



UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
CENTRO DE TECNOLOGIA
DEPARTAMENTO DE ENG. DE COMPUTAÇÃO E
AUTOMAÇÃO

CONTROLE INTELIGENTE

Professor: Fábio Meneghetti Ugulino de Araújo

Natal-RN, Julho de 2015

ÍNDICE

1	INTRODUÇÃO	4
1.1	CONCEITUAÇÃO DE INTELIGÊNCIA ARTIFICIAL	4
1.2	APRENDIZADO	5
1.3	FERRAMENTAS UTILIZADAS EM IA.....	5
1.3.1	<i>Sistemas de Produção</i>	5
1.3.2	<i>Lógica Nebulosa.....</i>	6
1.3.3	<i>Redes Neurais.....</i>	6
1.4	ALGUMAS APLICAÇÕES TÍPICAS DE IA.....	6
2	LÓGICA E DEDUÇÃO	7
2.1	CÁLCULO SENTENCIAL	7
2.1.1	<i>Dedução</i>	8
2.2	CÁLCULO DE PREDICADOS	9
2.2.1	<i>Resolução</i>	10
3	REPRESENTAÇÃO DE CONHECIMENTOS	12
3.1	PRINCIPAIS FORMAS DE REPRESENTAÇÃO DO CONHECIMENTO EM IA.....	12
3.1.1	<i>Representação Utilizando Lógica Matemática</i>	12
3.1.2	<i>Representação Utilizando Regras de Produção</i>	12
3.1.3	<i>Representação Utilizando Redes Semânticas</i>	13
3.1.4	<i>Representação Utilizando Frames e Scripts</i>	13
3.2	MÉTODOS DE BUSCA.....	14
3.2.1	<i>Algoritmo A*</i>	16
4	CONTROLADORES BASEADOS EM CONHECIMENTOS.....	17
4.1	EXEMPLO DE CONTROLADOR USANDO REGRAS DE PRODUÇÃO.....	18
4.2	EXEMPLO DE UM SUPERVISOR INTELIGENTE	19
4.3	EXEMPLO DE UM ASSISTENTE INTELIGENTE.....	20
5	INTRODUÇÃO À LÓGICA NEBULOSA	22
5.1	CONJUNTOS NEBULOSOS	22
5.2	LÓGICA NEBULOSA	26
6	CONTROLADORES EMPREGANDO LÓGICA NEBULOSA.....	28
6.1	ESTRUTURA DE CONTROLADORES EMPREGANDO LÓGICA NEBULOSA	29
6.1.1	<i>Conversor para Variáveis Nebulosas (“Nebulizador”)</i>	29
6.1.2	<i>Máquina de Inferência Nebulosa</i>	30
6.1.3	<i>Conversor para Variável Numérica (“Desnebulizador”)</i>	32
6.2	CONTROLADORES COM E(T) E U(T) QUANTIZADOS	32
6.3	ESTABILIDADE NO CONTROLE NEBULOSO EM MALHA FECHADA	34
6.4	CARACTERÍSTICAS DE CONTROLADORES NEBULOSOS	36
7	MÉTODOS DE OTIMIZAÇÃO NUMÉRICA.....	38
7.1	CONCEITUAÇÃO E UTILIDADE EM SISTEMAS INTELIGENTES	38
7.2	MÉTODOS DE OTIMIZAÇÃO UNIDIMENSIONAL	40
7.2.1	<i>Método da Busca Uniforme.....</i>	40
7.2.2	<i>Método da Secção Áurea.....</i>	41
7.3	MÉTODOS DE BUSCA EMPREGANDO GRADIENTE.....	42
7.3.1	<i>Método da Máxima Declividade</i>	43
7.3.2	<i>Método de Newton.....</i>	44
7.4	MÉTODOS DE BUSCA SEM EMPREGAR GRADIENTE	45
7.4.1	<i>Método da Busca Direta</i>	45
7.4.2	<i>Método dos Poliedros Flexíveis</i>	46
7.4.3	<i>Algoritmos Genéticos</i>	48
7.5	MÉTODOS ESTENDIDOS	50
7.5.1	<i>Zona Tabu</i>	50
7.5.2	<i>Recozimento Simulado</i>	51
7.6	SELEÇÃO DO MÉTODO DE OTIMIZAÇÃO	52
8	REDES NEURAIS ARTIFICIAIS.....	53

8.1	MODELOS E ARQUITETURAS.....	53
8.1.1	<i>O neurônio biológico (natural)</i>	54
8.1.2	<i>Definição de Rede Neural Artificial.....</i>	55
8.1.3	<i>Uma Estrutura Geral para Modelos de Redes Neurais Artificiais</i>	55
8.1.4	<i>Topologia de Redes Neurais Artificiais</i>	57
8.1.5	<i>Aprendizado em Redes Neurais Artificiais.....</i>	57
8.1.6	<i>Capacidade de Aproximação Universal</i>	58
8.2	APRENDIZADO COM SUPERVISÃO FORTE.....	59
8.2.1	<i>Programação Direta dos Pesos</i>	59
8.2.2	<i>Ajuste de Pesos Mediante Amostras de Pares Entrada-Saída.....</i>	64
8.2.3	<i>O Algoritmo Back-Propagation</i>	71
8.3	APLICAÇÕES DE REDES NEURAIS ARTIFICIAIS EM CONTROLE	74

1 INTRODUÇÃO

A Inteligência Artificial (IA) busca prover máquinas com a capacidade de realizar algumas atividades mentais do ser humano. Em geral são máquinas com algum recurso computacional, de variadas arquiteturas, que permitem a implementação de rotinas não necessariamente algorítmicas. As atividades realizadas por estas máquinas podem envolver a senso-percepção (tato, audição e visão), as capacidades intelectuais (aprendizado de conceitos e de juízos, raciocínio dedutivo e memória), a linguagem (verbais e gráficas) e atenção (decisão no sentido de concentrar as atividades sobre um determinado estímulo).

Em termos de tecnologia, IA permite que máquinas possam realizar tarefas complexas no lugar do operador humano, liberando-o de atividades enfadonhas, insalubres ou inseguras. Também pode aumentar a eficiência do humano na sua interação com equipamentos sofisticados. Permite, ainda, que conhecimentos possam ser compartilhados por muitas pessoas, sem que haja necessidade de consultas a especialistas.

Esta apostila baseia-se na obra de Nascimento Jr. e Yoneyama (2002) e não substitui em nenhum sentido tal livro, visando tão somente propiciar aos alunos um material de estudo resumido que apresente os principais conceitos explorados na supracitada obra.

1.1 Conceituação de Inteligência Artificial

Em 1948, N. Wiener talhou o termo cibernetica, compreendendo uma ciência que abrangeia, entre outros temas, o estudo da inteligência de máquinas. Em 1950, Alan Turing propôs um método para determinar se uma máquina poderia apresentar inteligência (Turing Test). O primeiro passo é, portanto, tentar estabelecer o que deve ser entendido como Inteligência Artificial. Contudo, conceituar Inteligência Artificial (IA) é uma tarefa difícil, seguem algumas das definições fornecidas por autores de renome na literatura especializada:

- a) *IA é o estudo das faculdades mentais através do uso de modelos computacionais (Charniak e McDermott, 1985);*
- b) *IA é o estudo de como fazer os computadores realizarem tarefas que, no momento, são feitas melhor por pessoas (Rich, 1983);*
- c) *IA é o estudo das idéias que permitem habilitar os computadores a fazerem coisas que tornam as pessoas inteligentes (Winston, 1977);*
- d) *IA é o campo de conhecimentos onde se estudam sistemas capazes de reproduzir algumas das atividades mentais humanas (Nilsson, 1986).*

Uma vez que as definições de IA envolvem termos como faculdades mentais, tarefas melhor feitas por pessoas, coisas que tornam as pessoas inteligentes e atividades mentais humanas, é interessante que se considere, inicialmente, os aspectos psicológicos da inteligência humana, ainda que o objetivo final não seja reproduzi-los.

Segundo livros do campo da psicologia (Paim, 1986) e da psiquiatria (Ey, Bernard e Brisset, 1985), as atividades psíquicas (do grego *psykhe* = alma) fundamentais são:

Sensação; Percepção; Representação; Conceituação;
Juízo; Raciocínio; Memória; Atenção; Consciência;
Orientação; Afetividade; Vontade; e Linguagem.

Um sistema empregando IA tenta imitar e integrar diversas atividades psíquicas de forma simultânea. Por exemplo, um veículo auto-guiado pode estar dotado de sistemas de senso-percepção, mecanismo para representação do ambiente, bem como bancos de conhecimentos e de dados, de forma a realizar inferências de forma autônoma e navegar de um ponto a outro, sem colidir com obstáculos na sala.

1.2 Aprendizado

Uma área fundamental em IA é o estudo do aprendizado. Para Kodratoff, 1986, aprendizado é a aquisição de conceitos e de conhecimentos estruturados. Esta aquisição certamente envolve o juízo, o raciocínio e a memória. Na prática, para que a aquisição de conhecimentos ocorra, há ainda a necessidade de sensação, percepção, representação, linguagem e, possivelmente, atenção. Portanto, aprendizado é um processo complexo e integrado que envolve múltiplas funções psíquicas no ser humano. Em consequência, o estudo do aprendizado pela máquina não pode prescindir da conceituação adequada destas funções psíquicas.

O aprendizado pode ocorrer com ou sem a presença de um tutor (professor). Quando o tutor orienta a aquisição dos conceitos e de conhecimentos estruturados tem-se o aprendizado supervisionado. Quando o aprendizado ocorre em função apenas dos estímulos primitivos tem-se o aprendizado não-supervisionado. Dependendo da intensidade de envolvimento do tutor, o aprendizado pode ser por descoberta, por exemplos ou por programação. Os mecanismos empregados no processo de aprendizado podem ser:

a) Numérico ou conceptual: O aprendizado numérico é aquele onde valores de certos parâmetros são ajustados no processo de armazenamento das informações. É o caso de redes neurais, onde os valores dos ganhos das sinapses alteram o conhecimento representado pela rede. O aprendizado é conceptual se as relações entre as diversas entidades são apreendidos pelo sistema, usualmente na forma simbólica.

b) Punição e Recompensa: É um mecanismo de aprendizado onde as tentativas e erros são disciplinadas por um supervisor que fornece ao aprendiz um sinal de realimentação na forma de punição ou recompensa para ações favoráveis ou desfavoráveis.

c) Empírico ou Racional: O aprendizado é empírico se ocorre com base em experimentação ou amostragem do mundo real. O aprendizado é racional se é direcionado por um mecanismo de inferência solidamente fundamentado.

d) Dedutivo/Indutivo/Inventivo: O aprendizado é dedutivo se o objetivo pode ser alcançado a partir de mecanismos de inferência sobre o conjunto de premissas fornecidas a priori. O aprendizado é indutivo se há necessidade de generalizações dos conceitos apreendidos a partir de exemplos e do conjunto de premissas fornecidas a priori, para se alcançar o objetivo. O aprendizado é inventivo se há necessidade de aquisicionar novas premissas, não obtentíveis através de generalizações das já disponíveis.

1.3 Ferramentas utilizadas em IA.

Dentre as muitas ferramentas de IA existentes hoje em dia, três constituem objeto de estudo neste texto, são elas:

1.3.1 Sistemas de Produção

Os sistemas de produção são aqueles que utilizam conjuntos de regras, usualmente do tipo Se (condição) Então (ação ou conclusão), aliado a uma base de dados (de conhecimentos) e mecanismos de controle (que indicam a seqüência de regras a serem usadas conjuntamente com a base de dados e, na existência de conflitos, providenciam a sua resolução).

Como exemplo, pode-se mencionar a expressão:

Se {(temperatura > 300°C) e (pressão > 2 atm)}; Então {válvula_1 = OFF}

São intimamente ligados ao cálculo de predicados, que por sua vez, é uma sub-área da Lógica Matemática. A aplicação direta da lógica se manifesta nos sistemas especialistas, freqüentemente baseados em regras de produção.

1.3.2 Lógica Nebulosa

Já na década de 30, J. Lukasiewicz havia formalizado um sistema lógico de 3 valores $\{0, 1/2, 1\}$, posteriormente estendido para os racionais e reais compreendidos entre 0 e 1. Também nesta época, M. Black havia introduzido o termo “*vagueness*” para expressar incertezas no campo da mecânica quântica. Em 1965, L.A. Zadeh introduziu os conjuntos nebulosos, reativando o interesse em estruturas matemáticas multivariadas. Em 1973, o próprio Zadeh propunha a lógica nebulosa como um novo enfoque para análise de sistemas complexos e processos de decisão.

Aplicações específicas sobre o uso da lógica nebulosa em controle surgem em trabalhos de E.H. Mamdani, em 1974.

A partir de meados dos anos 80, o emprego de controladores nebulosos adquiriu aceitação industrial, particularmente no Japão, com aplicações abrangendo desde máquinas fotográficas até processos industriais.

Em 1986, M. Togai e W. Watanabe apresentavam um chip VLSI implementando uma máquina de inferência nebulosa. Hoje em dia, a produção em massa deste tipo de chip tornou corriqueiro o uso de lógica nebulosa até em eletrodomésticos.

1.3.3 Redes Neurais

O conceito de neurônio, como constituinte estrutural primordial do cérebro pode ser atribuído a S. Ramón y Cajál, 1911. Circuitos que buscavam simular neurônios biológicos foram introduzidos por W.S. McCulloch e W. Pitts em 1943 e objetivavam realizar cálculos lógicos. McCulloch era psiquiatra e neuroanatomista, enquanto Pitts era matemático.

O livro “*The Organization of Behavior*” de D.O. Hebb, surgiu em 1949, onde o aprendizado era caracterizado como modificação das sinapses. Em 1956 A.M. Uttley demonstrou que redes neuronais com sinapses modificáveis poderiam classificar padrões binários. Em 1958, F. Rosenblatt introduziu o conceito de perceptron enquanto em 1960 B. Widrow e M.E. Hoff Jr. utilizaram o conceito de mínimos quadrados para formular o Adaline (*Adaptive Linear Element*). Em 1969, M. Minsky e S. Papert publicaram o seu livro sobre perceptrons em que se demonstravam as limitações de perceptrons de uma camada. Este fato, aliado a dificuldades de implementação de hardware, levou muitos pesquisadores a diminuírem as suas atividades neste campo.

Em 1982, J.J. Hopfield utilizou a idéia de funções energia e estabeleceu conexões com a física estatística, levando às redes de Hopfield. Também nesta data, 1982, T. Kohonen publicou seus resultados sobre mapas auto-organizáveis. Em 1983, A.R. Barto, R.S. Sutton e C.W. Anderson introduziram o conceito de *Reinforcement Learning*. Em 1986, o algoritmo de *Back-Propagation* foi desenvolvido por D.E. Rumelhart, G.E. Hinton e R.J. Williams. Em 1988, D.S. Broomhead e D. Lowe descreveram redes empregando funções de base radial (Radial Basis Functions).

1.4 Algumas aplicações típicas de IA

As ferramentas de IA têm se mostrado poderosas e úteis em diversas áreas do conhecimento humano, tais como:

- a) Jogos;
- b) Prova Automática de Teoremas;
- c) Sistemas Especialistas;
- d) Compreensão de Linguagem Natural;
- e) Percepção;
- f) Robótica.

2 LÓGICA E DEDUÇÃO

2.1 Cálculo Sentencial

O objetivo básico do **Cálculo Sentencial** é a verificação da veracidade de sentenças a partir da veracidade ou falsidade dos átomos. As sentenças mais complexas são formadas a partir da combinação de sentenças mais simples através do uso de conectivos sentenciais. As sentenças elementares são chamados de **átomos**.

No **Cálculo Sentencial**, a veracidade que se deseja estabelecer é quanto a legitimidade dos **argumentos** e não dos **átomos** em si.

É importante observar que o uso de símbolos iguais para objetos diferentes pode resultar em conclusões surpreendentes. Assim, usando-se o símbolo Águia para designar um ás da aviação, como o famoso piloto de aviões de caça, Manfred Von Richoffen, ter se ia

(Águia é ave)
(Manfred_Von_Richtoffen é Águia)
(Manfred_Von_Richtoffen é ave)

Os conectivos sentenciais usualmente empregados são \wedge , \vee , \rightarrow , \leftrightarrow , \neg correspondentes a "e", "ou", "implica", "equivale" e "não". Porém, pode-se demonstrar que todas as sentenças podem ser reescritas apenas com o uso de 2 conectivos: $\{\neg, \wedge\}$ ou $\{\neg, \vee\}$ ou $\{\neg, \rightarrow\}$.

Um exemplo de sentença é: "Se (a função $f(\cdot)$ é diferenciável) Então (a função $f(\cdot)$ é contínua)", ou simbolicamente, $a \rightarrow b$, onde $a = (\text{a função } f(\cdot) \text{ é diferenciável})$ e $b = (\text{a função } f(\cdot) \text{ é contínua})$.

Em princípio, a veracidade ou não de uma sentença pode ser aferida com o uso de **Tabelas Verdade**, onde são apresentados os valores da expressão, na forma V = Verdade e F = Falso, para cada combinação de valores para os átomos. Para o caso dos conectivos principais:

a	b	$a \wedge b$	$a \vee b$	$a \rightarrow b$	$a \leftrightarrow b$	$\neg a$
V	V	V	V	V	V	F
V	F	F	V	F	F	F
F	V	F	V	V	F	V
F	F	F	F	V	V	V

Para a verificação da veracidade de $a \rightarrow (b \vee c)$ em função de valores dos átomos a, b e c, pode-se construir a seguinte Tabela de Verdade:

a	b	c	$b \vee c$	$a \rightarrow (b \vee c)$
V	V	V	V	V
V	V	F	V	V
V	F	V	V	V
V	F	F	F	F
F	V	V	V	V
F	V	F	V	V
F	F	V	V	V
F	F	F	F	V

Neste exemplo, nota-se que a expressão $a \rightarrow (b \vee c)$ não assume o valor V para quaisquer valores de a, b, e c. Entretanto, existem expressões para o qual o valor é sempre V, independente do valor de seus átomos. Tais expressões são denominados de tautologias. Algumas tautologias importantes estão listadas a seguir:

1. $a \rightarrow a$	<i>Identidade</i>	(2.1)
2. $a \leftrightarrow a$	<i>Equivalência</i>	(2.2)
3. $(a \wedge a) \leftrightarrow a$	<i>Idempotência</i>	(2.3)
4. $(a \vee a) \leftrightarrow a$	<i>Idempotência</i>	(2.4)
5. $\neg(\neg a) \leftrightarrow a$	<i>Dupla Negação</i>	(2.5)
6. $(a \vee (\neg a))$	<i>Terceiro Excluído</i>	(2.6)
7. $\neg(a \wedge (\neg a))$	<i>Não Contradição</i>	(2.7)
8. $\neg a \rightarrow (a \rightarrow b)$	<i>Negação do Antecedente</i>	(2.8)
9. $(a \rightarrow b) \leftrightarrow (\neg b \rightarrow \neg a)$	<i>Contraposição</i>	(2.9)
10. $\neg(a \wedge b) \leftrightarrow ((\neg a) \vee (\neg b))$	<i>De Morgan</i>	(2.10)
11. $\neg(a \vee b) \leftrightarrow ((\neg a) \wedge (\neg b))$	<i>De Morgan</i>	(2.11)
12. $(a \vee b) \leftrightarrow \neg((\neg a) \wedge (\neg b))$		(2.12)
13. $(a \wedge b) \leftrightarrow \neg((\neg a) \vee (\neg b))$		(2.13)
14. $\neg(a \rightarrow b) \leftrightarrow (a \wedge (\neg b))$		(2.14)
15. $(a \vee b) \leftrightarrow ((\neg a) \rightarrow b)$		(2.15)

2.1.1 Dedução

Um tópico importante dentro do Cálculo Sentencial é o da **Dedução**.

Dedução é a obtenção de uma conclusão verdadeira, a partir de premissas consideradas verdadeiras e de axiomas válidos. Em princípio pode também ser verificada através do uso de tabelas de verdade.

Para denotar que “Os axiomas $a_1 - a_m$ permitem deduzir b , a partir das premissas $p_1 - p_n$ ” é usada a grafia:

$$p_1, \dots, p_n, a_1, \dots, a_m \Rightarrow b \quad (2.16)$$

Como exemplo, considere-se a verificação da legitimidade da dedução:

$$a \rightarrow b, \neg b \Rightarrow \neg a \quad (2.17)$$

a	b	$\neg b$	$a \rightarrow b$	$\neg a$
V	V	F	V	F
V	F	V	F	F
F	V	F	V	V
F	F	V	V	V

Observa-se nesta tabela de verdade que, quando $a \rightarrow b, \neg b$ são, simultaneamente, V tem-se que o valor de $\neg a$ é V.

Uma forma de reduzir o emprego de tabelas de verdade de grandes dimensões é utilizar algumas formas já conhecidas de dedução:

1. $a, a \rightarrow b \Rightarrow b$	<i>Modus Ponens</i>	(2.18)
2. $a \rightarrow b, \neg b \Rightarrow \neg a$	<i>Modus Tollens</i>	(2.19)
3. $a \rightarrow b, b \rightarrow c \Rightarrow a \rightarrow c$	<i>Silogismo Hipotético</i>	(2.20)
4. $\neg a, a \vee b \Rightarrow b$	<i>Silogismo Disjuntivo</i>	(2.21)
5. $a \vee c, (a \rightarrow b) \wedge (c \rightarrow d) \Rightarrow b \vee d$	<i>Dilema Construtivo</i>	(2.22)
6. $((\neg b) \vee (\neg d)), (a \rightarrow b) \wedge (c \rightarrow d) \Rightarrow (\neg a) \vee (\neg c)$	<i>Dilema Dedutivo</i>	(2.23)
7. $a \wedge b \Rightarrow a$	<i>Simplificação</i>	(2.24)
8. $a, b \Rightarrow a \wedge b$	<i>Conjunção</i>	(2.25)
9. $a \Rightarrow a \vee b$	<i>Adição</i>	(2.26)
10. $\neg a, a \Rightarrow b$	<i>Dedução por Absurdo</i>	(2.27)

2.2 Cálculo de Predicados

O cálculo de predicados também estuda a legitimidade ou não de sentenças, mas agora incluindo os efeitos dos quantificadores (\forall e \exists), bem como os predicados (verbos e objetos).

Considere a seguinte dedução, com o *quantificador universal* \forall :

$$\begin{array}{c} (\text{TODAS as crianças estudam}) \\ (\text{Alice é uma criança}) \\ \hline (\text{Alice estuda}) \end{array}$$

que pode ser representada simbolicamente por:

$$\{(\forall x [\text{Criança}(x) \rightarrow \text{Estuda}(x)]) \square (\text{Criança}(\text{Alice}) = V)\} \Rightarrow \text{Estuda}(\text{Alice}) = V. \quad (2.28)$$

A importância do quantificador pode ser facilmente ser percebida examinando-se a mesma estrutura anterior, com o *quantificador existencial* \exists no lugar de \forall :

$$\begin{array}{c} (\text{EXISTEM crianças que estudam}) \\ (\text{Alice é uma criança}) \\ \hline (\text{Alice estuda}) \end{array}$$

que pode ser representada simbolicamente por:

$$\{(\exists x [\text{Criança}(x) \wedge \text{Estuda}(x)]) \wedge (\text{Criança}(\text{Alice}) = V)\} \Rightarrow \text{Estuda}(\text{Alice}) = V \quad (2.29)$$

Obviamente, esta segunda dedução é incorrecta.

A importância do predicho é evidenciada pelo seguinte exemplo, onde $\text{Filho}(x,z)$ significa que x é filho de z e $\text{Irmão}(x,y)$ significa que x é irmão de y :

$$\begin{aligned} & \forall x \forall y (\exists z [\text{Filho}(x,z) \wedge \text{Filho}(y,z)] \rightarrow \text{Irmão}(x,y)) \\ & \text{Filho}(\text{Charles}, \text{Elizabeth}) = V \\ & \text{Filho}(\text{Andrew}, \text{Elizabeth}) = V \end{aligned}$$

A partir destas expressões devemos concluir que:

$$\text{Irmão}(\text{Andrew}, \text{Charles}) = V.$$

Até agora foram utilizadas nas representações as combinações (\forall com \rightarrow) e (\exists com \square):

$$\forall x [\text{Criança}(x) \rightarrow \text{Estuda}(x)] \text{ e } \exists x [\text{Criança}(x) \wedge \text{Estuda}(x)]$$

Será avaliada, então, a conveniência ou não das combinações (\forall com \wedge) e (\exists com \rightarrow):

$$\exists x [\text{Criança}(x) \rightarrow \text{Estuda}(x)] \text{ e } \forall x [\text{Criança}(x) \wedge \text{Estuda}(x)].$$

Lembrando que $(a \rightarrow b) \leftrightarrow ((\neg a) \vee b)$, o primeiro termo permite escrever

$$\exists x [\text{Criança}(x) \rightarrow \text{Estuda}(x)] \leftrightarrow \exists x [(\neg \text{Criança}(x)) \vee \text{Estuda}(x)]$$

ou seja, existe alguém que não seja criança ou que estude. É uma sentença de pouco interesse. Por outro lado, aplicando a propriedade que $a \wedge b \Rightarrow a$, tem-se a partir do segundo termo que $\text{Criança}(x) \wedge \text{Estuda}(x) \Rightarrow \text{Criança}(x)$, de onde se conclui que:

$$\forall x \text{ Criança}(x)$$

que é também de pouco interesse.

