o operador usual de mutação e o operador de seleção foi o *roulette wheel*. Para codificação em ponto flutuante, foi utilizado o *crossover* uniforme, mutação gaussiana (com média 0 e variância 0,1) e *roulette wheel*.

Os 3 algoritmos foram implementados na linguagem C++ e as simulações foram realizadas em uma estação Sun Sparc 4. Para cada um dos problemas, foram realizadas 10 simulações de cada um dos 3 algoritmos.

Para o problema do ou-exclusivo, foi escolhida uma rede neural com 2 entradas, 1 camada escondida com 2 neurônios e uma saída (2-2-1). Para os algoritmos genéticos, foram usados os seguintes parâmetros (sintonizados adequadamente para obtenção de bom

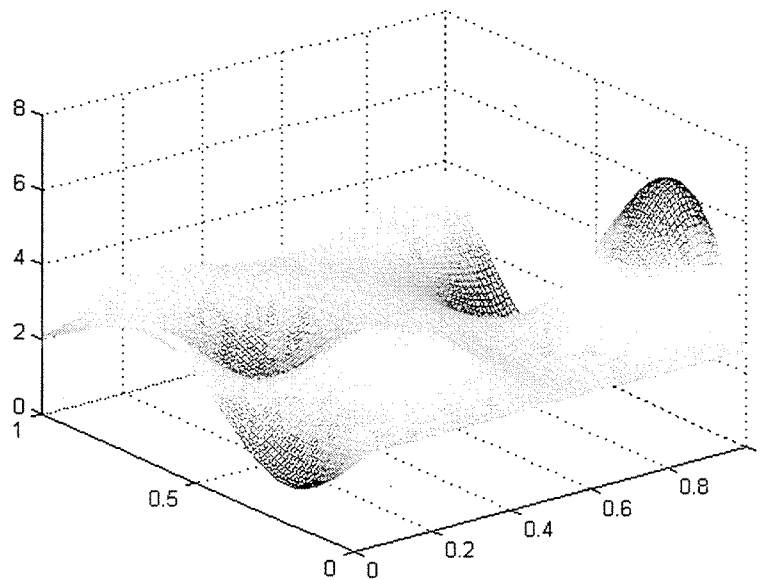


Figura 5.1 Função utilizada para testar algoritmos de treinamento de redes neurais.

desempenho):

	<u>Codificação binária</u>	<u>Codificação em ponto flutuante</u>
Tamanho da população:	100	50
Número de gerações:	1000	500
Probabilidade de <i>crossover</i> :	0,3	0,4
Probabilidade de mutação:	0,01	0,01

Para o algoritmo do gradiente conjugado escalonado, os pesos foram inicializados no intervalo $[-1, 1]$ e o critério de parada adotado foi norma do gradiente menor que 10^{-5} ou número de iterações igual a 500.

Na Tabela 5.1 apresentamos os resultados obtidos para o problema do ou-exclusivo. Na Tabela 5.1, AG-binário refere-se ao algoritmo genético com codificação binária, AG-real ao algoritmo genético com codificação em ponto flutuante e GCE ao gradiente conjugado escalonado. Melhor EQM é o menor erro quadrático médio (EQM) obtido nas 10 simulações, Pior EQM é o maior EQM obtido, EQM Médio é a médias dos EQM's nas 10 simulações e Tempo Médio corresponde ao tempo computacional médio de treinamento.

Analisando a Tabela 5.1, vemos que o melhor desempenho em termos de EQM foi obtido pelo algoritmo genético com codificação em ponto flutuante. Entretanto, em termos de tempo computacional, o algoritmo do gradiente conjugado escalonado obteve um desempenho muito superior aos demais algoritmos.

Para o problema de aproximação da função apresentada na Equação (5.1), foi escolhida uma arquitetura com 2 entradas, 1 camada escondida com 20 neurônios e 1 saída (2-20-1). Os parâmetros utilizados nos algoritmos genéticos foram:

Tabela 5.1 Resultados obtidos para o problema do ou-exclusivo.

	Melhor EQM	Pior EQM	EQM Médio	Tempo Médio
AG-Binário	0,01012	0,10695	0,08805	7min
AG-Real	$3,16694 \times 10^{-13}$	0,0625	0,01875	10s
GCE	$8,14471 \times 10^{-6}$	0,0833392	0,02084	0,08s

	<u>Codificação binária</u>	<u>Codificação em ponto flutuante</u>
Tamanho da população:	150	80
Número de gerações:	1000	500
Probabilidade de <i>crossover</i> :	0,3	0,4
Probabilidade de mutação:	0,01	0,01

(Aqui também os parâmetros foram sintonizados de tal forma a proporcionar desempenho satisfatório). Para o algoritmo do gradiente conjugado escalonado, os pesos foram inicializados em $[-1, 1]$ e o critério de parada adotado foi norma do gradiente do erro menor que 10^{-4} ou número de iterações igual a 1000.

Os resultados obtidos para o problema de aproximação são apresentados na Tabela 5.2. Para este problema, o algoritmo do gradiente conjugado escalonado obteve desempenho superior aos demais, tanto em termos de erro quadrático médio como também em tempo computacional. Os algoritmos genéticos apresentaram desempenho similar em termos de EQM, entretanto o algoritmo genético com codificação real apresentou custo computacional muito inferior.

No problema do ou-exclusivo, o algoritmo genético com codificação real apresentou desempenho superior (em termos de EQM) aos demais. Observe entretanto que este problema é bastante simples, e o desempenho médio do EQM do gradiente conjugado escalonado foi comparável ao do algoritmo genético com codificação real, e a um custo computacional muito menor. O problema de aproximação da função apresentada na Equação (5.1) é reconhecidamente difícil (HWANG *et al.*, 1994). Neste caso, o desempenho do algoritmo do gradiente conjugado escalonado apresentou desempenho muito superior ao dos outros

Tabela 5.2 Resultados obtidos para o problema de aproximação de funções.

	Melhor EQM	Pior EQM	EQM Médio	Tempo Médio
AG-Binário	0,00987	0,01031	0,01008	5h13min
AG-Real	0,0092622	0,0991044	0,01850073	37min
GCE	$9,33635 \times 10^{-4}$	0,0027966	0,00182104	5min

algoritmos.

Observando os resultados, vemos que o algoritmo genético com codificação em ponto flutuante apresentou desempenho superior ao algoritmo genético com codificação binária. Estes resultados podem ser explicados pelo fato de que a codificação binária não preserva relações de vizinhança no fenótipo, ou seja, uma modificação em apenas um bit pode provocar uma grande variação no valor do peso. Tal problema não é observado na codificação em ponto flutuante.

O desempenho superior do algoritmo do gradiente conjugado escalonado pode ser explicado da seguinte forma: um algoritmo genético, seja com codificação binária ou ponto flutuante, usa apenas informação de ordem zero a respeito da função objetivo, ou seja, usa apenas o valor da função em um determinado ponto. O algoritmo do gradiente conjugado, por sua vez, usa informação até ordem 2 (derivada segunda) da função objetivo no processo de otimização. Se tanto o algoritmo genético quanto o algoritmo do gradiente conjugado usam a informação a sua disposição com eficiência compatível, então parece bastante razoável esperar desempenho superior do algoritmo do gradiente conjugado, pois o mesmo usa uma quantidade maior de informação disponível a respeito da função objetivo do que o algoritmo genético.

É interessante notar que os autores citados na Seção 5.2.1, em geral, reportam resultados em que algoritmos evolutivos apresentam desempenho superior ao algoritmo de retropropagação. Entretanto, o algoritmo de retropropagação clássico é um algoritmo de primeira ordem e, como já enfatizado anteriormente, algoritmos de primeira ordem apresentam problemas de convergência lenta, e são comprovadamente inferiores a algoritmos de segunda ordem em termos de custo computacional. Se algum algoritmo de segunda ordem for utilizado nos problemas testados por esses autores, provavelmente apresentará desempenho superior aos métodos evolutivos propostos pelos mesmos.

Os resultados observados parecem confirmar a afirmação feita por SCHWEFEL (1994): “Ninguém deveria fazer uso de computação evolutiva quando bons e velhos métodos como programação dinâmica e linear, quase-Newton ou outro método bem fundamentado teoricamente funciona. Nenhum dos algoritmos evolutivos faria um trabalho melhor nem mesmo tão bom quanto eles. Computação evolutiva deveria ser levada em consideração se e

somente se métodos clássicos para o problema em questão não existem, não são aplicáveis, ou obviamente falham.”

5.3 Computação Evolutiva para Definição de Arquiteturas de Redes Neurais

Os métodos baseados em computação evolutiva têm sido reconhecidos como uma alternativa bastante adequada para definição automática da arquitetura de redes neurais artificiais. O problema de definição ótima de uma arquitetura de rede neural pode ser visto como uma busca por uma arquitetura que apresente melhor desempenho (de acordo com algum critério) em uma tarefa específica. Ou seja, devemos realizar uma busca em uma superfície definida pelo nível de desempenho de arquiteturas de redes neurais no espaço de arquiteturas, que é composto por todas as possíveis arquiteturas. MILLER *et al.* (1989) enumera algumas características desta superfície que tornam os algoritmos evolutivos candidatos bastante atraentes na implementação de métodos de busca:

- a dimensão da rede neural é ilimitada, portanto o espaço de arquiteturas é infinitamente grande;
- alterações no número de neurônios em uma camada escondida ou de conexões são discretas, e podem ter efeitos descontínuos no desempenho de uma rede neural, portanto a superfície de busca no espaço de arquiteturas é não-diferenciável;
- a superfície é complexa e ruidosa, pois o mapeamento entre uma arquitetura e seu desempenho após o treinamento é indireto e dependente das condições iniciais (inicialização dos pesos);
- redes estruturalmente similares podem apresentar habilidades de processamento de informação bastante distintas, de modo que a superfície é não-elucidativa (*deceptive*);
- redes estruturalmente distintas podem apresentar habilidades de processamento similares, portanto a superfície é multi-modal.

Nesta seção, apresentaremos alguns métodos evolutivos já propostos na literatura que abordam o problema de definição de arquitetura de redes neurais artificiais. Note que muitos destes métodos são utilizados na definição de apenas um ou poucos aspectos da arquitetura, considerando que os aspectos restantes já foram previamente definidos (provavelmente de

forma arbitrária, e portanto não-otimizada). Por exemplo, em alguns métodos o número de neurônios nas camadas escondidas é previamente determinado e o algoritmo evolutivo define a conectividade entre esses neurônios. Um ciclo típico de um algoritmo para evolução de arquiteturas de redes neurais é apresentado a seguir (adaptado de YAO (1993)), assumindo que já foi definida uma maneira de codificar geneticamente cada rede neural, que aqui será denominada um indivíduo da população:

1. Decodifique cada indivíduo da geração atual em uma arquitetura de rede neural.
2. Treine cada rede neural decodificada com um algoritmo de aprendizado fixo e pré-definido.
3. Calcule o *fitness* de cada indivíduo baseado nos resultados do treinamento.
4. Reproduza um número de indivíduos a partir da geração atual de acordo com algum critério de seleção.
5. Aplique operadores genéticos, tais como *crossover* e mutação, e obtenha a nova geração.

Observe que, em muitos dos métodos evolutivos já propostos na literatura, não existe o passo 2 do algoritmo acima, considerado uma estratégia de busca local em que os pesos da rede neural são determinados através de um algoritmo de treinamento. Ao invés disso, os pesos são representados nos indivíduos, e portanto são determinados pelo próprio processo evolutivo, não havendo a necessidade do passo 2.

Outro aspecto importante relacionado aos algoritmos evolutivos para definição de arquitetura de redes neurais diz respeito à codificação de arquiteturas em indivíduos para o processo de evolução (definição do genótipo). De um modo geral, os esquemas de codificação de redes neurais podem ser classificados em duas categorias principais (YAO & LIU, 1997; BALAKRISHNAN & HONAVAR, 1995):

- *Codificação direta* ou *esquema de especificação forte*: todas as conexões e nós de uma arquitetura são especificados no genótipo (por exemplo, através de representação binária). A transformação de genótipo em fenótipo é bastante simples. Um exemplo deste tipo de codificação é a matriz de conectividade, proposta por MILLER *et al.* (1989).
- *Codificação indireta* ou *esquema de especificação fraca*: apenas as características mais importantes de uma arquitetura, como número de camadas escondidas e número de neurônios escondidos são codificadas. Usualmente estes esquemas exigem um esforço

computacional considerável no processo de decodificação do genótipo em fenótipo. Um exemplo deste tipo de codificação pode ser visto em HARP *et al.* (1989).

A grande maioria dos algoritmos propostos na literatura utiliza esquema de codificação direta. A grande vantagem deste esquema é a sua simplicidade. Entretanto, arquiteturas com muitos neurônios e conexões podem levar a cromossomos de comprimento muito grande. Outra desvantagem relacionada a este esquema de codificação é que arquiteturas incorretas de redes neurais são geradas com frequência, exigindo que o algoritmo tenha que verificar se o indivíduo produzido é ou não válido. Por arquitetura incorreta entendemos, por exemplo, arquiteturas onde há neurônios cuja saída não se conecta a nenhum outro neurônio ou saída da rede.

Esquemas de codificação indireta tentam evitar os problemas apresentados por esquemas de codificação direta. Usualmente, estes esquemas são compactos e evitam o surgimento de arquiteturas incorretas. Entretanto, o esforço necessário para o processo de codificação/decodificação de indivíduos muitas vezes torna estes esquemas pouco atraentes do ponto de vista prático.

Uma comparação extensa entre diversos métodos de codificação pode ser encontrada em KOEHN (1994).

5.3.1 Revisão de Algoritmos Evolutivos para Definição de Arquiteturas de Redes Neurais

MILLER *et al.* (1989) propuseram um algoritmo com esquema de codificação direta, destinado a definir a topologia de uma rede neural. Os pesos eram determinados através de retropropagação. Foram testados quatro problemas simples, e o algoritmo obteve bons resultados em todos eles. Um fato interessante a ser destacado é que as melhores redes obtidas apresentavam arquiteturas não-convencionais, apresentando, por exemplo, conexões diretas entre nós de entrada e de saída.

Em WHITLEY *et al.* (1990) é proposto um método evolutivo para definição da conectividade de uma rede neural – ou seja, o número de neurônios na camada escondida é determinado previamente e um algoritmo genético é usado para determinar apenas as conexões entre estes neurônios. Um esquema de recompensa é proposto para privilegiar a

evolução de arquiteturas neurais parcimoniosas. Foram testados dois problemas binários simples, o somador de 2 bits e o ou-exclusivo.

KITANO (1990) propôs um algoritmo genético com codificação indireta para a evolução de arquiteturas de redes neurais. Os pesos eram determinados usando o algoritmo *quickprop*. Em seu método, um cromossomo representa regras gramaticais para geração de grafos usando codificação binária. O cromossomo é uma sequência de fragmentos, sendo que cada fragmento representa uma regra para a definição de um elemento do grafo. O método foi testado no problema do codificador/decodificador. Os resultados apresentados parecem mostrar uma superioridade deste método frente a métodos que usam codificação direta, especialmente quando o tamanho da rede aumenta (esta afirmação é contestada por SIDDIQI & LUCAS (1998), que apresentam resultados que mostram que esquemas de codificação direta podem apresentar desempenho tão bom quanto a codificação proposta por Kitano). KITANO (1994) apresentou uma extensão de sua proposta original, em que os pesos também são evoluídos, e posteriormente passam por um processo de ajuste fino por meio do algoritmo de retropropagação.

SCHIFFMANN *et al.* (1992) utiliza um algoritmo genético com esquema de codificação direta onde cada rede neural é representada por uma lista encadeada. Esta codificação procura evitar que arquiteturas inválidas sejam geradas ao longo da evolução. Esta codificação também permite que indivíduos de tamanho diferente sejam submetidos a *crossover*. Os autores testam este algoritmo em dois problemas: ou-exclusivo e um problema de predição do estado da glândula tiróide. O problema ou-exclusivo é bastante simples e o algoritmo conseguiu obter um arquitetura com apenas um neurônio escondido. O problema da tiróide é um problema real de alta dimensionalidade (21 atributos e 3 classes). Neste problema, o desempenho das arquiteturas obtidas pelo processo evolutivo é comparado com o desempenho de arquiteturas fixas (pré-definidas). É constatado que as arquiteturas evoluídas apresentam desempenho superior ao das arquiteturas fixas.

MANDISCHER (1993) usa um algoritmo genético com codificação indireta para determinação de topologias de redes neurais. Esta codificação também é proposta com o objetivo de evitar arquiteturas incorretas. Os pesos da rede são determinados através de retropropagação. Um aspecto interessante relativo à esta codificação é que os parâmetros de

treinamento para o algoritmo de retropropagação (taxa de aprendizado e momento) também são codificados no cromossomo. O autor apresenta resultados para problemas de detecção de bordas, reconhecimento de dígitos escritos à mão e aproximação de funções. O desempenho das redes obtidas por algoritmo genético é comparado ao desempenho de redes neurais pré-definidas. Também neste trabalho as redes evoluídas apresentaram desempenho superior em termos da qualidade da solução final.

Uma abordagem baseada em programação genética, denominada codificação celular, foi proposta por GRUAU (1994). Cada cromossomo é codificado em árvores gramaticais, sendo que cada árvore representa um conjunto de regras para construção de redes neurais artificiais. Este método foi usado na definição de arquiteturas neurais booleanas, e foi testado em vários problemas booleanos, como paridade, simetria e decodificador.

MANIEZZO (1994) propõe um algoritmo evolutivo denominado ANNA ELEONORA para determinação de conectividade e dos pesos de redes neurais. Uma característica interessante deste algoritmo é que a granularidade é também incorporada à codificação do indivíduo. A granularidade é o número de bits utilizado na codificação dos pesos da rede neural. Assim, o número de bits usado na codificação dos pesos da rede também é determinado ao longo do processo evolutivo. Outro aspecto interessante deste algoritmo é o uso de um operador local denominado *GA-simplex*. Este algoritmo foi testado em vários problemas booleanos e também na evolução de redes para o controle de “agentes” que jogam futebol.

Em LIU & YAO (1996) é apresentado um algoritmo baseado em programação evolutiva para definição de arquiteturas de redes neurais artificiais, incluindo a função de ativação de cada neurônio. Neste método, o algoritmo pode escolher entre duas possíveis funções de ativação: logística e gaussiana. Os pesos da rede são ajustados através do algoritmo de retropropagação combinado a um algoritmo de busca aleatória. O método é testado em um problema de predição de presença ou ausência de doença cardíaca. Para comparação, foram usados três modelos de redes neurais: redes neurais só com função de ativação logística, redes neurais só com função de ativação gaussiana e redes neurais com ambas as funções de ativação (todos os três modelos foram obtidos pelo algoritmo evolutivo). Os resultados obtidos para este problema em particular indicam que redes neurais com ambas

as funções de ativação apresentam capacidade de generalização maior que a dos outros dois modelos (desde que o conjunto de treinamento seja suficientemente grande).

YAO & LIU (1997) apresentaram uma outra versão do algoritmo proposto em LIU & YAO (1996). Neste novo algoritmo, apenas uma função de ativação é permitida para todos os neurônios da rede. Os pesos da rede são determinados através de um esquema híbrido de treinamento, envolvendo o algoritmo de retropropagação e o *simulated annealing* (AARTS & KORST, 1989). São propostos quatro operadores de mutação: remoção de neurônio, remoção de conexão, adição de neurônio e adição de conexão. Este método adota a seguinte estratégia na busca por redes neurais parcimoniosas: a remoção de nó ou de conexão é sempre tentada antes de uma adição de nó ou conexão. Se uma remoção é bem sucedida (*fitness* é maior do que o *fitness* do pior indivíduo da população) então nenhum outro operador de mutação é aplicado. Este método foi testado em uma ampla gama de problemas - paridade, diagnóstico médico, avaliação de cartão de crédito e predição de uma série temporal (série caótica de MacKey – Glass). Os resultados obtidos foram comparados aos resultados obtidos por outros algoritmos propostos na literatura, e o desempenho do método foi superior na grande maioria dos problemas.

FANG & XI (1997) propõem um algoritmo baseado em programação evolutiva para a definição simultânea da arquitetura e dos pesos da rede. É adotado um esquema de codificação direta simples, e a função de *fitness* utilizada é uma combinação entre o erro quadrático médio da rede, o número de neurônios escondidos e o número de conexões entre os neurônios. São empregados quatro operadores de mutação estrutural (adição de nó/conexão, remoção de nó/conexão) e um operador de mutação para os pesos. O método é testado em um problema de aproximação de funções.

CHEN *et al.* (1997) apresenta um algoritmo evolutivo simples para determinação da arquitetura de redes neurais com duas camadas escondidas de neurônios. O algoritmo emprega três operadores de mutação: alteração de λ (razão entre o número de neurônios da segunda camada escondida e o número de neurônios da primeira camada escondida), adição de conexões, remoção de conexões. O método é testado em problemas de aproximação de funções relacionados à indústria automobilística. As redes obtidas pelo algoritmo

apresentaram um menor número de neurônios e precisão maior do que redes anteriormente projetadas por especialistas.

Um método baseado em programação genética para determinação de arquiteturas de redes neurais é proposto por ZHANG *et al.* (1997). Neste algoritmo, as redes neurais são representadas por árvores. Esta representação tem a vantagem de dispensar procedimentos de codificação/decodificação de indivíduos, pois a avaliação do desempenho de uma rede pode ser feita diretamente na árvore. Uma característica marcante deste método é que ele permite o uso de neurônios com junção de soma ou junção multiplicativa. Um neurônio com junção de soma é aquele cuja saída é dada pelas equações (2.1) e (2.2). A saída de um neurônio com junção multiplicativa é dada por

$$y = f\left(\prod_{i=1}^n w_i x_i\right)$$

onde n é o número de entradas do neurônio, f é a função de ativação, w_i são os pesos associados ao neurônio e x_i são os sinais de entrada do neurônio. Os pesos das redes são determinados por outro algoritmo evolutivo (independente do algoritmo principal usado na determinação da arquitetura). O método foi aplicado em dois problemas de predição de séries temporais: série de MacKey-Glass e uma série obtida de um laser infravermelho.

ZHAO (1997) apresenta um algoritmo co-evolutivo para definição da arquitetura e dos pesos de redes neurais artificiais. Neste método, uma rede neural é decomposta em vários módulos, e cada módulo é otimizado individualmente via algoritmo genético. O algoritmo é aplicado a um problema de reconhecimento de dígitos escritos à mão.

Outro algoritmo co-evolutivo para determinação de pesos e arquitetura foi apresentado por MORIARTY & MIKKULAINEN (1998). Neste método, duas populações são evoluídas simultaneamente, uma população de neurônios e uma população de redes neurais. O método foi testado com sucesso em um problema de controle robótico.