Algumas expressões úteis envolvendo quantizadores estão listadas a seguir:

1. $\forall x A(x) \rightarrow \exists x A(x)$ (2.30)
2. $\forall x A(x) \leftrightarrow \neg(\exists x (\neg A(x)))$ (2.31)
3. $\neg(\forall x A(x)) \leftrightarrow \exists x (\neg A(x))$ (2.32)
4. $\exists x A(x) \leftrightarrow \neg(\forall x (\neg A(x)))$ (2.33)
5. $\neg(\exists x A(x)) \leftrightarrow \forall x (\neg A(x))$ (2.34)
6. $\forall x (A(x) \wedge B(x)) \leftrightarrow (\forall x A(x) \wedge \forall x B(x))$ (2.35)
7. $\exists x (A(x) \wedge B(x)) \rightarrow (\exists x A(x) \wedge \exists x B(x))$ (2.36)
8. $((\forall x A(x)) \vee (\forall x B(x))) \rightarrow \forall x (A(x) \vee B(x))$ (2.37)
9. $((\exists x A(x)) \vee (\exists x B(x))) \leftrightarrow \exists x (A(x) \vee B(x))$ (2.38)
10. $\forall x (A(x) \rightarrow B(x)) \rightarrow (\forall x A(x) \rightarrow \forall x B(x))$ (2.39)

O processo de dedução, no cálculo de predicados, é similar ao cálculo sentencial, exceto por passos adicionais para eliminação ou introdução de quantificadores.

A eliminação de \forall recebe o nome de IU (Instanciação Universal) e a introdução de \exists de GE (Generalização Existencial).

A regra IU não apresenta dificuldades, uma vez que a validade de $\forall x (A(x) \rightarrow B(x))$ permite efetuar uma substituição x' de x de modo que a instância $A(x') \rightarrow B(x')$ seja válida.

Também GE não apresenta dificuldades, pois se $(A(x) \wedge B(x))$ é válida para x genérico, então $\exists x (A(x) \wedge B(x))$.

Por outro lado deve-se tomar cuidado com a introdução de \forall que recebe o nome de GU (Generalização Universal) e a eliminação de \exists , ou IE (Instanciação Existencial).

Para ilustrar o efeito de não se ter cautela em dedução, considere o contra exemplo, onde Filho(x, z) significa que x é filho de z , ou alternativamente z é mãe de x :

- i. $\forall x (\exists z \text{Filho}(x,z))$; premissa. (2.40)
- ii. $\exists z \text{Filho}(x',z)$; IU em i. (2.41)
- iii. $\text{Filho}(x',z')$; IE em ii. (2.42)
- iv. $\forall x \text{Filho}(x,z')$; GU incorrecto em iii, pois anteriormente, em iii, ligou-se a variável z' a x' , afirmando no caso, que x' é filho de z' . (2.43)
- v. $\exists z (\forall x \text{Filho}(x,z))$; GE em iv. (2.44)

Enquanto a interpretação de (i) é "todos tem uma mãe", a de (v) é "existe uma mãe de todos".

2.2.1 Resolução

Uma forma de verificar a legitimidade ou não de uma sentença é a utilizar o processo de resolução. Para ilustrar o conceito fundamental envolvido na resolução, sejam as premissas:

$$(\neg \text{Musculoso}(x)) \vee \text{Forte}(x)$$

$$\text{Musculoso}(\text{Tyson})$$

das quais pode-se concluir que $\text{Forte}(\text{Tyson})$, pois não é admissível ter $\text{Musculoso}(\text{Tyson})$ e $\neg \text{Musculoso}(\text{Tyson})$. A resolução envolve, portanto, a utilização de premissas envolvendo conectivos \vee e critérios do tipo A e $\neg A$.

As sentenças constituídas de disjunções são ditas estarem na forma de cláusulas. As sentenças bem formadas podem sempre ser colocadas na forma de cláusula (as sentenças malformadas usualmente não fazem sentido, como $\forall x[A(x) \neg \rightarrow \exists \neg x]$).

O processo de redução de uma sentença à forma de cláusula pode ser ilustrada com o exemplo:

$$\forall x[\text{A}(x) \rightarrow ((\forall y \text{A}(y)) \vee (\neg \forall y (\neg \text{B}(x,y))))] \quad (2.45)$$

Passo 1: Eliminar \rightarrow usando $(a \rightarrow b) \leftrightarrow ((\neg a) \vee b)$

$$\forall x[\neg \text{A}(x) \vee ((\forall y \text{A}(y)) \vee (\neg \forall y (\neg \text{B}(x,y))))] \quad (2.46)$$

Passo 2: Trazer \neg junto aos átomos:

$$\forall x[\neg \text{A}(x) \vee ((\forall y \text{A}(y)) \vee (\exists y \text{B}(x,y)))] \quad (2.47)$$

Passo 3: Eliminar a repetição de símbolos iguais empregados para variáveis não relacionadas:

$$\forall x[\neg A(x) \vee ((\forall y A(y)) \vee (\exists z B(x,z)))] \quad (2.48)$$

Passo 4: Eliminar os quantificadores existenciais. Em uma expressão do tipo $\forall x[\exists z B(x,z)]$, para cada x , existe um z , possivelmente dependente de x , que afirma $B(x,z)$. Portanto, pode-se definir um mapa que associa a cada x , o seu z correspondente, $h:x \rightarrow z$, ou seja, $z = h(x)$. Este tipo de mapa é chamado de função de Skolem. No presente caso,

$$\forall x[\neg A(x) \vee ((\forall y A(y)) \vee B(x,h(x)))] \quad (2.49)$$

Passo 5: Converter à forma *prenex*. A forma prenex é aquela em que os quantificadores estão no início da expressão. Já que cada quantificador universal possui a sua própria *dummy variable*, não há possibilidade de embaralhamento das variáveis ao se mover o símbolo \forall para o início da sentença. A expressão entre os colchetes, recebe o nome de matriz

$$\forall x \forall y [\neg A(x) \vee (A(y) \vee B(x,h(x)))] \quad (2.50)$$

Passo 6: Colocar a matriz na forma conjuntiva normal, ou seja,

$$((a_{11} \vee a_{12} \vee \dots \vee a_{1n}) \wedge \dots \wedge (a_{m1} \vee a_{m2} \vee \dots \vee a_{mp})) \quad (2.51)$$

que, no caso em questão se reduz à eliminação de parênteses:

$$\forall x \forall y [\neg A(x) \vee A(y) \vee B(x,h(x))] \quad (2.52)$$

Estando na forma conjuntiva normal, pode-se escrever, após IU:

$$\begin{aligned} 1: & \quad a_{11} \vee a_{12} \vee \dots \vee a_{1n} \\ \vdots & \quad \dots \end{aligned} \quad (2.53)$$

m: $a_{m1} \vee a_{m2} \vee \dots \vee a_{mp}$
onde se aplica, facilmente, o processo de resolução.

Considere o seguinte exemplo de dedução, a ser avaliada quanto à sua legitimidade:

$$\begin{aligned} & \forall x[\text{Musculoso}(x) \rightarrow \text{Forte}(x)] \\ & \forall x[\text{Jóquei}(x) \rightarrow (\neg \text{Forte}(x))] \\ & \exists x[\text{Jóquei}(x) \wedge \text{Alto}(x)] \\ & \Rightarrow \\ & \exists x[\text{Alto}(x) \wedge (\neg \text{Musculoso}(x))] \end{aligned}$$

A avaliação da legitimidade pode ser feita por refutação, negando-se a tese e verificando que isso leva a contradição. Este método (prova por absurdo) é muito usado em cursos de matemática.

$$i. \forall x[\text{Musculoso}(x) \rightarrow \text{Forte}(x)] \quad ; \text{premissa.} \quad (2.54)$$

$$ii. \forall y[\text{Jóquei}(y) \rightarrow (\neg \text{Forte}(y))] \quad ; \text{premissa.} \quad (2.55)$$

$$iii. \exists z[\text{Jóquei}(z) \wedge \text{Alto}(z)] \quad ; \text{premissa.} \quad (2.56)$$

$$iv. \forall w[\neg \text{Alto}(w) \vee \text{Musculoso}(w)] \quad ; \text{premissa obtida por negação da tese.} \quad (2.57)$$

$$v. (\neg \text{Musculoso}(x)) \vee \text{Forte}(x) \quad ; \text{cláusula derivada de } i. \quad (2.58)$$

$$vi. (\neg \text{Jóquei}(y)) \vee (\neg \text{Forte}(y)) \quad ; \text{cláusula derivada de } ii. \quad (2.59)$$

$$vii. \text{Jóquei}(z') \wedge \text{Alto}(z') \quad ; \text{cláusula derivada de } iii. \quad (2.60)$$

$$viii. \text{Jóquei}(z') \quad ; \text{simplificação de } vii. \quad (2.61)$$

$$ix. \text{Alto}(z') \quad ; \text{simplificação de } vii. \quad (2.62)$$

$$x. \neg \text{Alto}(w) \vee \text{Musculoso}(w) \quad ; \text{cláusula derivada de } iv. \quad (2.63)$$

$$xi. \text{Musculoso}(z') \quad ; \text{resolução } ix, x. \quad (2.64)$$

$$xii. \text{Forte}(z') \quad ; \text{resolução } v, xi. \quad (2.65)$$

$$xiii. \neg \text{Jóquei}(z') \quad ; \text{resolução } vi, xii. \quad (2.66)$$

$$xiv. \exists x[\text{Alto}(x) \wedge (\neg \text{Musculoso}(x))] \quad ; \text{por absurdo, viii, xiii} \quad (2.67)$$

3 REPRESENTAÇÃO DE CONHECIMENTOS

Conhecimento pode ser definido como sendo um conjunto de informações que permitem articular os conceitos, os juízos e o raciocínio, usualmente disponíveis em um domínio particular de atuação. Empregando-se um sistema de símbolos, este conhecimento pode ser representado de modo a permitir atividades como inferência ou memorização.

3.1 Principais Formas de Representação do Conhecimento em IA

- Representação por lógica matemática;
- Representação por regras de produção;
- Representação por redes semânticas;
- Representação por quadros e roteiros (*frames* e *scripts*).

3.1.1 Representação Utilizando Lógica Matemática

A lógica é muito antiga, remontando ao tempo dos filósofos gregos. Este tipo de representação é bastante formal apresentando grande dificuldade de legibilidade e expressão do conhecimento por parte dos leigos.

3.1.2 Representação Utilizando Regras de Produção

Uma forma muito comum de representar conhecimentos heurísticos é usar regras de produção, tipicamente na forma Se (condições) Então (conclusões). Embora regras de produção sejam apenas casos particulares de sentenças tratáveis pela Lógica de Predicados, muitos sistemas práticos empregam esta estrutura de representação. Muitos sistemas especialistas utilizam regras de produção aliadas a uma máquina de inferência para realizar a sua tarefa. Como exemplo de regras de produção considere a clássica representação de conhecimentos sobre alguns animais.

i.	Se (produz_leite \wedge tem_pelos)	Então (mamífero)
ii.	Se (mamífero \wedge come_carne)	Então (carnívoro)
iii.	Se (mamífero \wedge possui_presas \wedge possui_garras)	Então (carnívoro)
iv.	Se (mamífero \wedge possui_casco)	Então (ungulado)
v.	Se (carnívoro \wedge pardo \wedge pintado)	Então (onça)
vi.	Se (carnívoro \wedge pardo \wedge listrado)	Então (tigre)
vii.	Se (ungulado \wedge pardo \wedge pintado)	Então (girafa)
viii.	Se (ungulado \wedge branco \wedge listrado)	Então (zebra)

onde, a regra Se (produz_leite \wedge tem_pelos) Então (mamífero) pode ser entendida como uma instância de " x [(produz_leite(x) \wedge tem_pelos(x)) \rightarrow mamífero(x)], como visto no capítulo 2.

A partir de fatos conhecidos sobre um dado animal A, do tipo {tem_pelos(A), produz_leite(A), come_carne(A), pardo(A), pintado(A)}, pode-se concluir que A é uma (onça). Quando se busca, a partir dos dados, verificar qual a conclusão a ser obtida, diz-se que o encadeamento é progressivo (acompanha a direção apontada pela \rightarrow).

Uma outra forma de usar os conhecimentos na forma de regras de produção é partir das conclusões que poderão ser provadas ou não. Conjecturando que A é (girafa), verifica-se, a partir de vii, que se necessitam das condições {ungulado, pardo, pintado}. As condições {pardo, pintado} já estão fornecidas. Entretanto, há necessidade de verificar se a condição {ungulado} é satisfeita. Para tal, deve-se examinar a regra iv que agora requer as condições {mamífero, possui_casco}. Como não há nenhuma regra que permita verificar se {possui_casco} é verdadeira e como esse fato também não é fornecido como dado, a tentativa de provar que A é (girafa) resulta em falha. Por outro lado, o mesmo procedimento aplicado a (onça) fornece condições que podem ser deduzidas dos dados fornecidos. Quando se busca, a partir dos objetivos (goals), as condições necessárias para a sua dedução, até que se casem com os dados fornecidos, diz-se que o encadeamento é retroativo (a busca ocorre no sentido oposto ao indicado pela \rightarrow).

3.1.3 Representação Utilizando Redes Semânticas

A rede semântica é uma coleção de nós conectados por arcos. Os nós representam objetos, conceitos ou eventos e os arcos representam as relações. Usualmente os nós e os arcos são etiquetados para indicar o que representam:

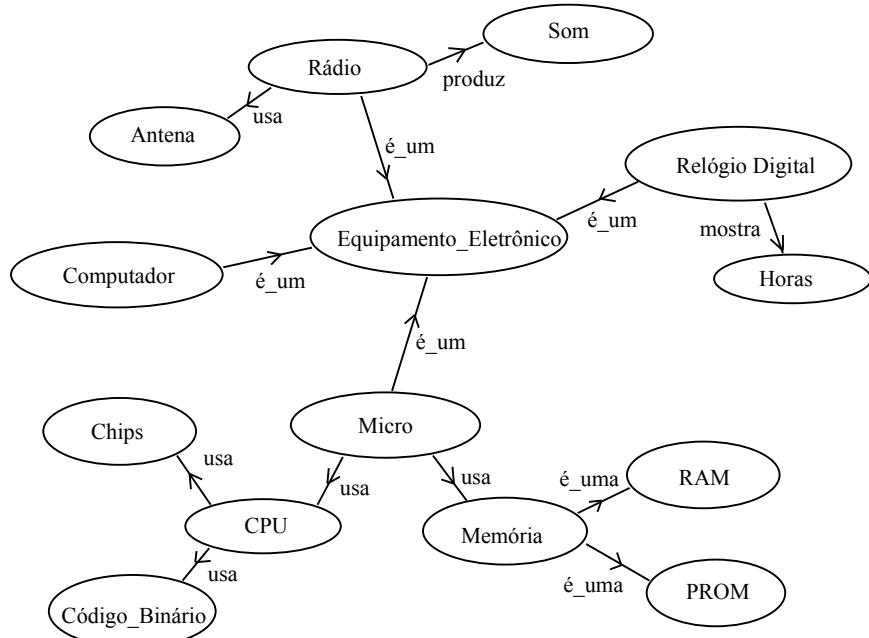


Figura 3-1. Uma rede semântica representando conhecimentos sobre equipamentos eletrônicos.

Uma forma de organizar redes semânticas simples é utilizando triplas (atributo, objeto, valor):

	<i>Atributo</i>	<i>objeto</i>	<i>valor</i>
Se	Corrente	cabo_29	<10mA
	Luminosidade	lâmpada_piloto	baixa
	Resposta	tecla_TESTE	negativa
Então	Carga	bateria	esgotada

3.1.4 Representação Utilizando Frames e Scripts

Os frames ou quadros são estruturas de dados com lacunas que podem ser preenchidas com informações declarativas ou procedimentais. No exemplo da figura 5.2, tanto XYZ quanto ABC são manipuladores mecânicos. No caso do manipulador XYZ, o efetor é do tipo garra, tipo bidigital, com sensor tátil, capacidade de 10kg e acionado por tendão. No caso do manipulador ABC, o efetor é uma pistola de pintura.

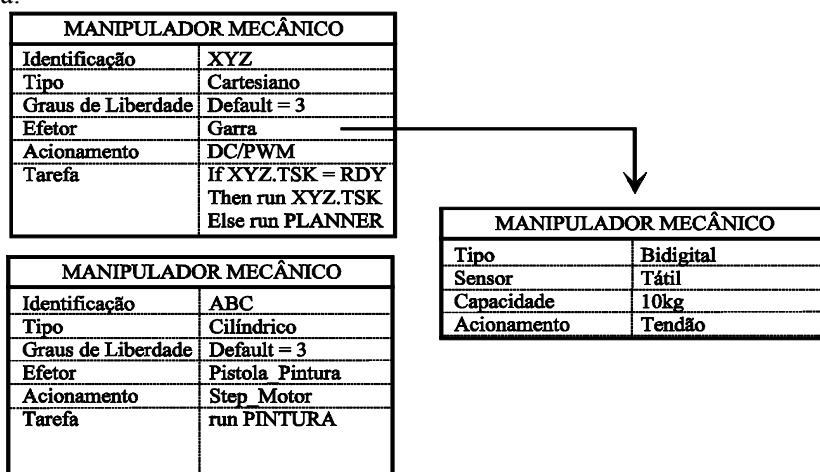


Figura 3-2. Representação de conhecimentos sobre manipuladores através de *frames*.

3.2 Métodos de busca.

Muitos problemas de IA necessitam de métodos de casamento (*matching*) e busca (*search*), durante o processo de solução.

Considere-se o problema de obter exatamente 1 litro de água em uma das jarras, a partir de uma jarra com 5 litros de capacidade, inicialmente cheia, e uma de 2 litros, inicialmente vazia. Representando as duas jarras por um par ordenado (x,y) , onde x é o número de litros de água na primeira jarra e y na segunda, os passos para a solução do problema podem ser apresentados na seguinte forma gráfica:

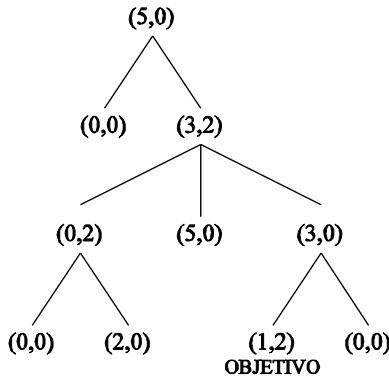


Figura 3-3. Árvore correspondente ao problema de jarras d'água.

O problema das jarras pode, ser resolvido mediante o uso de um algoritmo de busca em árvores. Aqui, a partir do nó $(5,0)$, é feita uma busca que terminaria ao se encontrar o nó objetivo $(1,2)$.

Como mencionado anteriormente, a busca pode ser realizada nos sentidos de encadeamento progressivo ou retroativo. A vantagem de um ou outro depende da particular estrutura de árvore a ser pesquisada. Em termos intuitivos, a verificação de se uma dada pessoa é avô de uma outra é mais facilmente feita no sentido neto \rightarrow pais \rightarrow avô do que no sentido avô \rightarrow pais \rightarrow neto. Tal característica se deve ao fato que cada um (neto) tem apenas 2 avôs (materno e paterno), portanto, apenas 2 candidatos a serem pesquisados. Por outro lado, se um avô gerou 5 filhos, e cada filho gerou 5 netos, tem-se que para verificar se uma dada pessoa é neto deste avô, há necessidade de pesquisa entre 25 candidatos.

O desenrolar de jogos, como xadrez, damas, e jogo da velha, também pode ser caracterizado por árvores. Para que se possam selecionar as opções mais promissoras para o jogo, há necessidade de uma heurística que permita fornecer, a cada jogada, um índice que reflete a vantagem posicional alcançada pelo movimento escolhido. Uma heurística possível é contar, em cada situação de jogo, a diferença entre o número de alinhamentos possíveis para cada jogador. Para simplificar a explicação, considere-se uma notação análoga à utilizada em teoria de matrizes, ou seja, a posição na i -ésima linha e j -ésima coluna é denotada (i,j) . Nestas condições, a situação ilustrada na Figura 3-4 possui um símbolo (O) na posição $(1,2)$ e um símbolo (x) na posição $(2,2)$. A partir desta situação, o jogador que utiliza o símbolo (O) pode obter 4 alinhamentos enquanto o que utiliza (x) consegue 6. Portanto, o jogador (x) atribui um valor $+2 = +6 - 4$ para esta situação.

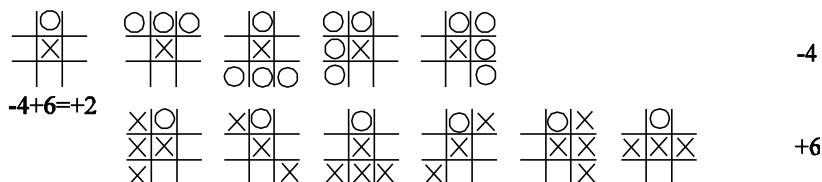


Figura 3-4. A primeira linha apresenta os alinhamentos possíveis (4) para o jogador (O) e a segunda os possíveis (6) para o jogador (x). A diferença $-4+6=+2$ é o índice para a configuração no canto superior esquerdo.

A árvore mostrada na Figura 3-5 ilustra o mecanismo que o jogador (x) utiliza para posicionar a sua ficha. Caso o jogador (x) positione a ficha na posição $(1,1)$, o jogador (O) poderá obter valor -1

através da ocupação de (2,2), que é a situação mais favorável para (O). Se o jogador (O) ocupar a posição (2,1), por exemplo, obtém apenas valor +1, ou seja pior para (O). Note-se que o melhor caso para (O) será também o pior caso para (x). Assim, considerando-se as possibilidades para a primeira jogada de (x), os valores associados são:

- (x) em (1,1) -> valor -1,
- (x) em (2,2) -> valor +1,
- (x) em (2,1) -> valor -2.

Portanto, considerando o melhor dos piores casos, o jogador (x) deve optar por colocar a sua ficha em (2,2), que corresponde ao máximo entre -1, +1 e -2. Por outro lado, o valor -1 corresponde ao mínimo entre 1, 0, 1, 0 e -1 da sub-árvore à esquerda da Figura 3-5. Analogamente, o valor -2 corresponde ao mínimo entre -1, -2, 0, 0, e -1, da sub-árvore à direita da Figura 3-5. Esta estratégia de decisão é chamada de ‘max-min’, ou seja, ‘maximizar o ganho considerando os piores casos’.

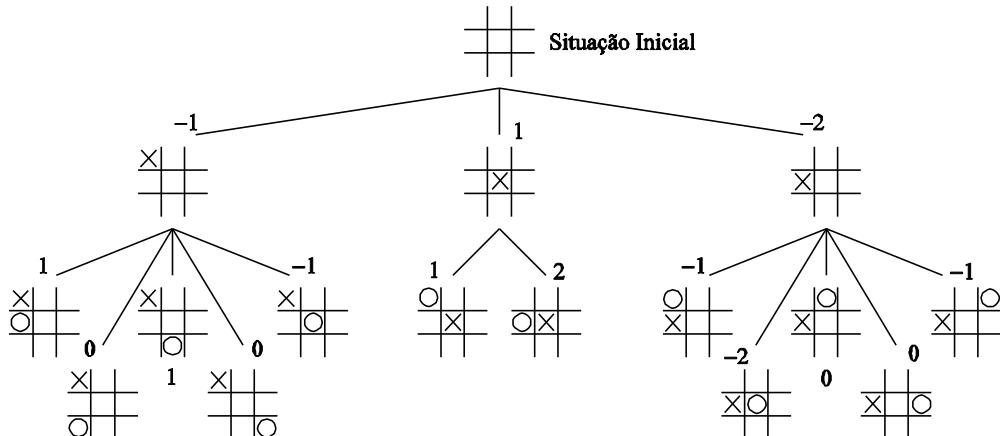


Figura 3-5. Construção da árvore correspondente ao jogo da velha, utilizando a heurística de diferenças entre o número de alinhamentos possíveis para cada jogador, a partir do tabuleiro vazio (situação inicial). Note que as alternativas simétricas não estão aqui ilustradas.