PUJOL & POLI (1998) apresentam um algoritmo baseado em programação genética para evolução de arquiteturas, pesos e função de ativação de redes neurais. Neste algoritmo, é utilizada uma representação dual para redes neurais – as redes são representadas por cromossomos lineares e, em algumas situações, convertidas em grafos. Foi proposto um novo operador de *crossover*, bastante complexo. Foram apresentados resultados da aplicação deste

método em diversos problemas booleanos. Foram realizados dois testes: o primeiro considerando redes com apenas uma função de ativação (função sinal) e o segundo tendo a função de ativação de cada neurônio escolhida pelo algoritmo evolutivo. No segundo teste poderiam ser escolhidas duas funções de ativação – sinal ou sigmoidal. Como foram testados apenas problemas booleanos, houve um predomínio da função sinal nas soluções obtidas no segundo teste. Apesar disso, o desempenho da redes com duas funções foi levemente inferior ao de redes com apenas uma função.

5.4 Conclusões

Neste capítulo, apresentamos alguns dos principais métodos envolvendo computação evolutiva para o projeto de redes neurais artificiais. Foram apresentados métodos evolutivos para treinamento de redes neurais e para definição de arquiteturas de redes neurais, onde os pesos podem ou não ser definidos pelo processo evolutivo. Apresentamos também algumas simulações com o objetivo de comparar o desempenho entre um algoritmo genético clássico e um método de segunda ordem no treinamento de redes neurais artificiais. Foi observada uma clara superioridade do método de segunda ordem, tanto em termos de custo computacional como em termos da qualidade da solução obtida. Parece claro que a computação evolutiva deve ser empregada apenas quando o espaço de busca não é diferenciável, como é o caso do espaço de arquiteturas de redes neurais. Por fim, baseados nos resultados já apresentados na literatura, devemos ressaltar que é plenamente justificável evoluir a arquitetura de uma rede neural caso se queira obter uma solução dedicada e completamente adaptada ao contexto do problema que se deseja resolver.

Capítulo 6

Arquiteturas Híbridas de Redes Neurais Artificiais

Em capítulos anteriores, apresentamos as redes neurais multicamadas, juntamente com o algoritmo de retropropagação. Estudamos também algoritmos construtivos associados a estas arquiteturas, como o aprendizado por busca de projeção (*Projection Pursuit Learning* – PPL) e o A*, bem como técnicas de projeto de redes neurais utilizando computação evolutiva.

Considerando os algoritmos e arquiteturas estudadas, e também outras arquiteturas presentes na literatura, podemos verificar que os algoritmos diferem entre si em três aspectos básicos:

- no método usado na determinação da dimensão da rede neural;
- no procedimento adotado na escolha da função de ativação de cada neurônio;
- no tipo de composição das ativações dos neurônios escondidos usada para produzir a saída da rede.

Os algoritmos de aprendizado mais avançados são projetados para tratar todos estes aspectos simultaneamente, levando a soluções dedicadas. Neste capítulo, apresentaremos uma arquitetura de rede neural híbrida e evolutiva, que permite a definição de diferentes funções de ativação para os neurônios escondidos e também o uso de composições aditivas ou multiplicativas destas funções, ou seja, capaz de lidar com todos os três aspectos acima simultaneamente.

6.1 Introdução

No Capítulo 2, apresentamos as redes neurais multicamadas e o algoritmo de retropropagação (*backpropagation*), usado no treinamento destas redes. Estas arquiteturas neurais, com uma única camada de neurônios escondidos e submetidas a treinamento supervisionado,

têm sido aplicadas com sucesso em problemas de aproximação de funções não-lineares contínuas e definidas em espaços funcionais compactos. Os resultados mais avançados também fornecem taxas de convergência, estipulando quantos neurônios escondidos, com uma dada função de ativação (a mesma para todos os neurônios) devem ser usados para obter uma ordem de aproximação específica (BARRON, 1993; DEVORE *et al.*, 1989; MHASKAR & MICCHELLI, 1994).

Como já enfatizado no Capítulo 2, o aprendizado de um mapeamento de entrada-saída a partir de um conjunto de exemplos pode ser considerado como a síntese de um modelo de aproximação $\hat{g}(\cdot)$ para a função desconhecida $g(\cdot): \Omega \subset \mathfrak{R}^m \rightarrow \mathfrak{R}^r$, onde Ω é uma região compacta.

Para $l = 1, \dots, N$, sejam: $\mathbf{x}_l \in \mathfrak{R}^m$ o vetor coluna de variáveis independentes, $\mathbf{s}_l \in \mathfrak{R}^r$ o vetor coluna de saídas desejadas, e $\varepsilon_l \in \mathfrak{R}^r$ um vetor coluna de ruído estocástico aditivo, com média zero e variância fixa. Então, um processo de amostragem dado por

$$\mathbf{s}_l = g(\mathbf{x}_l) + \varepsilon_l, l = 1, \dots, N \quad (6.1)$$

produz uma matriz de entrada $\mathbf{X} \in \mathfrak{R}^{N \times m}$ e uma correspondente matriz de saída $\mathbf{S} \in \mathfrak{R}^{N \times r}$, com $\mathbf{x}_l^T$ e $\mathbf{s}_l^T$ como suas respectivas linhas. O objetivo é usar as matrizes $\mathbf{X}$ e $\mathbf{S}$ para construir o melhor modelo de aproximação $\hat{g}(\cdot)$, e adotar este modelo para obter uma estimativa $\hat{\mathbf{s}}$, dado qualquer $\mathbf{x} \in \Omega$, tal que $\hat{\mathbf{s}} = \hat{g}(\mathbf{x})$.

Os modelos conexionistas tradicionais usados em aproximação de funções, baseados em redes neurais com uma única camada escondida, podem ser vistos como consistindo de uma composição aditiva ponderada das ativações dos neurônios escondidos, mais um termo de polarização. Cada ativação é também produzida por uma composição aditiva ponderada do vetor de entrada atual $\mathbf{x} = [x_1 \dots x_m]^T$, mais um termo de polarização. Assim, o componente estimado $\hat{s}_k$, $k = 1, \dots, r$, do vetor de saída $\mathbf{s} = [s_1 \dots s_r]^T$ pode ser escrito na forma (veja Figura 6.1):

$$\hat{s}_k = w_{k0} + \sum_{j=1}^n \left[w_{kj} f \left(\sum_{i=1}^m v_{ji} x_i + v_{j0} \right) \right], \quad (6.2)$$

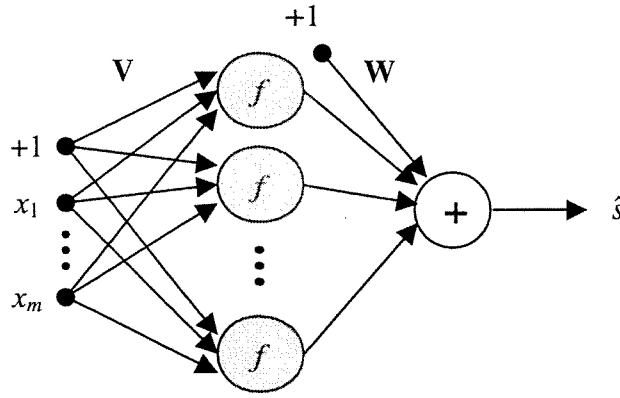


Figura 6.1 Modelo conexcionista tradicionalmente usado em problemas de aproximação de funções. Todos os neurônios escondidos possuem a mesma função de ativação. Para não sobrecarregar a figura, ilustramos uma rede neural com uma única saída ($r = 1$).

onde:

- n é o número de neurônios escondidos;
- f é a função de ativação dos neurônios escondidos;
- v_{ji} , $i \neq 0$, representa o peso sináptico que conecta a i -ésima entrada ao j -ésimo neurônio escondido;
- v_{j0} é a polarização do j -ésimo neurônio escondido;
- w_{kj} , $j \neq 0$, representa o peso que conecta o j -ésimo neurônio escondido ao k -ésimo neurônio de saída.
- w_{k0} representa a polarização do k -ésimo neurônio de saída;

Observe que este modelo apresenta uma severa limitação: todos os neurônios escondidos apresentam a mesma função de ativação $f(\cdot)$. Se a função de ativação de cada neurônio escondido puder ser apropriadamente e automaticamente definida, então melhores taxas de convergência poderão ser obtidas. Este é exatamente o propósito do algoritmo construtivo usando técnicas de aprendizado por busca de projeção (*Projection Pursuit Learning* - PPL), apresentado no Capítulo 4.

Nos modelos gerados pelo PPL, cada neurônio escondido tem sua própria função de ativação f_j , $j = 1, \dots, n$. A saída da rede neural pode então ser escrita na forma (veja Figura 6.2)

$$\hat{s}_k = w_{k0} + \sum_{j=1}^n \left[w_{kj} f_j \left(\sum_{i=1}^m v_{ji} x_i + v_{j0} \right) \right] \quad (6.3)$$

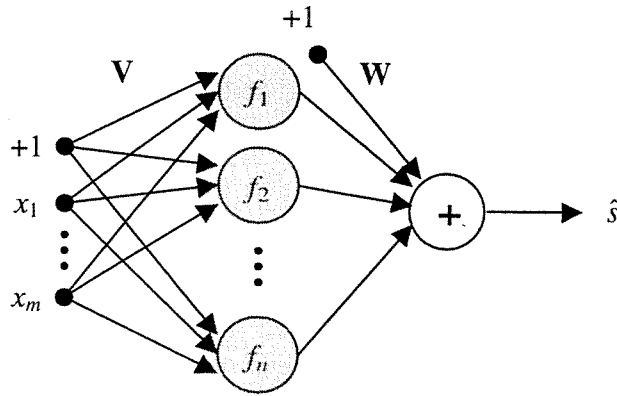


Figura 6.2 Modelo associado ao processo de aprendizado construtivo usando aprendizado por busca de projeção (PPL). Aqui, cada neurônio escondido pode ter sua própria função de ativação e, mais uma vez, a rede é apresentada com uma única saída ($r = 1$).

Agora, o processo de aprendizado supervisionado é responsável não apenas por determinar valores ótimos para os pesos v_{ji} ($i = 0, \dots, m; j = 1, \dots, n$) e w_{kj} ($j = 0, \dots, n; k = 1, \dots, r$), mas também a forma ótima das funções de ativação não-lineares f_j ($j = 1, \dots, n$).

Observe que, quando as funções de ativação f_j ($j = 1, \dots, n$) são idênticas, então as funções de transferência das Equações (6.2) e (6.3) são completamente equivalentes, ou seja, o modelo da Figura 6.1 é um caso particular do modelo da Figura 6.2.

A despeito de sua maior flexibilidade, a função de transferência associada ao modelo gerado pelo PPL (Figura 6.2) ainda apresenta uma forte restrição: composição aditiva é a única maneira de combinar as n funções de ativação distintas. Para superar esta limitação, propomos aqui uma arquitetura neural híbrida, que aceita também o operador de multiplicação.

6.2 Redes Neurais Híbridas

Os resultados mais avançados disponíveis na literatura a respeito de treinamento supervisionado em redes neurais com uma camada escondida podem ser classificados em três abordagens distintas:

Classe 1 as funções de ativação são as mesmas para cada neurônio escondido, e são definidas arbitrariamente ou baseadas em alguma heurística;

Classe 2 a melhor função de ativação para cada neurônio escondido é selecionada a partir de um número finito de candidatos;

Classe 3 a forma das funções de ativação de cada neurônio escondido é definida durante o aprendizado, de forma paramétrica ou não-paramétrica.

A principal restrição destas abordagens é que as ativações dos neurônios escondidos são combinadas empregando composição puramente aditiva. Composições puramente multiplicativas também podem ser consideradas (DONOHO & JOHNSTONE, 1989). Entretanto, quando a natureza do mapeamento a ser aproximado não é puramente aditivo ou puramente multiplicativo, é certamente possível obter melhores desempenhos considerando uma composição híbrida, com alguns neurônios contribuindo de forma aditiva e outros de forma multiplicativa, produzindo o que denominamos *rede neural híbrida* (IYODA & VON ZUBEN, 1999).

Na Figura 6.3, ilustramos o modelo híbrido proposto, sendo que, sem perda de generalidade, iremos considerar uma única saída ($r = 1$). A saída da rede neural híbrida é dada por

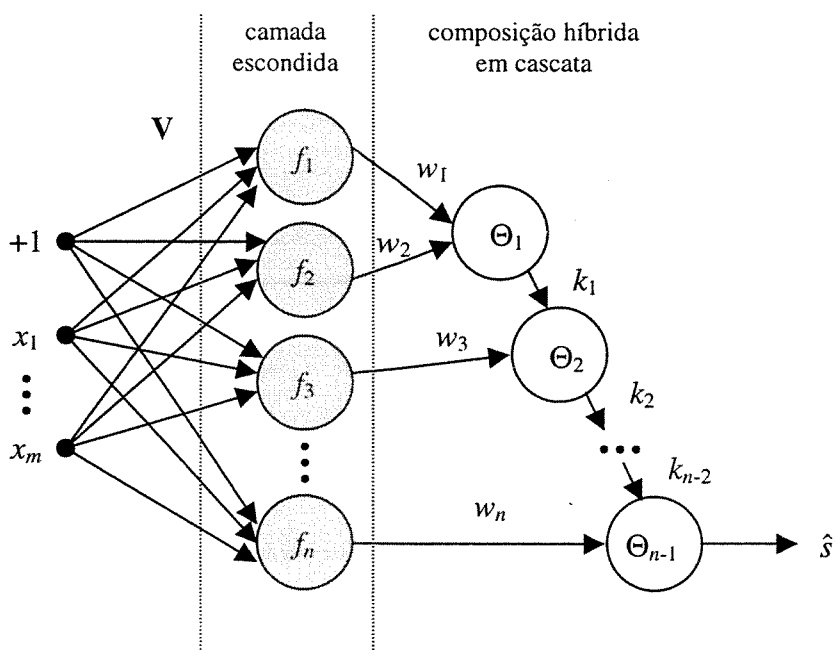


Figura 6.3 Arquitetura híbrida proposta: cada neurônio escondido pode ter sua própria função de ativação, e elas podem ser combinadas de forma aditiva ou multiplicativa: $\Theta_i, i = 1, \dots, n-1$ representa o operador soma ou multiplicação.

$$\hat{s} = (\cdots((z_1 \Theta_1 z_2) k_1 \Theta_2 z_3) k_2 \Theta_3 \cdots) k_{n-2} \Theta_{n-1} z_n \quad (6.4)$$

onde cada Θ_i , $i = 1, \dots, n-1$ representa ou o operador de soma ou o operador de multiplicação, e k_j , $j = 1, \dots, n-2$, representam os pesos conectando a cascata de composições. Os termos z_j , $j = 1, \dots, n$ são dados por

$$z_j = w_j f_j \left(\sum_{i=1}^m v_{ji} x_i + v_{j0} \right), \quad j = 1, \dots, n \quad (6.5)$$

isto é, z_j é a ativação ponderada do j -ésimo neurônio escondido.

Observe que, se os operadores Θ_i , $i = 1, \dots, n-1$, representam apenas adição e os pesos k_j , $j = 1, \dots, n-2$ são todos iguais a 1, então as redes neurais das Figuras 6.2 e 6.3 são completamente equivalentes, o que implica que o modelo PPL da Figura 6.2 é um caso particular do modelo híbrido da Figura 6.3.

Como o modelo híbrido da Figura 6.3 engloba os modelos de redes neurais artificiais apresentados nas Figuras 6.1 e 6.2 é evidente que seu poder de aproximação é superior ao apresentado pelos dois anteriores. No entanto, da mesma forma que a manipulação do modelo da Figura 6.2 é bem mais complexa que aquela empregada para ajustar os parâmetros do modelo da Figura 6.1, a determinação, para cada problema de aplicação:

- 1) do número ótimo de neurônios da camada intermediária;
- 2) da melhor função de ativação para cada neurônio;
- 3) de como elas devem ser compostas; e
- 4) dos pesos da rede neural,

não é um problema elementar.

A solução proposta neste trabalho está vinculada à aplicação de um algoritmo evolutivo com busca local.

Neste caso, foi desenvolvido um algoritmo genético para selecionar a melhor combinação de funções de ativação para os neurônios da camada escondida, dentre um conjunto de funções candidatas. O número máximo admitido de neurônios escondidos é dado, sendo que o processo evolutivo é responsável por eliminar os neurônios considerados em excesso, se for o caso. O algoritmo genético é também responsável pela determinação da seqüência de operadores na cascata de composições. Cada rede neural candidata, membro da

população em uma dada geração do processo evolutivo, tem seus pesos ajustados por um algoritmo de gradiente conjugado (processo de busca local), especificamente adaptado para manipular qualquer seqüência de operadores e qualquer conjunto de funções de ativação.

A seguir, examinaremos com detalhes aspectos relacionados ao treinamento da rede híbrida, e também detalharemos o algoritmo genético desenvolvido para escolha de funções de ativação e seqüência de operadores.

6.3 Treinamento da Rede Neural Híbrida: Busca Local

Nesta seção, apresentaremos a derivação de uma expressão para o gradiente do erro quadrático médio da rede neural híbrida. Esta expressão é usada no processo de busca local. O procedimento que apresentaremos é basicamente o mesmo já descrito no Capítulo 2.

Observando a Figura 6.3, vemos que os neurônios que formam a rede (neurônios da camada escondida e da composição híbrida em cascata) podem ser interpretados como casos particulares de um modelo generalizado de neurônio, apresentado na Figura 6.4. O termo v_j é o nível de ativação interna do neurônio generalizado, e é dado por

$$v_j = \begin{cases} \sum_{i=1}^p w_{ji} q_i, & \text{se } \Theta_j \text{ é o operador adição} \\ \prod_{i=1}^p w_{ji} q_i, & \text{se } \Theta_j \text{ é o operador multiplicação} \end{cases} \quad (6.6)$$

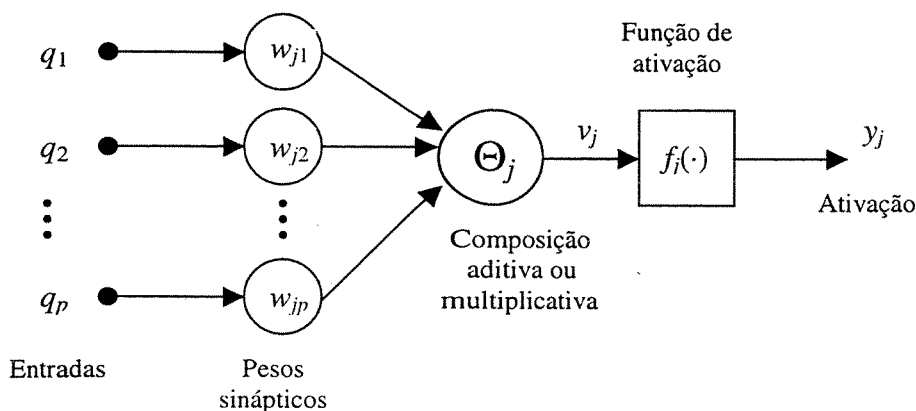


Figura 6.4 Modelo generalizado de neurônio.

A ativação y_j associada ao j -ésimo neurônio na iteração l (isto é, apresentação do l -ésimo padrão de treinamento, $l = 1, \dots, N$) é

$$y_j(l) = f_j(v_j(l)) \quad (6.7)$$

Os neurônios da camada escondida empregam composição aditiva e funções de ativação possivelmente distintas. Por outro lado, os neurônios responsáveis pela composição híbrida em cascata empregam composições possivelmente distintas (aditiva ou multiplicativa) e função de ativação fixa (linear).

Nosso objetivo aqui é o mesmo da Seção 2.4, isto é, derivar uma expressão para o gradiente do erro quadrático instantâneo em relação aos pesos sinápticos da rede. Os passos para obtenção desta expressão são exatamente os mesmos da Seção 2.4, entretanto algumas expressões devem ser modificadas para levar em consideração o fato de que podemos ter agora composição aditiva ou multiplicativa. Assim, não repetiremos aqui todos os passos necessários para a derivação do gradiente, e sim enfatizaremos as expressões que precisam ser modificadas em relação às já apresentadas na Seção 2.4.

Considere novamente a Figura 2.5, onde agora j é um neurônio generalizado. As equações (2.5) a (2.13) permanecem inalteradas. Entretanto, a expressão (2.14), que mostra a derivada do nível de ativação interno em relação a um peso sináptico deve ser modificada para:

$$\frac{\partial v_j(l)}{\partial w_{ji}(l)} = \begin{cases} y_i(l), & \text{se } \Theta_j \text{ é o operador adição} \\ y_i(l) \prod_{\substack{k=0 \\ k \neq i}}^p w_{jk}(l) y_k(l), & \text{se } \Theta_j \text{ é o operador multiplicação} \end{cases}, \quad (6.8)$$

de modo que a equação (2.15) deve ser reescrita como:

$$\frac{\partial \varepsilon(l)}{\partial w_{ji}(l)} = \begin{cases} -e_j(l) f'_j(v_j(l)) y_i(l), & \text{se } \Theta_j \text{ é o operador adição} \\ -e_j(l) f'_j(v_j(l)) y_i(l) \prod_{\substack{k=0 \\ k \neq i}}^p w_{jk}(l) y_k(l), & \text{se } \Theta_j \text{ é o operador multiplicação} \end{cases}. \quad (6.9)$$

Definindo, como na Seção 2.4, o gradiente local $\delta_j(l)$ na forma

$$\delta_j(l) = e_j(l) f'_j(v_j(l)) \quad (6.10)$$

podemos rescrever (6.9) como

$$\frac{\partial \varepsilon(l)}{\partial w_{ji}(l)} = \begin{cases} -\delta_j(l) y_i(l), & \text{se } \Theta_j \text{ é operador adição} \\ -\delta_j(l) y_i(l) \prod_{\substack{k=0 \\ k \neq i}}^p w_{jk}(l) y_k(l), & \text{se } \Theta_j \text{ é operador multiplicação} \end{cases} \quad (6.11)$$

Observe que aqui também teremos que considerar os dois casos distintos estudados na Seção 2.4, ou seja, devemos considerar separadamente o caso em que o neurônio j é um neurônio de saída da rede e o caso em que o neurônio j é um neurônio escondido. O primeiro caso, que denominamos caso I, é trivial, pois conhecemos o valor desejado para a saída do neurônio j , e pode-se facilmente calcular o gradiente local $\delta_j(l)$ usando a equação (6.10). Assim, nos deteremos a partir de agora na análise do caso II, isto é, a computação de $\delta_j(l)$ considerando que o neurônio j é um neurônio escondido.