De um modo geral, os processos de busca podem ser, conforme mencionado anteriormente, do tipo encadeamento progressivo (*data driven*) ou retroativo (*goal driven*). Ainda, dependendo de como se executa o controle sobre a busca, pode ser do tipo busca em profundidade ou busca em largura.

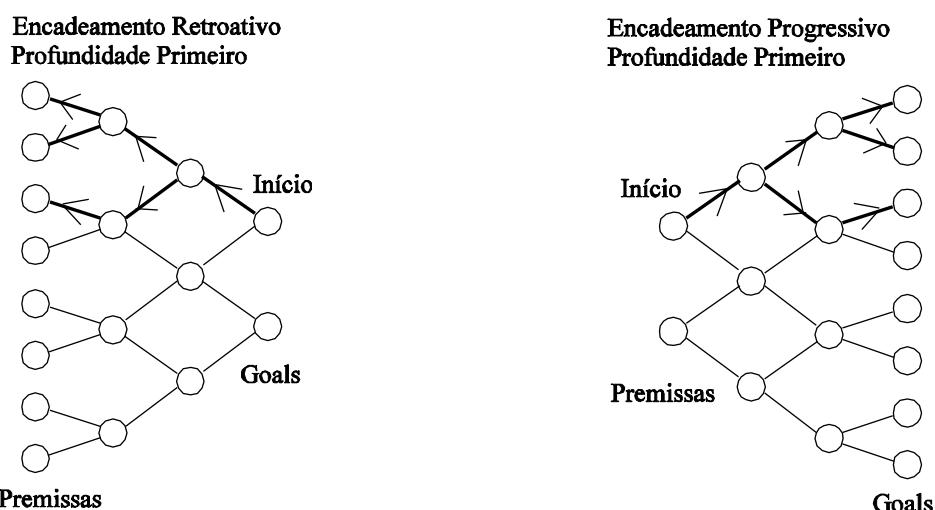


Figura 3-6. Ilustração das formas de encadeamento retroativo e progressivo para o caso de busca em profundidade.

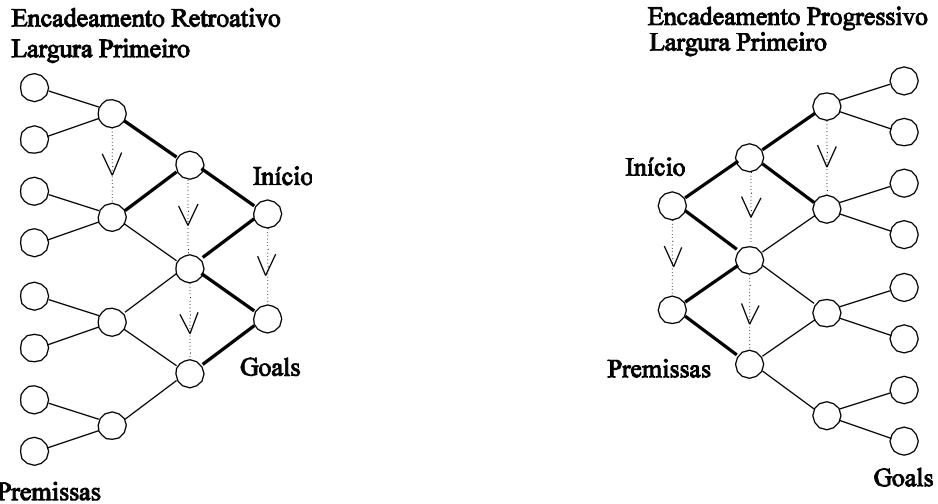


Figura 3-7. Ilustração das formas de encadeamento retroativo e progressivo / busca em largura.

3.2.1 Algoritmo A*

Um exemplo de método de busca muito usado é o algoritmo A*. Este algoritmo é aplicável quando se dispõe de uma função de avaliação heurística do custo, denotada $f'(\cdot)$. Em muitas aplicações, o custo incorrido ao percorrer, a partir de S , um certo caminho até o nó N , pode ser calculado através de uma função $g(\cdot)$. Se for disponível uma função heurística $h'(\cdot)$ que permite estimar o custo para percorrer a árvore de N até o nó objetivo G (goal), então, $f'(N) = g(N) + h'(N)$. No caso de navegação de um robô móvel, a distância direta (sem considerar obstáculos como morros e lagos) poderia ser considerada uma função heurística para avaliar a conveniência ou não de se tomar uma decisão sobre a direção a ser seguida.

Passo 0. Criar um grafo de busca G , inicialmente contendo somente o nó de partida S .

Passo 1. Inicializar uma lista chamada OPEN com o nó de partida S e uma lista chamada CLOSED com \emptyset . (OPEN é a lista dos nós ainda não selecionados para expansão, enquanto CLOSED é a lista dos nós já expandidos)

Passo 2. LOOP: Se $OPEN = \emptyset$

Então FIM e reportar Falha.

Passo 3. Tomar um nó de OPEN com o menor f' .

Chamar este nó de N .

Retirá-lo de OPEN e colocá-lo em CLOSED.

Passo 4. Se N é o nó buscado (*goal*)

Então FIM e fornecer a solução percorrendo os ponteiros de N a S .

Passo 5. Expandir N , gerando o conjunto de nós SUCESSORES e então adicioná-los no grafo.

Passo 6. Para cada membro M de SUCESSORES:

Se $M \notin \{OPEN, CLOSED\}$

Então adicionar M em OPEN;

Se $M \in OPEN$

Então chamar a cópia de M em OPEN de M^{old} ; verificar se $g(M) < g(M^{old})$:

em caso afirmativo, tem-se que é melhor chegar a M através do caminho atual do que a indicada pelo ponteiro de M^{old} . Portanto, descarta-se M^{old} e faz-se o M apontar para N .
em caso negativo, basta que se remova M .

Se $M \in CLOSED$

Então chamar a cópia de M em CLOSED de M^{old} ; de modo similar ao caso ($M \in OPEN$), verificar qual o melhor caminho para M e atualizar os valores de g e f' .

Passo 7. Reordenar OPEN com base no f' .

Passo 8. Return LOOP;

4 CONTROLADORES BASEADOS EM CONHECIMENTOS

Regras de produção do tipo **Se (condições) Então (ações)** podem ser utilizadas para incorporar à máquina a experiência heurística do operador humano. Entre as tarefas que podem ser realizadas por sistemas baseados em conhecimento estão: Interpretação de dados, predição, diagnose, síntese, planejamento, monitoração, correção de falhas, treinamento e controle ativo.

A arquitetura mais simples de um sistema baseado em conhecimentos envolve um banco de conhecimentos, onde as regras estão armazenadas, um banco de dados, onde as informações sobre as condições da planta a ser controlada e as medidas estão armazenadas e uma máquina de inferência que deverá deduzir as ações a serem tomadas, em função das informações dos bancos de dados e de conhecimentos.

A máquina de inferência executa, usualmente, os seguintes passos:

Passo 1: Busca de regras no banco de conhecimentos que tenham as condições satisfeitas, em termos do conteúdo do banco de dados (casamento do antecedente).

Passo 2: Se houver uma ou mais regras com as condições satisfeitas
Então selecionar uma (resolução de conflito)
Senão, retornar ao Passo 1.

Passo 3: Executar a ação descrita na regra selecionada e retornar ao Passo 1.

As regras de produção podem ser utilizadas em sistemas de controle em três níveis hierárquicos:

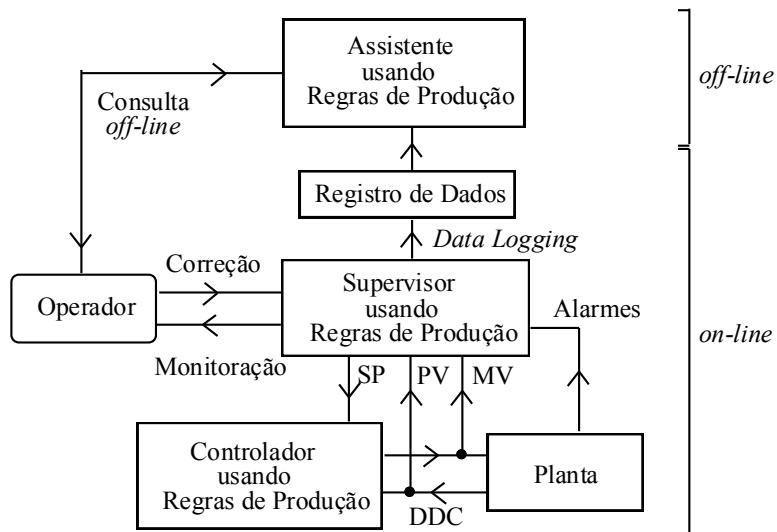


Figura 4-2. Esquema de um sistema de controle empregando representação de conhecimentos na forma de regras de produção. As siglas utilizadas são: SP = Set Point, PV = Process Variable, MV = Manipulated Variable e DDC = Direct Digital Control.

O nível hierárquico inferior corresponde ao Controle Direto Digital, empregando controladores baseados em conhecimentos, no caso, representados por regras de produção.

O nível hierárquico intermediário compreende a utilização de bases de conhecimentos para supervisionar a operação geral da planta controlada, em tempo real, aquisicionando sinais de alarmes e

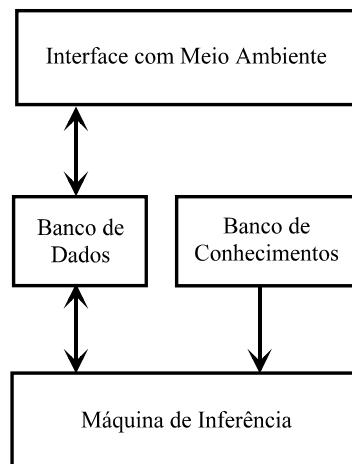


Figura 4-1. Arquitetura típica de um sistema baseado em conhecimentos.

de controle, podendo interagir com o operador humano em termos de correção e monitoração do sistema. O supervisor pode ter funções de decisão como reconfiguração de controladores após testes de desempenho do sistema, partida e desligamento inteligentes, detecção e acomodação de falhas, entre outras.

O nível hierárquico superior corresponde a operação *off-line* para dar assistência ao operador, quer na forma de consulta sobre procedimentos operacionais de rotina, quer no treinamento de novos recursos humanos. Pode, por exemplo, auxiliar equipes de manutenção, em função de diagnósticos aprofundados a partir dos dados armazenados durante a operação da planta.

4.1 Exemplo de Controlador usando Regras de Produção.

Como exemplo de um controlador usando regras de produção, será apresentada uma versão desenvolvida no trabalho de Cardozo e Yoneyama, 1987 e Yoneyama, 1988, onde se considera uma planta não linear descrita por uma equação diferencial ordinária não linear e invariante no tempo:

$$\begin{aligned}\frac{dx_1(t)}{dt} &= x_2(t) \\ \frac{dx_2(t)}{dt} &= -\beta[x_2(t)]^3 + u(t)\end{aligned}\quad (4.1)$$

onde, $\beta = 1$ é uma constante e, no instante inicial $t=0$, o sistema está em repouso ($x_1(t) = 0$ e $x_2(t) = 0$).

Um controlador que pode ser utilizado para esta planta é da forma:

$$u(t) = -k \times \text{sat}[K, (y(t) - y_r(t))] \quad (4.2)$$

onde a função $\text{sat}[\dots]$ é dada por:

$$\text{sat}[M, z] = \begin{cases} z, & \text{se } |z| \leq M \\ M \text{sign}(z), & \text{se } |z| > M \end{cases} \quad (4.3)$$

De modo heurístico, foram propostas as seguintes regras para ajuste *on-line* de k e K , admitindo que o nível de saturação do controle é $K = 5.0$ e velocidades adequadas estão na faixa de $V = 0.18$:

REGRA 1: Se $(x_2(t) \leq V) \wedge (|y(t) - y_r(t)| < 0.2) \wedge (x_1(t) < 1.0)$
Então $(k \leftarrow -50.0)$

REGRA 2: Se $(x_2(t) > V) \wedge (|y(t) - y_r(t)| < 0.2) \wedge (x_1(t) > 1.0)$
Então $(k \leftarrow 50.0)$

REGRA 3: Se $(x_2(t) \leq -V) \wedge (|y(t) - y_r(t)| < 0.2) \wedge (x_1(t) > 1.0)$
Então $(k \leftarrow -50.0)$

REGRA 4: Se $(|x_2(t)| \leq -V) \wedge (|y(t) - y_r(t)| < 0.2)$
Então $(k \leftarrow 1.0)$

REGRA 5: Se $(|x_2(t)| \leq -V) \wedge (x_1(t) < 1.0)$
Então $(k \leftarrow 1.0)$

Uma vez que as regras 1 a 5 promovem um controle com saída levemente oscilatória, foi introduzida uma regra adicional:

REGRA 6: Se $(t > 2.0)$
Então $(tc \leftarrow tc + 0.05) \wedge (V \leftarrow V/tc)$

4.2 Exemplo de um Supervisor Inteligente

Como exemplos de supervisores inteligentes podem ser citados os sistemas desenvolvidos em Pereira, Yoneyama e Cardozo, 1990, Pereira, Cardozo e Yoneyama, 1991 e Fujito e Yoneyama, 1992. A grande virtude de se utilizar supervisores inteligentes é a possibilidade de realizar processamentos simbólico e numérico de forma simultânea, valendo-se das vantagens de ambos.

Um destes exemplos consiste de um supervisor inteligente para sintonização fina automática de controladores PID.

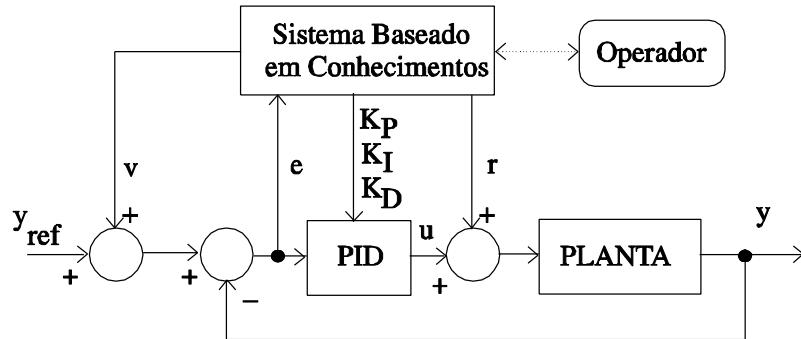


Figura 4-3. Sintonização fina automática de Controladores PID usando base de conhecimentos.

Considera-se aqui que o controlador já foi sintonizado inicialmente utilizando métodos como os de Ziegler-Nichols. Nestas condições, pretende-se, obter uma sintonização fina através do emprego de regras heurísticas:

- i) Aplica-se um sinal $v(t) = \text{degrau}$ e a partir de $e(t)$, calcula-se $y(t)$

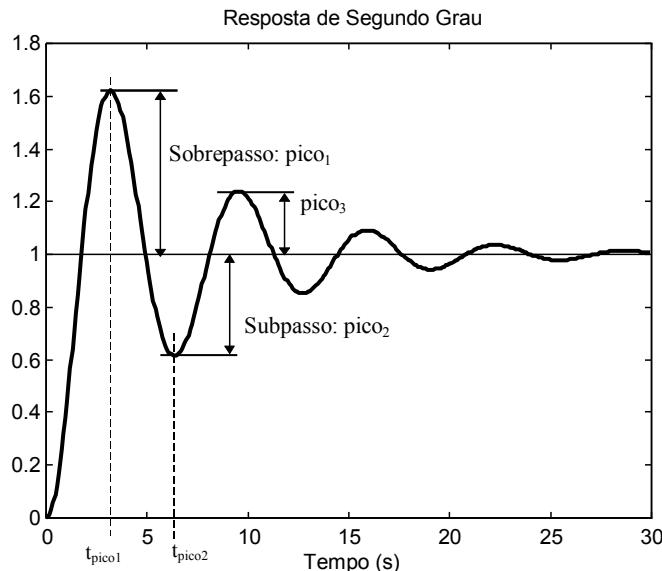


Figura 4-4. Parâmetros para o refinamento da heurística na sintonização de controladores PID.

- ii) Detectam-se os três primeiros picos na saída $y(t)$ e calculam-se os parâmetros:

- Sobrepasso Máximo da Variável de Saída;
- Subpasso Máximo da Variável de Saída;
- Período de Oscilação Amortecida: $T_0 = 2 \times (t_{pico2} - t_{pico1})$;
- Razão entre os dois primeiros picos: $r_{12} = \frac{|pico2|}{|pico1|}$;

- Índice de Convergência: $cnv = \frac{|pico2| + |pico3|}{|pico1| + |pico2|}$;

iii) Verifica-se a adequação da atual resposta do sistema. Em caso afirmativo, o supervisor entra em modo de espera. Senão, calcula-se a contribuição de fase, na frequência $2\pi/T_0$

$$\phi_{PID} = \tan^{-1} \left\{ \frac{2\pi K_D}{T_0 K_P} - \frac{T_0 K_I}{2\pi K_I} \right\} \quad (4.4)$$

e reajustam-se K_D e K_I de modo que ϕ_{PID} seja minimizado.

iv) K_P é multiplicado pela razão entre o sobrepasso desejado e o sobrepasso medido.

v) Recalcula-se K_I e K_D e retorna-se ao passo i.

Aplicando-se estas regras ao sistema:

$$G(s) = \frac{10 \exp(-0.5s)}{s + 3} \quad (4.5)$$

onde se especificou sobrepasso máximo de 15% e subpasso máximo de 7.5%, resultou

Iteração	K_P	K_I	K_D	Sobrepasso	Subpasso
1º	0.268	0.417	0.051	23.5%	17.6%
2º	0.268	0.625	0.052	55.3	30.7
3º	0.147	0.342	0.029	6.25	0.84

Uma dificuldade com este tipo de enfoque é a necessidade de regras heurísticas de qualidade para que a planta seja operada com segurança, o que não é sempre disponível. Ainda, é importante observar que regras podem ser omissas na ocorrência de imprevistos ou pode até mesmo existir regras conflitantes, uma vez que a base de dados pode, eventualmente crescer de forma significativa.

4.3 Exemplo de um Assistente Inteligente

Como exemplo de um assistente inteligente é apresentado o TOAST, que possui capacidades para simulação de eventos e diagnósticos de falhas em sistemas elétricos de potência (Assistentes poderiam, também, serem operados em tempo real, caso em que são chamados de Fase 2. Os Assistentes de Fase 1 são aqueles operando *off-line* e utilizados para pesquisa, treinamento e demonstração de conceitos, podendo evoluir, mais tarde, para Fase 2). O TOAST é um assistente de Fase 1, desenvolvido em 1985 na Universidade de Carnegie-Mellon.

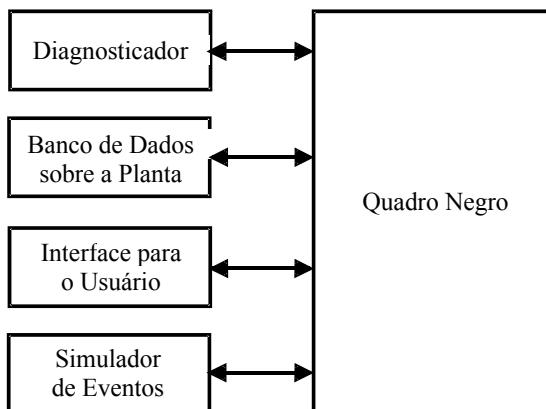


Figura 4-5. Arquitetura de um assistente inteligente.

O Diagnosticador é um SE que permite identificar a falha que ocasionou a alteração acidental na configuração da rede de distribuição de energia elétrica. Ele é escrito em OPS 5, com cerca de 150 regras. O Simulador de Eventos é utilizado para fornecer a assinatura de eventos que seriam decorrentes de uma dada falha tomada como hipótese diagnóstica. É escrito em OPS 5 e LISP, com

cerca de 150 regras e 25 funções LISP. A Interface com o usuário, também inteligente, proporciona ao usuário ou *trainee*, facilidades de interação com a máquina. Escrito em OPS 5, possui aproximadamente 175 regras.

5 INTRODUÇÃO À LÓGICA NEBULOSA

A lógica nebulosa (*fuzzy logic*) permite o tratamento de expressões que envolvem grandezas descritas de forma não exata e que envolvam variáveis lingüísticas. A lógica nebulosa baseia-se no conceito de conjuntos nebulosos.

5.1 Conjuntos Nebulosos

Um conjunto é uma coleção de objetos. Na teoria clássica, um objeto possui apenas duas possibilidades quanto à sua relação com um conjunto, ou seja, um dado objeto é ou não é um elemento do conjunto. Na teoria de conjuntos nebulosos, um objeto possui variados graus de pertinência. Pode-se ilustrar a utilidade de tal tipo de teoria examinando exemplos do dia a dia. Na teoria clássica, um paciente é dito estar com febre se a sua temperatura axilar ultrapassa 37.8°C . Assim, alguém que tenha 37.7° não pertence ao conjunto de pacientes febris. Também, homens com alturas entre 1.65 e 1.75m são considerados de estatura mediana e, acima de 1.75m, estatura alta. Nestas condições, 1.76m é a estatura de homem alto, mas 1.74 é a de um homem mediano:

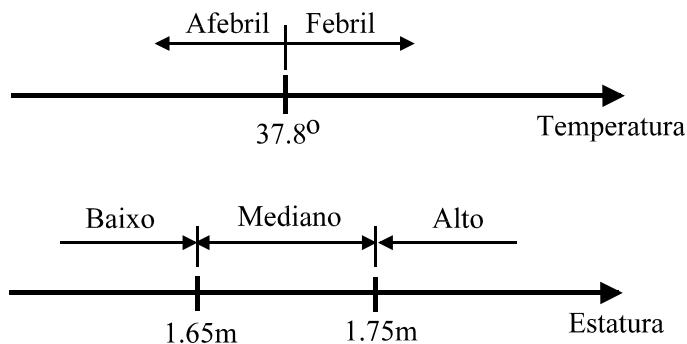


Figura 5-1. Representação de condições (afebril/febril, baixo/mediano/alto) por faixas precisamente definidas.

Por outro lado, usando o conceito de conjuntos nebulosos, as incertezas quanto ao que seria estando febril (ou ao que seria alto) ficariam representados por um grau de pertinência:

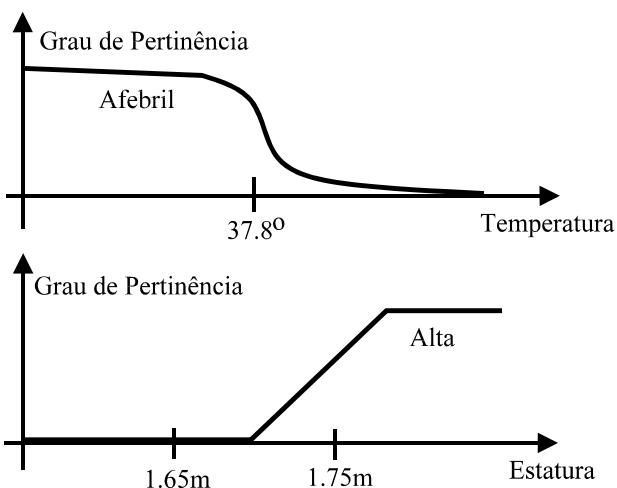


Figura 5-2. Representação de condições afebril e estatura alta utilizando graus de pertinência.

A formalização do conceito de conjuntos nebulosos pode ser obtida estendendo-se a teoria clássica de conjuntos. Assim, na teoria clássica, um conjunto pode ser caracterizado pela sua função indicadora, ou seja, dado um conjunto A no universo X , define-se $I_A(x): X \rightarrow [0,1]$ por:

$$I_A(x) = \begin{cases} 1 & \text{se } x \in A \\ 0 & \text{se } x \notin A \end{cases} \quad (5.1)$$

Se X for o conjunto \mathbb{R}^+ e A um intervalo fechado,

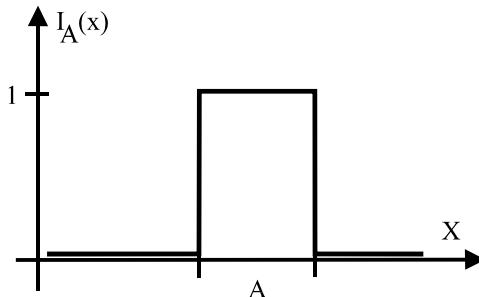


Figura 5-3. Função indicadora do conjunto A .

Portanto, um conjunto clássico pode ser representado por $A = \{x \in X \mid I_A(x) = 1\}$, ou abreviadamente, $A = \{x, I_A\}$. De forma análoga, os conjuntos nebulosos são definidos por $A = \{x \in X \mid \mu_A(x) = 1\}$, onde $\mu_A(x): X \rightarrow [0,1]$ é a função de pertinência que expressa o quanto um dado elemento x pertence a A .

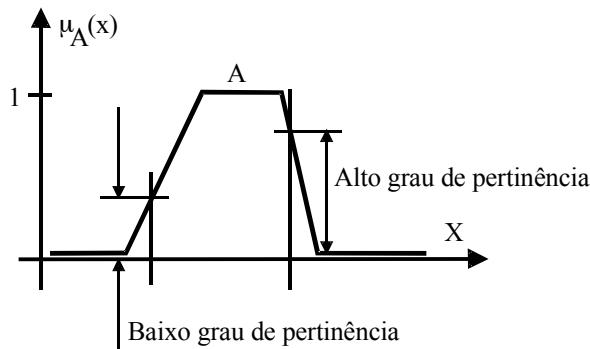


Figura 5-4. Função de pertinência para um conjunto nebuloso A .

A teoria de conjuntos nebulosos busca, portanto, traduzir em termos formais a informação imprecisa que ocorre de maneira espontânea na representação dos fenômenos da natureza, como descrito por humanos utilizando uma linguagem corriqueira.

Na teoria clássica, quando um elemento x pertence a um conjunto A ou a um conjunto B , diz-se que $x \in A \cup B$. Quando um elemento x pertence simultaneamente aos conjuntos A e B , diz-se que $x \in A \cap B$. Também no caso de conjuntos nebulosos deseja-se fazer associações semânticas análogas e que resultariam em interpretações úteis para expressões do tipo $(\alpha \vee \beta)$, $(\alpha \wedge \beta)$, $(\neg \alpha)$. Portanto, é necessário que sejam definidas, para conjuntos nebulosos, as operações com conjuntos: $A \cup B$, $A \cap B$ e A^c .

Dados conjuntos nebulosos A e B sobre um universo X e caracterizados pelas funções de pertinência μ_A e μ_B , podem ser definidas as seguintes operações:

a) União: o conjunto nebuloso $C = A \cup B$ é caracterizado pela função de pertinência:

$$\forall x \in X, \mu_C(x) = \max \{ \mu_A(x), \mu_B(x) \} \quad (5.2)$$

A expressão (5.2) representa apenas uma forma particular de definir μ_C , com $C = A \cup B$, a partir de μ_A e μ_B , como será detalhado mais adiante.

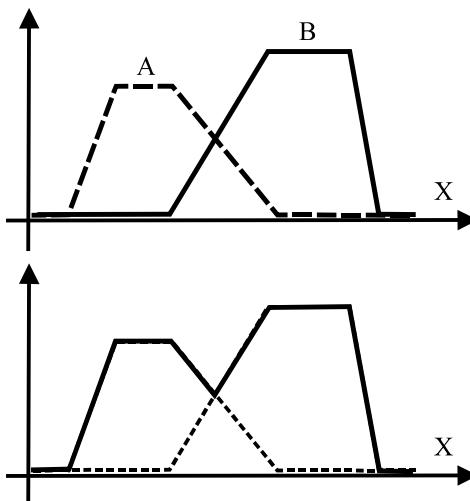


Figura 5-5. União de conjuntos nebulosos $C = A \cup B$.

b) Intersecção: o conjunto $D = A \cap B$ é caracterizado pela função de pertinência:

$$\forall x \in X, \mu_D(x) = \min \{ \mu_A(x), \mu_B(x) \} \quad (5.3)$$

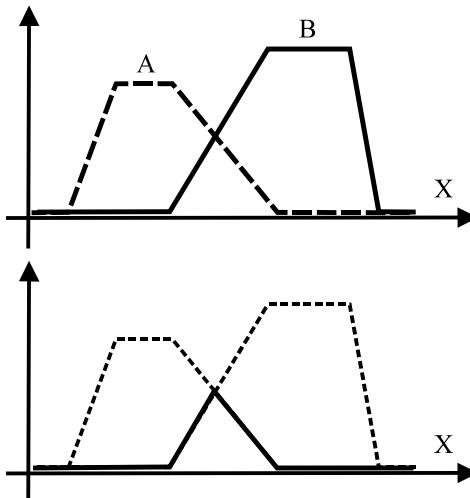


Figura 5-6. Intersecção de conjuntos nebulosos $D = A \cap B$.

c) Complementação: o conjunto $E = A^c$ é caracterizado pela função de pertinência:

$$\forall x \in X, \mu_E(x) = 1 - \mu_A(x) \quad (5.4)$$

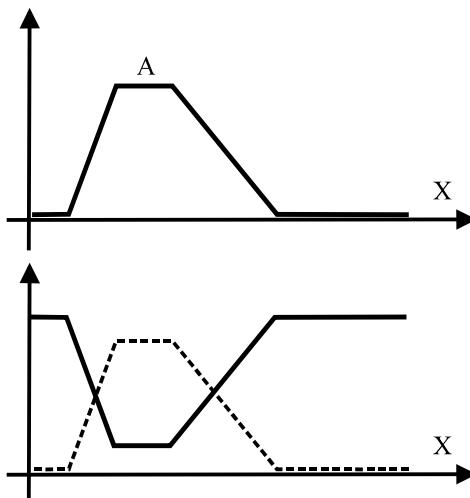


Figura 5-7. Complementação de um conjunto nebuloso $E = A^c$.

d) Produto: o conjunto $F = A \times B$ é caracterizado pela função de pertinência:

$$\forall x \in X, \mu_F(x) = \mu_A(x) \cdot \mu_B(x) \quad (5.5)$$

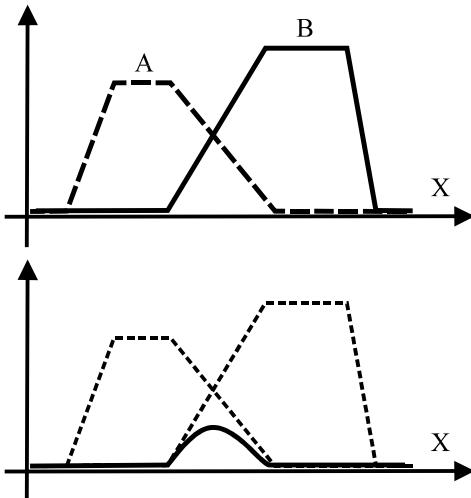


Figura 5-8. Produto de dois conjuntos nebulosos $F = A \times B$.

No caso particular de $\mu_A(x)$ e $\mu_B(x)$ serem idênticos a $I_A(x)$ e $I_B(x)$, respectivamente, as operações com conjuntos nebulosos reproduzem as operações clássicas com conjuntos.

Pode-se verificar que as operações com conjuntos nebulosos satisfazem as seguintes propriedades:

- | | | |
|---|--|--------|
| a) $(A^c)^c = A$ | <i>; Involução</i> | (5.6) |
| b) $(A \cap B)^c = A^c \cup B^c$ | <i>; De Morgan</i> | (5.7) |
| c) $(A \cup B)^c = A^c \cap B^c$ | <i>; De Morgan</i> | (5.8) |
| d) $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$ | <i>; Distributividade da \cap</i> | (5.9) |
| e) $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$ | <i>; Distributividade da \cup</i> | (5.10) |
| f) $(A \cap B) \cup A = A$ | <i>; Absorção</i> | (5.11) |
| g) $(A \cup B) \cap A = A$ | <i>; Absorção</i> | (5.12) |
| h) $A \cup A = A$ | <i>; Idempotência</i> | (5.13) |
| i) $A \cap A = A$ | <i>; Idempotência</i> | (5.14) |

Definindo-se $\mu_X(x) = 1$ para $\forall x \in X$ e $\mu_\emptyset(x) = 0$ para $\forall x \in X$, tem-se ainda as propriedades:

j) $A \cap \emptyset = \emptyset$	(5.15)
k) $A \cap X = A$	(5.16)
l) $A \cup \emptyset = A$	(5.17)
m) $A \cup X = X$	(5.18)

Atentar, porém, que para conjuntos nebulosos algumas operações legítimas com conjuntos clássicos não são necessariamente válidas:

$$n) A \cup A^c \neq X \quad (5.19)$$

$$o) A \cap A^c \neq \emptyset \quad ; \text{Não vale o princípio da exclusão} \quad (5.20)$$

Os conceitos de União, Intersecção e Complementação associados a (5.2), (5.3) e (5.4) podem ser tratados de forma mais geral através do emprego de operações conhecidas como normas e conormas triangulares.

Uma operação $t:[0,1] \times [0,1] \rightarrow [0,1]$ é dita ser uma t-norma se:

$$a) t(0,0) = 0 \quad (5.21)$$

- b) $t(a,b) = t(b,a)$; comutatividade (5.22)
c) $t(a,t(b,c)) = t(t(a,b),c)$; associatividade (5.23)
d) $(a \leq c) \wedge (b \leq d) \Rightarrow t(a,b) \leq t(c,d)$; monotonicidade (5.24)
e) $t(a,1) = a$ (5.25)

Se uma operação s possui as propriedades {a,b,c,e} e ainda satisfaz

$$a') s(1,1) = 1 \quad (5.26)$$

$$b') s(b,0) = b \quad (5.27)$$

então é chamada de t-conorma.

Pode-se verificar que $\min(a,b)$ é uma t-norma e $\max(a,b)$ é uma t-conorma, de forma tal que a intersecção e a união expressas em (5.3) e (5.2) são instâncias particulares de uma família de operações. Algumas das outras operações que satisfazem as propriedades de t-norma e t-conorma estão apresentadas na Tabela 5-1.

Tabela 5-1

Nome	t-norma	t-conorma
Zadeh	$\min(a,b)$	$\max(a,b)$
Probabilístico	$a.b$	$a+b-ab$
Lukasiewicz	$\max(a+b-1,0)$	$\min(a+b,1)$
Yager	$1-\min(1,(1-(1-a)^w + (1-b)^w)^{1/w})$	$\min(1,(a^w + b^w)^{1/w})$
Dubois e Prade	$\frac{ab}{\max(a,b,\alpha)}$	$\frac{a+b-ab-\min(a,b,1-\alpha)}{\max(1-a,1-b,\alpha)}$

Um operador $n:[0,1] \rightarrow [0,1]$ é dito ser de negação se

$$a) n(0) = 1 \quad (5.28)$$

$$b) n(1) = 0 \quad (5.29)$$

$$c) a > b \Rightarrow n(a) \leq n(b) \quad (5.30)$$

$$d) n(n(a)) = a, \forall a \in [0,1] \quad (5.31)$$

São exemplos de negações:

$$a) n(a) = 1 - a \quad (5.32)$$

$$b) n(a) = \frac{1-a}{1+\lambda a}, \quad \lambda \in (-1, \infty) \quad ; \text{negação de Sugeno} \quad (5.33)$$

$$c) n(a) = (1 - a^w)^{\frac{1}{w}}, \quad w \in (0, \infty) \quad ; \text{negação de Yager} \quad (5.34)$$

Assim, na equação (5.4), tem-se um caso particular de complementação, caracterizada por:

$$\mu_{A^c}(x) = n(\mu_A(x)) = 1 - \mu_A(x) \quad (5.35)$$

5.2 Lógica Nebulosa

Conceitos nebulosos expressos em universos distintos podem apresentar relações entre si. Eventualmente podem estar envolvidos mais de dois universos.

Sejam X_1, X_2, \dots, X_n universos de discurso. Uma relação nebulosa R em $X_1 \times X_2 \times \dots \times X_n$ é definida pelo mapeamento:

$$\mu_R: X_1 \times X_2 \times \dots \times X_n \rightarrow [0,1] \quad (5.36)$$

Quando se tem duas relações que devem ser encadeadas, por exemplo $A \rightarrow B$ e $B \rightarrow C$, seria interessante que fosse disponível um mecanismo de composição. Suponha que X , Y e Z são três universos de discurso distintos, R_1 uma relação sobre $X \times Y$ e R_2 uma outra relação sobre $Y \times Z$. Para se obter a composição $R_1 \circ R_2$ que relaciona X e Z , estende-se, inicialmente, R_1 e R_2 para $X \times Y \times Z$. Agora, uma vez que R_1 e R_2 passaram a ter o mesmo ‘domínio’, a relação entre X e Z é obtida mediante restrição,

$$\mu_{R_1 \circ R_2}(x, z) = \sup_{y \in Y} \left[\min \left\{ \mu_{R_1}^{\text{ext}}(x, y, z), \mu_{R_2}^{\text{ext}}(y, z) \right\} \right] \quad (5.37)$$

onde

$$\begin{aligned} \mu_{R_1}^{\text{ext}}(x, y, z) &= \mu_{R_1}(x, y) \forall z \in Z \\ \mu_{R_2}^{\text{ext}}(x, y, z) &= \mu_{R_2}(y, z) \forall x \in X \end{aligned} \quad (5.38)$$

Esta operação é conhecida como **extensão cilíndrica** e a composição expressa em (7.36) é denominada de **sup-min**.

Uma relação muito importante é a expressa por “Se A Então B ”. Por exemplo, se $A \subset W =$ temperatura_baixa e $B \subset Y =$ tensão_grande, onde W poderia ser um conjunto de temperaturas medidas por um termistor, enquanto Y poderia ser um conjunto de tensões a serem aplicadas na resistência de aquecimento de uma estufa. Assim, uma forma de expressar a relação $R: A \rightarrow B$ seria:

$$R = A^c \cup B \quad ; \quad \mu_R(w, y) = \max \{ 1 - \mu_A(w), \mu_B(y) \} \quad (5.39)$$

em vista de $A \rightarrow B$ ser equivalente a $B \vee (\neg A)$ que é associado a $A^c \cup B$.

Existem, entretanto, outras associações úteis. Para se utilizar conjuntos nebulosos em dedução, é conveniente que se disponha de uma forma de realizar *modus ponens* em versão nebulosa. Originalmente, *modus ponens* consiste em obter B a partir de $A \wedge (A \rightarrow B)$. Como se sabe que $A \wedge (A \rightarrow B) \Leftrightarrow (A \wedge B)$, e por sua vez faz-se a associação $(A \wedge B)$ com $A \cap B$, uma versão nebulosa interessante para R seria:

$$\mu_R(w, y) = \min \{ \mu_A(w), \mu_B(y) \} \quad (5.40)$$

que é conhecido como a **regra de Mamdani**.

Uma vez que a relação $R: A \rightarrow B$ esteja caracterizada, a inferência nebulosa (ou *modus ponens generalizado*):

Se (**a** é A) Então (**b** é B)

Tem-se A'

Portanto, B'

pode ser obtida fazendo-se a composição de R com A' , usando, por exemplo, **sup-min**.

Quando se dispõe de várias regras envolvendo a mesma variável, por exemplo denotada **a**, necessita-se de uma forma de amalgamar as conclusões implicadas por cada regra:

R1: Se (**a** é A_1) Então (**b** é B_1)

⋮

R2: Se (**a** é A_m) Então (**b** é B_m)

Uma heurística para combinar as regras seria tomar como conclusão $B = B'_1 \cup \dots \cup B'_m$.

6 CONTROLADORES EMPREGANDO LÓGICA NEBULOSA

A lógica nebulosa, na forma como é utilizada aqui, é uma extensão do conceito de sistemas de produção, no sentido de incluir imprecisões na descrição dos processos controlados. Portanto, regras nebulosas podem ser empregadas para controle direto, supervisão ou assistência ao operador. Aqui, será dedicado maior atenção ao controle direto, uma vez que é a modalidade que apresenta maiores diferenças em relação a sistemas tratados anteriormente.

O problema típico de controle direto consiste em obter, a partir do modelo da planta, uma lei de controle que atenda as especificações fornecidas a priori:

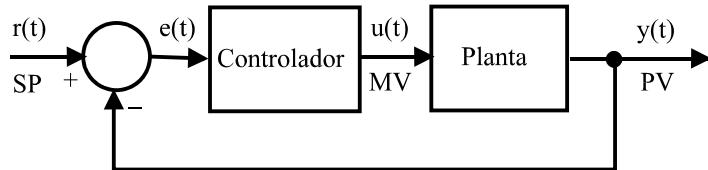


Figura 6-1. Esquema clássico onde o controlador calcula o valor da variável manipulada (MV) a partir do sinal de erro $e = SP - PV$, onde SP é o *set-point* e PV é o *process variable*.

Para efeito de comparação, sejam três tipos possíveis de controladores que poderiam ser empregados como ilustrado na figura 8.1:

a) Controlador baseado em equação:

$$u(t) = \sqrt[3]{e(t)} \quad (6.1)$$

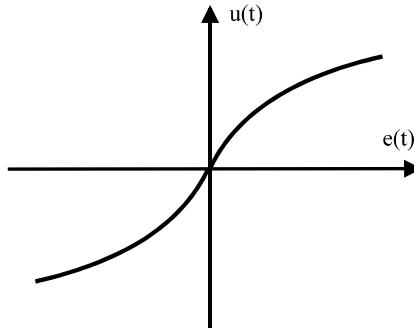


Figura 6-2. Característica entrada-saída de um controlador não-linear contínuo e sem dinâmica.

b) Controlador baseado em regras de produção não nebulosas:

Se ($ e(t) < k_1$)	Então ($u(t) = 0$)
Se ($ e(t) \in [k_1, k_2]$)	Então ($u(t) = \sqrt[3]{k_1}$)
Se ($ e(t) \in [k_2, k_3]$)	Então ($u(t) = \sqrt[3]{k_2}$)
Se ($ e(t) > k_3$)	Então ($u(t) = \sqrt[3]{k_3}$)

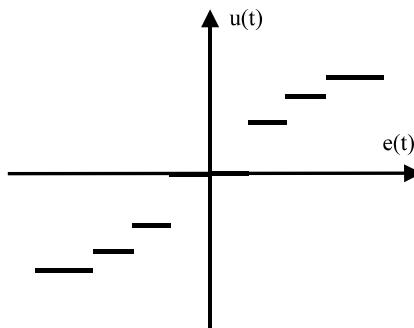


Figura 6-3. Característica I/O de um controlador baseado em regras de produção e sem incertezas.

c) Controlador baseado em regra de produção nebulosa:

Se ($e(t) \sim EQZ$)	Então ($u(t) \sim UQZ$)
Se ($e(t) \sim EMP$)	Então ($u(t) \sim UMP$)
Se ($e(t) \sim EMN$)	Então ($u(t) \sim UMN$)
Se ($e(t) \sim EGP$)	Então ($u(t) \sim UGP$)
Se ($e(t) \sim EGN$)	Então ($u(t) \sim UGN$)

onde QZ significa Quase_Zero, MP=Médio_Positivo, MN=Médio_Negativo, GP=Grande_Positivo e BN=Grande_Negativo.

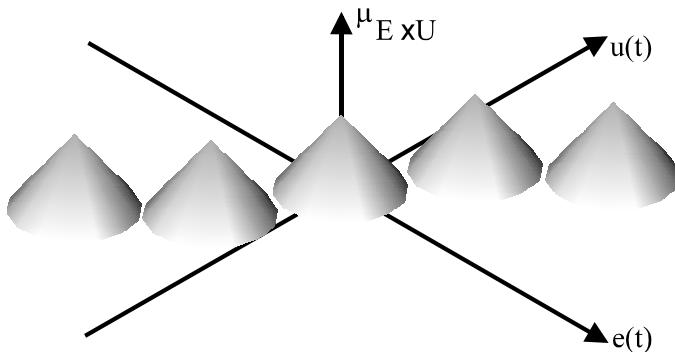


Figura 6-4. Tentativa de ilustrar a característica entrada-saída de um controlador nebuloso.

6.1 Estrutura de Controladores empregando Lógica Nebulosa

Os controladores empregando lógica nebulosa possuem, de modo geral, três blocos:

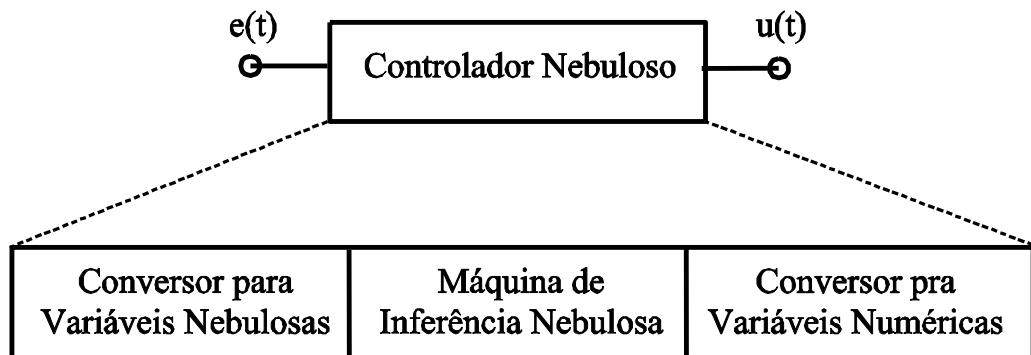


Figura 6-5. Estrutura típica de um controlador nebuloso.

6.1.1 Conversor para Variáveis Nebulosas (“Nebulizador”)

O sinal $e(t)$ é, usualmente, do tipo numérico. Por outro lado, as regras são expressas em termos da pertinência ou não destas grandezas a conjuntos nebulosos. Assim, há a necessidade de se converter $e(t)$ para uma entidade apropriada para inferência nebulosa. Tipicamente, são calculados os valores das funções de pertinência $\mu_{A_i}(e(t))$ para cada um dos A_i associado à grandeza $e(t)$.

Considere-se que os conjuntos associados a $e(t)$ são $A_1 = EP$ (erro pequeno), $A_2 = EM$ (erro moderado) e $A_3 = EG$ (erro grande). Da figura 8.6, para os instantes t_1 , t_2 e t_3 tem-se:

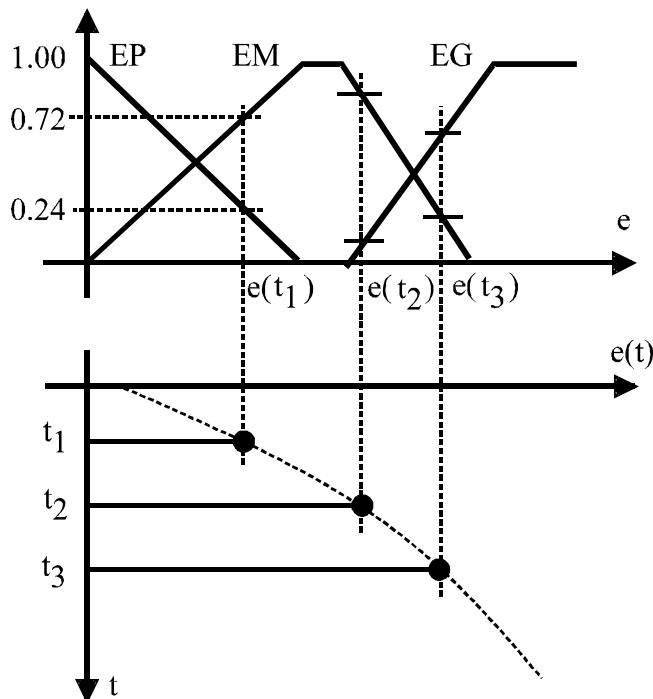


Figura 6-6. Valores da função de pertinência para o sinal de erro em diferentes instantes de tempo.

Para $e(t_1)$, tem-se, no caso, a partir da Figura 6-6:

$$c(t_1) \in \begin{cases} EP & \text{com } \mu_{EP} = 0.24 \\ EM & \text{com } \mu_{EM} = 0.72 \end{cases} \quad (6.2)$$

6.1.2 Máquina de Inferência Nebulosa

A máquina de inferência deve receber as informações sobre $e(t)$, já convertidas em termos associados a variáveis linguísticas, especificamente graus de pertinência a conjuntos nebulosos e gerar, a partir de uma base de regras fornecidas pelo usuário, uma saída também do tipo linguístico, na forma de funções de pertinência ($\mu_U(\cdot)$) que deve ser, posteriormente, convertido para uma variável numérica $u(t)$.