Considere a Figura 2.6, onde os neurônios j e k agora são neurônios generalizados do tipo apresentado na Figura 6.4. As Equações (2.19) a (2.24) não precisam ser alteradas. A partir da Equação (2.25), devemos considerar o fato de que podemos ter composição aditiva ou multiplicativa das entradas aplicadas ao neurônio. Assim, devemos rescrever a Equação (2.25), que dá o nível de ativação interna do neurônio k , na forma

$$v_k(l) = \begin{cases} \sum_{j=0}^t w_{kj}(l) y_j(l), & \text{se } \Theta_j \text{ é o operador adição} \\ \prod_{j=0}^t w_{kj}(l) y_j(l), & \text{se } \Theta_j \text{ é o operador multiplicação} \end{cases} \quad (6.12)$$

onde t é o número total de entradas aplicadas ao neurônio k . Diferenciando a equação (6.12) em relação a $y_j(l)$, obtemos

$$\frac{\partial v_k(l)}{\partial y_j(l)} = \begin{cases} w_{kj}(l), & \text{se } \Theta_k \text{ é o operador adição} \\ w_{kj}(l) \prod_{\substack{m=0 \\ m \neq j}}^t w_{km}(l) y_m(l), & \text{se } \Theta_k \text{ é o operador multiplicação} \end{cases} \quad (6.13)$$

Substituindo as Equações (2.24) e (6.13) na Equação (2.22), obtemos

$$\frac{\partial \varepsilon(l)}{\partial y_j(l)} = \begin{cases} -\sum_k e_k(l) f'_k(v_k(l)) w_{kj}(l), & \text{se } \Theta_k \text{ é o operador adição} \\ -\sum_k e_k(l) f'_k(v_k(l)) w_{kj}(l) \prod_{\substack{m=0 \\ m \neq j}}^t w_{km}(l) y_m(l), & \text{se } \Theta_k \text{ é o operador multiplicação} \end{cases} \quad (6.14)$$

Usando a definição de gradiente local $\delta_k(l)$, dada na Equação (6.10), podemos rescrever a Equação (6.14) como

$$\frac{\partial \varepsilon(l)}{\partial y_j(l)} = \begin{cases} -\sum_k \delta_k(l) w_{kj}(l), & \text{se } \Theta_k \text{ é o operador adição} \\ -\sum_k \delta_k(l) w_{kj}(l) \prod_{\substack{m=0 \\ m \neq j}}^t w_{km}(l) y_m(l), & \text{se } \Theta_k \text{ é o operador multiplicação} \end{cases} \quad (6.15)$$

Substituindo (6.15) em (2.19), obtemos o gradiente local $\delta_j(l)$ para o neurônio escondido j :

$$\delta_j(l) = \begin{cases} f'_j(v_j(l)) \sum_k \delta_k(l) w_{kj}(l), & \text{se } \Theta_k \text{ é o operador adição} \\ f'_j(v_j(l)) \sum_k \delta_k(l) w_{kj}(l) \prod_{\substack{m=0 \\ m \neq j}}^t w_{km}(l) y_m(l), & \text{se } \Theta_k \text{ é o operador multiplicação} \end{cases} \quad (6.16)$$

Para finalizar, vamos resumir os resultados obtidos nesta seção. Nosso objetivo era derivar uma expressão para o gradiente do erro quadrático médio em relação aos pesos sinápticos da rede híbrida da Figura 6.3. Cada componente do vetor gradiente é dado por

$$\frac{\partial \varepsilon(l)}{\partial w_{ji}(l)} = \begin{cases} -\delta_j(l) y_i(l), & \text{se } \Theta_j \text{ é operador adição} \\ -\delta_j(l) y_i(l) \prod_{\substack{k=0 \\ k \neq i}}^p w_{jk}(l) y_k(l), & \text{se } \Theta_j \text{ é operador multiplicação} \end{cases} \quad (6.17)$$

onde o cálculo do gradiente local $\delta_j(l)$ depende da localização do neurônio j na rede, isto é, se o neurônio j é um neurônio escondido ou um neurônio de saída da rede:

1. Se j é um neurônio de saída, então $\delta_j(l)$ é dado por

$$\delta_j(l) = e_j(l) f'_j(v_j(l)) \quad (6.18)$$

2. Se j é um neurônio escondido, então $\delta_j(l)$ é dado por

$$\delta_j(l) = \begin{cases} f'_j(v_j(l)) \sum_k \delta_k(l) w_{kj}(l), & \text{se } \Theta_k \text{ é o operador adição} \\ f'_j(v_j(l)) \sum_k \delta_k(l) w_{kj}(l) \prod_{\substack{m=0 \\ m \neq j}}^l w_{km}(l) y_m(l), & \text{se } \Theta_k \text{ é o operador multiplicação} \end{cases} \quad (6.19)$$

onde o índice k refere-se a todos os neurônios que estão conectados à saída do neurônio j .

De posse do gradiente, podemos utilizar qualquer método de otimização não-linear que utilize informação de primeira ordem para ajustar adequadamente os pesos sinápticos da rede neural (veja, por exemplo, BAZARAA *et al.* (1992) ou LUENBERGER (1984)). Também é possível utilizar métodos de segunda ordem que usam a informação de primeira ordem para estimar a informação de segunda ordem, como o método do gradiente conjugado escalonado (MØLLER, 1993). Este foi o método utilizado neste trabalho. É possível também calcular de forma exata a informação de segunda ordem a partir da informação de primeira ordem e da aplicação de um operador diferencial (PEARLMUTTER, 1994).

6.4 A Abordagem Evolutiva

Neste trabalho, utilizamos um algoritmo genético para escolha do melhor conjunto de funções de ativação f_j ($j = 1, \dots, n$) dos neurônios da camada escondida, e da melhor sequência de operadores Θ_i ($i = 1, \dots, n-1$) da camada de composição híbrida. As funções de ativação f_j serão escolhidas dentre um conjunto finito de candidatos, apresentados na Tabela 6.1.

Na Figura 6.5, apresentamos os gráficos das funções candidatas. A razão da escolha dos 7 primeiros candidatos (funções 0 a 6) é que elas estão entre as classes mais frequentes de função de ativação encontradas na literatura. O oitavo candidato (função 7) foi incluído para criar a possibilidade de poda em redes com número excessivo de neurônios. Isto implica que o algoritmo genético também vai atuar na definição do número ótimo de neurônios, a partir de um número máximo fornecido.

O algoritmo genético implementado é basicamente o algoritmo genético tradicional, estudado no Capítulo 3. A seguir, descrevemos os principais aspectos relacionados ao algoritmo genético implementado neste trabalho.

Tabela 6.1 Conjunto de candidatos a função de ativação

Nr.	Nome	Função
0	Tangente hiperbólica normalizada	$s = 1,716 \operatorname{tgh}\left(\frac{2}{3}x\right)$
1	Logística	$s = \frac{1}{1 + e^{-x}}$
2	Linear	$s = x$
3	Cosseno	$s = \cos(x)$
4	Gaussiana	$s = e^{-0,5x^2}$
5	Seno hiperbólico	$s = \sinh(x)$
6	Cosseno hiperbólico	$s = \cosh(x)$
7	Função nula	$s = 0$

6.4.1 Codificação

No esquema de codificação adotado, cada indivíduo da população representa uma rede neural híbrida de tamanho máximo fixo, isto é, com um número máximo predeterminado n de neurônios na camada escondida e $n - 1$ neurônios na camada de composição híbrida em cascata.

A codificação é feita da seguinte forma: cada neurônio da camada escondida é codificado em um arranjo binário de 3 bits, com uma única sequência de bits representando cada uma das funções da Tabela 6.1. Se mais funções forem incorporadas ao conjunto de candidatos à função de ativação, então o número de bits deve ser adequadamente redefinido. Cada neurônio da camada de composição híbrida, por sua vez, é codificado em 1 bit, onde 0 representa o operador adição e 1 representa o operador multiplicação. Na Figura 6.6, ilustramos um exemplo deste esquema de codificação. Observe que a divisão do cromossomo em camada escondida e operadores foi feita para fins de uma melhor visualização do processo de codificação. Para propósitos de evolução, o cromossomo da Figura 6.6 é considerado um único arranjo binário de 19 bits. Para um dado n , a dimensão do arranjo é dada por $3 \times n + (n - 1) = 4 \times n - 1$.

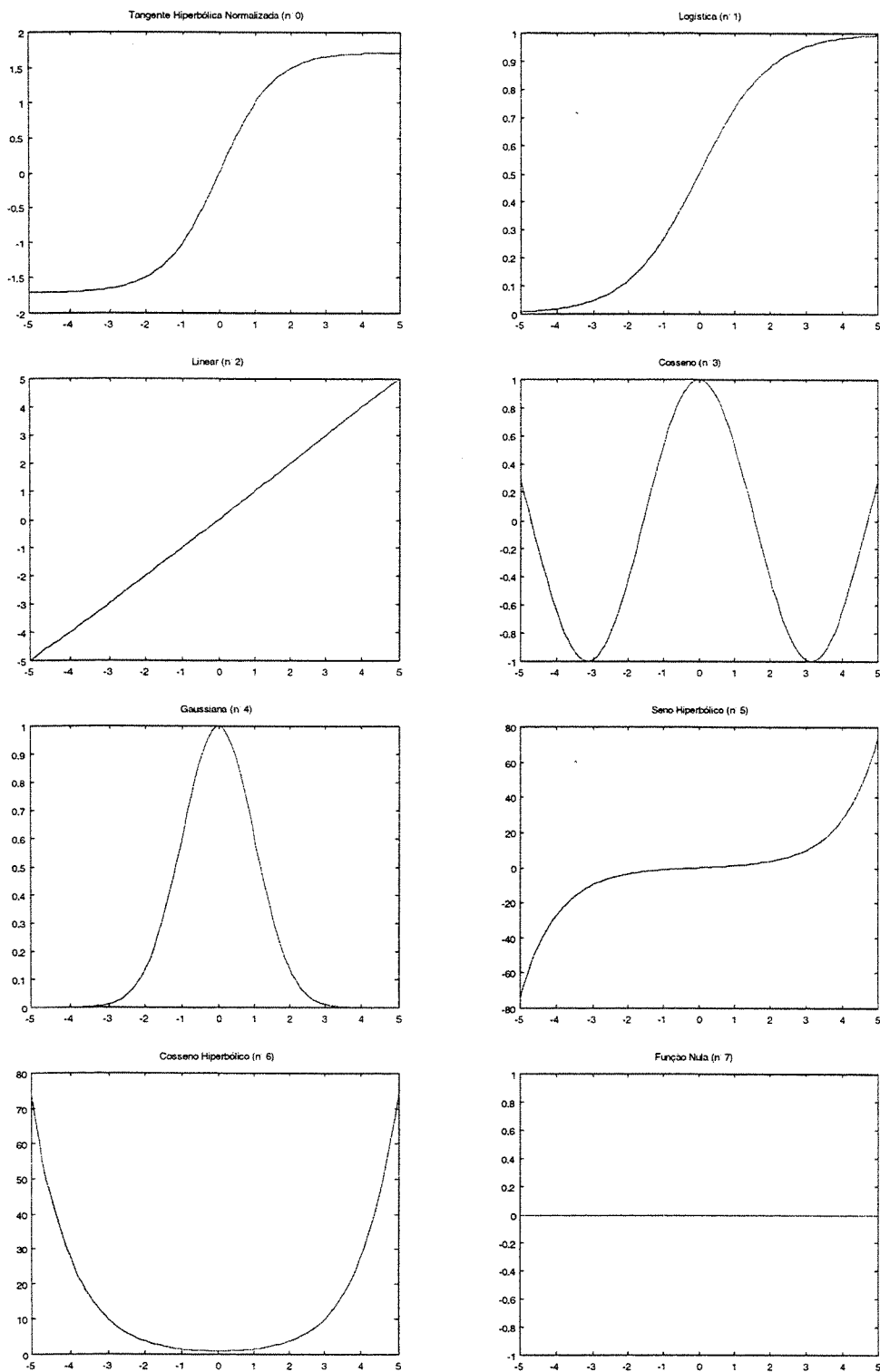


Figura 6.5 Conjunto de funções candidatas à função de ativação.

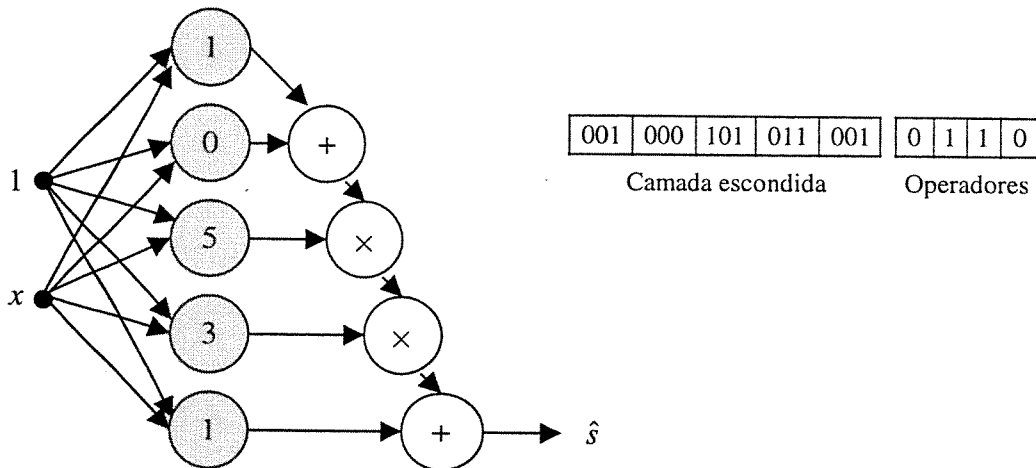


Figura 6.6 Exemplo do esquema de codificação: rede neural híbrida (fenótipo) e cromossomo correspondente (genótipo).

Observe que os pesos da rede neural não são considerados na codificação, pois estes são determinados por um algoritmo de gradiente conjugado (busca local).

6.4.2 Inicialização da População

A população do algoritmo genético utilizado tem tamanho fixo, isto é, um número fixo de indivíduos em todas as gerações. A população inicial é gerada de forma aleatória.

6.4.3 Avaliação dos Indivíduos

Para avaliar os indivíduos da população, cada indivíduo é decodificado na rede neural correspondente e então esta é treinada 5 vezes utilizando o algoritmo do gradiente conjugado escalonado, proposto por MØLLER (1993). Em cada etapa de treinamento, uma condição inicial diferente é atribuída aos pesos. Como a busca é local, e portanto depende da condição inicial, o melhor desempenho nas 5 tentativas é considerado um bom indicador do potencial efetivo da rede neural.

Sendo assim, o *fitness* de um indivíduo é definido por

$$fitness = -menor_eqm + C \quad (6.20)$$

onde *menor_eqm* é o menor erro quadrático médio obtido das 5 etapas de treinamento e *C* é uma constante apropriadamente definida de forma a evitar valores negativos de *fitness*.

Note que um indivíduo (rede neural híbrida) não é treinado até a convergência (isto é, até um mínimo local ser obtido precisamente), mas apenas por um número reduzido de

iterações (neste caso 50). Treinar os indivíduos até a convergência, além de não trazer contribuições significativas quanto ao potencial relativo das redes neurais, aumenta muito o custo computacional do algoritmo, o que torna seu uso inviável em problemas de grande porte. Assim, preferimos limitar o número de iterações de treinamento, embora de forma arbitrária, e usar o erro quadrático médio obtido após a execução deste número limitado de iterações como uma medida aproximada do *potencial* de um indivíduo. Para uma ilustração desta argumentação, veja a Figura 6.7.

6.4.4 Operador de Seleção

A cada indivíduo da população é atribuída uma probabilidade de passar para a próxima geração, que é proporcional ao *fitness* do indivíduo. Assim, quanto maior o *fitness* do indivíduo, maior a chance deste indivíduo passar para a próxima geração. Neste trabalho, usamos o esquema de seleção elitista: o melhor indivíduo é preservado, ou seja, sempre passa para a próxima geração, e o restante dos indivíduos é selecionado por *roulette wheel* (apresentado no Capítulo 3).

Um aspecto importante a salientar é que quando um indivíduo passa de uma geração a outra sem sofrer *crossover* nem mutação (descritos a seguir), ele não é reavaliado, passando para a próxima geração com o valor de *fitness* calculado na geração atual. Assim, consegue-se poupar um tempo computacional considerável.

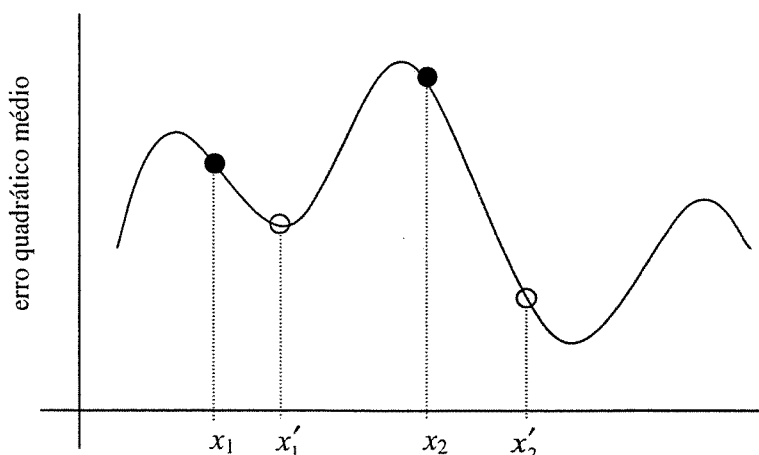


Figura 6.7 O indivíduo x_1 tem erro quadrático médio menor que x_2 . Entretanto x_2 claramente tem um *potencial* maior que x_1 . Esta relação entre os potenciais associados a x_1 e x_2 pode ser detectada facilmente com poucos passos de treinamento, sem necessidade de treinarmos os indivíduos até a convergência.

6.4.5 Operador de *Crossover*

O operador de *crossover* utilizado foi o *crossover* uniforme, descrito no Capítulo 3. Quando o algoritmo genético chega em gerações avançadas, observa-se que a população torna-se bastante homogênea (baixo nível de diversidade), o que aumenta muito a probabilidade de termos *crossover* entre dois indivíduos idênticos, produzindo indivíduos idênticos ao pais. Quando isto ocorre, adota-se o seguinte procedimento:

1. avalie cada um dos dois filhos obtidos usando o procedimento descrito em 6.4.3;
2. compare o melhor *fitness* obtido dos filhos com o *fitness* dos pais;
3. se o melhor *fitness* dos filhos é melhor que o *fitness* dos pais, então substitua o *fitness* dos pais, e de todos os indivíduos idênticos a eles que por ventura houver na população, pelo *fitness* do filho mais adaptado.

O procedimento acima executa uma nova busca local na tentativa de melhorar um indivíduo e também assegura que indivíduos idênticos não tenham valores distintos de *fitness*.

6.4.6 Operador de Mutação

O operador de mutação utilizado foi o operador de mutação convencional, onde um bit selecionado para mutação tem seu valor complementado.

No caso da mutação, também existe a possibilidade (embora muito baixa) de ser gerado um indivíduo já existente na população. Neste caso, após a avaliação do indivíduo mutado, verifica-se se o *fitness* deste indivíduo é melhor que o dos outros indivíduos idênticos a ele; se isto ocorre, então o *fitness* de todos os indivíduos idênticos ao indivíduo mutado assumem o seu valor. Caso contrário, este indivíduo repetido assume o melhor *fitness* dos clones já existentes.

6.5 Resultados de Simulações

Para avaliar o desempenho das redes neurais híbridas, dividimos as simulações em duas etapas principais. Na primeira, testamos a capacidade das arquiteturas em aproximar funções pertencentes ao próprio conjunto de funções candidatas. Na segunda etapa, comparamos o desempenho da rede neural híbrida com o de outras arquiteturas em problemas de aproximação de funções cujas soluções não pertencem ao espaço de funções possíveis de

serem produzidas pelo processo evolutivo. Sendo assim, a solução produzida pelo processo evolutivo será sempre a melhor aproximação possível, ou seja, o elemento do espaço de funções representáveis que mais se aproxime da solução exata.

6.5.1 Aproximação de Funções do Conjunto de Candidatos

O objetivo desta etapa é verificar se o erro de aproximação da rede híbrida cai assintoticamente a zero quando o problema de aproximação de funções pode ser resolvido de forma exata. Isto ocorre quando a função a ser aproximada $g(\cdot)$ pertence à classe de funções que podem ser geradas pelo modelo de aproximação $\hat{g}(\cdot)$.

Em todas as simulações realizadas nesta etapa usamos populações fixas composta por redes neurais híbridas com 5 neurônios na camada escondida, e 50 dados de entrada/saída para treinamento, uniformemente amostrados no intervalo $[-5, 5]$. Sob estas condições, o algoritmo genético foi capaz de encontrar um conjunto de funções de ativação e uma sequência de operadores tais que o erro quadrático médio sempre atingia valores muito próximos de zero ($\cong 10^{-6}$).

Esta etapa de simulação foi subdividida em 3 estágios, que serão descritos a seguir. Em todos os estágios, o objetivo é resolver problemas de aproximação de funções baseados na escolha de um modelo de aproximação $\hat{g}(\cdot)$ composto de uma rede neural híbrida com um dado número de neurônios na camada escondida.

Estágio 1: A rede neural híbrida foi usada para aproximar dados gerados a partir de uma função $g(\cdot)$ suposta desconhecida, que é completamente equivalente a uma rede neural híbrida com apenas um neurônio na camada escondida, cuja função de ativação foi escolhida dentre os candidatos descritos na Tabela 6.1. Escolhemos a tangente hiperbólica normalizada e a função linear como dois exemplos:

$$g_1(x) = 1,716 \tanh\left(\frac{2}{3}x\right) \text{ e } g_2(x) = x$$

Os parâmetros usados no algoritmo genético foram:

- Tamanho da população: 5
- Número máximo de gerações: 10
- Probabilidade de *crossover*: 0,25

- Probabilidade de mutação: 0,01

A Figura 6.8 ilustra uma solução encontrada para $g_1(\cdot)$ e para $g_2(\cdot)$. Um aspecto interessante a observar tanto na Figura 6.8(a) como em (b), é a existência de valores nulos associados a alguns dos pesos sinápticos, cujo efeito é cancelar o efeito da combinação de ativações produzidas anteriormente àquela conexão. Como consequência prática, a rede neural resultante é de fato uma rede com apenas um neurônio na camada escondida. Como esperado, as saídas das redes nas Figuras 6.8(a) e (b) são dadas por $\hat{s} \cong 1,716 \tanh(2x/3)$ e $\hat{s} \cong x$ respectivamente.

Estágio 2: Neste estágio, tentamos aproximar composições de duas funções supostas desconhecidas, ambas escolhidas dentre as funções candidatas descritas na Tabela 6.1. As seguintes funções foram consideradas:

$$g_3(x) = \frac{1}{1 + e^{-x}} + \cos(x) \text{ e } g_4(x) = \cos^2(x)$$

A função $g_3(\cdot)$ é uma composição aditiva das funções 1 e 3, ambas descritas na Tabela 6.1. A função $g_4(\cdot)$ é uma composição multiplicativa da função 3 com ela mesma. Os parâmetros do algoritmo genético foram:

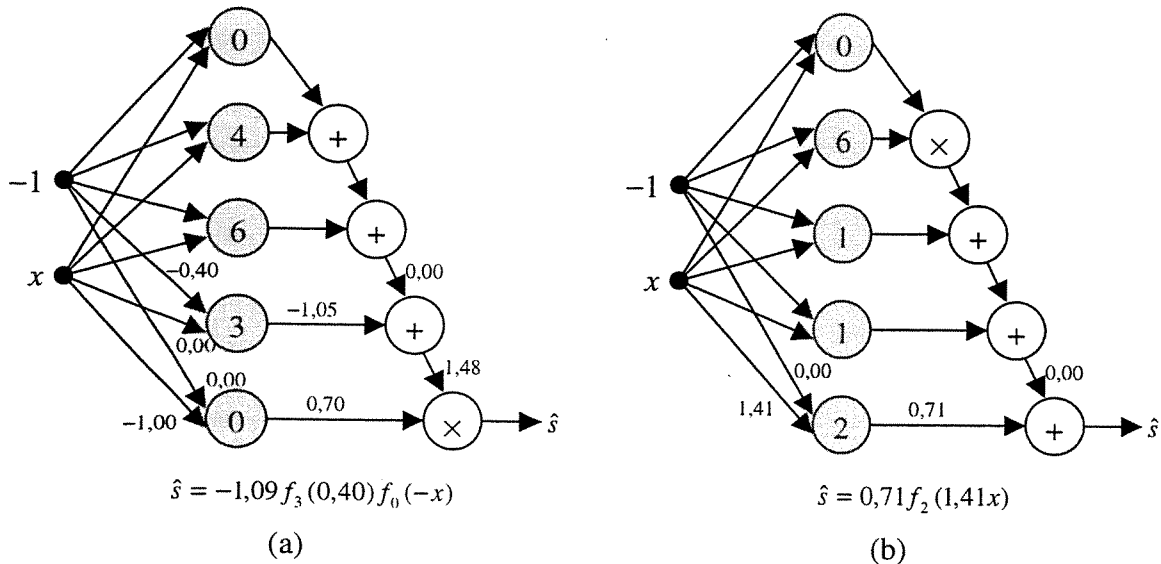


Figura 6.8 Exemplos de soluções obtidas no estágio 1 de simulações. Em (a) temos uma solução obtida para $g_1(\cdot)$ e em (b) temos uma solução obtida para $g_2(\cdot)$. Os pesos cujos valores não foram apresentados têm participação desprezível na solução final.

- Tamanho da população: 10
- Número máximo de gerações: 10
- Probabilidade de *crossover*: 0,25
- Probabilidade de mutação: 0,01

A Figura 6.9 apresenta dois exemplos de soluções obtidas neste estágio. Os mesmo comentários feitos no estágio anterior se aplicam também aqui: o algoritmo sempre encontrou redes neurais com erro de aproximação muito pequeno. Um fato interessante a se observar na Figura 6.9(b) é que a sub-rede acima da linha pontilhada produz uma saída praticamente constante ($\approx 1,2$) na região de aproximação, funcionando assim como uma espécie de polarização. Observe que as redes da Figura 6.9 correspondem aproximadamente a $g_3(\cdot)$ e $g_4(\cdot)$, e foram produzidas pelo processo evolutivo. Entretanto, estas soluções não são únicas (já que o erro quadrático médio final pode ser ligeiramente maior que zero), sendo que foram obtidas soluções com composições diferentes, inclusive com mais de 2 neurônios efetivos na camada escondida.

Estágio 3: Neste estágio, consideramos composições mais complexas das funções de ativação candidatas. As funções consideradas foram:

$$g_5(x) = x + \cos(2x) \text{ e } g_6(x) = \frac{1}{1 + e^{-x}} + xe^{-0.5x^2}$$

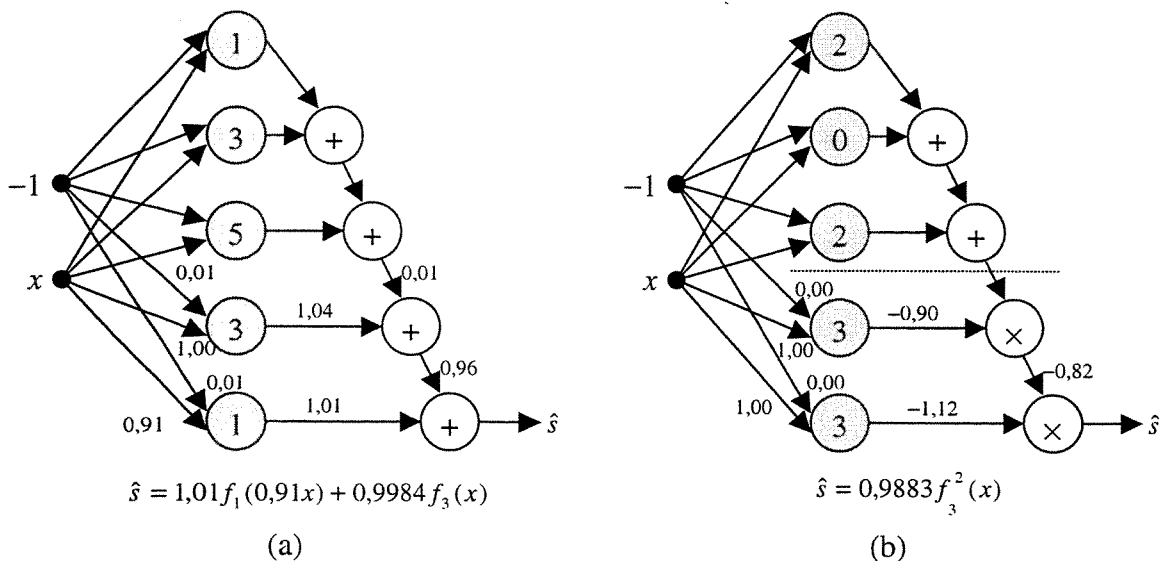


Figura 6.9 Exemplos de soluções obtidas no estágio 2. Em (a) temos uma solução obtida para $g_3(\cdot)$ e em (b) temos uma solução obtida para $g_4(\cdot)$.

Os parâmetros usados no algoritmo genético foram:

- Tamanho da população: 20
- Número máximo de gerações: 50
- Probabilidade de *crossover*: 0,25
- Probabilidade de mutação: 0,01

Na Figura 6.10 ilustramos as melhores soluções obtidas neste estágio. Aqui, soluções com diferentes composições e com mais de 2 (para $g_5(\cdot)$) e 3 (para $g_6(\cdot)$) neurônios escondidos foram mais freqüentes. Entretanto a solução final sempre tinha menos de 5 neurônios efetivos.

A Tabela 6.2 mostra um sumário dos melhores resultados obtidos nos três estágios acima descritos. Aqui, n é o número final de neurônios efetivos na camada escondida, FA é o conjunto de funções de ativação encontrado e SC é a sequência de composições correspondentes.

6.5.2 Comparação com Outras Arquiteturas

Nesta seção, o desempenho da arquitetura híbrida proposta será comparado com o desempenho de duas outras arquiteturas muito encontradas na literatura (VON ZUBEN *et al.*, 1999):

- o perceptron com uma camada escondida e submetido a treinamento supervisionado por

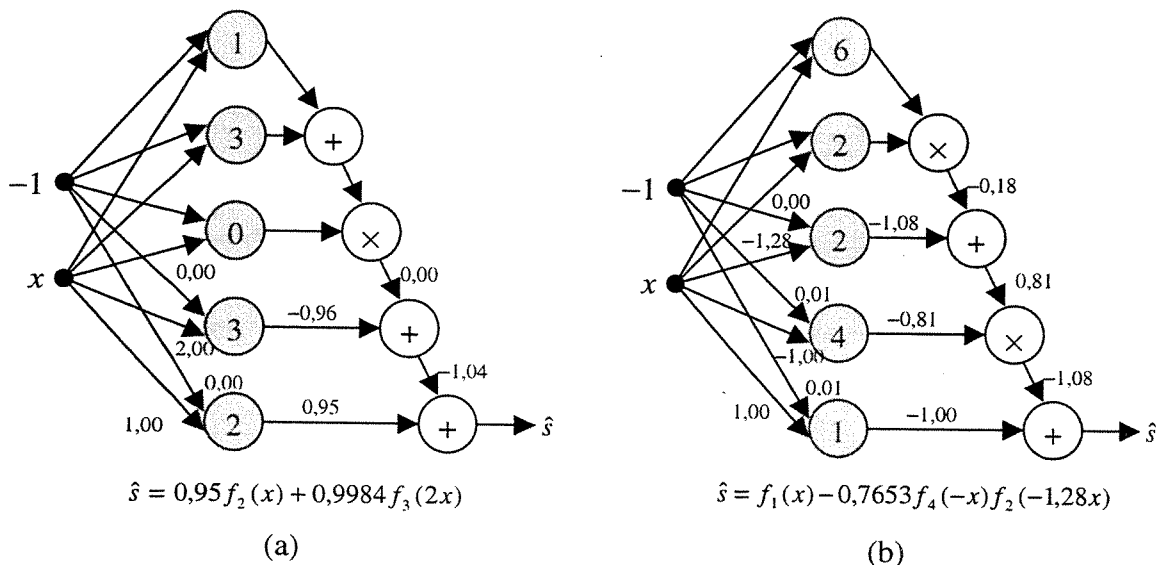


Figura 6.10 Exemplos de soluções obtidas no estágio 3. Em (a) temos uma solução obtida para $g_5(\cdot)$ e em (b) temos uma solução obtida para $g_6(\cdot)$.

Tabela 6.2 Resultados de simulações: melhores resultados obtidos na primeira etapa de testes.

Função	n	FA	SC
g_1	2	3-0	×
g_2	1	2	
g_3	2	3-1	+
g_4	2	3-3	×
g_5	2	3-2	+
g_6	3	2-4-1	×+

gradiente conjugado; e

- o aprendizado por busca de projeção.

Aqui, também dividimos as simulações em três estágios, cada um deles correspondendo a um problema de aproximação de funções distinto, particularmente escolhidos para enfatizar os principais aspectos das diferentes abordagens.

Notação: **P+CG:** Perceptron com uma camada escondida + gradiente conjugado

PPL: Aprendizado por busca de projeção

RNH: Rede neural híbrida

n : número de neurônios na camada escondida

SC: sequência de composições

EQM: erro quadrático médio

CC: custo computacional (número de Kflops)

RCA: região compacta de aproximação

NA: número de amostras de entrada-saída para treinamento

Estágio 4: Neste estágio aplicamos as três arquiteturas em um problema de aproximação de funções extremamente simples, que é a aproximação da função tangente hiperbólica:

$$g_7 = 1,716 \operatorname{tgh}\left(\frac{2}{3}x\right).$$

Tabela 6.3 Resultados obtidos no Estágio 4.

	n	SC	EQM	CC
P+CG	1		10^{-8}	378
PPL	1		10^{-6}	790
RNH	1		10^{-18}	22328

Os resultados obtidos são mostrados na Tabela 6.3. Observe que todas as abordagens obtiveram a mesma solução. Embora a rede híbrida tenha obtido um erro quadrático médio muito menor que as outras abordagens, seu custo computacional foi bem superior.

Estágio 5: Neste estágio, tentamos aproximar uma combinação de funções do conjunto de candidatos (Tabela 6.1):

$$g_8(x, y) = \cos(x + y) + e^{-0.5(x-y)^2}.$$

Os melhores resultados obtidos neste estágio encontram-se na Tabela 6.4. Observe que a arquitetura híbrida obteve o melhor resultado, pois as funções que compõem $g_8(\cdot)$ fazem parte do conjunto básico, embora a um custo computacional superior.

Estágio 6: Neste estágio, tentamos aproximar a seguinte função

$$g_9(x, y) = 1,9 \left\{ 1,35 + e^{x-y} \sin[13(x-0,6)^2] \sin(7y) \right\}.$$

O problema de aproximar $g_9(\cdot)$ é uma tarefa bastante custosa, como já observado por HWANG *et al.* (1994), no contexto de PPL. A Tabela 6.5 mostra os resultados obtidos. Novamente, a arquitetura híbrida foi capaz de encontrar uma solução melhor que as outras arquiteturas consideradas, embora a um custo computacional bem mais elevado.

Tabela 6.4 Resultados obtidos no Estágio 5.

	n	SC	EQM	CC
P+CG	12	+	10^{-4}	85012
PPL	2	+	10^{-4}	98242
RNH	2	+	10^{-7}	171244

Tabela 6.5 Resultados obtidos no estágio 6.

	n	SC	EQM	CC
P+CG	20	+	10^{-2}	792870
PPL	5	++++	10^{-2}	223709
RNH	4	+x+	10^{-2}	10891603

Deve-se salientar aqui a existência de dois custos computacionais envolvidos na solução do problema de aproximação. Um é aquele que está sendo medido basicamente pela última coluna (CC) das Tabelas 6.3 a 6.5, referente ao *custo de se obter a solução*. O outro é aquele que está sendo medido basicamente pela primeira coluna (n) das Tabelas 6.3 a 6.5, referente ao número de neurônios empregados na solução, diretamente vinculado ao *custo de se implementar a solução obtida*. O resultado obtido no estágio 6 é particularmente significativo (veja Tabela 6.5) por demonstrar que composição híbrida de funções de ativação pode conduzir a soluções cuja implementação envolve menos recursos computacionais. Nenhum outro resultado da literatura havia resolvido o problema de aproximar $g_9(\cdot)$ com menos de 5 neurônios.

6.6 Conclusão

Neste capítulo, apresentamos uma nova arquitetura híbrida de rede neural. Esta arquitetura permite uma composição aditiva ou multiplicativa em cascata de diferentes funções de ativação, selecionados a partir de uma conjunto finito de candidatos.

Um algoritmo genético foi empregado para encontrar o melhor conjunto de funções de ativação e a melhor seqüência de operadores para um dado problema, enquanto que um algoritmo de gradiente conjugado foi responsável pela determinação dos pesos da rede.

Os resultados obtidos foram bastante promissores em termos de qualidade da solução produzida. Na primeira etapa de simulações, o algoritmo de aprendizado foi capaz de fornecer funções de ativação pertinentes e composições adequadas, ao mesmo tempo que encontrou arquiteturas dedicadas. Na segunda etapa de simulação, onde seu desempenho foi comparado ao de outras arquiteturas, a rede híbrida sempre foi capaz de encontrar soluções melhores, seja

em termos de erro quadrático médio, seja em termos de arquitetura final produzida (embora a um custo computacional superior).

Capítulo 7

Redes *Neurofuzzy* Heterogêneas

Neste capítulo, iremos apresentar mais uma estrutura híbrida de inteligência computacional. Esta estrutura é baseada numa sinergia entre os três principais paradigmas da inteligência computacional: redes neurais artificiais, sistemas *fuzzy* e computação evolutiva. Em linhas gerais, usaremos um algoritmo genético para otimizar parâmetros de uma arquitetura *neurofuzzy*, que pode ser definida como um arquitetura *fuzzy* com a capacidade de aprendizado das redes neurais artificiais. Veremos que o problema com o qual nos defrontaremos é análogo ao apresentado no capítulo anterior.

Recentemente, diversas arquiteturas *neurofuzzy* têm sido propostas na literatura. Estas arquiteturas diferem basicamente no tipo de neurônio utilizado, no tipo de informação processada pela rede e na natureza das conexões. Algumas arquiteturas utilizam neurônios lógicos, processam sinais reais e utilizam pesos sinápticos também reais. Entre estas arquiteturas podemos citar CAMINHAS (1997) e FIGUEIREDO *et al.* (1995). Outras arquiteturas processam sinais *fuzzy* e/ou empregam pesos *fuzzy*, e utilizam neurônios do tipo perceptron com alguma função de ativação não-linear. Este tipo de arquitetura emprega aritmética *fuzzy* para realizar operações de soma e multiplicação e usa o princípio da extensão (PEDRYCZ & GOMIDE, 1998) para computar a saída de um neurônio. Uma visão geral deste tipo de rede *neurofuzzy* pode ser encontrada em BUCKLEY & HAYASHI (1994).

Computação evolutiva também tem sido usada em conjunto com arquiteturas *fuzzy* ou *neurofuzzy*. Algoritmos genéticos podem ser utilizados na definição de parâmetros de sistemas *fuzzy*, como número de funções de pertinência e número de regras (TAKAGI & LEE, 1993), ou mesmo no treinamento de redes *neurofuzzy* (BUCKLEY & HAYASHI, 1994; PEDRYCZ, 1995). Algoritmos genéticos também podem ser

usados para otimizar a própria arquitetura de uma rede *neurofuzzy* (KASABOV & WATTS, 1997).

Neste capítulo, apresentaremos em detalhes a arquitetura *neurofuzzy* proposta por CAMINHAS (1997), e como uma extensão importante junto a esta arquitetura, aplicaremos então um algoritmo genético para definir o melhor conjunto de operadores para modelar conectivos lógicos utilizados nos neurônios desta rede *neurofuzzy*.

7.1 Fundamentos de Sistemas *Fuzzy*

Os conjuntos *fuzzy* foram propostos por ZADEH (1965) como uma alternativa inovadora para formalizar o tratamento de conceitos como imprecisão e incerteza. Na teoria clássica de conjuntos, cada elemento do universo de discurso (domínio ou espaço em que as variáveis associadas ao problema podem assumir valores) apresenta exclusão completa ou então pertinência completa a um determinado conjunto. Na teoria de conjuntos *fuzzy*, a cada elemento do universo de discurso são atribuídos *graus contínuos de pertinência* entre 0 (exclusão completa) e 1 (pertinência completa) a cada conjunto definido no mesmo universo de discurso. Os valores de pertinência expressam os graus com os quais cada elemento é compatível com as propriedades dos conjuntos do universo (PEDRYCZ & GOMIDE, 1998). Formalmente, conjuntos *fuzzy* são definidos como segue:

Definição 7.1 Um conjunto *fuzzy* $\mathfrak{S}$ é caracterizado por uma função de pertinência F mapeando elementos do universo de discurso X para o intervalo unitário $[0, 1]$. Isto é, $F: X \rightarrow [0, 1]$. $\square$

Assim, um conjunto *fuzzy* $\mathfrak{S}$ em X pode ser representado como um conjunto de pares ordenados de um elemento genérico $x \in X$ e seu grau de pertinência: $\mathfrak{S} = \{(F(x)/x) \mid x \in X\}$.

Definição 7.2 Um *sistema fuzzy* é composto por uma base de regras e um mecanismo de inferência *fuzzy*. Um sistema *fuzzy* faz o mapeamento $U \rightarrow V$ de um espaço de entrada para um espaço de saída, onde $U = U_1 \times \dots \times U_n \subset \mathfrak{R}^n$, $V \subset \mathfrak{R}$, e $\times$ é o produto cartesiano. A base de regras é constituída de M regras do tipo:

$$R^{(l)}: \text{se } x_1 \text{ é } F_1^l \text{ e } x_2 \text{ é } F_2^l \text{ e } \dots \text{ e } x_n \text{ é } F_n^l \text{ então } y \text{ é } G^l$$

onde F_i^l ($i = 1, \dots, n$; $l = 1, \dots, M$) e G^l ($l = 1, \dots, M$) são subconjuntos *fuzzy* definidos respectivamente em $U_i \subset \mathfrak{R}$ ($i = 1, \dots, n$) e $V \subset \mathfrak{R}$, com $\mathbf{x} = [x_1 \dots x_n]^T \in U$ e $y \in V$ sendo variáveis lingüísticas de entrada e saída, respectivamente. $\square$

Não detalharemos aqui o mecanismo de inferência *fuzzy*, sendo que uma abordagem aprofundada deste conceito pode ser encontrada em PEDRYCZ & GOMIDE (1998). Observe que a rede *neurofuzzy* que iremos apresentar implementa uma base de regras *fuzzy*, junto com o seu mecanismo de inferência.

7.2 Normas Triangulares

Normas triangulares (PEDRYCZ & GOMIDE, 1998) desempenham um papel fundamental na teoria de conjuntos *fuzzy*. As normas triangulares fornecem modelos genéricos para as operações de união e interseção em conjuntos *fuzzy*, e devem possuir as propriedades de comutatividade, associatividade e monotonicidade. Condições de contorno também devem ser satisfeitas. Portanto, normas triangulares formam classes genéricas de operadores de união e interseção. Formalmente, elas são definidas como segue.

Definição 7.3 Uma *norma triangular* (*t-norma*) é um operador $t: [0, 1]^2 \rightarrow [0, 1]$ satisfazendo as seguintes propriedades:

- Comutatividade: $x \ t \ y = y \ t \ x$
- Associatividade: $x \ t \ (y \ t \ z) = (x \ t \ y) \ t \ z$
- Monotonicidade: Se $x \leq y$ e $w \leq z$, então $x \ t \ w \leq y \ t \ z$
- Condições de contorno: $0 \ t \ x = 0$, $1 \ t \ x = x$. $\square$

Definição 7.4 Uma *s-norma*, também conhecida como *co-norma triangular*, é um operador $s: [0, 1]^2 \rightarrow [0, 1]$ satisfazendo as seguintes propriedades:

- Comutatividade: $x \ s \ y = y \ s \ x$
- Associatividade: $x \ s \ (y \ s \ z) = (x \ s \ y) \ s \ z$
- Monotonicidade: Se $x \leq y$ e $w \leq z$, então $x \ s \ w \leq y \ s \ z$
- Condições de contorno: $x \ s \ 0 = x$, $x \ s \ 1 = 1$. $\square$

Das definições apresentadas acima, pode-se concluir que o operador $\min$ ($\wedge$) é um exemplo de *t-norma*, enquanto que o operador $\max$ ($\vee$) é um exemplo de *s-norma*. A seguir,

apresentamos mais alguns exemplos de normas triangulares, extraídos de PEDRYCZ & GOMIDE (1998).