Ressalte-se que não há restrições quanto a $e(t)$ ser do tipo vetorial e nem em se realizar operações auxiliares. Um exemplo de operação auxiliar é o cálculo de $\Delta e(t) = e(t) - e(t - \Delta t)$, o que permitiria regras do tipo:

Se $\{(e(t) \sim EP) \wedge (\Delta e(t) \sim DEPG)\}$ Então $(u(t) \sim UNG)$

onde DEPG poderia significar Δe positivo grande e UNG, u negativo grande.

As regras podem envolver operações intrincadas como:

Se $\{ \sum_{k=0}^p |c(t - k\Delta t)| \exp(-kT) \sim SG \}$ Então $(\Delta u(t) \sim DUN)$

significando que se a soma dos módulos do erro, com esquecimento exponencial, numa janela de comprimento $p+1$, for grande (SG = Somatória Grande), então o controle deve ser diminuído (Delta u Negativo).

Uma forma de se representar o conjunto de regras envolvendo apenas uma variável, por exemplo o erro $e(t)$, é através de tabela do tipo:

Tabela 6-1. Apresentação compacta, 5 regras, onde NG = Negativo-Grande, NM = Negativo-Mediano, Z = Quase Zero, PM = Positivo-Moderado e PG = Positivo-Grande

		e(t)					
		NG	NM	Z	PM	PG	
u(t)	PG	PM	Z	NM	NG		

No caso de se ter 2 variáveis, por exemplo $e(t)$ e $\Delta e(t)$, pode-se expressar as regras na forma da tabela:

Tabela 6-2. Apresentação da base de regras para 2 entradas (e , Δe) e uma saída (u).

		e				
		NG	NM	Z	PM	PG
Δe	N	PG	PM	Z	NM	NG
	Z	PM	PM	Z	NM	NM
	P	Z	Z	Z	NM	NG

Para ilustrar um mecanismo de inferência, considere-se um caso em que há apenas 2 regras:

Regra 1: Se ($e \sim EP$) Então ($u \sim UP$)

Regra 2: Se ($e \sim EG$) Então ($u \sim UG$)

onde as funções de pertinência $\mu_{EP}(\cdot)$, $\mu_{UP}(\cdot)$, $\mu_{EG}(\cdot)$, $\mu_{UG}(\cdot)$ associados a Erro-Pequeno, U-Pequeno, Erro-Grande e U-Grande estão ilustrados na Figura 6-7. Para a relação Se (...) Então (...) é utilizada a regra min de Mamdani e para a relação (Regra 1) ou (Regra 2) é utilizada a forma max.

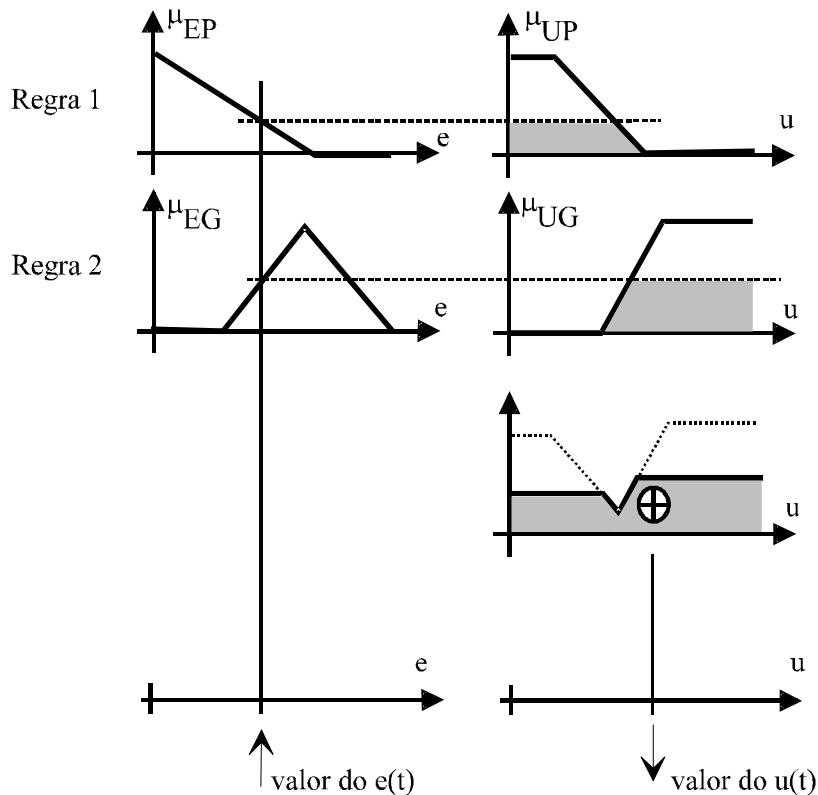


Figura 6-7. Mecanismo de inferência utilizando o esquema proposto por Mamdani.

6.1.3 Conversor para Variável Numérica (“Desnebulizador”)

As regras nebulosas produzem como saída conjuntos nebulosos. Como, na prática, necessita-se de um valor bem definido para $u(t)$, defini-se o processo de “desnebulização” (às vezes também referido como “defuzzificação”). Esse processo consiste em obter um valor para $u(t)$ a partir de $\mu_U(u)$.

Os processos mais conhecidos de conversão para variáveis numéricas são:

a) **Centro de Área (COA)**: É como imaginar a área sob a curva $\mu_U(u)$ como uma placa e determinar a coordenada u do seu centro de massa:

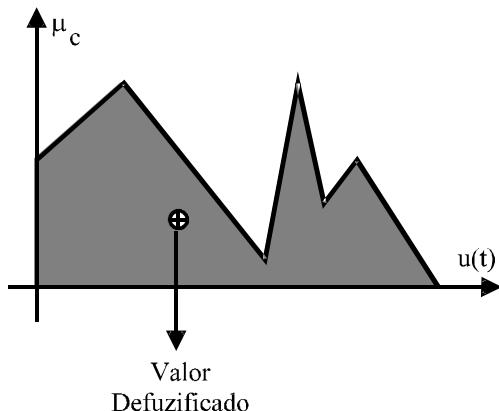


Figura 6-8. Conversão para variável numérica empregando centro de massa.

b) **Média dos Máximos (MOM)**: É valorizar os picos de $\mu_U(u)$, sendo que no caso de ser único, o valor “defuzificado” é a própria abscissa deste pico:

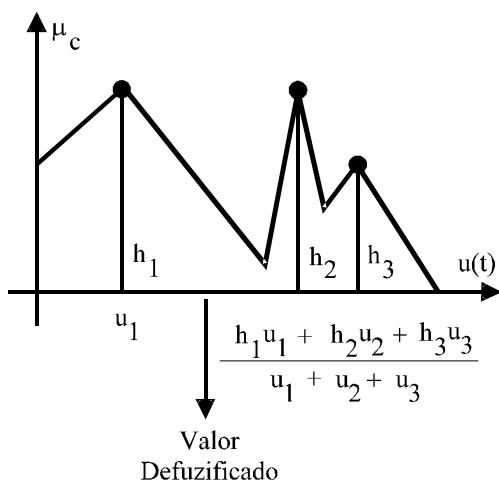


Figura 6-9. Conversão para variável numérica empregando média dos máximos.

6.2 Controladores com $e(t)$ e $u(t)$ quantizados

Considerando-se que, muitas vezes, o universo de discurso X é discreto, ou seja, assumindo $e \in \{1, 2, 3, \dots, n_e\}$ e $u \in \{1, 2, 3, \dots, n_u\}$, após escalonamento conveniente, ter-se-ia $X = \{1, 2, 3, \dots, n_e\} \times \{1, 2, 3, \dots, n_u\}$. Observa-se, então, que uma forma conveniente de representar $\mu_R(e, u)$ é como uma matriz $[R]$ onde o elemento $r_{ij} = \mu_R(i, j)$. Neste caso, $\mu_E(e)$ seria representado por uma matriz linha $[E]$ com elemento $e_i = \mu_E(i)$, permitindo a obtenção rápida de $\mu_U(u)$ usando regra de multiplicação de matrizes, com o produto substituído pela operação min e a soma pela max: $[U] = [E] \otimes [R]$.

Como exemplo, considere as regras:

R_1 : Se ($e \sim EP$) Então ($u \sim UG$)

R_2 : Se ($e \sim EG$) Então ($u \sim UP$)

onde:

EP significa Erro_Pequeno, caracterizado pela matriz $[EP] = [0.5 \ 1 \ 1 \ 0 \ 0]$

EG Erro_Grande, $[EG] = [0 \ 0 \ 0.5 \ 1 \ 0.5]$

UG Controle_Grande, $[UG] = [0 \ 0 \ 0.5 \ 1 \ 1]$

UP Controle_Pequeno, $[UP] = [1 \ 1 \ 1 \ 0.5 \ 0]$

A regra R_1 , usando min de Mamdani, é dada por

R_1	u_1	u_2	u_3	u_4	u_5
e_1	0	0	0.5	0.5	0.5
e_2	0	0	0.5	1	1
e_3	0	0	0.5	1	1
e_4	0	0	0	0	0
e_5	0	0	0	0	0

A regra R_2 , de forma análoga, é dada por

R_2	u_1	u_2	u_3	u_4	u_5
e_1	0	0	0	0	0
e_2	0	0	0	0	0
e_3	0.5	0.5	0.5	0.5	0
e_4	1	1	1	0.5	0
e_5	0.5	0.5	0.5	0.5	0

Tomando-se a regra $R = R_1 \vee R_2$, o conjunto nebuloso correspondente, também denotado R , com $\mu_R(e,u)$ obtido usando max, fica representado pela matriz R :

R	u_1	u_2	u_3	u_4	u_5
e_1	0	0	0.5	0.5	0.5
e_2	0	0	0.5	1	1
e_3	0.5	0.5	0.5	1	1
e_4	1	1	1	0.5	0
e_5	0.5	0.5	0.5	0.5	0

Supondo que, em um dado instante obteve-se a informação de que o sinal de erro é descrito por:

$$[E] = [0.2 \ 0.7 \ 0.2 \ 0.1 \ 0]$$

Neste caso, as regras indicam que deve ser gerado um sinal de controle dado por:

$$[U] = [0.2 \ 0.7 \ 0.2 \ 0.1 \ 0] \otimes [R] = [0.2 \ 0.2 \ 0.7 \ 0.7 \ 0.7]$$

ou seja, o sinal de controle a ser aplicado deve ser algo grande.

Na prática, o valor do sinal de erro é calculado a partir de saídas de sensores, assumindo a forma

$$[E] = [0 \ 0 \ 0 \ 1 \ 0]$$

que é do tipo não nebuloso, no sentido que $e(t) = 4$ (valor normalizado). Neste caso, a operação $[U] = [E] \otimes [R]$ resulta, simplesmente, na 4^a linha de $[R]$.

6.3 Estabilidade no Controle Nebuloso em Malha Fechada

A relação entrada-saída de um controlador nebuloso é, de modo geral, não-linear. Assim, a análise de estabilidade de sistemas em malha fechada empregando controladores nebulosos requer o uso de métodos sofisticados, como o de Lyapunov.

Como ilustração, considere o caso de um controlador nebuloso de uma entrada e uma saída:

Tabela 6-3. Regras utilizadas no exemplo de estudo de estabilidade.

e(t)					
	NG	NM	Z	PM	PG
u(t)	NG	NM	Z	PM	PG

As funções de pertinência empregadas nas regras apresentadas na Tabela 6-3 estão ilustradas na Figura 6-10:

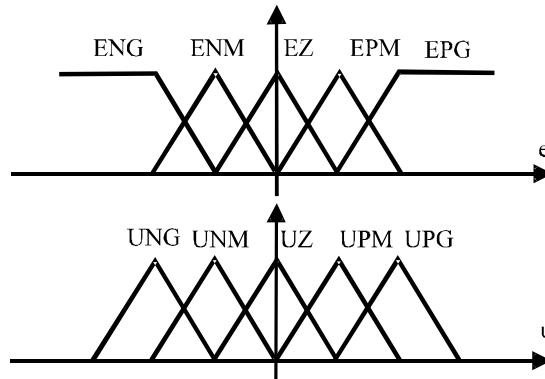


Figura 6-10. Funções de pertinência do exemplo de estudo de estabilidade, onde N significa negativo, P positivo, M moderado, G grande e Z quase zero.

A partir das regras apresentadas na Tabela 6-3 é possível calcular a saída u a partir de uma dada entrada e . O gráfico da relação $u=\phi(e)$ é apresentado na Figura 6-11.

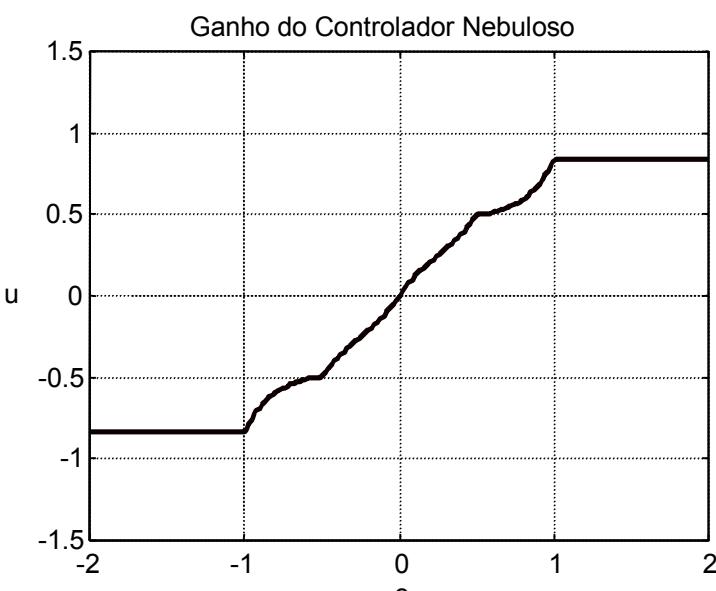


Figura 6-11. Relação entrada-saída ($u=\phi(e)$) para o exemplo de análise de estabilidade.

Uma vez que a característica entrada-saída do controlador nebuloso é conhecida, pode-se buscar uma metodologia de análise de estabilidade de sistemas não-lineares que seja aplicável para o caso em questão.

Para o exemplo considerado, é aplicável a teoria de estabilidade absoluta de Lur'e.

Considerando $r(.) \equiv 0$, o diagrama de blocos da Figura 6-1 assume a forma ilustrada na Figura 6-12.

Caso o gráfico de $\phi(.)$ esteja contido em um setor limitado por retas de inclinações a e b , conforme ilustrado na Figura 6-13, existem condições que garantem a estabilidade de sistemas como o da Figura 6-12. Um setor limitado por retas de inclinações a e b é denotado setor(a, b).

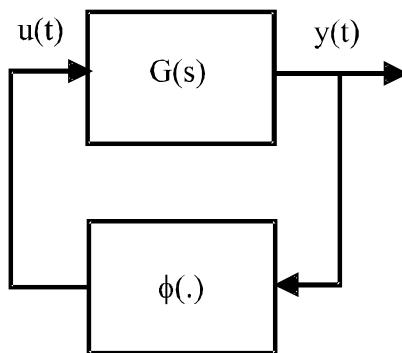


Figura 6-12. Diagrama de blocos de um sistema para o qual podem ser estabelecidas condições de estabilidade.

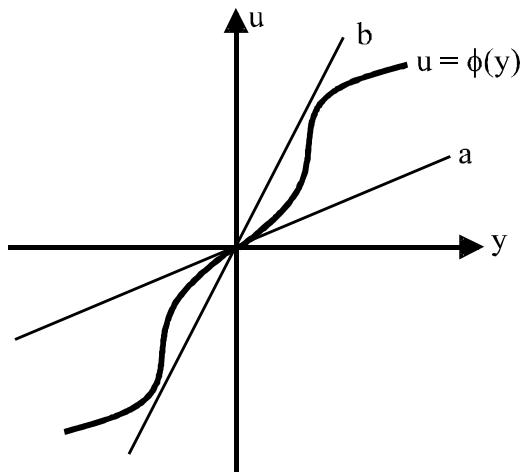


Figura 6-13. Não linearidade do tipo setor, onde o gráfico de $\phi(.)$ está contido na região limitada por retas de inclinações a e b .

Teorema 6.1: Se $G(s)$ é estável (ou seja, não possui pólos no semi-plano direito), $G(s)$ é estritamente próprio (ou seja, o grau do denominador é estritamente maior que o do numerador), $\phi(.)$ está contido no setor $(0, k)$ com $k > 0$, então o sistema em malha fechada (Figura 6-12) é estável se

$$\text{Re}[1 + kG(j\omega)] > 0, \quad \forall \omega \in \mathbb{R}. \quad (6.3)$$

No caso particular do exemplo, se o controlador nebuloso caracterizado pelas regras da Tabela 6-3 for utilizado com uma planta $G(s)$ estável e estritamente própria, basta que

$$\operatorname{Re}[1 + G(j\omega)] > 0, \forall \omega \in \mathbb{R}. \quad (6.4)$$

6.4 Características de Controladores Nebulosos

a) Aspectos Favoráveis:

- Não necessidade de modelamento do sistema a ser controlado;
- Possibilidade de incorporar conhecimentos heurísticos do operador humano;
- Aplicabilidade a sistemas de dinâmica complicada, incluindo não linearidades;
- Possibilidade de explorar a não-linearidade dos controladores para obter maior desempenho;
- Disponibilidade de componentes dedicados.

b) Aspectos Desfavoráveis:

- Ausência de diretrizes precisas para o projeto do controlador, resultando em enfoque artesanal e pouco sistematizado.
- Impossibilidade de demonstração, nos casos gerais, de propriedades como a estabilidade, não se podendo garantir, por exemplo, ausência de ciclos limite.
- Precisão de regulagem eventualmente insuficiente
- Consistência das regras não garantidas *a priori*.

c) Capacidade de Aproximação Universal

Um controlador nebuloso pode assumir a forma de um sistema aditivo nebuloso como ilustrado na figura 8.14:

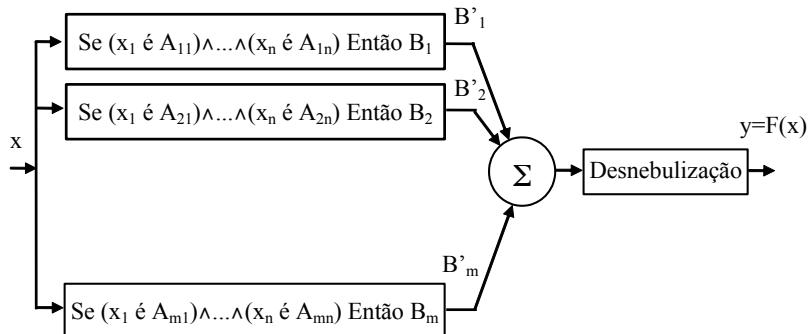


Figura 6-14. Sistema aditivo nebuloso

O vetor de entradas x , em função do grau de pertinência do componente x_j ao conjunto A_{ij} produz saídas B'_i que, combinadas, resultam no conjunto B . Aplicando-se a desnebulização sobre este conjunto obtém-se um valor numérico y . Assim, a saída y é uma função de x , ou seja $y = F(x)$, sendo que as características de $F(\cdot)$ podem ser modificadas através da alteração das funções de pertinência dos conjuntos A_{ij} e B_i nas regras “Se $(x_1 \text{ é } A_{i1}) \wedge \dots \wedge (x_n \text{ é } A_{in})$ Então B_i ”.

No caso de um controlador lógico nebuloso do tipo Mamdani, o mapa $y = F(x)$ é da forma:

$$F(x) = \frac{\sum_{i=1}^m \min_j \{u_{A_{ij}}(x_j)\} a_i c_i}{\sum_{i=1}^m \min_j \{u_{A_{ij}}(x_j)\} a_i}$$

onde a_i é a área de B_i e c_i é a abscissa do centro de massa de B_i . Portanto, uma vez que os conjuntos A_{ij} e B_i são conhecidos, a saída y pode ser calculada para cada x através de $y = F(x)$.

Uma questão interessante é sobre a existência de solução para o problema inverso: “Dada uma função $F(\cdot)$, seria possível construir um controlador lógico nebuloso, de modo que $y = F(x)$?”.

Em vista dos resultados de Wang e Mendel (1992), é possível construir um controlador lógico nebuloso tal que $y = F_a(x)$, de modo que $F_a(\cdot)$ aproxima $F(\cdot)$ em termos de norma $\|\cdot\|_\infty$, no espaço de funções contínuas, desde que o universo de discurso X seja compacto.

Portanto, controladores lógicos nebulosos são “aproximadores universais” para uma certa classe de funções, mas persiste a dificuldade em se caracterizar os conjuntos A_{ij} e B_i , bem como o número de

regras necessárias. Uma forma de tentar determinar estas características de forma numérica é através do emprego de métodos de otimização como o de “Poliedros Flexíveis” e “Algoritmos Genéticos”.

7 MÉTODOS DE OTIMIZAÇÃO NUMÉRICA

São várias as aplicações dos métodos de otimização numérica na implementação de sistemas incorporando técnicas de inteligência artificial. Podem ser mencionados o treinamento de redes neurais artificiais, o ajuste adaptativo de controladores nebulosos, a utilização de otimização da produção em sistemas supervisionados por estações inteligentes e outras.

Em vista de existirem diversos textos especializados em otimização, não se pretende aqui aprofundar demasiadamente no tema e o objetivo deste capítulo é apenas oferecer algumas noções sobre as principais técnicas disponíveis.

7.1 Conceituação e Utilidade em Sistemas Inteligentes

Otimização é, no presente contexto, a determinação de uma ação que proporciona um máximo de benefício, medido por um critério de desempenho pré-estabelecido. Muitas vezes a ação é a seleção de um vetor de parâmetros $w = (w_1, w_2, \dots, w_n) \in R^n$, onde n é um número natural fixo, e o critério de desempenho é uma função $J: R^n \rightarrow R$ que deve ser minimizada caso represente um custo ou, então, ser maximizada caso esteja associada a retorno.

Uma ampla classe de problemas de otimização pode ser expressa na forma:

$$\begin{aligned} & \min_{w \in R^n} J(w) \\ & \text{sujeito a} \\ & g_i(w) \leq 0 \quad i = 1, 2, \dots, n_g \\ & h_j(w) = 0 \quad j = 1, 2, \dots, n_h \end{aligned} \tag{7.1}$$

onde $g_i: R^n \rightarrow R$ são chamadas de restrições (ou vínculos) de desigualdade e $h_j: R^n \rightarrow R$ são chamados de restrições (ou vínculos) de igualdade. O problema expresso formalmente em (7.1) é determinar um ponto $w \in R^n$ tal que resulte em um mínimo valor de $J(\cdot)$, enquanto as desigualdades $g_i(w) \leq 0$, $i=1,2,\dots,n_g$ e as igualdades $h_j(w) = 0$, $j=1,2,\dots,n_h$ são satisfeitas.

Exemplo: Para $n = 1$, $n_g = 1$, $n_h = 1$, $w = (w_1, w_2) \in R^2$ e

$$\begin{aligned} & J(w) = 3w_1 + 2w_2 \\ & g(w) = w_1^2 + w_2^2 - 4 \\ & h(w) = w_1 - 1 \end{aligned} \tag{7.2}$$

O problema pode ser escrito, assumindo como objetivo a minimização de $J(\cdot)$, como sendo:

$$\begin{aligned} & \min_{w \in R^2} \{3w_1 + 2w_2\} \\ & s.a. \\ & w_1^2 + w_2^2 - 4 \leq 0 \\ & w_1 + 1 = 0 \end{aligned} \tag{7.3}$$

cuja solução pode ser obtida gráfica ou analiticamente, fornecendo o ponto de ótimo $w^* = (1, -\sqrt{3})$.

Quando as funções $J(\cdot)$, $g(\cdot)$ e $h(\cdot)$ são não lineares e n é um número grande, pode ser impraticável a obtenção de soluções por métodos gráficos ou analíticos. Nesses casos, é interessante que se disponha de métodos numéricos que possam ser codificados para utilização em computador digital.

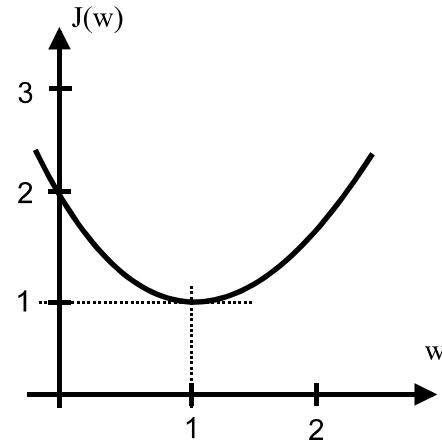
De fato, métodos numéricos de otimização podem ser empregados no treinamento de redes neurais, no ajuste de funções de pertinência de sistemas difusos, no ajuste sistematizado de parâmetros de controladores ou na construção de leis sub-ótimas de controle.