Exemplo 7.1 Exemplos de t-normas:

- $x t_1 y = \frac{1}{1 + \sqrt[p]{((1-x)/x)^p + ((1-y)/y)^p}}, p > 0$
- $x t_2 y = \max(0, (1+p)(x+y-1) - pxy), p \geq -1$
- $x t_3 y = 1 - \min\left(1, \sqrt[p]{(1-x)^p + (1-y)^p}\right), p > 0$
- $x t_4 y = xy$
- $x t_5 y = \frac{xy}{p + (1-p)(x+y-xy)}, p \geq 0$
- $x t_6 y = \frac{1}{\sqrt[p]{1/x^p + 1/y^p - 1}}$
- $x t_7 y = \sqrt[p]{\max(0, x^p + y^p - 1)}, p > 0$
- $x t_8 y = \frac{xy}{\max(x, y, p)}, p \in [0, 1]$
- $x t_9 y = \log_p \left[1 + \frac{(p^x - 1)(p^y - 1)}{p - 1} \right], p > 0, p \neq 1$
- $x t_{10} y = \frac{1}{1 + \sqrt[p]{((1-x)/x)^p + ((1-y)/y)^p}}, p > 0$
- $x t_{11} y = \begin{cases} x, & \text{se } y = 1 \\ y, & \text{se } x = 1 \\ 0, & \text{caso contrário} \end{cases}$

□

Exemplo 7.2 Exemplos de s-normas:

- $x s_1 y = \frac{1}{1 + \sqrt[p]{(x/(1-x))^p + (y/(1-y))^p}}, p > 0$
- $x s_2 y = \min(1, x + y - pxy), p \geq 0$

- $x s_3 y = \min\left(1, \sqrt[p]{x^p + y^p}\right), p > 0$
- $x s_4 y = x + y - xy$
- $x s_5 y = \frac{x + y - xy - (1-p)xy}{1 - (1-p)xy}, p \geq 0$
- $x s_6 y = 1 - \frac{1}{\sqrt[p]{1/(1-x)^p + 1/(1-y)^p} - 1}$
- $x s_7 y = 1 - \max\left(0, \sqrt[p]{(1-x)^p + (1-y)^p} - 1\right), p > 0$
- $x s_8 y = 1 - \frac{(1-x)(1-y)}{\max((1-x), (1-y), p)}, p \in [0, 1]$
- $x s_9 y = \log_p \left[1 + \frac{(p^{1-x} - 1)(p^{1-y} - 1)}{p - 1} \right], p > 0, p \neq 1$
- $x s_{10} y = \frac{1}{1 - \sqrt[p]{(x/(1-x))^p + (y/(1-y))^p}}, p > 0$
- $x s_{11} y = \begin{cases} x, & \text{se } y = 0 \\ y, & \text{se } x = 0 \\ 1, & \text{caso contrário} \end{cases}$

□

As operações sobre conjuntos podem ser interpretadas como conectivos lógicos. Na lógica proposicional binária, os operadores de interseção e união podem ser identificados com os conectivos conjuntivos (AND) e disjuntivos (OR), respectivamente. Este também é o caso da lógica *fuzzy*, onde t-normas e s-normas são vistas, respectivamente, como conectivos conjuntivos e disjuntivos genéricos.

7.3 Neurônios Lógicos

Os neurônios lógicos utilizados em sistemas *neurofuzzy* podem ser divididos em duas categorias principais (PEDRYCZ & ROCHA, 1993): neurônios de agregação e neurônios de referência. Dentre os neurônios de agregação, podemos citar os neurônios AND e OR (PEDRYCZ & ROCHA, 1993), OR/AND (HIROTA & PEDRYCZ, 1994) e neurônios de limiar

(*threshold*) (PEDRYCZ & GOMIDE, 1998). Como exemplo de neurônios de referência, podemos citar os neurônios de comparação (*matching*), diferença, inclusão e dominância (PEDRYCZ & ROCHA, 1993). Nesta seção, iremos analisar apenas os neurônios de agregação do tipo AND e OR, pois a arquitetura *neurofuzzy* que iremos considerar utiliza apenas estes dois tipos de neurônios.

Sejam $\mathbf{x} = [x_1 \ x_2 \ \dots \ x_n]^T$ e $\mathbf{w} = [w_1 \ w_2 \ \dots \ w_n]^T$ vetores pertencentes ao hipercubo unitário $[0, 1]^n$. O neurônio OR realiza um mapeamento $[0, 1]^n \rightarrow [0, 1]$ descrito por

$$y = \text{OR}(\mathbf{x}; \mathbf{w}) = \text{OR}\{x_1 \text{ AND } w_1, x_2 \text{ AND } w_2, \dots, x_n \text{ AND } w_n\},$$

onde o vetor $\mathbf{w}$ representa os pesos de conexão do neurônio. Em sistemas *fuzzy*, conectivos lógicos são usualmente implementados através de alguma s- ou t-norma. Assim, obtemos a seguinte expressão para o neurônio OR (Figura 7.1(a)):

$$y = \mathbf{S}_{i=1}^n [x_i \ t \ w_i]. \quad (7.1)$$

Para o neurônio AND os operadores OR e AND são invertidos, produzindo a expressão:

$$y = \text{AND}(\mathbf{x}; \mathbf{w}) = \text{AND}\{x_1 \text{ OR } w_1, x_2 \text{ OR } w_2, \dots, x_n \text{ OR } w_n\},$$

ou em notação de normas triangulares (Figura 7.1(b)):

$$y = \mathbf{T}_{i=1}^n [x_i \ s \ w_i]. \quad (7.2)$$

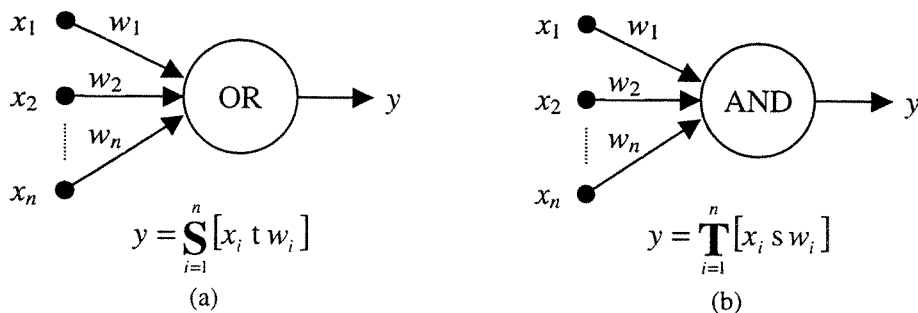


Figura 7.1 Neurônios lógicos: (a) neurônio OR e (b) neurônio AND.

Os neurônios AND e OR realizam operações lógicas “puras” sobre as entradas. O papel dos pesos é determinar o nível de impacto que uma entrada individual pode ter no resultado da agregação. Observe que, quanto maior o valor de uma conexão em um neurônio OR, a entrada correspondente exerce uma influência maior na saída do neurônio. O efeito oposto é observado no neurônio AND: quanto mais próximo um peso for de 1, menor será a influência da respectiva entrada na saída do neurônio.

7.4 A Rede Neurofuzzy AND/OR

A rede *neurofuzzy* AND/OR, descrita em CAMINHAS (1997), encontra-se ilustrada na Figura 7.2. Esta arquitetura foi originalmente proposta para problemas de classificação de padrões. Um problema de classificação de padrões consiste em, dado um conjunto de padrões $\mathbf{x}_p = [x_{p1} \dots x_{pi} \dots x_{pn}]^T$, $p = 1, \dots, P$, e um conjunto de classes $C = \{C_1, C_2, \dots, C_{N_c}\}$ determinar

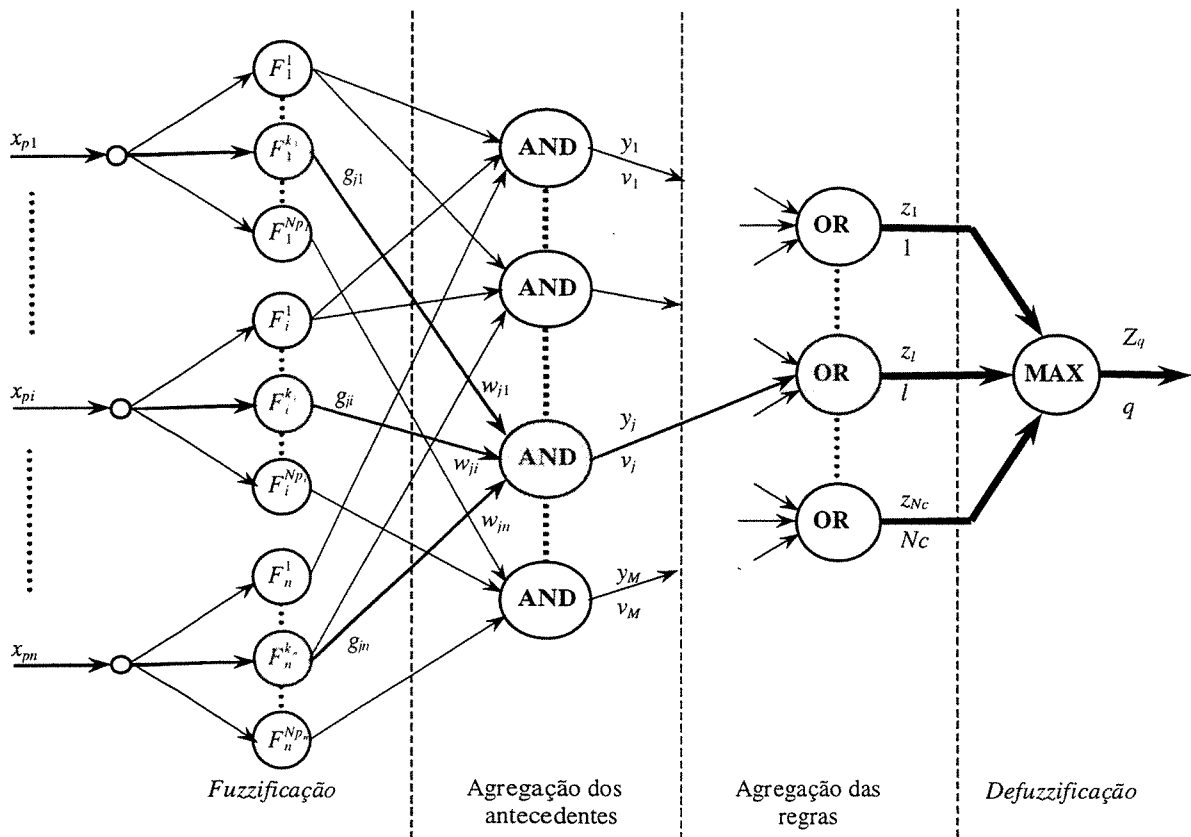


Figura 7.2 Rede neurofuzzy AND/OR.

a classe a que cada padrão pertence. Observe que a rede da Figura 7.2 implementa uma base de regras *fuzzy* do tipo (veja fluxo em negrito e hachurado na Figura 7.2):

“Se $(x_{p1} \text{ é } F_1^{k_1} \text{ com incerteza } w_{j1}) \dots \text{ e } (x_{pi} \text{ é } F_i^{k_i} \text{ com incerteza } w_{ji}) \dots \text{ e } (x_{pn} \text{ é } F_n^{k_n} \text{ com incerteza } w_{jn})$ então x_p pertence à classe l com certeza v_j .”

Com relação à Figura 7.2, temos:

- Np_i é o número de subconjuntos *fuzzy* da entrada i ;
- j é o índice que indexa um neurônio do tipo AND. Da forma como foi estruturada a rede, o índice j é determinado a partir dos índices dos subconjuntos *fuzzy* das coordenadas de entrada k_i , da seguinte forma

$$j = f(\mathbf{k}) = k_n + \sum_{i=2}^n (k_{n-i+1} - 1) \left(\prod_{j=1}^{i-1} Np_{n+1-j} \right) \quad (7.3)$$

sendo $\mathbf{k}$ o vetor que contém os índices dos subconjuntos *fuzzy*, ou seja, $\mathbf{k} = [k_1 \dots k_i \dots k_n]^T$. Para exemplificar, suponha que $x_{p1} \text{ é } F_1^1$, $x_{p2} \text{ é } F_2^3$, $x_{p3} \text{ é } F_3^2$ e $Np_1 = Np_2 = Np_3 = 3$. Neste caso tem-se $k_1 = 1$, $k_2 = 3$ e $k_3 = 2$ e

$$j = 2 + (k_2 - 1)Np_3 + (k_1 - 1)Np_3Np_2 = 2 + (3 - 1) \times 3 + (1 - 1) \times (3 \times 3) = 8.$$

- x_{p1} a x_{pn} são as n coordenadas do padrão x_p (entradas da rede);
- g_{ji} é o grau de pertinência da variável x_{pi} ao subconjunto *fuzzy* $F_i^{k_i}$ na regra j ;
- w_{ji} é o peso da conexão da entrada i para o neurônio AND indexado por j ;
- y_j é o valor da saída do neurônio AND indexado por j , obtido usando a Equação (7.2);
- v_j é o peso da conexão do neurônio AND indexado por j ao neurônio OR indexado por l ;
- z_l é o valor da saída do neurônio OR indexado por l , que corresponde ao grau de pertinência do padrão x_p à classe l , obtido da Equação (7.1).
- Z_q é o valor que corresponde ao maior grau de pertinência e q é a classe a que pertence o padrão, obtida a partir da *defuzzificação*.

A rede *neurofuzzy* AND/OR é composta de 4 camadas de neurônios, detalhadas a seguir:

- a primeira camada é responsável pela *fuzzificação* das entradas, ou seja, é responsável pela transformação das coordenadas de entrada em graus de pertinência de subconjuntos *fuzzy*. Assim, cada neurônio nesta camada representa uma função de pertinência *fuzzy*.
- a segunda camada é composta de neurônios AND, e é responsável pela agregação dos antecedentes das regras *fuzzy*.
- a terceira camada é composta de neurônios OR, e é responsável pela agregação das regras.
- a quarta camada é composta apenas de um neurônio, responsável pela *defuzzificação*. O critério de *defuzzificação* adotado é o do máximo.

7.5 Algoritmo de Treinamento da Rede *Neurofuzzy* AND/OR

O algoritmo empregado no treinamento da rede *neurofuzzy* AND/OR é do tipo competitivo, similar ao LVQ (*Learning Vector Quantization*), não necessitando de derivadas para ajustar os pesos das conexões. O algoritmo de treinamento é descrito a seguir:

1. gere as funções de pertinência;
2. gere as conexões da rede e inicialize os pesos w_{ji} e v_j , usando o procedimento descrito na Seção 7.5.2;
3. processo iterativo:
 - 3.1. apresente um padrão p à rede (normalmente escolhido de forma aleatória);
 - 3.2. determine os neurônios AND e OR ativos;
 - 3.3. efetue a *fuzzificação*;
 - 3.4. determine a classe vencedora;
 - 3.5. atualize os pesos w_{ji} e v_j com o objetivo de reduzir o erro atual produzido pela rede;
 - 3.6. teste a condição de parada (todos os padrões classificados corretamente ou número de iterações alcançado):
 - se é satisfeita, vá para o passo 4;
 - se não é satisfeita, vá para o passo 3.1;
4. fim.

A seguir, detalharemos cada um dos passos do algoritmo acima.

7.5.1 Geração das Funções de Pertinência

Dois tipos de função de pertinência serão discutidos:

- tipo 1: triangulares normais, uniformemente distribuídas;
- tipo 2: triangulares normais, não uniformemente distribuídas.

Para os dois tipos, as funções de pertinência são complementares, ou seja, a soma de duas funções sucessivas é 1.

7.5.1.1 Geração de Funções de Pertinência do Tipo 1

Neste caso têm-se funções de pertinência como as representadas na Figura 7.3. As funções de pertinência são definidas pelos parâmetros x_{imin} e x_{imax} . O centro da função de pertinência indexada por r é a_{ir} , que corresponde ao ponto onde a função possui o valor máximo.

Da Figura 7.3 tem-se

$$\Delta_i = \frac{x_{imax} - x_{imin}}{Np_i - 1}$$

Com isto, o cálculo de uma função de pertinência r , $2 \leq r \leq Np_i$ é feito da seguinte forma:

$$\mu_{F_i'}(x_{pi}) = \begin{cases} \frac{(x_{pi} - a_{ir})}{\Delta_i} + 1, & a_{ir-1} \leq x_{pi} \leq a_{ir} \\ -\frac{(x_{pi} - a_{ir})}{\Delta_i} + 1, & a_{ir} \leq x_{pi} \leq a_{ir+1} \\ 0, & \text{caso contrário} \end{cases}$$

7.5.1.2 Geração de Funções de Pertinência do Tipo 2

Em alguns casos, pode não ser interessante o uso de funções de pertinência do tipo 1, por exemplo em situações onde há concentração de dados em uma região e dispersão em

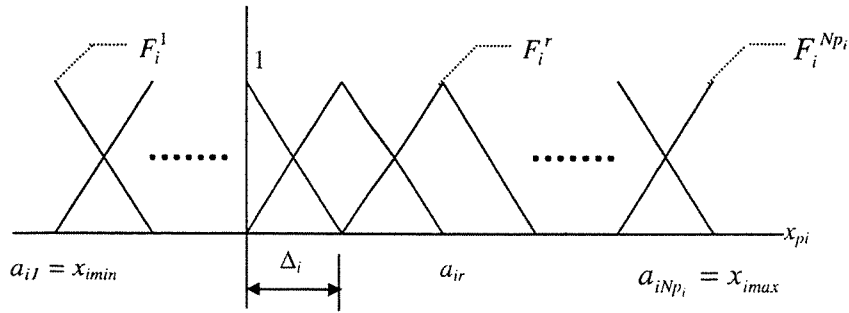


Figura 7.3 Funções de pertinência do tipo 1.

outras. Sendo assim, pode ser mais interessante o uso de funções de pertinência não uniformemente distribuídas.

Um método para gerar automaticamente as funções de pertinência é usar um algoritmo de agrupamento dos padrões de entrada (*clustering*), onde os centros dos grupos gerados correspondem ao centro das funções de pertinência, a_{ir} (Figura 7.4). O algoritmo de *clustering* usado neste trabalho é baseado na rede neural auto-organizada de Kohonen (KOHONEN, 1989), ilustrada na Figura 7.5. São utilizadas n redes neurais auto-organizadas, uma para cada coordenada do padrão. Os pesos das redes correspondem assim aos valores dos centros das funções de pertinência. O número de neurônios (Np_i^0), que é um dado definido a priori, corresponde ao número de funções de pertinência para a coordenada i ($i = 1, \dots, n$). Ao final do treinamento da rede, este número pode ser reduzido através de um algoritmo de poda (DE CASTRO & VON ZUBEN, 1999).

A seguir, apresentamos o algoritmo para treinamento da rede neural auto-organizada usada para geração automática das funções de pertinência:

1. inicializações:

- pesos a_{ir} :

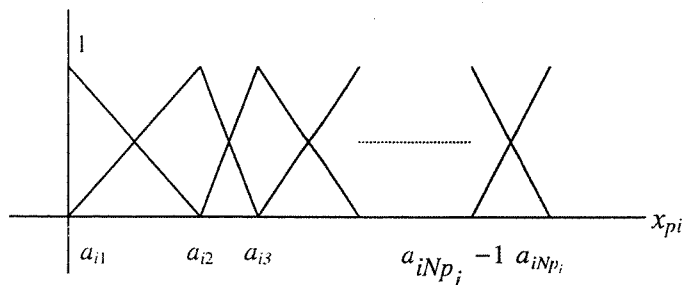


Figura 7.4 Funções de pertinência do tipo 2.

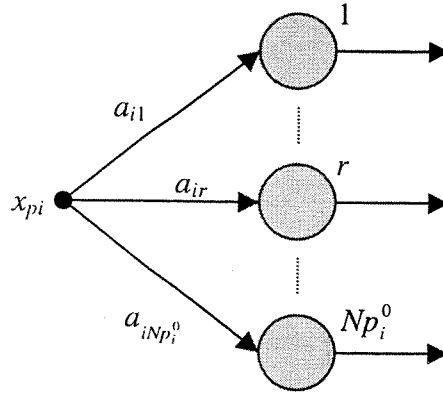


Figura 7.5 Rede neural auto-organizada para determinação dos centros das funções de pertinência.

$$a_{i1} = x_{imin}$$

$$a_{ir} = a_{i(r-1)} + \Delta_i, \quad \text{para } r = 2, 3, \dots, Np_i^0$$

- índice de desempenho: $id_i(r) = 0$, para $r = 1, 2, \dots, Np_i^0$, ou seja, as funções de pertinência iniciais são do tipo 1.

2. processo iterativo:

- 2.1. apresente um padrão p à rede e atualize o peso da conexão do neurônio vencedor da seguinte forma

$$a_{iL}(p+1) = a_{iL}(p) + \alpha(p)[x_{pi} - a_{iL}(p)]$$

onde L é o índice do neurônio vencedor, que é aquele cujo peso da conexão possui o valor mais próximo de x_{pi} , ou seja,

$$L = \arg \left\{ \min_r |x_{pi} - a_{ir}| \right\}$$

- 2.2. reduzir o passo $\alpha(p)$;

- 2.3. atualize o índice de desempenho do neurônio vencedor, fazendo:

$$id_i(L) = id_i(L) + 1$$

- 2.4. verifique o teste de parada (por exemplo, se nenhuma mudança significativa for percebida nos pesos da rede):

- se não é satisfeito, volte ao passo 2.1;

- se é satisfeito, vá para o passo 3;
- elimine todos os neurônios cujo valor de id_i seja menor que um limiar N (inteiro positivo). Seja Nne_i o número de neurônios eliminados para a coordenada i . O número de subconjuntos *fuzzy* para a coordenada i é então dado por

$$Np_i = Np_i^0 - Nne_i.$$

- fim.

7.5.2 Geração das Conexões da Rede e Inicialização dos Pesos w_{ji} e v_j

Neste passo, são estabelecidas as conexões entre os neurônios AND e os neurônios OR. Este passo equivale à geração das regras *se-então*. O algoritmo apresentado aqui é uma adaptação da proposta original de NOZAKI *et al.* (1996).

- Para $j = 1, \dots, M$, faça

- 1.1. geração das conexões: compute β_l para $l = 1, 2, \dots, Nc$

$$\beta_l = \sum_{x_p \in \text{classe } l} \mathbf{T}^n x_{ji}, \quad (7.4)$$

onde β_l é a soma das compatibilidades dos padrões em cada classe aos antecedentes (ISHIBUCHI *et al.*, 1992). Observe que cada neurônio lógico AND determina uma partição no espaço dos padrões, de modo que, quanto maior a presença de padrões pertencentes à classe l na partição definida pelo neurônio AND j , maior o valor de β_l . Determine então a classe q tal que

$$q = \arg \left\{ \max_l \{ \beta_1, \dots, \beta_l, \dots, \beta_{Nc} \} \right\}. \quad (7.5)$$

Isto define a conexão do neurônio AND indexado por j ao neurônio OR indexado por q . Se $\beta_q = 0$, ou se existir mais de um máximo para β_q , então o neurônio AND j não se conecta a nenhuma classe de saída.

- 1.2. Inicialize os pesos w_{ij} e v_j fazendo:

$$w_{ji} = 0, \forall i = 1, 2, \dots, n \quad \text{e} \quad j = 1, 2, \dots, M,$$

$$v_j = \frac{\beta_q - \sum_{i \neq q} \beta_i / (Nc - 1)}{\sum \beta_i}. \quad (7.6)$$

Na Equação (7.6), quanto maior a diferença entre β_q e as demais classes, significa que há uma predominância de padrões pertencentes à classe q na partição determinada pelo neurônio AND j , e mais próximo o valor de v_j é de 1. Quando existirem duas ou mais classes dominantes na partição do espaço definida pelo neurônio AND j , o valor de v_j é próximo de 0. Lembre que v_j representa o grau de certeza que se tem a respeito da entrada correspondente do neurônio OR. A inicialização dos pesos v_j segundo a expressão (7.6) reduz o tempo de treinamento, pois melhora a superfície de separação inicial (CAMINHAS, 1997).

Exemplo 7.3 Considere o problema de classificação de padrões ilustrado na Figura 7.6(a), onde os padrões ilustrados por círculos correspondem à classe 1 e os padrões representados por triângulos correspondem à classe 2. Suponha que empregamos duas funções de pertinência do tipo 1 para cada coordenada do espaço dos padrões. Observe as partições formadas pelas funções de pertinência; cada partição corresponde a um neurônio AND. Os rótulos em cada partição na Figura 7.6(a) correspondem aos índices dos neurônios AND obtidos através da Equação (7.3). Usando as Equações (7.4), (7.5) e (7.6), obtemos a seguinte tabela:

Neurônio AND	β_1	β_2	Neurônio OR conectado	Peso v_j
1	49,7608	11,6342	1	0,6210
2	18,5939	18,2169	1	0,1024
3	23,0692	26,7551	2	0,0740
4	8,5760	43,3938	2	0,6700

Segue uma explicação intuitiva para o conjunto de valores apresentados na tabela acima: na partição 1 da Figura 7.6(a) vemos claramente que há uma predominância de padrões pertencentes à classe 1, daí o valor elevado de β_1 ; portanto o neurônio AND₁ conecta-se ao neurônio OR₁, com peso de conexão 0,6201. Na partição 2, não há predominância de nenhuma das classes, daí os valores próximos de β_1 e β_2 ; como $\beta_1 > \beta_2$ o neurônio AND₂

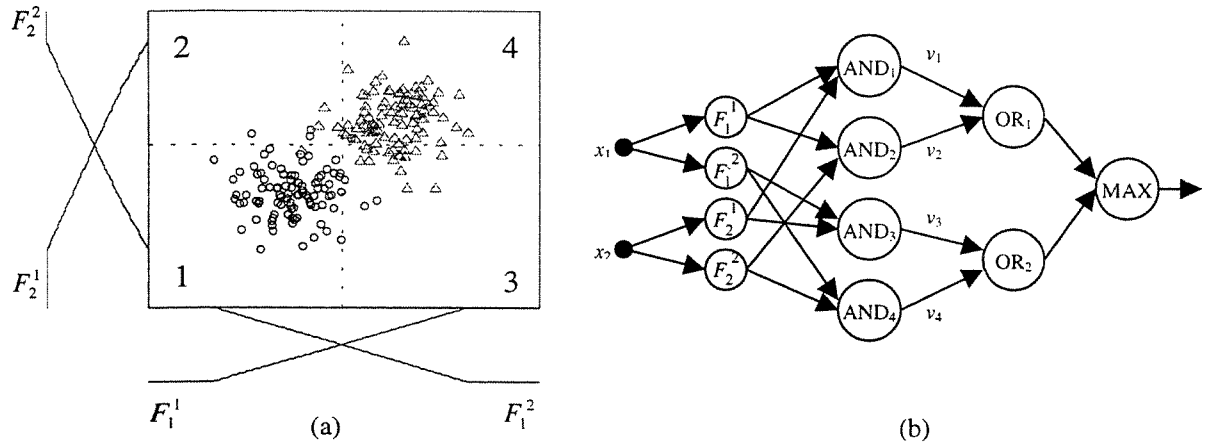


Figura 7.6 (a) Um problema de classificação de padrões. (b) Rede *neurofuzzy* AND/OR resultante.

conecta-se ao neurônio OR_1 , com peso de conexão 0,1024. Este valor baixo para o peso de conexão indica que temos pouca certeza na classificação de um padrão contido na partição 2 do espaço de padrões. E assim por diante. Na Figura 7.6(b) apresentamos a configuração da rede *neurofuzzy* AND/OR para este problema. $\square$

7.5.3 Determinação dos Neurônios AND e OR Ativos

Considerando funções de pertinência como mostradas nas Figuras 7.3 e 7.4, para cada padrão apresentado os respectivos graus de pertinência serão diferentes de zero para no máximo dois subconjuntos *fuzzy*. Estes serão definidos como subconjuntos ativos, que por sua vez definem os neurônios AND ativos. Com isto, dos M neurônios AND, no máximo 2^n estarão ativos. Estes neurônios são determinados da seguinte maneira:

1. dado um padrão $\mathbf{x}_p = [x_{p1} \cdots x_{pi} \cdots x_{pn}]^T$, sejam $\mathbf{k}^1 = [k_1^1 \cdots k_i^1 \cdots k_n^1]^T$ o vetor que contém os índices da primeira função de pertinência diferente de zero para cada coordenada do padrão e $\mathbf{k}^2 = [k_1^2 \cdots k_i^2 \cdots k_n^2]^T$ um vetor tal que:

$$k_i^2 = \begin{cases} k_i^1 + 1, & \text{se } \mu_{k_i^1}(x_{pi}) \neq 1 \\ k_i^1, & \text{caso contrário} \end{cases},$$

onde $\mu_{k_i^1}(x_{pi})$ representa o grau de pertinência de x_{pi} ao subconjunto *fuzzy* indexado por k_i^1 . O número de neurônios AND ativo, Na , é igual a 2^{Pa} , que é no máximo igual a 2^n ,

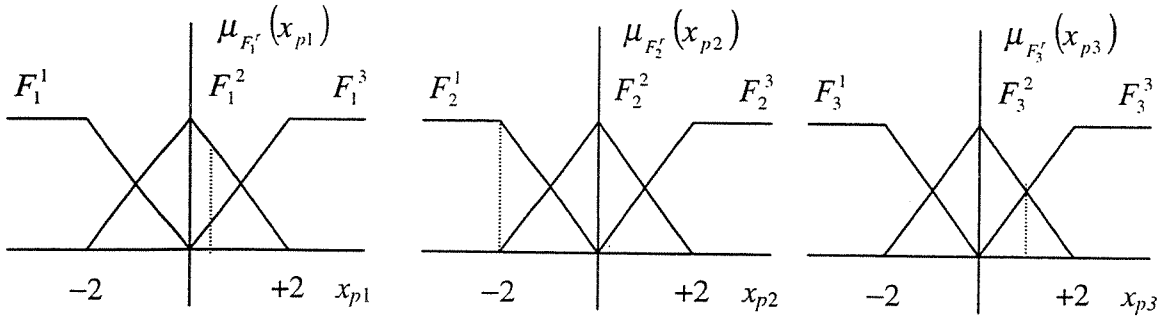


Figura 7.7 Subconjuntos *fuzzy* utilizados no Exemplo 7.4.

sendo P_a o número de elementos tal que $k_i^1 \neq k_i^2$, para $i = 1, 2, \dots, n$.

2. Os neurônios AND ativos podem ser determinados a partir da combinação dos elementos dos vetores $\mathbf{k}^1$ e $\mathbf{k}^2$, como ilustrado no seguinte exemplo, retirado de CAMINHAS (1997).

Exemplo 7.4 Suponha uma rede com 3 entradas definidas no intervalo $[-2, +2]$ e considere 3 subconjuntos *fuzzy* para cada entrada (Figura 7.7). Para um determinado padrão p de coordenadas $\mathbf{x}_p = [0,5 \ -2 \ 1]^T$, tem-se: para $x_{p1} = 0,5$ os subconjuntos *fuzzy* ativos são F_1^2 e F_1^3 , para $x_{p2} = -2$ somente o subconjunto *fuzzy* F_2^1 está ativo, e para $x_{p3} = 1$ os subconjuntos ativos são F_3^2 e F_3^3 , conforme ilustrado na Figura 7.7. Os vetores $\mathbf{k}^1$ e $\mathbf{k}^2$ serão $\mathbf{k}^1 = [2 \ 1 \ 2]^T$ e $\mathbf{k}^2 = [3 \ 1 \ 3]^T$. Combinando $\mathbf{k}^1$ e $\mathbf{k}^2$ chega-se aos seguintes vetores: $[2 \ 1 \ 2]^T$, $[2 \ 1 \ 3]^T$, $[3 \ 1 \ 2]^T$ e $[3 \ 1 \ 3]^T$, que, a partir da Equação (7.3), permitem identificar os neurônios AND ativos indexados por j como 11, 12, 20 e 21. $\square$

No exemplo acima, observe que dos 27 neurônios AND, somente $N_a = 2^{P_a} = 2^2 = 4$ são ativos. A vantagem do emprego desse passo do algoritmo é que somente os neurônios AND ativos serão considerados para efeito de cálculo nos passos seguintes.

Seja $\mathbf{L}^*$ o conjunto dos neurônios AND ativos. Este conjunto e as conexões estabelecidas no passo 7.5.2 determinam o conjunto $\mathbf{Q}^*$ que indexa neurônios OR ativos.

7.5.4 Fuzzificação

Nesta etapa, são calculados os graus de pertinência das coordenadas de entrada, somente para os subconjuntos ativos definidos em $\mathbf{k}^1$. No caso em que $k_i^1 \neq k_i^2$ calcula-se também $\mu_{k_i^2}(x_{pi})$. Como as funções de pertinência são complementares, tem-se

$$\mu_{k_i^2}(x_{pi}) = 1 - \mu_{k_i^1}(x_{pi}).$$

Observe que é necessário calcular apenas $2n$ graus de pertinência.

7.5.5 Determinação da Classe Vencedora

Para determinar a classe vencedora faça

1. Compute $z_l, \forall l \in Q^*$ (ou seja, somente para os neurônios OR ativos)

$$z_l = \mathbf{S}_{j \in L^*} (v_j \text{ t } y_j),$$

onde

$$y_j = \mathbf{T}_{i=1}^n (w_{ji} \text{ s } g_{ji}).$$

2. *Defuzzificação* para determinação da classe vencedora: o índice da classe vencedora q , é determinado usando o operador max como critério:

$$Z_q = \max_{l \in Q^*} \{z_l\}$$

7.5.6 Atualização dos Pesos

Os pesos das conexões são atualizados segundo uma estratégia de punição e recompensa. Se um padrão é classificado corretamente, então os pesos mais significativos para esta classificação são aumentados; caso contrário, são reduzidos. Os pesos w_{ij} e v_j são então atualizados da seguinte forma:

Seja q o índice da classe vencedora, determinada no passo 7.5.5.

Se q é o índice da classe correta, então faça

$$v_j(k+1) = v_j(k) + \alpha_1[1 - v_j(k)] \quad (7.7)$$

$$w_{ji}(k+1) = w_{ji}(k) + \alpha_2[1 - w_{ji}(k)] \quad (7.8)$$

senão faça

$$v_j(k+1) = v_j(k) - \alpha_3 v_j(k) \quad (7.9)$$

$$w_{ji}(k+1) = w_{ji}(k) - \alpha_4 w_{ji}(k) \quad (7.10)$$

sendo I e J os indexadores das conexões mais significativas para a classificação, os quais são determinados da seguinte forma

$$J = \arg \left\{ \max_j \left\{ \left[v_j \text{ t } y_j \right] \forall j \in L^* \text{ e } x_p \in \text{classe } q \right\} \right\}$$

$$I = \arg \left\{ \min_i \left\{ \left[x_{ji} \text{ s } w_{ji} \right] \forall i = 1, 2, \dots, n \text{ e } j = J \right\} \right\}$$

Para as Equações (7.7) a (7.10), α_1 , α_2 , α_3 e α_4 são constantes de aprendizado tais que $0 < \alpha_1 < \alpha_3 < 1$ e $0 < \alpha_2 < \alpha_4 < 1$.

7.6 Redes Neurofuzzy Heterogêneas

A arquitetura *neurofuzzy* apresentada na seção anterior, baseada nos resultados de CAMINHAS (1997), possui as seguintes vantagens:

- geração automática da topologia da rede;
- estratégia de aprendizado competitiva, não havendo necessidade de cálculo de derivadas, apresentando por isso iterações muito rápidas;
- flexibilidade quanto à utilização de diversas t e s-normas;
- possibilidade de extração de regras diretamente da topologia da rede.

Apesar da grande flexibilidade apresentada por esta arquitetura, a escolha adequada de t-normas e s-normas a serem usadas nos neurônios lógicos fica a cargo do projetista da rede. A escolha é feita através de um processo de tentativa e erro e/ou da própria experiência anterior do projetista com problemas semelhantes. Usualmente, em um sistema *fuzzy*, uma única t-norma e uma única s-norma é utilizada para modelar conjunções e disjunções. As escolhas mais usuais para t-normas são min e produto e para s-normas, max e soma probabilística.

No contexto da arquitetura *neurofuzzy* AND/OR, usualmente é empregada uma única t-norma e uma única s-norma em todos os neurônios lógicos da rede. Observe que a t-norma e a s-norma escolhidas influenciam na definição da superfície de separação final obtida: CAMINHAS (1997) mostra alguns exemplos dessa influência em problemas didáticos de classificação de padrões. Daqui em diante, denominaremos uma rede *neurofuzzy* AND/OR com uma única t-norma e uma única s-norma de *rede neurofuzzy homogênea*.

Nesta seção, propomos uma alternativa para a seleção automática de normas triangulares a serem usadas em redes *neurofuzzy* AND/OR. Usaremos um algoritmo genético para escolher, a partir de um conjunto finito de normas, as normas triangulares mais adequadas para cada um dos neurônios da rede *neurofuzzy* AND/OR. Observe que, diferente dos sistemas *fuzzy* tradicionais, esta abordagem permite que cada neurônio lógico da Figura 7.2, ou equivalentemente, cada regra de uma base de regras *fuzzy* use um par específico de t-normas e s-normas. Denominamos estas redes *neurofuzzy* com diferentes t-normas e s-normas de *redes neurofuzzy heterogêneas* (IYODA *et al.*, 1999).

Nosso objetivo é propor um método eficiente e automático para escolha de t-normas e s-normas para sistemas *fuzzy*, que substitua o processo de tentativa e erro, e tal que as estruturas geradas sejam adaptadas a um determinado problema, apresentando portanto um melhor desempenho para uma mesma dimensão de rede *neurofuzzy*.

7.6.1 A Abordagem Evolutiva

Iremos descrever agora o algoritmo genético utilizado na seleção de normas triangulares para uma rede *neurofuzzy* heterogênea. Como explicado anteriormente, o algoritmo genético será responsável pela escolha da t-norma e da s-norma a ser usada em cada um dos neurônios lógicos de uma rede *neurofuzzy* AND/OR. As normas triangulares serão escolhidas de um conjunto finito de normas candidatas, apresentado na Tabela 7.1. Note que esta tabela contém as mesmas normas apresentadas nos Exemplos 7.1 e 7.2, mas com parâmetros p escolhidos arbitrariamente.

O problema aqui é completamente análogo ao problema apresentado no capítulo 6 – lá, precisávamos escolher um conjunto de funções de ativação a partir de um conjunto finito de funções candidatas; aqui temos de escolher um conjunto de normas triangulares a partir de conjunto finito de normas candidatas.

Usaremos o algoritmo genético simples, apresentado no Capítulo 3. A seguir, detalharemos cada um dos passos do algoritmo.

7.6.1.1 Codificação

No algoritmo implementado, cada indivíduo de uma população representa uma rede *neurofuzzy* heterogênea. Uma rede *neurofuzzy* é codificada em um cromossomo da seguinte forma: cada neurônio lógico é codificado em dois genes, o primeiro representando a t-norma a

ser usada pelo neurônio e o segundo a s-norma. Cada gene pode assumir um valor inteiro no intervalo [0, 10], representando as normas apresentadas na Tabela 7.1.

7.6.1.2 Avaliação dos Indivíduos

O *fitness* de um indivíduo é calculado da seguinte forma: um indivíduo é treinado usando o algoritmo apresentado na Seção 7.5 um número pré-especificado de épocas e o

Tabela 7.1 t-normas e s-normas selecionadas pelo GA.

t-normas	s-normas
$x \ t_0 \ y = \min(x, y)$	$x \ s_0 \ y = \max(x, y)$
$x \ t_1 \ y = \frac{1}{1 + \sqrt{((1-x)/x)^2 + ((1-y)/y)^2}}$	$x \ s_1 \ y = \frac{1}{1 + \sqrt{((x/1)-x)^2 + ((y/1)-y)^2}}$
$x \ t_2 \ y = \max(0, x + y - 1)$	$x \ s_2 \ y = \min(1, x + y)$
$x \ t_3 \ y = 1 - \min(1, \sqrt{(1-x)^2 + (1-y)^2})$	$x \ s_3 \ y = \min(1, \sqrt{x^2 + y^2})$
$x \ t_4 \ y = xy$	$x \ s_4 \ y = x + y - xy$
$x \ t_5 \ y = \frac{xy}{x + y - xy}$	$x \ s_5 \ y = \frac{x + y - 2xy}{1 - xy}$
$x \ t_6 \ y = \frac{1}{\sqrt{1/x^2 + 1/y^2} - 1}$	$x \ s_6 \ y = 1 - \frac{1}{\sqrt{1/(1-x)^2 + 1/(1-y)^2} - 1}$
$x \ t_7 \ y = \sqrt{\max(0, x^2 + y^2 - 1)}$	$x \ s_7 \ y = 1 - \max(0, \sqrt{(1-x)^2 + (1-y)^2} - 1)$
$x \ t_8 \ y = \frac{xy}{\max(x, y, 0)}$	$x \ s_8 \ y = 1 - \frac{(1-x)(1-y)}{\max((1-x), (1-y), 0)}$
$x \ t_9 \ y = \log_{10} \left[1 + \frac{(10^x - 1)(10^y - 1)}{9} \right]$	$x \ s_9 \ y = \log_{10} \left[1 + \frac{(10^{1-x} - 1)(10^{1-y} - 1)}{9} \right]$
$x \ t_{10} \ y = \begin{cases} x, & \text{se } y = 1 \\ y, & \text{se } x = 1 \\ 0, & \text{caso contrário} \end{cases}$	$x \ s_{10} \ y = \begin{cases} x, & \text{se } y = 0 \\ y, & \text{se } x = 0 \\ 1, & \text{caso contrário} \end{cases}$

fitness é então dado por

$$fitness = \frac{N_{pcc}}{m}$$

onde N_{pcc} é o número de padrões corretamente classificados após o treinamento e m é o número total de padrões.

7.6.1.3 Operador de Seleção

O operador de seleção utilizado foi o *roulette wheel*, onde a probabilidade de um indivíduo passar para a próxima geração é diretamente proporcional ao seu *fitness*.

7.6.1.4 Crossover e Mutação

O operador de *crossover* utilizado foi o *crossover* uniforme, e o operador de mutação foi o operador de mutação usual, onde simplesmente troca-se o valor de um gene escolhido para mutação por outro valor, de forma aleatória (MICHALEWICZ, 1996).

7.6.2 Problemas de Teste

Nesta seção, apresentaremos os problemas utilizados para testar o desempenho da rede *neurofuzzy* heterogênea. Foram selecionados quatro problemas de teste, descritos abaixo:

- Classes: este problema é composto de 5 classes não disjuntas, geradas artificialmente, sendo cada amostra composta de dois atributos. As classes foram geradas com médias (0, 0), (-1, 1), (1, 1), (-1, -1) e (1, -1) com distribuição normal de variância 1. Foram geradas 40 amostras para cada classe. Para uma ilustração das classes, veja Figura 7.8(a),

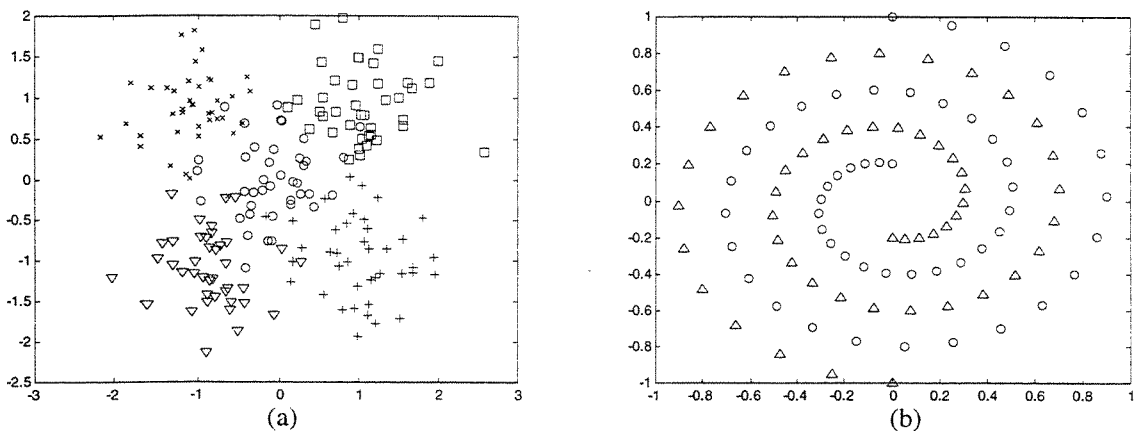


Figura 7.8 Ilustração dos problemas classes (a) e espirais (b)

onde os pontos associados a cada classe são plotados usando símbolos distintos.

- Espirais: este problema é composto de duas classes, sendo que cada amostra é composta de dois atributos. As classes estão dispostas como apresentado na Figura 7.8(b). Foram geradas 50 amostras para cada classe em coordenadas polares, de modo que os atributos são o raio e ângulo de cada amostra.
- Íris: este é um dos problemas mais utilizados na literatura para avaliar o desempenho de sistemas para classificação de padrões. Este problema é composto de três classes com 50 amostras cada, onde a classe se refere a uma das espécies de planta: *Iris setosa*, *Iris versicolor* ou *Iris virginica*. Cada amostra possui quatro atributos, todos medidos em centímetros: comprimento da sépala, largura da sépala, comprimento da pétala e largura da pétala.
- Tiróide: este problema consiste em tentar predizer se paciente apresenta hipertireoidismo, hipotireoidismo ou eutireoidismo (tiróide normal). Cada amostra é composta de cinco atributos (para uma descrição de cada atributo veja COOMANS *et al.* (1983)). As classes são distribuídas da seguinte forma: 150 amostras para a classe normal, 35 para a classe hiper e 30 para a classe hipo.