O problema de otimização pode apresentar diversas dificuldades, como ilustrado a seguir:

a) Caso clássico irrestrito (sem dificuldades maiores): O ponto de mínimo, neste caso, corresponde a w^* tal que $dJ(w^*)/dw = 0$.

$$J(w) = w^2 - 2w + 2$$

$$\frac{dJ}{dw} \Big|_{w=w^*} = 0$$



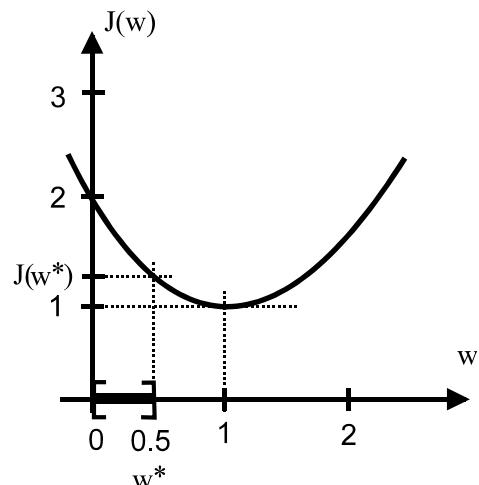
b) Caso clássico restrito: Um problema de minimização com restrições, onde $dJ(w)/dw \neq 0$ no ponto w^* .

$$J(w) = w^2 - 2w + 2$$

$$g_1(w) = w - 0.5$$

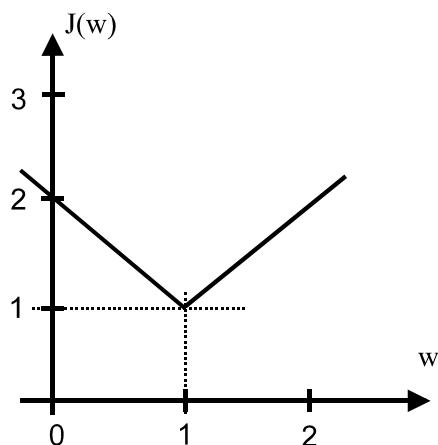
$$g_2(w) = -w$$

$$\frac{dJ}{dw} \Big|_{w=w^*} \neq 0$$



c) Caso não diferenciável: Um caso de minimização onde existe o ponto de mínimo, mas este não é caracterizado por $dJ(w)/dw = 0$.

$$J(w) = |w - 1| + 1$$



7.2 Métodos de Otimização Unidimensional

Considera-se aqui a otimização (minimização ou maximização) de funcionais $J: \mathbb{R} \rightarrow \mathbb{R}$, ou seja, aquele em que w é um número real. Variantes como o caso onde w é restrito a ser um número inteiro fogem do escopo do presente texto.

7.2.1 Método da Busca Uniforme

Uma forma de se resolver numericamente um problema de minimização unidimensional é gerar uma sequência w_0, w_1, \dots, w_n de modo que $J(w_n) \rightarrow J(w^*)$ à medida que $n \rightarrow \infty$. Para tal, pode-se prover um mecanismo para obtenção de um novo ponto w_{k+1} a partir de um ponto w_k arbitrário, de modo que $J(w_{k+1}) \leq J(w_k)$, de preferência com a desigualdade no sentido estrito.

Passo 0: Arbitrar w_0 e $h_0 \in \mathbb{R}$.

Arbitrar um número pequeno tol.

Fazer $k=0$.

Passo 1: Calcular $w_{k+1} = w_k + h_k$

Passo 2: Calcular $\delta = J(w_k) - J(w_{k+1})$

Passo 3: Critério de Parada:

Se ($|\delta| < tol$) Então ($w_{aprox}^* = w_{k+1}$ e fim).

Passo 4: Critério para Redução do Passo e Mudança de Sentido:

Se ($\delta < 0$) Então ($h_{k+1} = -h_k/2$)

Passo 5: Retorna para nova iteração:

Fazer $w_{k+1} = w_k$ e $k = k+1$

Retornar ao Passo 1.

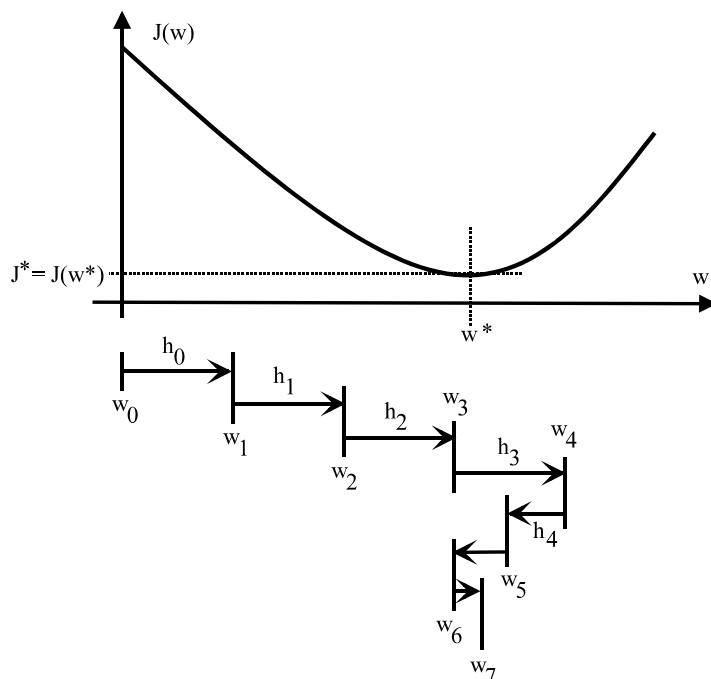


Figura 7-1. Busca Uniforme

7.2.2 Método da Secção Áurea

Este método consiste em estimar, a cada passo k , um intervalo $[w_k^{\min}, w_k^{\max}]$ onde estaria localizado w^* e seccioná-lo, usando a razão áurea. Se o comprimento do intervalo inicial é L , a secção áurea produz dois intervalos de comprimentos $m = (3 - \sqrt{5})L/2$ e $M = (\sqrt{5} - 1)L/2$ de modo que M está para L assim como m está para M . Obtém-se dois intervalos $I_1 = [w_k^{\min}, w_k^{\min} + M]$ e $I_2 = [w_k^{\min} + m, w_k^{\max}]$. A partir de um critério de menores valores de $J(\cdot)$ nos extremos seleciona-se I_1 ou I_2 que será o próximo intervalo $[w_{k+1}^{\min}, w_{k+1}^{\max}]$, até que o critério de parada seja ativado.

Passo 0: Arbitrar w_0^{\min} e w_0^{\max} .

Escolher um número real tol.

Fazer $k=0$

Passo 1: Calcular:

$$L = w_k^{\max} - w_k^{\min}$$

Passo 2 Critério de Parada:

$$\text{Se } (L < \text{tol}) \text{ Então } (w_{aprox}^* = w_k^{\min} + \frac{L}{2} \text{ e fim})$$

Passo 3 Calcular:

$$\begin{aligned} m &= \frac{3 - \sqrt{5}}{2} L \\ M &= \frac{\sqrt{5} - 1}{2} L \end{aligned} \quad (7.4)$$

Passo 4 Seleção do Próximo Intervalo

$$\begin{array}{ll} \text{Se } (J_m \leq J_M) \text{ Então } (w_{k+1}^{\min} = w_k^{\min} \text{ e } w_{k+1}^{\max} = w_k^{\min} + M) & ; \text{ onde: } J_m = J(w_k^{\min} + m) \\ \text{Se } (J_m > J_M) \text{ Então } (w_{k+1}^{\min} = w_k^{\min} + m \text{ e } w_{k+1}^{\max} = w_k^{\max}) & J_M = J(w_k^{\min} + M) \end{array}$$

Passo 5 Fazer $k = k + 1$ e retornar ao Passo 1.

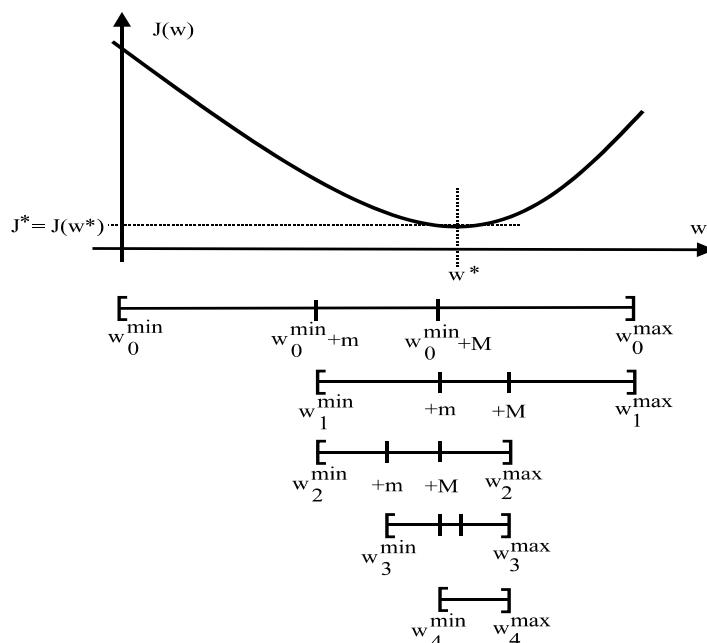


Figura 7-2. Otimização unidimensional empregando Secção Áurea

7.3 Métodos de Busca empregando Gradiente

Em muitos problemas de otimização é possível a determinação do gradiente de J no ponto w , denotado por $\nabla J(w)$, $w \in R^n$ e definido por:

$$\nabla J(w) = \begin{bmatrix} \frac{\partial J}{\partial w_1}(w) & \dots & \frac{\partial J}{\partial w_n}(w) \end{bmatrix}^T \quad (7.5)$$

De fato, utilizando-se a fórmula para expansão em série de Taylor, verifica-se que quando se realiza um deslocamento de h a partir de um certo ponto w , a função $J(\cdot)$ varia segundo a expressão:

$$J(w+h) = J(w) + \nabla^T J(w)h + o(\|h\|^2) \quad (7.6)$$

ou seja,

$$J(w+h) - J(w) \approx \langle \nabla J(w), h \rangle \quad (7.7)$$

onde $\langle \cdot, \cdot \rangle$ denota produto escalar.

Lembrando que o produto escalar é calculado através de

$$\langle x | y \rangle = \|x\| \|y\| \cos(\phi) \quad (7.8)$$

onde ϕ é o ângulo entre x e y , daí:

$$J(w+h) - J(w) = 0 \quad \text{se } h \perp \nabla J(w) \quad (7.9)$$

$$J(w+h) - J(w) = \text{máx} \quad \text{se } h // \nabla J(w) \quad (7.10)$$

De onde conclui-se, que $\Delta J(w) = J(w+h) - J(w)$ é máximo quando o deslocamento h está na direção de $\nabla J(w)$. Ou seja, $\nabla J(w)$ aponta na direção de máximo crescimento de $J(\cdot)$ a partir do ponto w .

Por outro lado, $\Delta J(w)$ é nulo se h é perpendicular a $\nabla J(w)$. Ou seja, a função $J(\cdot)$ não varia à medida que caminhamos (infinitesimalmente) na direção h a partir de w . Como o lugar geométrico $\Gamma = \{w \in R^n \mid J(w) = \text{constante}\}$ é uma curva de nível, verifica-se que $\nabla J(w)$ é perpendicular às curvas de nível de $J(\cdot)$.

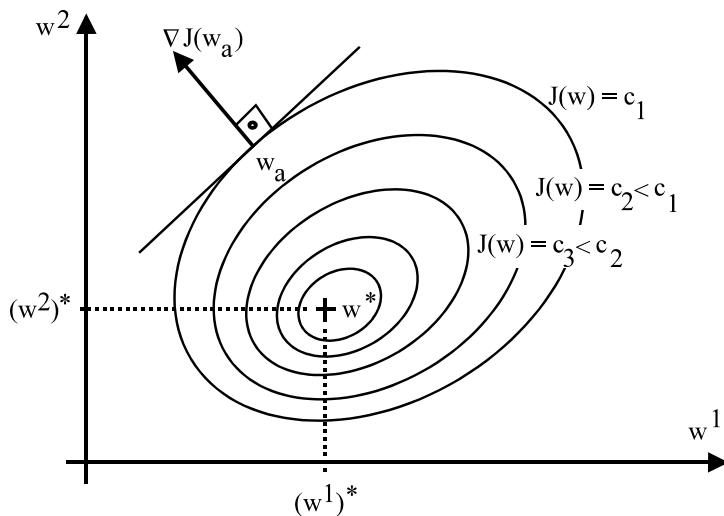


Figura 7-3. Curvas de Nível e Gradientes de uma função $J: \mathbb{R}^2 \rightarrow \mathbb{R}$.

7.3.1 Método da Máxima Declividade

Uma vez que o gradiente $\nabla J(w)$ aponta na direção de máximo crescimento de $J(\cdot)$, uma idéia é realizar buscas do ponto de mínimo através de deslocamento em sentidos negativos na direção de $\nabla J(w)$.

Passo 0: Arbitrar w_0 e tol.

Fazer $k=0$.

Passo 1: Calcular $h_k = \nabla J(w_k)$

Passo 2: Critério de Parada:

Se ($\|h_k\| \leq tol$) Então ($w_{aprox}^* = w_k$ e fim)

Passo 3: Obter λ_k^* , a solução do problema auxiliar de otimização unidimensional na variável λ

$$\min_{\lambda \in \mathbb{R}} J(w_k + \lambda h_k)$$

Passo 4: Fazer $w_{k+1} = w_k + \lambda_k^* h_k$, $k=k+1$ e retornar ao Passo 1.

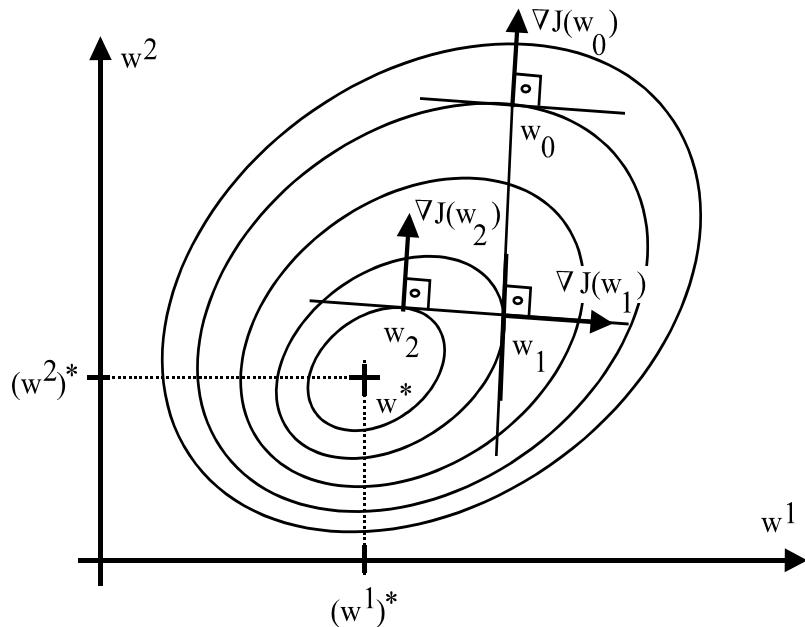


Figura 7-4. Algumas iterações do método da máxima declividade.

O método de máxima declividade costuma apresentar uma boa estabilidade, pois sempre avança em direção a pontos com menor valor da função.

Sua convergência é considerada lenta, principalmente nas regiões mais próximas da solução, onde pode apresentar um movimento em forma de “zig-zag” em torno da solução.

7.3.2 Método de Newton

É de se esperar que um algoritmo de otimização que empregue aproximação quadrática tenda a possuir uma velocidade de convergência maior que um utilizando aproximação linear. De fato, para funções quadráticas o ponto estacionário pode ser calculado resolvendo-se a equação que se obtém igualando a sua derivada a zero. Se a matriz H for positivo definida, o ponto estacionário será um ponto de mínimo.

Seja uma função quadrática

$$Q(h) = \frac{1}{2} h^T H h + b^T h + c \quad (7.11)$$

Assumindo que H seja positivo definida, o ponto de mínimo de $Q(\cdot)$ é dado pela solução de

$$Hh + b = 0 \Rightarrow h = -H^{-1}b \quad (7.12)$$

No caso de uma função não necessariamente quadrática, pode-se utilizar novamente a expansão em série de Taylor:

$$J(w+h) = J(w) + \nabla^T J(w)h + \frac{1}{2} h^T H(w)h + o(\|h\|^3) \quad (7.13)$$

onde a matriz $H(w)$ é a Hessiana calculada no ponto w :

$$H(w) = \begin{bmatrix} \frac{\partial^2 J}{\partial w_1 \partial w_1}(w) & \dots & \frac{\partial^2 J}{\partial w_n \partial w_1}(w) \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 J}{\partial w_1 \partial w_n}(w) & \dots & \frac{\partial^2 J}{\partial w_n \partial w_n}(w) \end{bmatrix} \quad (7.14)$$

O valor de h que minimiza $Q(h) := J(w+h)$ é, portanto

$$h = -H^{-1}(w) \nabla J(w) \quad (7.15)$$

Utilizando-se estes conceitos, pode-se descrever o método de Newton.

A idéia é, em torno de cada ponto w_k , utilizar uma aproximação quadrática para calcular h_k , empregando expressões do tipo (7.15):

Passo 0: Arbitrar w_0 e tol.

Fazer $k=0$.

Passo 1: Calcular: $h_k = \nabla J(w_k)$ e $H_k^{-1} = H(w_k)$

Passo 2: Critério de Parada:

Se ($\|h_k\| \leq tol$) Então ($w_{aprox}^* = w_k$ e fim)

Passo 3: Obter λ_k^* , a solução do problema auxiliar de otimização unidimensional na variável λ

$$\min_{\lambda \in R} J(w_k + \lambda H_k h_k)$$

Passo 4: Fazer $w_{k+1} = w_k + \lambda_k^* h_k$, $k=k+1$ e retornar ao Passo 1.

A dificuldade em se utilizar o método de Newton está na necessidade de derivadas segundas para caracterizar a Hessiana $H(\cdot)$ e de inversão de matrizes para o cálculo de $H^{-1}(\cdot)$.

7.4 Métodos de Busca sem empregar Gradiente

7.4.1 Método da Busca Direta

Uma versão de busca direta é a realização de explorações, a partir de um dado ponto w_k , em direções paralelas aos eixos de coordenadas. Após a realização de explorações em todas as coordenadas é gerada uma direção h_k para busca unidimensional do passo λ_k^* , permitindo a atualização $w_{k+1} = w_k + \lambda_k^* h_k$.

Passo 0. Arbitrar w_0 e Δx_0 . Escolher um número positivo tol.

Fazer $k=0$.

Passo 1. Iniciar exploração na direção da coordenada i :

Fazer $i=1$

Passo 2: Verificar se o sentido indicado por Δx_k^i é adequada:

$$\text{Se } \left(J \begin{pmatrix} w_k^1 \\ \vdots \\ w_k^i + \Delta x_k^i \\ \vdots \\ w_k^n \end{pmatrix} \leq J \begin{pmatrix} w_k^1 \\ \vdots \\ w_k^i \\ \vdots \\ w_k^n \end{pmatrix} \right) \text{ Então } (h_k^i = \Delta x_k^i)$$

Senão ($\Delta x_k^i = -\Delta x_k^i$).

Passo 3: Caso tenha sido mudado o sinal de Δx_k^i , verificar se o seu valor é adequado:

$$\text{Se } \left(J \begin{pmatrix} w_k^1 \\ \vdots \\ w_k^i + \Delta x_k^i \\ \vdots \\ w_k^n \end{pmatrix} \leq J \begin{pmatrix} w_k^1 \\ \vdots \\ w_k^i \\ \vdots \\ w_k^n \end{pmatrix} \right) \text{ Então } (h_k^i = \Delta x_k^i)$$

Senão $\left(\Delta x_k^i = \frac{\Delta x_k^i}{2} \right)$.

Passo 4: Se $(\|\Delta x_k^i\| \leq tol)$

Então ($h_k^i = 0$, $i = i + 1$ e retornar ao Passo 2 se $i \leq n$).

Passo 5: Critério de Parada:

Se $(\|h_k\| \leq tol)$ Então ($w_{aprox}^* = w_k$ e fim)

Passo 6: Obter λ_k^* , a solução do problema auxiliar de otimização unidimensional na variável λ

$$\min_{\lambda \in R} J(w_k + \lambda h_k)$$

Passo 7: Fazer $w_{k+1} = w_k + \lambda_k^* h_k$, $k=k+1$ e retornar ao Passo 2.

7.4.2 Método dos Poliedros Flexíveis

A idéia básica no método dos poliedros flexíveis é deformar, a cada iteração, um poliedro, de modo que este caminhe em uma direção descendente.

Passo 0: Escolher $n+1$ pontos w_0^1, \dots, w_0^{n+1} .

Fazer $k=0$.

Escolher números reais positivos tol , α , β , γ e δ .

Passo 1: Determinar os vértices com pior (w_k^H) e melhor (w_k^L) custo associado $J(\cdot)$:

$$w_k^H \text{ tal que } J(w_k^H) = \max_{i=1, \dots, n+1} \{J(w_k^i)\}$$

$$w_k^L \text{ tal que } J(w_k^L) = \min_{i=1, \dots, n+1} \{J(w_k^i)\}$$

Passo 3: Critério de Parada:

$$\text{Calcular } P = \sum_{i=1}^{n+1} |w_k^i - w_k^L|.$$

$$\text{Se } (P < \text{tol}) \text{ Então } \left(w_{\text{aprox}}^* = \frac{1}{n+1} \sum_{i=1}^{n+1} w_k^i \text{ e fim} \right)$$

Passo 4: Determinar o centróide da face oposta ao vértice w_k^H :

$$c = \frac{1}{n} \left[\sum_{i=1}^{n+1} w_k^i - w_k^H \right] \quad (7.16)$$

Passo 5. Reflexão:

$$w_k^R = c + \alpha(c - w_k^H)$$

Passo 6: Expansão, se for o caso:

$$\text{Se } (J(w_k^R) < J(w_k^L)) \text{ Então } \left(w_k^E = c + \gamma(w_k^R - c) \right)$$

e

$$\text{Fazer } w_{k+1}^i = \begin{cases} w_k^E & \text{se } i \text{ corresponde a } w_k^H \\ w_k^i & \text{para os demais vértices} \end{cases}$$

Passo 7: Redução, se for o caso:

$$\text{Se } (J(w_k^R) \geq J(w_k^H))$$

$$\text{Então } \left(\begin{array}{l} w_{k+1}^i = w_k^L + \delta(w_k^i - w_k^L) \forall w_k^i \neq w_k^L \\ w_{k+1}^i = w_k^i \text{ para } w_k^i = w_k^L \end{array} \right)$$

Passo 8: Contração, se for o caso:

$$\text{Se } (J(w_k^R) > J(w_k^L) \forall w_k^i \neq w_k^H) \text{ Então } \left(w_k^C = c + \beta(w_k^H - c) \right)$$

e

$$\text{Fazer } w_{k+1}^i = \begin{cases} w_k^C & \text{se } i \text{ corresponde a } w_k^H \\ w_k^i & \text{para os demais vértices} \end{cases}$$

Passo 9: Fazer $k = k+1$ e Retornar ao Passo 1

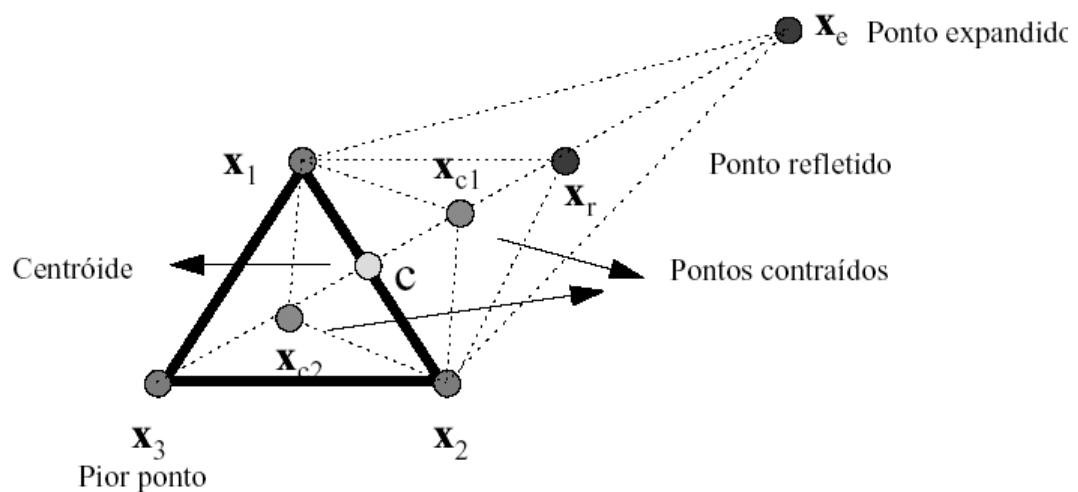


Figura 7-5. Poliedro Flexível.