Os dados dos problemas Iris e Tiróide podem ser encontrados em BLAKE & MERZ (1998). Veja Tabela 7.2 para um resumo dos problemas acima.

Foram utilizados cinco algoritmos para resolver os problemas acima

1. Mapa auto-organizado de Kohonen (SOM) (KOHONEN, 1989).
2. *Learning vector quantization* (LVQ) (KOHONEN, 1989).
3. Rede *neurofuzzy* AND/OR homogênea, com mínimo como t-norma e máximo como s-norma (RNF₁).

Tabela 7.2 Problemas utilizados para avaliar o desempenho da rede *neurofuzzy* heterogênea.

	Número de amostras	Número de classes	Número de atributos
Classes	200	5	2
Espiral	100	2	2
Íris	150	3	4
Tiróide	215	3	5

4. Rede *neurofuzzy* AND/OR homogênea, com produto como t-norma e soma probabilística como s-norma (RNF₂).
5. Rede *neurofuzzy* heterogênea (RNFH), com pares de t-normas e s-normas selecionados através de GA, de acordo com a Seção 7.6.1.

Nas simulações envolvendo a rede *neurofuzzy* heterogênea, foram utilizadas funções de pertinência do tipo 2, apresentadas na Seção 7.5.1. Na Tabela 7.3, apresentamos a quantidade de neurônios AND que se conectam a neurônios OR após o passo 7.5.2 do algoritmo de treinamento (camada de agregação de antecedentes para a camada de agregação de conseqüentes – veja a Figura 7.2). Note que o número de neurônios OR é igual ao número de classes do problema. Observe também que, dos dados desta tabela, podemos calcular o tamanho dos cromossomos em cada problema. Por exemplo, para o problema classes cada cromossomo é um vetor de $(30 + 5) \times 2 = 70$ elementos.

7.6.3 Resultados de Simulações

Nesta seção, apresentaremos os resultados obtidos de simulações computacionais. A Tabela 7.4 apresenta o percentual de amostras corretamente classificadas obtido a partir de cada um dos algoritmos em cada um dos problemas citados na Seção 7.6.2. Como esperado, a rede *neurofuzzy* heterogênea obteve o melhor desempenho para todos os problemas testados. Ou seja, o algoritmo genético foi capaz de encontrar uma configuração de normas triangulares capaz de melhorar o desempenho da rede *neurofuzzy*. O único problema onde o algoritmo genético não foi capaz de melhorar o desempenho da rede foi o problema das duas espirais.

A Figura 7.9 apresenta a *fitness* (taxa de classificação correta) do melhor indivíduo da população ao longo das gerações para cada um dos problemas testados. Também é mostrado o

Tabela 7.3 Número de neurônios AND/OR conectados. O número de neurônios OR é igual ao número de classes.

	AND	OR
Classes	30	5
Espirais	29	2
Íris	151	3
Tiróide	144	3

Tabela 7.4 Percentual (%) de classificações corretas de cada estratégia para os problemas testados.

	SOM	LVQ	RNF ₁	RNF ₂	RNFH
Classes	89,0	91,0	93,0	91,0	94,5
Espiraais	57,0	91,0	94,0	95,0	95,0
Íris	89,0	93,0	98,7	98,7	99,3
Tiróide	88,8	91,6	95,8	96,3	98,8

valor médio do *fitness* da população em cada geração.

Na Figura 7.10, a título de ilustração, apresentamos as fronteiras de decisão obtidas pelas arquiteturas RNF₁, RNF₂ e RNFH para o problema Classes.

O gráfico de barras da Figura 7.11 mostra uma análise estatística das t-normas e s-

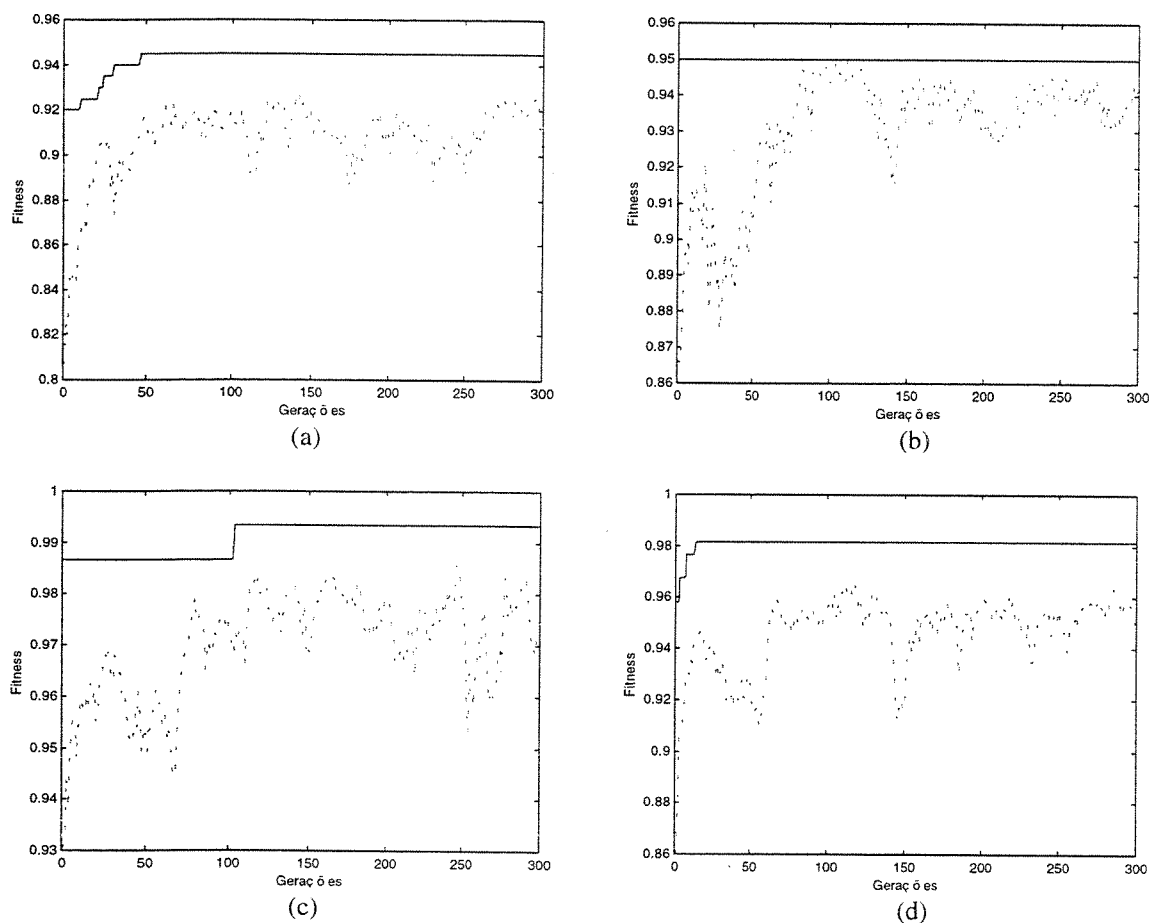


Figura 7.9 Evolução do melhor indivíduo (— linha cheia) e do valor médio do *fitness* da população (---- linha tracejada) ao longo das gerações. Problemas: (a) Classes, (b) Espiraais, (c) Íris e (d) Tiróide.

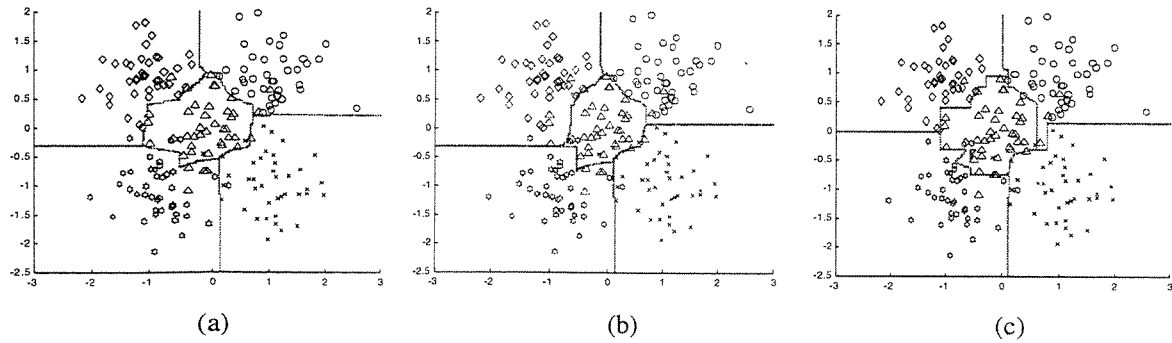


Figura 7.10 Fronteiras de decisão obtidas para o problema Classes para as arquiteturas: (a) RNF₁, (b) RNF₂ e (c) RNFH.

normas mais selecionadas para o melhor indivíduo obtido em todos os problemas. Observe que as normas t_4 e s_0 foram as mais selecionadas para os neurônios AND, embora com pouca diferença frente às outras normas, e as normas t_1 e s_7 foram as mais selecionadas para os neurônios OR.

7.7 Conclusões

Neste capítulo, apresentamos uma nova arquitetura *neurofuzzy* híbrida, que denominamos rede *neurofuzzy* heterogênea. Esta arquitetura difere das arquiteturas usuais porque permite o uso de diferentes normas triangulares em cada neurônio da rede. Isto é equivalente a termos diferentes pares de normas triangulares modelando conjunções e disjunções *fuzzy* em cada uma das regras de uma base de regras *fuzzy*. Na abordagem adotada neste trabalho, t-normas e s-normas são escolhidas por um algoritmo genético a partir de um

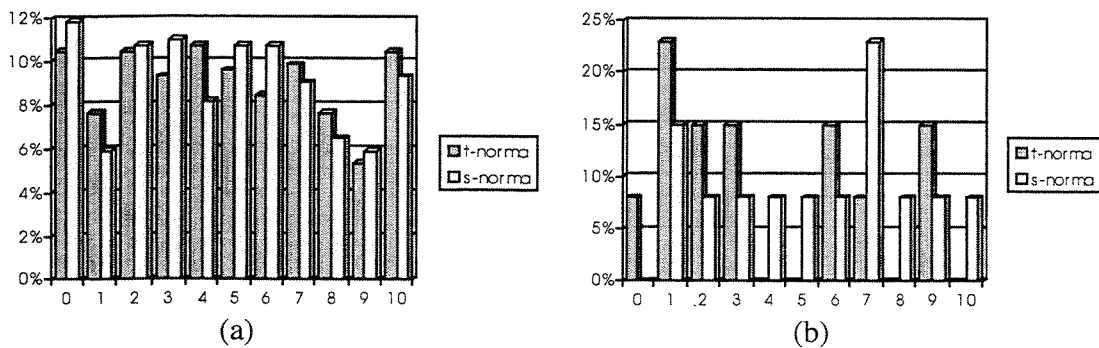


Figura 7.11 t-normas e s-normas mais selecionadas pelo algoritmo genético: (a) neurônios AND e (b) neurônios OR.

conjunto finito de normas triangulares.

Os resultados mostram que o uso de algoritmo genético para escolha de t-normas e s-normas para modelar neurônios AND e OR é efetivo, no sentido de que a rede torna-se mais flexível e adaptada ao problema em questão, aumentando o desempenho em relação a arquiteturas tradicionais.

Capítulo 8

Conclusão

Sistemas híbridos de inteligência computacional têm despertado um interesse crescente na comunidade científica. Os métodos que exploram possíveis combinações entre os principais paradigmas da inteligência computacional (redes neurais artificiais, sistemas *fuzzy* e computação evolutiva) permitem a geração automática de arquiteturas dedicadas ao problema em questão, e que frequentemente apresentam desempenho superior ao de arquiteturas mais genéricas.

Este trabalho objetivou investigar algumas possíveis interações entre os paradigmas acima citados. Entre as principais contribuições desta tese, podemos mencionar:

- uma comparação entre três algoritmos construtivos para definição de arquiteturas de redes neurais artificiais;
- uma revisão bibliográfica das principais técnicas já propostas de combinações entre redes neurais artificiais e computação evolutiva, incluindo uma comparação entre algoritmos genéticos e um algoritmo de gradiente conjugado para treinamento de redes neurais;
- uma extensão do modelo de aprendizado construtivo por busca de projeção, produzindo uma rede neural híbrida;
- uma rede *neurofuzzy* heterogênea.

Os algoritmos construtivos estudados foram o *cascade-correlation* (CASCOR), o aprendizado por busca de projeção (*Projection Pursuit Learning* – PPL) e o A*. Foram apresentados resultados comparando o desempenho dos três algoritmos, em termos de dimensão da arquitetura final produzida.

A revisão bibliográfica de combinações entre redes neurais artificiais e computação evolutiva foi dividida em duas partes: computação evolutiva para ajuste de parâmetros de redes neurais artificiais e computação evolutiva para definição de arquiteturas de redes

neurais artificiais. Na seção referente a treinamento de redes neurais usando computação evolutiva, foram apresentados resultados comparando o desempenho de algoritmos genéticos (com codificação binária e real) e o algoritmo do gradiente conjugado escalonado. Os resultados mostraram uma nítida superioridade do algoritmo do gradiente conjugado escalonado.

A rede neural híbrida corresponde a uma extensão do modelo de aprendizado construtivo por busca de projeção onde, além de neurônios escondidos com funções de ativação distintas, a saída da rede é produzida através de uma cascata de combinações aditivas e/ou multiplicativas das funções de ativação. As funções de ativação dos neurônios escondidos e a natureza das combinações (aditiva ou multiplicativa) é determinada através da aplicação de um algoritmo genético. A arquitetura híbrida resultante mostrou-se extremamente flexível, e produziu bons resultados em problemas de aproximação de funções.

A rede *neurofuzzy* heterogênea corresponde a uma arquitetura *neurofuzzy* onde, diferentemente das arquiteturas *neurofuzzy* tradicionais, podemos ter neurônios lógicos com diferentes pares de normas triangulares. As normas triangulares são escolhidas por um algoritmo genético a partir de conjuntos de normas triangulares candidatas. Os resultados mostraram que a escolha adequada de normas triangulares pode tornar a rede *neurofuzzy* mais adaptada ao problema em questão, melhorando o desempenho da rede.

8.1 Novas Estratégias de Solução de Problemas de Engenharia

Este trabalho apresentou duas estratégias poderosas para o tratamento de problemas de engenharia, como aproximação de funções e classificação de padrões:

- definição de um algoritmo para geração automática de uma arquitetura híbrida de rede neural, que permite construir modelos bastante flexíveis e parcimoniosos de aproximação de funções;
- definição de um algoritmo para geração de uma rede *neurofuzzy* heterogênea, que permite obter de forma automática as normas triangulares mais apropriadas a um determinado problema, gerando uma arquitetura dedicada a problemas de classificação de padrões, dentre outros.

Quando comparadas às soluções produzidas por estratégias alternativas, as soluções obtidas a partir da aplicação desses dois algoritmos apresentam desempenho superior no tratamento de problemas de elevada complexidade, além de requererem uma quantidade menor de recursos computacionais na sua implementação, por representarem soluções dedicadas à natureza do problema. No entanto, embora os dois algoritmos sejam computacionalmente tratáveis, os recursos computacionais necessários para produzir estas soluções dedicadas são elevados.

8.2 Extensões

Em relação às redes neurais híbridas, podemos citar como possíveis extensões:

- Estudo de propriedades de generalização.
- Estudo de robustez à dados de treinamento sujeitos a ruído.
- Otimização de parâmetros relacionados às funções de ativação.
- Uso de um algoritmo genético onde os cromossomos tenham comprimento variável, a fim de determinar de forma automática o número de neurônios na camada escondida.
- Hibridização da camada de neurônios escondidos, isto é, considerar possíveis combinações multiplicativas também entre a camada de entrada e a camada de neurônios escondidos;
- A cascata de composição híbrida parece privilegiar a última função da cascata ($f_n(\cdot)$ na Figura 6.3). Assim, é interessante estudar outras formas de composição das funções de ativação, de forma a distribuir melhor a contribuição de cada função de ativação para a saída da rede.
- Aplicação a problemas originados a partir de dados reais.

Em relação às redes *neurofuzzy* heterogêneas, podemos sugerir como possíveis extensões:

- Estudo de propriedades de generalização.
- Otimização dos parâmetros relacionados às normas triangulares.
- Estudo mais aprofundado da distribuição de t-normas e s-normas em cada problema.
- Problemas com grande número de entradas podem levar a uma explosão combinatorial no número de neurônios AND da rede *neurofuzzy* AND/OR. Assim, é interessante estudar a

possibilidade de aplicação de algum algoritmo de poda de regras *fuzzy* (por exemplo, YEUNG *et al.* (1999)) antes da aplicação do algoritmo genético.

- Estudar o possível uso de operadores OWA (YAGER, 1988), que são vistos como operadores de agregação genéricos. Dessa forma, ao invés de um conjunto finito de normas triangulares, poderíamos empregar um algoritmo para ajuste dos pesos dos operadores OWA.
- Aplicação da idéia de pares distintos de normas triangulares em outras arquiteturas *fuzzy*.
- Aplicação de rede *neurofuzzy* heterogêneas a outros tipos de problema, tais como controle de sistemas dinâmicos e aproximação de funções.

Bibliografia

- AARTS, E. & KORST, J. *Simulated Annealing and Boltzmann Machines – A Stochastic Approach to Combinatorial Optimization and Neural Computing*, John Wiley & Sons, 1989.
- ATMAR, W. “Notes on the Simulation of Evolution”, *IEEE Transactions on Neural Networks*, 5(1): 130-147, 1994.
- BÄCH, T., HAMMEL, U. & SCHWEFEL, H. -P. “Evolutionary Computation: Comments on the History and Current State”, *IEEE Transactions on Evolutionary Computation*, 1(1): 3-17, 1997.
- BALAKRISHNAN, K. & HONAVAR, V. “Evolutionary Design of Neural Architectures – A Preliminary Taxonomy and Guide to Literature”, *Technical Report CS TR #95-01*, Artificial Intelligence Research Group, Department of Computer Science, Iowa State University, EUA, 1995.
- BARRON, A. R. “Universal Approximation Bounds for Superpositions of a Sigmoidal Function”, *IEEE Transactions on Information Theory*, 39(3): 930-945, 1993.
- BAZARAA, M. S., SHERALY, H. D. & SHETTY, C. *Nonlinear Programming: Theory and Algorithms*, 2^a edição, John Willey & Sons, 1992.

- BELEW, R. K., MCINERNEY, J. & SCHRAUDOLPH, N. N. "Evolving Networks: Using the Genetic Algorithm with Connectionist Learning", *CSE Technical Report #CS90-174*, Computer Science & Engineering Department, University of California at San Diego, EUA, 1990.
- BERTONI, A. & DORIGO, M. "Implicit Parallelism in Genetic Algorithms", *Artificial Intelligence*, 61(2): 307-314, 1993.
- BEZDEK, J. C. "What is Computational Intelligence?", em Zurada, J. M., Marks II, R. J. & Robinson, C. J. (eds.), *Computational Intelligence – Imitating Life*, IEEE Press, 1-12, 1994.
- BISHOP, C. M. *Neural Networks for Pattern Recognition*, Clarendon Press, 1995.
- BLAKE, C. L. & MERZ, C. J. *UCI Repository of machine learning databases* [<http://www.ics.uci.edu/~mllearn/MLRepository.html>], Irvine, CA: University of California, Department of Information and Computer Science, 1998
- BOERS, E. G. W. & KUIPER, H. "Biological Metaphors and the Design of Modular Artificial Neural Networks", Tese de Mestrado, Universidade de Leiden, Holanda, 1992.
- BREIMAN, L. & FRIEDMAN, J. H. "Predicting Multivariate Responses in Multiple Linear Regression", *Journal of the Royal Statistical Society B*, 59(1): 3-54, 1997.
- BUCKLEY, J. J. & HAYASHI, Y. "Fuzzy Neural Networks: A Survey", *Fuzzy Sets and Systems*, 66: 1-13, 1994.
- CAMINHAS, W. M. *Estratégias de Detecção e Diagnóstico de Falhas em Sistemas Dinâmicos*, Tese de Doutorado, Faculdade de Engenharia Elétrica e de Computação, Unicamp, 1997.

- CANTÚ-PAZ, E. “A Survey of Parallel Genetic Algorithms”, *Calculateurs Parallèles, Réseaux et Systems Repartis*, 10(2): 141-171, 1998.
- CHEN, C. –H. “Automatic Design of Neural Networks Based on Genetic Algorithms”, *Proceedings of the 1998 International Computer Symposium – Workshop on Artificial Intelligence*, 8-13, 1998.
- CHEN, Z., XIAO, J. & CHENG & J. “PASS: A Program for Automatic Structure Search”, *Proceedings of the 1997 International Conference on Neural Networks*, (1): 308-311, 1997.
- COOMANS, D., BROECKAERT, M., JONCKHEER, M. & MASSART, D. L. “Comparison of Multivariate Discriminant Techniques for Clinical Data – Application to the Thyroid Functional State”, *Methods of Information in Medicine*, 22: 93-101, 1983.
- CYBENKO, G. “Approximation by Superpositions of a Sigmoidal Function”, *Mathematics of Control, Signals and Systems*, 2: 303-314, 1989.
- DAHMEN, W. & MICCHELLI, C. A. “Some Remarks on Ridge Functions”, *Approximation Theory and its Applications*, 3(2-3): 139-143, 1987.
- DE CASTRO, L. N. *Análise e Síntese de Estratégias de Aprendizado para Redes Neurais Artificiais*, Tese de Mestrado, Faculdade de Engenharia Elétrica e de Computação, Unicamp, 1998.
- DE CASTRO, L. N., IYODA, E. M., VON ZUBEN, F. J. & GUDWIN, R. “Feedforward Neural Network Initialization: an Evolutionary Approach”, *Proceedings of the Vth Brazilian Symposium on Neural Networks*, 1: 43-48, 1998.

- DE CASTRO, L. N., IYODA, E. M., PINHEIRO, E. & VON ZUBEN, F. “Redes Neurais Construtivas: Uma Abordagem Comparativa”, *Anais do IV Congresso Brasileiro de Redes Neurais*, 102-107, 1999.
- DE CASTRO, L. N. & VON ZUBEN, F. J. “Uma Alternativa Simples e Robusta para Inicialização de Pesos em Redes Neurais Multicamada”, *Anais do V Simpósio Brasileiro de Redes Neurais*, 2: 97-102, 1998.
- DE CASTRO, L. N. & VON ZUBEN, F. J. “An Improving Pruning Technique with Restart for the Kohonen Self-Organizing Feature Map”, *Proceedings of the 1999 IEEE International Joint Conference on Neural Networks*, artigo #395, 1999.
- DEVORE, R. A., HOWARD, R. & MICCHELLI, C. “Optimal Nonlinear Approximation”, *Manuscripta Mathematica*, 63(4): 469-478, 1989.
- DOERING, A., GALICKI, M. & WITTE, H. “Structure Optimization of Neural Networks with the A*-Algorithm”, *IEEE Transactions on Neural Networks*, 8(6): 1434-1445, 1997.
- DONOHU, D. L. & JOHNSTONE, I. M. “Projection-Based Approximation and a Duality with Kernel Methods”, *The Annals of Statistics*, 17(1): 58-106, 1989.
- EIBEN, Á. E., HINTERDING, R. & MICHALEWICZ, Z. “Parameter Control in Evolutionary Algorithms”, *IEEE Transactions on Evolutionary Computation*, 3(2): 124-141, 1999.
- ENSLEY, D. & NELSON, D. E. “Extrapolation of MacKey-Glass Data Using Cascade Correlation”, *Simulation*, 58(5): 333-339, 1992.
- ESHELMAN, L. J., CARUANA, R. A. & SCHAFFER, J. D. “Biases in the Crossover Landscape”, em Schaffer, J. (ed.), *Proceedings of the Third International Conference on Genetic Algorithms*, Morgan Kaufmann Publishers, 10-19, 1989.

- FAHLMAN, S. E. "An Empirical Study of Learning Speed in Back-Propagation Networks", *Technical Report CMU-CS-88-162*, School of Computer Science, Carnegie Mellon University, EUA, 1988.
- FAHLMAN, S. E. & LEBIERE, C. "The Cascade-Correlation Learning Architecture", *Technical Report CMU-CS-90-100*, School of Computer Science, Carnegie Mellon University, EUA, 1990.
- FANG, J. & XI, Y. "Neural Network Design Based on Evolutionary Programming", *Artificial Intelligence in Engineering*, 11: 155-161, 1997.
- FIGUEIREDO, M., GOMIDE, F. & PEDRYCZ, W. "A Fuzzy Neural Network: Structure and Learning", em Bien, Z., Min, K. C. (eds.), *Fuzzy Logic and its Applications, Information Sciences, and Intelligent Systems*, Kluwer Academic Publishers, 177-186, 1995.
- FOGEL, D. B. "An Introduction to Simulated Evolutionary Computation", *IEEE Transactions on Neural Networks*, 5(1): 3-14, 1994.
- FOGEL, D. B. *Evolutionary Computation – Toward a New Philosophy of Machine Intelligence*, 2ª edição, IEEE Press, 1999.
- FOGEL, D. B., FOGEL, L. J. & PORTO, V. W. "Evolving Neural Networks", *Biological Cybernetics*, 63: 487-493, 1990.
- FOGEL, L. J., OWENS, A. J. & WALSH, M. J. *Artificial Intelligence Through Simulated Evolution*, John Wiley, 1966.
- FRIEDMAN, J. H. "SMART User's Guide", *Report LCM001*, Department of Statistics, Stanford University, EUA, 1984.

- FRIEDMAN, J. H. & STUETZLE, W. "Projection Pursuit Regression", *Journal of the American Statistical Association (JASA)*, 76(376): 817-823, 1981.
- FRIEDMAN, J. H., STUETZLE, W. & SCHROEDER, A. "A Projection Pursuit Density Estimation", *Journal of the American Statistical Association (JASA)*, 79(387): 599-608, 1984.
- FRIEDMAN, J. H. & TUKEY, J. "A Projection Pursuit Algorithm for Exploratory Data Analysis", *IEEE Transactions on Computers*, 23(9): 881-890, 1974.
- FUNAHASHI, K. "On the Approximate Realization of Continuous Mappings by Neural Networks", *Neural Networks*, 2(3): 183-192, 1989.
- GOLDBERG, D. E. *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, 1989(a).
- GOLDBERG, D. E. "Messy Genetic Algorithms: Motivation, Analysis, and First Results", *Complex Systems*, 3: 493-530, 1989(b).
- GOLDBERG, D. E. "A Meditation on the Application of Genetic Algorithms", *IlligAL Report No. 98003*, Department of General Engineering, University of Illinois at Urbana-Champaign, EUA, 1998.
- GRUAU, F. "Genetic Micro Programming of Neural Networks", em Kinnear Jr., K. (ed.), *Advances in Genetic Programming*, The MIT Press, 495-518, 1994.
- HARP, S. A., SAMAD, T. & GUHA, A. "Towards the Genetic Synthesis of Neural Networks", em Schaffer, J. D. (ed.), *Proceedings of the Third International Conference on Genetic Algorithms*, Morgan Kaufmann Publishers, 360-369, 1989.

- HARTL, D. L. & CLARK, A. G. *Principles of Population Genetics*, Sinauer, 1989.
- HAYKIN, S. *Neural Networks – A Comprehensive Foundation*, 2ª edição, Macmillan, 1999.
- HIROTA, K. & PEDRYCZ, W. “OR/AND Neuron in Modeling Fuzzy Set Connectives”, *IEEE Transactions on Fuzzy Systems*, 2(2): 151-161, 1994.
- HIROTA, K. & SUGENO, M. (eds.). *Industrial Applications of Fuzzy Technology in the World*, World Scientific, 1995.
- HOLLAND, J. H. *Adaptation in Natural and Artificial Systems*, 2ª edição, MIT Press, 1992.
- HORNIK, K., STINCHCOMBE, M. & WHITE, H. “Multilayer Feedforward Neural Networks Are Universal Approximators”, *Neural Networks*, 2: 359-366, 1989.
- HUBER, P. J. “Projection Pursuit (with Discussion)”, *The Annals of Statistics*, 13(2): 435-475, 1985.
- HWANG, J. -N., LAY, S. -R., MAECHLER, M., MARTIN, R. D. & SCHIMERT, J. “Regression Modelling in Back-Propagation and Projection Pursuit Learning”, *IEEE Transactions on Neural Networks*, 5(3): 342-353, 1994.
- ISHIBUCHI, H., NOZAKI, K. & TANAKA, H. “Distributed representation of fuzzy rules and its application to pattern classification”, *Fuzzy Sets and Systems*, 52: 21-32, 1992.
- IYODA, E. M., DE CASTRO, L. N., GOMIDE, F. & VON ZUBEN, F. J. “Evolutionary Design of Neurofuzzy Networks for Pattern Classification”, *Proceedings of the 1999 Congress on Evolutionary Computation*, 2: 1237-1244, 1999.

- IYODA, E. M. & VON ZUBEN, F. J. "Evolutionary Hybrid Composition of Activation Functions in Feedforward Neural Networks", *Proceedings of the 1999 International Joint Conference on Neural Networks*, artigo #396, 1999.
- KASABOV, N. K. & WATTS, M. J. "Genetic Algorithms for Structural Optimisation, Dynamic Adaptation and Automated Design of Fuzzy Neural Networks", *Proceedings of the 1997 International Conference on Neural Networks*, 4: 2546-2549, 1997.
- KIM, Y. K. & RA, J. B. "Weight Value Initialization for Improving Training Speed in the Backpropagation Network", *Proceedings of the IEEE International Joint Conference on Neural Networks*, 3: 2396-2401, 1991.
- KITANO, H. "Designing Neural Networks Using Genetic Algorithm with Graph Generation System", *Complex Systems*, 4: 461-479, 1990.
- KITANO, H. "Neurogenetic learning: an integrated method of designing and training neural networks using genetic algorithms", *Physica D*, 75: 225-238, 1994.
- KOEHN, P. *Combining Genetic Algorithms and Neural Networks: The Encoding Problem*, Master Thesis, University of Tennessee, Knoxville, EUA, 1994.
- KOHONEN, T. *Self-organization and associative memory*, 3^a edição, Springer, 1989.
- KORNING, P. G. "Training Neural Networks by means of Genetic Algorithms Working on Very Long Chromosomes", *DAIMI Report PB-486*, Computer Science Department, Aarhus University, Dinamarca, 1994.
- KOZA, J. R. *Genetic Programming: on the Programming of Computers by Means of Natural Selection*, MIT Press, 1992.

- KOZA, J. R., BENNET III, F. H., ANDRE, D., KEANE, M. A. & DUNLAP, F. “Automated Synthesis of Analog Electrical Circuits by Means of Genetic Programming”, *IEEE Transactions on Evolutionary Computation*, 1(2): 109-128, 1997.
- KWOK, T. -Y., & YEUNG, D. -Y. “Constructive Algorithms for Structure Learning in Feedforward Neural Networks for Regression Problems”, *IEEE Transactions on Neural Networks*, 8(3): 630-645, 1997.
- LIU, Y. & YAO, X. “Evolutionary Design of Artificial Neural Networks with Different Nodes”, *Proceedings of the 1996 International Conference on Evolutionary Computation*, 670-675, 1996.
- LUENBERGER, D. G. *Linear and Nonlinear Programming*, 2^a edição, Addison Wesley, 1984.
- MANDISCHER, M. “Representation and Evolution of Neural Networks”, em Albrecht, R. F., Reeves, C. R. & Steele, N. C. (eds.), *Artificial Neural Nets and Genetic Algorithms – Proceedings of the International Conference in Innsbruck, Austria*, Springer, 643-649, 1993.
- MANIEZZO, V. “Genetic Evolution of the Topology and Weight Distribution of Neural Networks”, *IEEE Transactions on Neural Networks*, 5(1): 39-53, 1994.
- MCCULLOCH, W. S. & PITTS, W. “A Logical Calculus of the Ideas Immanent in Nervous Activity”, *Bulletin of Mathematical Biophysics*, 5, 1943.
- MHASKAR, H. N. & MICCHELLI, C. A. “How to Choose an Activation Function”, em Cowan, J. D., Tesauro, G., Alspector, J. (eds.), *Advances in Neural Information Processing Systems*, Morgan Kaufmann Publishers, 6: 319-326, 1994.

- MICHALEWICZ, Z. *Genetic Algorithms + Data Structures = Evolution Programs*, 3^a edição, Springer, 1996.
- MICHALEWICZ, Z. & SCHOENAUER, M. “Evolutionary Algorithms for Constrained Parameter Optimization Problems”, *Evolutionary Computation*, 4(1): 1-32, 1996.
- MILLER, G. F., TODD, P. M. & HEDGE, S. U. “Designing Neural Networks using Genetic Algorithms”, em Schaffer, J. D. (ed.), *Proceedings of the Third International Conference on Genetic Algorithms*, Morgan Kaufmann Publishers, 379-384, 1989.
- MINSKY, M. & PAPERT, S. *Perceptrons: an Introduction to Computational Geometry*, edição expandida, MIT Press, 1988.
- MØLLER, M. F. “A Scaled Conjugate Gradient Algorithm for Fast Supervised Learning”, *Neural Networks*, 6: 525-533, 1993.
- MORIARTY, D. E. & MIKKULAINEN, R. “Hierarchical Evolution of Neural Networks”, *Proceedings of the 1998 Conference on Evolutionary Computation*, 428-433, 1998.
- NGUYEN, D. & WIDROW, B. “Improving the Learning Speed of Two-Layer Neural Networks by Choosing Initial Values of Adaptive Weights”, *Proceedings of the International Joint Conference on Neural Networks*, 3: 21-26, 1990.
- NILSSON, N. *Principles of Artificial Intelligence*, Springer-Verlag, 1986.
- NOZAKI, K., ISHIBUCHI, H. & TANAKA, H. “Adaptive Fuzzy Rule-Based Classification Systems”, *IEEE Transactions on Fuzzy Systems*, 4(3):238-250, 1996.
- PEARLMUTTER, B. A. “Fast Exact Multiplication by the Hessian”, *Neural Computation*, 6: 147-160, 1994.

- PEDRYCZ, W. "Genetic Algorithms for Learning in Fuzzy Relational Structures", *Fuzzy Sets and Systems*, 69: 37-52, 1995.
- PEDRYCZ, W. & GOMIDE, F. *An Introduction to Fuzzy Sets*, MIT Press, 1998.
- PEDRYCZ, W. & ROCHA, A. "Fuzzy-Set Based Models of Neurons and Knowledge-Based Networks", *IEEE Transactions on Fuzzy Systems*, 1(4): 254-265, 1993.
- PUJOL, J. C. F. & POLI, R. "Evolving Neural Networks Using a Dual Representation with a Combined Crossover Operator", *Proceedings of the 1998 International Conference on Evolutionary Computation*, 416-421, 1998.
- RECHENBERG, I. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*, Frommann-Holzboog, 1973.
- REED, R. "Pruning Algorithms – A Survey", *IEEE Transactions on Neural Networks*, 4(5): 740-747, 1993.
- RUMELHART, D. E., HINTON, G. E. & WILLIAMS, R. J. "Learning Internal Representations by Error Propagation", em Rumelhart, D. E., McClelland, J. L. (eds.), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, vol. 1, MIT Press, 1986.
- SCHIFFMANN, W., JOOST, M. & WERNER, R. "Synthesis and Performance Analysis of Multilayer Neural Network Architectures", *Technical Report 16/1992*, Institute für Physics, University of Koblenz, Alemanha, 1992.
- SCHWEFEL, H. –P. "On the Evolution of Evolutionary Computation", em Zurada, J. M., Marks II, R. J. & Robinson, C. J. (eds.), *Computational Intelligence – Imitating Life*, IEEE Press, 1994.

SCHWEFEL, H. –P. *Evolution and Optimum Seeking*, John Wiley, 1995.

SIDDIQI, A. A. & LUCAS, S. M. “A comparison of matrix rewriting versus direct encoding for evolving neural networks”, *Proceedings of the 1998 International Conference on Evolutionary Computation*, 392-397, 1998.

SIMPSON, P. K. (ed.). *Neural Networks Applications*, IEEE Press, 1996.

SYSWERDA, G. “Uniform Crossover in Genetic Algorithms”, em Schaffer, J. D. (ed.), *Proceedings of the Third International Conference on Genetic Algorithms*, Morgan Kaufmann Publishers, 2-9, 1989.

TAKAGI, H. & LEE, M. “Neural Networks and Genetic Algorithms Approaches to Auto-Design of Fuzzy Systems”, *Proceedings of Fuzzy Logic in Artificial Intelligence*, 1993.

VON ZUBEN, F. J. *Modelos Paramétricos e Não-Paramétricos de Redes Neurais Artificiais e Aplicações*, Tese de Doutorado, Faculdade de Engenharia Elétrica e de Computação, Unicamp, 1996.

VON ZUBEN, F. J., IYODA, E. M. & NETTO, M. L. A. “Evolutionary and Constructive Approaches to Supervised Learning in Hybrid Neural Networks”, *Proceedings of the World Multiconference on Systemics, Cybernetics and Informatics*, 3: 255-260, 1999.

VON ZUBEN, F. J. & NETTO, M. L. A. “Unit-Growing Learning Optimizing the Solvability Condition for Model-Free Regression”, *Proceedings of the IEEE International Conference on Neural Networks*, 2: 795-800, 1995.

- VON ZUBEN, F. J. & NETTO, M. L. A. "Projection Pursuit and the Solvability Condition Applied to Constructive Learning", *Proceedings of the IEEE Joint International Conference on Neural Networks*, 2: 1062-1067, 1997.
- WHITE, D. A. & SOFGE, D. A. *Handbook of Intelligent Control – Neural, Fuzzy and Adaptive Approaches*, Van Nostrand Reinhold, 1992.
- WHITLEY, D., STARKWEATHER, T. & BOGART, C. "Genetic Algorithms and Neural Networks: Optimizing Connections and Connectivity", *Parallel Computing*, 14: 347-361, 1990.
- YAGER, R. R. "On Ordered Weighted Averaging Aggregation Operations in Multicriteria Decision Making", *IEEE Transactions on Systems, Man and Cybernetics*, 18(1): 183-190, 1988.
- YAM, Y., BARANYI, P. & YANG, C. -T. "Reduction of Fuzzy Rule Base Via Singular Value Decomposition", *IEEE Transactions on Fuzzy Systems*, 7(2): 120-132, 1999.
- YAO, X. "A Review of Evolutionary Artificial Neural Networks", *International Journal of Intelligent Systems*, 8(4): 539-567, 1993.
- YAO, X. & LIU, Y. "A New Evolutionary System for Evolving Artificial Neural Networks", *IEEE Transactions on Neural Network*, 8(3): 694-713, 1997.
- ZADEH, L. A. "Fuzzy Sets", *Information and Control*, 8: 338-353, 1965; re-impresso em Klir, G. J. & Juan, B. (eds.), *Fuzzy Sets, Fuzzy Logic and Fuzzy Systems: Selected Papers by Lotfi A. Zadeh*, World Sci, 1996.
- ZHANG, B. -T., OHM, P. & MÜHLENBEIN, H. "Evolutionary Induction of Sparse Neural Trees", *Evolutionary Computation*, 5(2): 213-236, 1997.

ZHAO, Q. "A Co-Evolutionary Algorithm for Neural Network Learning", *Proceedings of the 1997 International Conference on Neural Networks*, (1): 432-437, 1997.

Índice de Autores

AARTS & KORST (1989)	38, 92
ATMAR (1994)	32, 33, 34
BÄCK <i>et al.</i> (1997)	31, 32, 45
BALAKRISHNAN & HONAVAR (1995)	88
BARRON (1993)	96
BAZARAA <i>et al.</i> (1992)	18, 26, 105
BERTONI & DORIGO (1993)	49
BEZDEK (1994)	2
BISHOP (1995)	7
BLAKE & MERZ (1998)	140
BOERS & KUIPER (1992)	24
BREIMAN & FRIEDMAN (1997)	78
BUCKLEY & HAYASHI (1994)	119
CAMINHAS (1997)	119, 120, 125, 132, 134, 136
CANTÚ-PAZ (1998)	51
CHEN (1998)	82
CHEN <i>et al.</i> (1997)	92
COOMANS <i>et al.</i> (1983)	140
CYBENKO (1989)	28
DAHMEN & MICCHELLI (1987)	63
DE CASTRO & VON ZUBEN (1998)	24
DE CASTRO & VON ZUBEN (1999)	129
DE CASTRO (1998)	26
DE CASTRO <i>et al.</i> (1998)	24, 44
DE CASTRO <i>et al.</i> (1999)	76

DEVORE <i>et al.</i> (1989)	96
DOERING <i>et al.</i> (1997)	74
DONOHO & JOHNSTONE (1989)	99
EIBEN <i>et al.</i> (1999)	50
ENSLEY & NELSON (1992)	60
ESHELMAN <i>et al.</i> (1989)	42
FAHLMAN & LEBIERE (1990)	57, 59, 60
FAHLMAN (1988)	57
FANG & XI (1997)	92
FIGUEIREDO <i>et al.</i> (1995)	119
FOGEL (1994)	32, 36, 40, 45
FOGEL (1999)	32, 34
FOGEL <i>et al.</i> (1966)	36
FOGEL <i>et al.</i> (1990)	81
FRIEDMAN & STUETZLE (1981)	61, 63
FRIEDMAN & TUKEY (1974)	62
FRIEDMAN (1984)	71
FRIEDMAN <i>et al.</i> (1984)	62
FUNAHASHI (1989)	28
GOLDBERG (1989a)	51
GOLDBERG (1989b)	49
GOLDBERG (1998)	51
GRUAU (1994)	35, 91
HARP <i>et al.</i> (1989)	89
HARTL & CLARK, (1989)	32
HAYKIN (1999)	7, 12, 22, 24
HIROTA & PEDRYCZ (1994)	123
HIROTA & SUGENO (1995)	3
HOLLAND (1992)	35, 39, 43, 46, 48, 49
HORNIK <i>et al.</i> (1989)	28

HUBER (1985).....	63
HWANG <i>et al.</i> (1994)	67, 71, 78, 85, 116
ISHIBUCHI <i>et al.</i> (1992).....	131
IYODA & VON ZUBEN (1999)	99
IYODA <i>et al.</i> (1999).....	137
KASABOV & WATTS (1997)	120
KIM & RA (1991)	24
KITANO (1990)	90
KITANO (1994)	90
KOEHN (1994)	89
KOHONEN (1989).....	129, 140
KORNING (1994).....	82
KOZA (1992).....	35
KOZA <i>et al.</i> (1997).....	35
KWOK & YEUNG (1997).....	54, 55
LIU & YAO (1996).....	91, 92
LUENBERGER (1984)	18, 26, 105
MØLLER (1993)	23, 26, 82, 105, 108
MANDISCHER (1993)	90
MANIEZZO (1994).....	91
MCCULLOCH & PITTS (1943).....	10
MHASKAR & MICCHELLI (1994)	96
MICHALEWICZ & SCHOENAUER (1996)	43, 44, 45
MICHALEWICZ (1996).....	36, 37, 38, 39, 42, 44, 45, 46, 48, 49, 50, 139
MILLER <i>et al.</i> (1989)	87, 88, 89
MINSKY & PAPERT (1988).....	11
MORIARTY & MIKKULAINEN (1998).....	93
NGUYEN & WIDROW (1990).....	24
NILSSON (1986)	74
NOZAKI <i>et al.</i> (1996).....	131

PEARLMUTTER (1994)	26, 105
PEDRYCZ & GOMIDE (1998).....	119, 120, 121, 122, 124
PEDRYCZ & ROCHA (1993).....	123, 124
PEDRYCZ (1995)	119
PUJOL & POLI (1998).....	93
RECHENBERG (1973)	36
REED (1993)	53
RUMELHART <i>et al.</i> (1986).....	17
SCHIFFMANN <i>et al.</i> (1992)	90
SCHWEFEL (1994).....	31, 86
SCHWEFEL (1995).....	36
SIDDIQI & LUCAS (1998).....	90
SIMPSON (1996).....	3
SYSWERDA (1989)	42
TAKAGI & LEE (1993)	119
VON ZUBEN & NETTO (1995).....	71, 72
VON ZUBEN (1996).....	7, 27, 53, 61, 63, 64, 66, 67, 71, 72
VON ZUBEN <i>et al.</i> (1999).....	114
WHITE & SOFGE (1992).....	7
WHITLEY <i>et al.</i> (1990)	81, 89
YAGER (1988)	148
YAO & LIU (1997).....	88, 92
YAO (1993)	81, 88
YEUNG <i>et al.</i> (1999)	148
ZADEH (1965).....	120
ZHANG <i>et al.</i> (1997).....	93
ZHAO (1997).....	93