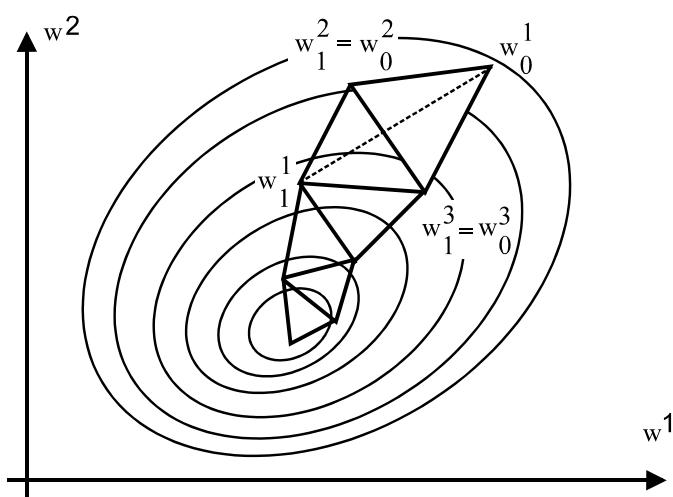


Figura 7-6. Método dos Poliedros Flexíveis.

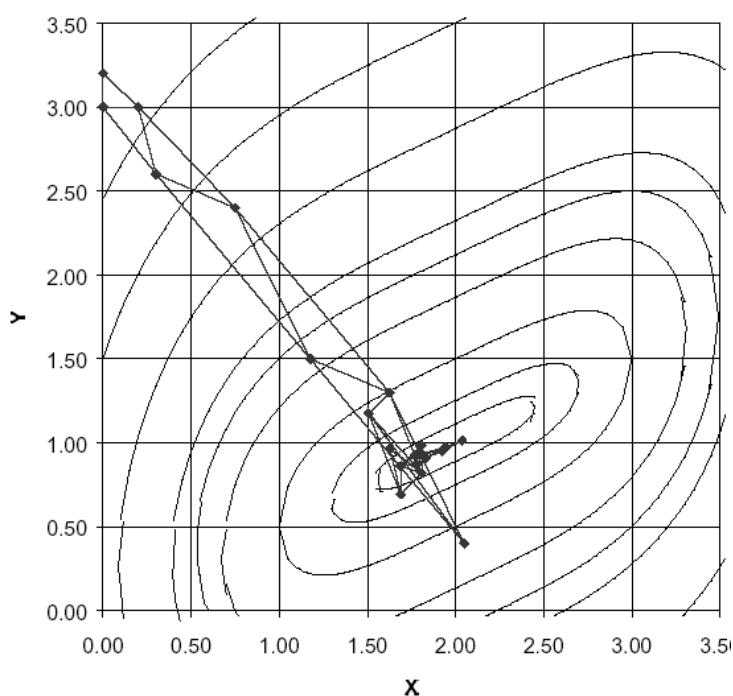


Figura 7-7. Exemplo de Aplicação do Método dos Poliedros Flexíveis.

7.4.3 Algoritmos Genéticos

Os algoritmos genéticos utilizam um conceito análogo ao de seleção natural da teoria da evolução de Darwin. A idéia básica, no caso da minimização de funcionais de custo $J(w)$ em relação a w , é codificar o valor de w em uma cadeia de dados que recebe o nome de cromossomo. Em uma dada população de cromossomos existem aqueles, associados a w , que resultam em valores baixos de $J(w)$. Estes cromossomos são considerados mais aptos para reprodução. A reprodução consiste em selecionar pares de cromossomos que sofrem os efeitos de crossover e mutação. O crossover consiste em trocar segmento da cadeia de dados de um cromossomo com o seu par, eventualmente originando cadeias de dados novas que ainda não existia na população. A mutação é a alteração aleatória de algum dado codificado no cromossomo.

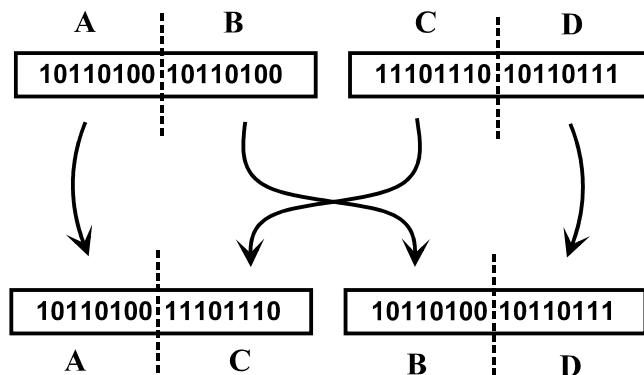


Figura 7-8. Mecanismo de crossover nos algoritmos genéticos.

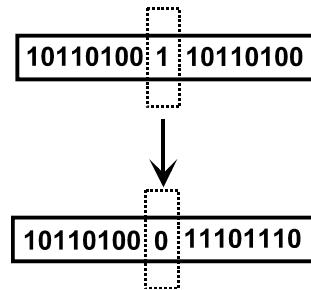


Figura 7-9. Mecanismo de mutação nos algoritmos genéticos.



Figura 7-10. Exemplo de Roleta para Seleção.

Passo 0: Selecionar os tamanhos da população inicial N e população máxima M.

Fazer $k = N$.

Escolher um número real positivo tol e um número inteiro $0 < t < N$ para teste de convergência.

Passo 1: Geração da população inicial

$$P = \{ (cromo_1, J(w_1)), \dots, (cromo_N, J(w_N)) \}$$

$$\text{onde } cromo_i = w_i^1 : w_i^2 : \dots : w_i^n$$

Passo 2: Ordenar a lista P de modo que $J(w_{i-1}) \leq J(w_i)$

Passo 3: Seleção do par de cromossomos para reprodução

Gerar dois números aleatórios r_1 e r_2 distribuídos sobre o intervalo $(0,1)$, com elevada densidade próximo a 0 e baixa próximo a 1.

Calcular os índices dos cromossomos sorteados

$$m = \text{IFIX}(r_1 * N) + 1$$

$$f = \text{IFIX}(r_2 * N) + 1$$

Passo 4: Mutação:

Alterar, aleatoriamente, 1 bit de $cromo_m$ e 1 bit de $cromo_f$

Passo 5 Crossover:

Quebrar $cromo_m$ em dois fragmentos $frag_m^1$ e $frag_m^2$ de comprimento aleatório.

Quebrar $cromo_f$ em dois fragmentos $frag_f^1$ e $frag_f^2$ de comprimentos idênticos aos $frag_m^1$ e $frag_m^2$.

Montar $cromo_{d1} = frag_f^1 e frag_m^2$ e $cromo_{d2} = frag_m^1 e frag_f^2$

Passo 6: Fazer $k = k + 2$ e inserir $cromo_{d1}$ e $cromo_{d2}$ na lista P, na posição correta da ordenação.

Passo 7: Seleção Natural - eliminação dos cromossomos de pior desempenho em termos de $J(.)$

Se ($k \geq M$)

Então ($P = P - \{(cromo_{N+1}, J(w_{N+1})), \dots, (cromo_k, J(w_k))\}$ e $k = M$)

Passo 8: Critério de Parada:

Se ($J(w_t) - J(w_1) < tol$) Então $(w_{aprox}^* = w_1)$

Senão (retornar ao Passo 2).

7.5 Métodos Estendidos

7.5.1 Zona Tabu

Muitos algoritmos de otimização possuem a tendência de serem capturados por pontos de mínimo que podem não ser ainda o ponto de mínimo global.

Uma forma de se tentar determinar o ponto de mínimo global é repetir o algoritmo que tende a convergir para um ponto de mínimo local, mas a partir de diferentes pontos iniciais. O método da zona tabu tenta disciplinar a geração de novos pontos iniciais para a aplicação dos algoritmos de busca de mínimo local.

A idéia fundamental no método da zona tabu é a construção de listas de pontos iniciais que tendem a resultar em convergência para pontos de mínimo piores que o melhor já conhecido.

Passo 0: Arbitrar w_0 .

Selecionar um número real $p \in (0.5, 1)$.

Selecionar σ , tol e o comprimento máximo de listas M

Selecionar um algoritmo de minimização local min_local:

$$w_{local}^* = \min_local(w_0)$$

Passo 1: Montar a lista ZT = vazio (lista de Zona Tabu)

Montar a lista MP = $(w_0, J(w_0))$ (lista de Melhores Pontos)

Passo 2: Se ($ZT = \text{vazio}$) Então (Selecionar um elemento qualquer de MP)

Senão (Selecionar um elemento qualquer $(w_S, J(w_S))$ de MP com probabilidade p ou de ZT com probabilidade $(1-p)$)

Passo 3: Promover espalhamento em torno de w_S :

Gerar vetor aleatório r de distribuição normal $N(0, \sigma^2 I)$
Fazer $w_C = w_S + r$

Passo 4: Obtenção de um mínimo local a partir de w_C :

$$w_C^* = \min_local(w_C)$$

Passo 5: Avaliação de w_C^* :

Se $(J(w_C^*) < J(w_S))$ Então (transferir todos os elementos de MP para TZ e inserir $(w_C, J(w_C))$ em MP)

Se $(|J(w_C^*) - J(w_S)| < tol)$ Então (inserir $(w_C, J(w_C))$ em MP)

Se $(J(w_C^*) > J(w_S))$ Então (inserir $(w_C, J(w_C))$ em ZT)

Passo 6: Critério de Parada

Se (comprimento(MP) $\geq M$) Então ($w_{approx}^* = w_k, w_k \in MP$)

Senão (retornar ao Passo 2)

7.5.2 Recozimento Simulado

O mecanismo de recozimento simulado (*simulated annealing*) procura evitar que um algoritmo de busca descendente seja capturado por um ponto de mínimo local e para tal, promove a geração de direções não localmente favoráveis, com uma probabilidade não nula. A analogia física que se faz é com relação à agitação térmica imposta às moléculas de um dado material durante um processo de témpera (ou recozimento).

Entretanto, à medida que o algoritmo de busca se aproxima do ponto de mínimo, possivelmente global, a temperatura deve ser reduzida, de modo a diminuir a agitação térmica.

O mecanismo de recozimento simulado pode ser incorporado em diversos algoritmos de busca. Como exemplo, é apresentado, a seguir, a adaptação para o caso do método de máxima declividade.

Passo 0: Arbitrar w_0 e tol.

Fazer $k=0$.

Escolher um número real T (Temperatura Inicial).

Escolher um número real $\rho \in (0,1)$ (Taxa de Resfriamento)

Passo 1: Calcular $h_k = -\nabla J(w_k)$

Passo 2: Critério de Parada:

Se $(\|h_k\| \leq tol)$ Então ($w_{aprox}^* = w_k$ e fim)

Passo 3: Geração de uma direção alternativa, por exemplo aleatória, h_A

Passo 4: Critério de Aceitação (ou de Metropolis):

Calcular a diferença de desempenho entre h_k e h_A :

$$\delta = J(w_k + h_A) - J(w_k + h_k)$$

Calcular a probabilidade de aceitação p_a pela fórmula:

$$p_a = \begin{cases} 1 & se \delta < 0 \\ e^{-\frac{\delta}{T}} & se \delta > 0 \end{cases}$$

Passo 5: Escolha da direção de busca:

Gerar um número aleatório uniformemente φ distribuído entre $[0,1]$

Se ($\varphi \leq p_a$) Então ($h_k = h_A$)

Passo 6: Obter λ_k^* , a solução do problema auxiliar de otimização unidimensional na variável λ

$$\min_{\lambda \in R} J(w_k + \lambda h_k)$$

Passo 7: Fazer $w_{k+1} = w_k + \lambda_k^* h_k$, $k=k+1$, $T = \rho T$ e retornar ao Passo 1.

Existem outras formas de promover o ‘resfriamento’ diferentes da incluída no Passo 7: $T = \rho T$, tais como o $T = T - \rho_1$, $T = \rho_1/(1+\rho_2*T)$ e $T = \rho_1/\log(1+t)$, onde ρ_1 e ρ_2 são constantes convenientemente escolhidas.

7.6 Seleção do Método de Otimização

A escolha do método para solucionar um dado problema de otimização requer uma análise cuidadosa das condições exigidas para a aplicabilidade de cada um. Entre os diversos fatores, estão a disponibilidade ou não, de expressões para as derivadas do funcional de custo (por exemplo, no caso dos métodos de máxima declividade e de Newton), quantidade de memória que pode ser alocada para o armazenamento de dados (zona tabu, algoritmos genéticos), utilização ou não em ambientes de tempo real, existência ou não de uma boa estimativa para w^* (facilitando a escolha de w_0), convexidade da função a ser minimizada (caso em que a solução existe e é única se o problema não possui restrições), simplicidade de programação e outros.

8 REDES NEURAIS ARTIFICIAIS

8.1 Modelos e Arquiteturas

O funcionamento da grande maioria dos computadores digitais em uso atualmente é baseado no princípio de centralizar todas as operações em um processador muito poderoso e complexo. Essa é a idéia básica da arquitetura de Von Neumann, assim chamada, pois foi proposta por John Von Neumann, um dos pioneiros da computação moderna, em 1947.

A maneira tradicional de usar tais computadores tem sido o chamado enfoque algorítmico que consiste em planejar uma seqüência precisa de passos (um programa de computador) que é executada pelo computador. Algumas vezes estas seqüências envolvem operações com valores numéricos. Nestas condições tem-se o enfoque numérico. Por outro lado, baseado na hipótese de que o processo de pensamento do especialista humano pode ser modelado usando um conjunto de símbolos e um conjunto de regras lógicas, os computadores são também utilizados para realizar inferências lógicas. Este é o chamado enfoque simbólico. Nestes enfoques é necessária a intervenção de uma pessoa que entenda do processo (o especialista humano) e também de alguém para programar o computador.

O enfoque algorítmico, tanto os de natureza numérica quanto simbólica, podem ser muito úteis para a classe de problemas onde é possível encontrar uma seqüência precisa de operações matemáticas ou regras, por exemplo, inversão de matrizes e o diagnóstico médico de certas doenças que já são bem compreendidas. Entretanto, tais enfoques podem apresentar limitações:

a) Processamento predominantemente seqüencial: devido à centralização do processamento em torno de um processador, as instruções têm que ser executadas seqüencialmente, mesmo que dois conjuntos de instruções não sejam inter-relacionadas. Surge então um "gargalo" em torno do processador central que impõe uma limitação na velocidade máxima de processamento.

b) Representação Local: o conhecimento é localizado no sentido de que um conceito ou regra é localizado em uma área precisa na memória do computador. Tal representação não é resistente a danos. Também uma pequena corrupção em uma das instruções que serão executadas pelo processador (p. ex. erro em apenas 1 bit) pode facilmente arruinar a computação seqüencial. Outro problema é a diminuição da confiabilidade de um programa à medida que sua complexidade aumenta, visto que aumenta a probabilidade de erros na programação.

c) Dificuldade de aprendizado: utilizando-se a definição de que aprendizado em computação é a construção ou a modificação de alguma representação ou modelo computacional (Thornton, 1992), é difícil simular "aprendizado" usando os enfoques algorítmico ou simbólico, uma vez que não é trivial incorporar os dados adquiridos via interação com o ambiente no modelo computacional.

Em geral, pode se dizer que os computadores digitais atuais podem resolver problemas que são árduos para humanos, mas freqüentemente são muito difíceis de serem usados para automatizar tarefas que humanos podem realizar com pouco esforço, tal como dirigir um carro ou reconhecer faces e sons nas situações encontradas no mundo real.

O estudo de Redes Neurais Artificiais (RNA, computação neural, conexionismo ou processamento paralelo distribuído) fornece um enfoque alternativo a ser aplicado em problemas onde os enfoques algorítmico e simbólico não são julgados muito adequados.

As RNAs, diferentemente de computadores digitais convencionais, executam sua tarefa usando um grande número de processadores, cada um desses muito simples, mas com uma elevado grau de interconexão entre eles. Ou seja, esses processadores operam em paralelo. A representação do conhecimento é distribuída pelas conexões (sinapses) e o aprendizado é feito alterando-se os valores associados a estas conexões. Todavia, os métodos de aprendizado ainda precisam ser programados e para cada problema específico, um método de aprendizado apropriado deve ser escolhido.

As redes neurais artificiais são, às vezes, modelos tão grosseiros de sistemas nervosos biológicos que seria difícil justificar a palavra neural. A palavra neural é usada hoje mais por razões históricas, dado que os primeiros pesquisadores do assunto vieram das áreas de Biologia ou Psicologia, não das áreas de Engenharia ou Computação.

8.1.1 O neurônio biológico (natural)

O cérebro humano contém aproximadamente 10^{11} células nervosas elementares chamadas de neurônios. Cada um desses neurônios está conectado a cerca de 10^3 a 10^4 outros neurônios. Portanto, estima-se que o cérebro humano teria entre 10^{14} e 10^{15} conexões.

Existem vários tipos de neurônios, cada um com sua função, forma e localização específica, porém os componentes principais de um neurônio de qualquer tipo são: o corpo de neurônio (chamado soma), os dendritos e o axônio, como ilustrado na Figura 8-1.

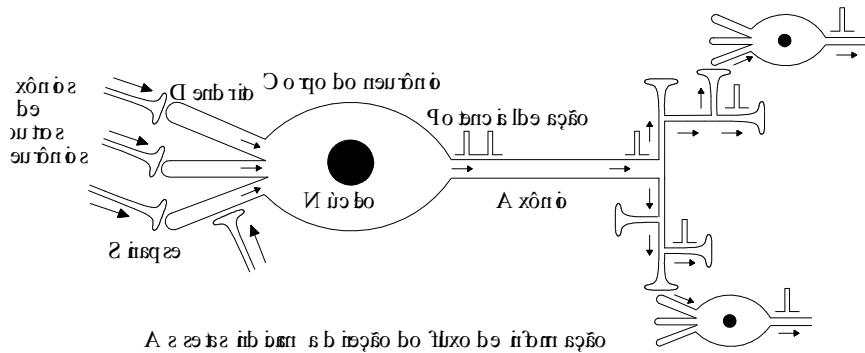


Figura 8-1. Representação esquemática de um neurônio biológico.

O corpo do neurônio mede entre 5 a 10 mm de diâmetro e contém o núcleo da célula. A maioria das atividades bioquímicas necessárias para manter a vida do neurônio, tais como a síntese de enzimas, são realizadas no seu corpo. Os dendritos atuam como canais de entrada para os sinais externos provenientes de outros neurônios e o axônio atua como canal de saída e transmissão.

O final de um dos ramos do axônio tem a forma de um botão com diâmetro em torno de 1 mm e é conectado ao dendrite de um outro neurônio. Tal conexão é chamada de sinapse. Normalmente tal conexão não é física (o axônio e o dendrite não se tocam), mas, eletroquímica.

O corpo do neurônio pode gerar atividade elétrica na forma de pulsos de voltagem chamados de potencial de ação. O axônio transporta o potencial de ação do corpo do neurônio até as sinapses onde certas moléculas, chamadas de neurotransmissores, são liberadas. Estas moléculas atravessam a fenda sináptica e modificam o potencial da membrana do dendrite. Dependendo do tipo do neurotransmissor predominante na sinapse, o potencial da membrana do dendrite é aumentado (sinapse excitatória) ou diminuído (sinapse inibidora). Estes sinais recebidos pelos dendritos de vários neurônios são propagados até o corpo do neurônio onde são aproximadamente somados. Se esta soma dentro de um pequeno intervalo de tempo for acima de um determinado limite o corpo do neurônio gera um potencial de ação que é então transmitido pelo axônio aos outros neurônios.

Depois que o potencial de ação é gerado existe um período refratário quando o neurônio não gera outro potencial de ação, mesmo que receba uma grande excitação. O período refratário que dura entre 3 e 4 ms determina a máxima frequência de disparo do neurônio. Por exemplo, considerando que o mínimo período de um potencial de ação é de 4 ms (1 ms como a mínima duração do pulso + 3 ms como a mínima duração do período refratário) a máxima frequência de trabalho de um neurônio seria 250 Hz.

A velocidade de propagação do potencial de ação ao longo do axônio varia de 0.5 a 130 m/s, dependendo do diâmetro do axônio e da existência da bainha de mielina (uma substância isolante). Dois terços dos axônios do corpo humano têm um pequeno diâmetro (entre 0.0003 e 0.0013 mm) e não são cobertos pela bainha de mielina. Estes axônios transmitem sinais como a temperatura corporal. O outro terço dos axônios possuem um diâmetro largo (0.001 a 0.022 mm), são cobertos pela bainha de mielina e são usados para transmitir sinais vitais que precisam ser processados rapidamente, por exemplo, os sinais de visão.

É interessante ressaltar que o sinal transmitido pelo neurônio é modulado em frequência (FM). Usando este tipo de modulação, o sinal gerado no corpo do neurônio pode ser transmitido pelo axônio até outros neurônios por longas distâncias sem distorções significativas no conteúdo da informação.

8.1.2 Definição de Rede Neural Artificial

A seguinte definição formal de uma Rede Neural Artificial foi proposta por Hecht-Nielsen, 1990:

“Uma Rede Neural Artificial é uma estrutura que processa informação de forma paralela e distribuída e que consiste de unidades computacionais (as quais podem possuir uma memória local e podem executar operações locais) interconectadas por canais unidirecionais chamados de conexões. Cada unidade computacional possui uma única conexão de saída que pode ser dividida em quantas conexões laterais se fizer necessário, sendo que cada uma destas conexões transporta o mesmo sinal, o sinal de saída da unidade computacional. Este sinal de saída pode ser contínuo ou discreto. O processamento executado por cada unidade computacional pode ser definido arbitrariamente com a restrição de que ele deve ser completamente local, isto é, ele deve depender somente dos valores atuais dos sinais de entrada que chegam até a unidade computacional via as conexões e dos valores armazenados na memória local da unidade computacional”.

A definição acima mostra que redes neurais artificiais podem ser vistas como uma subclasse de uma classe geral de arquitetura computacional para processamento paralelo chamada Múltipla Instrução Múltiplos Dados (*Multiple Instruction Multiple Data - MIMD*).

8.1.3 Uma Estrutura Geral para Modelos de Redes Neurais Artificiais

Existem muitos modelos diferentes de Redes Neurais Artificiais, porém cada modelo pode ser definido formalmente pelas seguintes oito características principais:

- Um conjunto de unidades computacionais,
- Um estado de ativação para cada unidade (a_i),
- Uma função de saída F_j para cada unidade, $out_j = F(a_j)$,
- Um padrão de conectividade entre as unidades, ou seja, a topologia da rede, a qual é definida pela matriz de pesos W ,
- Uma regra de combinação usada para propagar os estados de ativação das unidades pela rede (cálculo de net_j),
- Uma regra de ativação usada para atualizar o estado de ativação de cada unidade usando o valor atual do estado de ativação e as entradas recebidas de outras unidades $a_j(k+1) = g[a_j(k), net_j(k)]$,
- Um ambiente externo que fornece informação para a rede e/ou interage com ela,
- Uma regra de aprendizado, usada para modificar o padrão de conectividade da rede, usando informação fornecida pelo ambiente externo, ou seja, para modificar a matriz de pesos W .

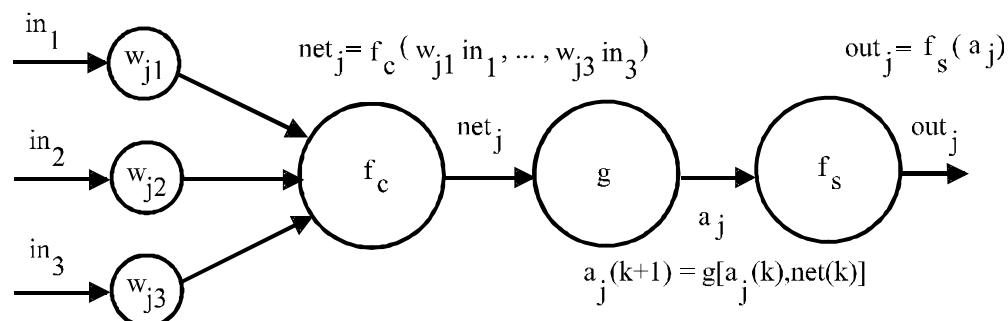


Figura 8-2. Estrutura geral de uma unidade computacional.

8.1.3.1 Alguns Tipos de Unidades Computacionais:

a) Unidade Tipo Linear:

O tipo de unidade computacional mais simples é o tipo linear, onde se tem:

$$\text{out}_i = \text{net}_i = \sum_j w_{ij}x_j + \text{bias}_i \quad (8.1)$$

O valor do limiar (*bias*) pode simplesmente ser interpretado como sendo um outro peso vindo de uma unidade cuja saída é sempre 1.

b) Unidade Tipo TLU:

Em 1943 W. McCulloch e W. Pitts propuseram modelar o neurônio biológico como uma "unidade lógica tipo limiar" (*Threshold Logic Unit*, TLU) com L entradas binárias (0 ou 1) e uma saída binária. Os pesos associados com cada entrada são ± 1 . A saída de tal unidade é alta (1) apenas quando a combinação linear de todas as entradas é maior ou igual a um certo limiar, e baixa (0) no caso contrário.

$$\text{out}_i = F_i(\text{net}_i) = F_i\left(\sum_j w_{ij}x_j + \text{bias}_i\right) \quad (8.2)$$

onde $F_i(z) = 1$ se $z \geq 0$ e $F_i(z) = 0$ se $z < 0$, w_{ij} é o peso vindo da unidade j para a unidade i ($i \leftarrow j$).

c) Unidade Tipo Sigmóide ou Tangente Hiperbólica:

Este é o tipo de unidade mais comumente usado em aplicações de redes neurais. Neste caso tem-se que:

$$\text{net}_i = \sum_j w_{ij}x_j + \text{bias}_i \quad (8.3)$$

e para a unidade tipo sigmóide:

$$\text{out}_i = \text{sig}(\text{net}_i) = \frac{1}{1 + \exp(-\text{net}_i)} \quad (8.4)$$

enquanto, para a unidade tipo tangente hiperbólica:

$$\text{out}_i = \tanh(\text{net}_i) = 2 \text{sig}(\text{net}_i) - 1 \quad (8.5)$$

A saída da unidade tipo sigmóide varia entre 0 e 1, enquanto que a saída da unidade tipo tangente hiperbólica varia entre -1 e 1.

Note que para este tipo de unidade: a) a curva de saída apresenta uma forma de S, b) que tal curva pode ser vista como uma versão suave (isto é, com derivada limitada) da curva de saída da unidade tipo TLU.

d) Unidade tipo Função de Base Radial:

Neste caso tem-se:

$$\text{net}_i = \|x - C_i\| \quad (8.6)$$

onde C_i é um vetor que determina o centro da unidade e:

$$\text{out}_i = \exp[-(\text{net}_i)^2] \quad (8.7)$$

Desta maneira, a saída da unidade é máxima quando $x = C_i$ e decai suavemente à medida que x se afasta do centro C_i .

8.1.4 Topologia de Redes Neurais Artificiais

De acordo com a topologia, uma rede neural artificial pode ser classificada como *feedforward* (sem realimentação local) ou *feedback* (com realimentação local ou recorrente). Em uma rede neural artificial tipo *feedforward* uma unidade envia sua saída apenas para unidades das quais ela não recebe nenhuma entrada direta ou indiretamente (via outras unidades). Em outras palavras em uma rede tipo *feedforward* não existem laços (*loops*). Em uma rede tipo *feedback* os laços existem. A Figura 8-3 mostra os tipos de redes neurais artificiais.

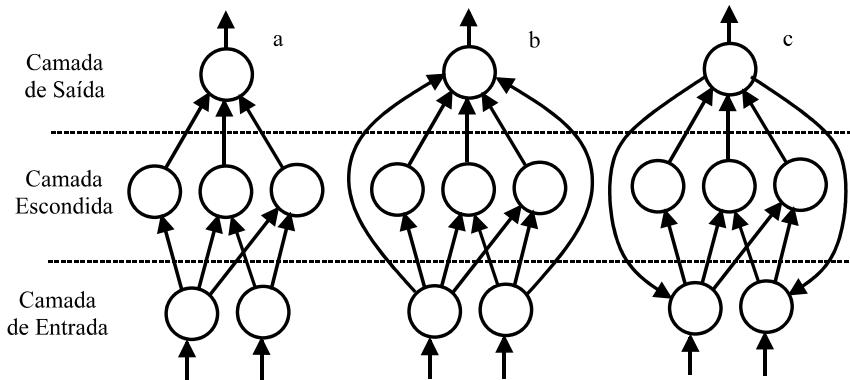


Figura 8-3. Algumas topologias para rede neural artificial: (a) sem realimentação e camadas isoladas, (b) sem realimentação e conexões entre camadas de entrada e saída, (c) com realimentação.

Em redes neurais artificiais tipo *feedforward* as unidades podem ser enumeradas de forma tal que a matriz de pesos W é triangular inferior com diagonal nula (enumere as unidades da entrada para a saída e denote por W_{ij} o peso recebido pela unidade i vindo da unidade j).

Uma rede neural artificial separada em camadas, onde todas as unidades enviam suas saídas apenas para as unidades situadas na próxima camada, é dita ser estritamente feedforward (vide Figura 8-3 a).

Uma rede neural artificial tipo *feedforward* implementa um mapeamento estático do seu espaço de entrada para o seu espaço de saída. Em certos casos necessita-se de camadas escondidas (ou camadas internas) para que a rede possa implementar o mapeamento entrada-saída desejado.

Uma rede tipo *feedback* é, em geral, um sistema dinâmico não linear. Neste último caso, a estabilidade da rede torna-se um tópico de grande importância.

Uma aplicação típica de uma rede neural artificial tipo *feedforward* é o desenvolvimento de modelos não lineares usados em reconhecimento de padrões/classificação. Uma aplicação típica de redes neurais artificiais tipo *feedback* é como uma memória endereçada por conteúdo onde a informação que deve ser gravada corresponde a pontos de equilíbrio estável da rede.

8.1.5 Aprendizado em Redes Neurais Artificiais

Na fase de aprendizado (treinamento) de uma rede neural artificial, uma regra é usada para alterar os elementos da matriz de pesos W (e outros parâmetros modificáveis que a rede possa ter) usando a informação do meio externo, disponibilizada pelo supervisor do aprendizado. Neste contexto, os termos treinamento, aprendizado ou adaptação são sinônimos e são interpretados como alterações nos parâmetros modificáveis da rede neural.

Os diferentes métodos de aprendizado podem ser classificados de acordo com a participação do supervisor no processo de aprendizado.

No grau de supervisão mais forte possível o supervisor fornece diretamente para a rede neural os valores dos pesos. Este é o caso, por exemplo, das Redes de Hopfield. Este tipo de supervisão pode ser qualificado de Muito Forte e, na literatura, redes que utilizam tal tipo de estratégia são referidos como sendo de pesos fixos.

Em um grau de supervisão menor, que denominamos de Supervisão Forte, o supervisor fornece para a rede neural um conjunto de treinamento, ou seja, um conjunto de entradas e suas respectivas saídas desejadas. Deseja-se então que a rede aprenda a "imitar" o supervisor, ou seja, ajuste os seus

pesos de forma a produzir, para cada entrada do conjunto de treinamento a saída desejada fornecida pelo supervisor. Este é o caso de redes tipo *feedforward* treinadas com algoritmos de correção de erro, como o algoritmo *Back-Propagation*. Nestes algoritmos, para cada entrada do conjunto de treinamento a saída produzida pela rede neural é comparada com a saída desejada fornecida pelo supervisor e os pesos são alterados de forma a diminuir esta diferença. Frequentemente, este tipo de aprendizado é denominado na literatura de aprendizado supervisionado.

Na redução seguinte do nível de supervisão, denominada de Supervisão Fraca, o supervisor faz apenas o papel de um crítico fornecendo uma avaliação grosseira da saída da rede neural (por exemplo, certo ou errado, erro grande ou erro pequeno, sucesso ou fracasso) ao invés de fornecer a saída desejada. Nesta categoria estão os algoritmos de aprendizado por reforço (*reinforcement learning*) ou de punição/recompensa (*reward/punishment*).

No menor grau de supervisão, que denominamos de Supervisão Muito Fraca, o algoritmo de treinamento da rede neural tenta descobrir categorias dos dados de entrada e o supervisor participa apenas fornecendo os rótulos para estes agrupamentos. Um exemplo deste tipo de aprendizado está presente nas Redes de Kohonen. Na literatura este tipo de aprendizado é denominado *aprendizado não supervisionado*, um nome que, estritamente falando, não estaria totalmente apropriado, pois ainda ocorre uma pequena (mas efetiva) participação do supervisor no processo de aprendizagem.

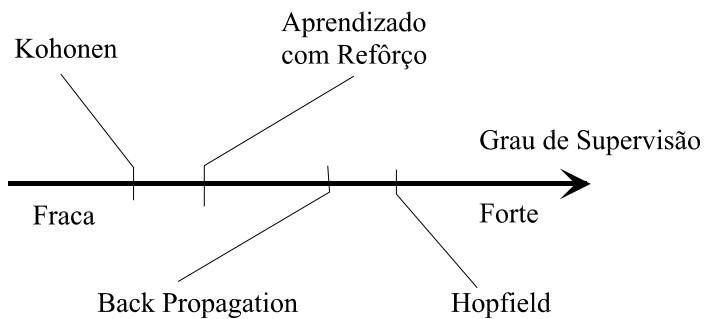


Figura 8-4. Graus de envolvimento do supervisor nas diferentes formas de treinamento de redes neurais.

8.1.6 Capacidade de Aproximação Universal

No Congresso Internacional de Matemática, realiza em Paris no ano de 1900, o eminentíssimo matemático, Prof. David Hilbert apresentou 23 problemas, de cuja discussão poderiam advir significativos avanços na ciência.

O 13º problema provocou, de fato, um impacto decisivo no campo de redes neurais. O problema envolvia responder à questão: “Existem funções contínuas de 3 variáveis que não são representáveis como superposições e composições de funções de 1 variável?” No contexto de uma rede neural, a questão poderia ser adaptada para: “Existem funções contínuas de 3 variáveis que não são sintetizáveis através de ajuste de pesos de um perceptron multi-camadas?” Em termos formais, dada uma função $f: \mathbb{R}^3 \rightarrow \mathbb{R}$, existiria um conjunto de pesos w de forma que $f(x) = \text{rede_neural}(x, w)$, para alguma rede_neural? De forma mais geral, o problema de aproximação universal consiste em caracterizar o conjunto de funções $f: \mathbb{R}^n \rightarrow \mathbb{R}$ que podem ser representados na forma $f(x) = f^{\text{RN}}(x, w)$, onde para um dado conjunto de pesos $f^{\text{RN}}(., w) : \mathbb{R}^n \rightarrow \mathbb{R}$ representa o mapa entrada-saída de uma rede_neural.

Um resultado fundamental a esta questão é fornecido pelo Teorema de Kolmogorov.

Teorema de Kolmogorov: Uma função contínua $f: [0,1]^n \rightarrow \mathbb{R}$, com $n \geq 2$, pode ser representada na forma

$$f(x_1, \dots, x_n) = \sum_{j=1}^{2n+1} h_j \left(\sum_{i=1}^n g_{ij}(x_i) \right) \quad (8.8)$$

onde h_j e g_{ij} são funções contínuas de 1 variável e g_{ij} são funções monotônicas crescentes, fixas e independentes da especificação de f . (Prova: Kolmogorov, 1957)

A dificuldade do Teorema de Kolmogorov reside no fato que não são fornecidos procedimentos construtivos para h e g e também não se garante que se possa arbitrar h e g , tomando-se, por exemplo, funções sigmoidais ou lineares, como apresentam, usualmente, as redes neurais.

Diversos autores como Hornik, Stinchcombe e White, 1989, Funahashi, 1989 e Irie & Miyake, 1988 estenderam e adaptaram os resultados para o caso específico de redes neurais. Aqui é apresentado um resultado obtido por Cybenko que faz referência a funções $\sigma: \mathbb{R} \rightarrow \mathbb{R}$, ditas sigmoidais:

$$\sigma(x) = \begin{cases} 1 & \text{se } t \rightarrow +\infty \\ 0 & \text{se } t \rightarrow -\infty \end{cases} \quad (8.9)$$

Teorema de Cybenko: Seja $f: [0,1]^n \rightarrow \mathbb{R}$ uma função contínua e $\sigma: \mathbb{R} \rightarrow \mathbb{R}$ uma função contínua sigmoidal. Então, dado $\epsilon > 0$, existe:

$$f^{\text{RN}}(x) = \sum_{j=1}^N v_j \sigma(w_j^T x + b) \quad (8.10)$$

onde $w_j \in \mathbb{R}^n$, $b_j \in \mathbb{R}^n$ e $v_j \in \mathbb{R}$, de forma que:

$$|f^{\text{RN}}(x) - f(x)| < \epsilon \quad \forall x \in [0,1]^n \quad (8.11)$$

(Prova: Teorema 2 de Cybenko, 1989).

Em vista do Teorema de Cybenko, redes neurais com uma camada de neurônios dotados de funções de ativação do tipo sigmoidal e uma camada do tipo linear são suficientes para aproximar funções contínuas sobre $[0,1]^n$. Com uma camada adicional de entrada pode-se realizar um escalonamento, de modos que o domínio pode ser estendido a paralelepípedos $[x^{\min}, x^{\max}]$.

Resultados similares são disponíveis para redes neurais empregando funções de base radial. (Nascimento Jr. e Yoneyama, 2002)

8.2 Aprendizado com Supervisão Forte

São apresentados dois métodos de aprendizado onde a participação do supervisor pode ser considerado forte. O primeiro método trata de redes de Hopfield onde os pesos são fixos e ajustados (programados) *a priori*. Neste caso os pesos fixos devem ser calculados de modo que os pontos de equilíbrio de uma rede recorrente sejam os desejados. O segundo método é o de ajuste de pesos baseado na minimização de erro quadrático em relação a uma amostra de pares entrada-saída. Aqui, métodos de otimização numérica são utilizados para a determinação do ponto de mínimo.

8.2.1 Programação Direta dos Pesos

Selecionou-se para estudo, a rede neural de Hopfield. Trata-se de uma rede neural com realimentação e, como tal, são sistemas dinâmicos não lineares que podem exibir comportamentos bastante complicados.

A importância do estudo de tais tipos de redes neurais advém do fato de que possuem aplicações notáveis, particularmente no campo de memórias associativas e para a obtenção de solução para algumas classes de problemas de otimização. No caso de memórias associativas, a rede neural é projetada de modo que os padrões que devem ser memorizados correspondem aos seus pontos de equilíbrio estáveis. No caso da solução de problemas de otimização, a rede neural é projetada de modo que haja convergência para pontos de equilíbrio estáveis e que, eventualmente correspondem a soluções ótimas.

8.2.1.1 Memórias associativas

São entidades que armazenam um conjunto de padrões $\Pi = \{S_1, S_2, \dots, S_M\}$ de modo que, quando lhes é apresentado um novo padrão Y , retornam $S_i \in \Pi$ que mais se aproximam de Y , em um sentido estabelecido a priori. Podem ser interpretadas como memórias endereçáveis por conteúdo, pois permitem que um determinado padrão que esteja armazenado seja recuperado mediante uma entrada incompleta ou corrompida (*pattern completion* e *pattern recognition*). São exemplos de aplicações, o reconhecimento de caracteres manuscritos e o reconhecimento de fisionomias.

8.2.1.2 Redes de Hopfield de tempo discreto

Em função de conveniência matemática, assuma que são empregados os valores 1 e -1 no lugar de 0 e 1, usualmente adotados em representações binárias.

Considere uma rede neural artificial de uma camada e com realimentação, empregando saída bipolar, como a ilustrada na Figura 8-5.

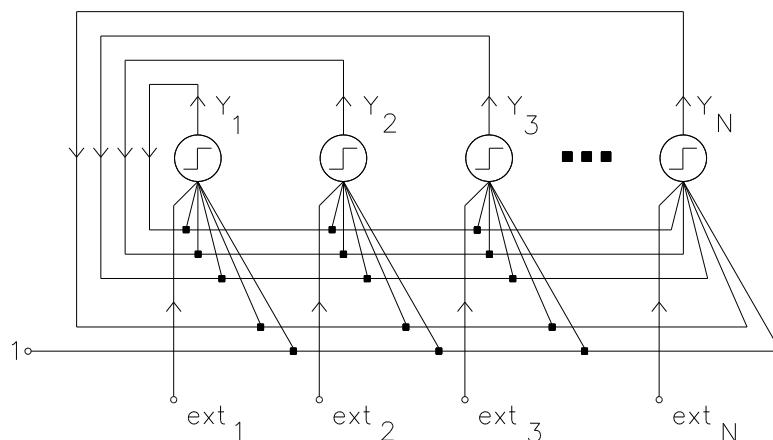


Figura 8-5. Uma RNA de uma camada e com realimentação

onde:

$$y_i = \text{sign}(\text{net}_i) = \begin{cases} 1 & \text{se } \text{net}_i > 0 \\ -1 & \text{se } \text{net}_i \leq 0 \end{cases} \quad (8.12)$$

$$\text{net}_i = \sum_{j=1}^N w_{ij} y_j + \text{bias}_i + \text{ext}_i \quad (8.13)$$

onde, N é o número de unidades na rede, bias_i e ext_i são variáveis constantes internas e externas, respectivamente.

Um padrão $Y = (y_1, \dots, y_N)$ é dito corresponder a um ponto de equilíbrio estável da rede, se:

$$\text{sign}\left(\sum_{j=1}^N w_{ij} y_j\right) = y_i \quad \forall i = 1, \dots, N \quad (8.14)$$

pois, neste caso, a saída não se alteraria na atualização, caso se apresentasse Y como entrada.

Se o padrão que se deseja armazenar é $S = (s_1, \dots, s_N)$, tem-se que os pesos w_{ij} da rede devem ser

$$w_{ij} = k s_i s_j \quad (8.15)$$

com $k > 0$, pois, neste caso:

$$\begin{aligned} \text{sign}\left(\sum_{j=1}^N w_{ij} s_j\right) &= \text{sign}\left(\sum_{j=1}^N k s_i s_j s_j\right) \\ &= \text{sign}(k N s_i) \\ &= s_i \quad \forall i = 1, \dots, N \end{aligned} \quad (8.16)$$

desde que $s_j s_j = 1$, significando que S é um ponto de equilíbrio da rede. Por conveniência, se $k = N^{-1}$,

$$W = \frac{1}{N} SS^T \quad (8.17)$$

onde W é a matriz de pesos com elementos w_{ij} .

No caso de se ter M padrões, uma proposta simples para a determinação da matriz de pesos W , pode ser obtida por superposição, ou seja:

$$W = \frac{1}{N} \sum_{pat=1}^M S^{pat} (S^{pat})^T \quad (8.18)$$

Para que a rede apresente a função de memória associativa, deve-se ter:

$$\text{sign}\left(\sum_{j=1}^N w_{ij} s_j\right) = s_i \quad \forall i = 1, \dots, N \quad (8.19)$$

ou, utilizando a expressão (8.18),

$$\text{sign}\left(\frac{1}{N} \sum_{j=1}^N \sum_{pat=1}^M s_i^{pat} s_j^{pat} s_j\right) = s_i \quad \forall i = 1, \dots, N \quad (8.20)$$

Denotando por S^P um padrão particular que se deseja memorizar na rede, o argumento de $\text{sign}()$ na equação (8.20) é:

$$\begin{aligned} \frac{1}{N} \sum_{j=1}^N \sum_{pat=1}^M s_i^{pat} s_j^{pat} s_j^P &= \frac{1}{N} \sum_{j=1}^N s_i^{pat} s_j^{pat} s_j^P + \frac{1}{N} \sum_{j=1}^N \sum_{pat=2}^M s_i^{pat} s_j^{pat} s_j^P \\ &= s_i^P + (IC) \quad \forall i = 1, \dots, N \end{aligned} \quad (8.21)$$

onde (IC) representa a interferência cruzada entre S^P e outros padrões. Portanto, se a magnitude de (IC) for inferior a 1, não haverá mudança de sinal de s_i^P e a condição de estabilidade do padrão S^P estará satisfeita.

A equação (8.18) expressa uma regra conhecida como a de Hebb. Uma rede neural com realimentação utilizando a regra de Hebb, e com atualização da saída realizada assíncronamente, é chamada de rede de Hopfield de tempo discreto (Hopfield, 1982).

Uma questão importante é obter uma estimativa para o número M de padrões diferentes que pode ser armazenado em uma rede como a da Figura 8-5, com N neurônios. Esta estimativa depende, entre outros fatores, da taxa de erros que se admitiria na recuperação da informação armazenada. Caso não se admitam erros (perfect recall), McEliece et al, 1987, mostram que:

$$M \leq \frac{N}{4 \ln N} \quad (8.22)$$

o que resultaria, para $N = 256$, apenas $M \leq 11$.

Note-se que, caso os padrões a serem armazenados sejam ortogonais entre si, ou seja:

$$(S^{pat_i})^T S^{pat_k} = \begin{cases} 0 & \text{se } pat_i \neq pat_k \\ N & \text{se } pat_i = pat_k \end{cases} \quad (8.23)$$

a capacidade de memória seria $M = N$. Entretanto, com a regra de Hebb, W seria a matriz identidade, o que, do ponto de vista prático, seria desinteressante.

Um conceito interessante utilizado por Hopfield, 1982, é a de função energia em um dado instante kT :

$$H(k) = -\frac{1}{2} Y^T(k) W Y(k) \quad (8.24)$$

Pode-se verificar que $H(k+1) \leq H(k)$, sendo que a igualdade ocorre quando y_i não varia na atualização. Portanto, a tendência da rede neural quando recebe uma entrada S é evoluir dinamicamente, na direção em que há decréscimo da função energia, até que a saída se estabilize em um dado ponto Y^S . Caso a rede esteja treinada, Y^S corresponde à versão recuperada de S , ou seja, um padrão completo e não ruidoso que havia sido armazenado a priori e que melhor se aproxima do S apresentado.

8.2.1.3 Redes de Hopfield com atualização contínua

Após propor o uso de redes neurais bipolares realimentadas e com atualização assíncrona como memórias associativas, Hopfield, 1984, verificou que propriedades computacionais similares poderiam ser obtidas empregando uma versão de tempo contínuo e com saídas contínuas. Redes neurais com realimentação deste tipo são conhecidos como redes de Hopfield do tipo gradiente (Zurada, 1992) ou redes de Hopfield com atualização contínua (Hertz *et al*, 1991).

Uma forma de se implementar redes de Hopfield com atualização contínua é na forma de um circuito. A Figura 8-6 apresenta um possível circuito elétrico analógico para o neurônio:

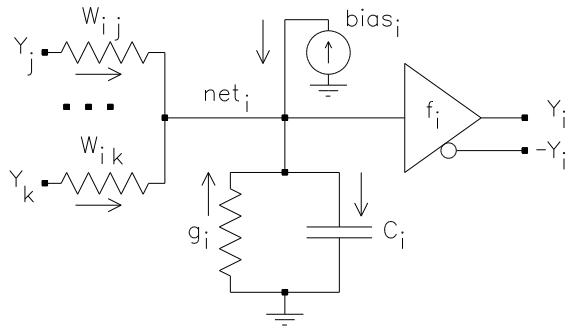


Figura 8-6. Circuito elétrico implementando um neurônio para utilização em redes de Hopfield.

As variáveis net_i e y_j são tensões, $bias_i$ é uma corrente, w_{ij} e g_i são condutâncias, C_i é uma capacitância e f_i é um amplificador de tensão tal que $V_{out} = f(V_{in})$, ou seja, $y_i = f_i(net_i)$, com impedância de entrada com valor muito elevado que pode ser considerado infinito para efeitos práticos.

A dinâmica deste circuito, assumido ser a i -ésima unidade da rede, é descrita pela equação diferencial ordinária:

$$i_i C = C_i \frac{dnet_i}{dt} = bias_i + \sum_{j=1}^N w_{ij}(y_j - net_i) - g_i net_i \quad (8.25)$$

Definindo-se $C = [C_1, \dots, C_N]$ e $G = \text{diag}[G_1, \dots, G_N]$, onde

$$G_i = g_i + \sum_{j=1}^N w_{ij} \quad (8.26)$$

a equação (8.25) pode ser reescrita de forma compacta

$$C \frac{dnet}{dt} = -Gnet + Wf(net) + bias \quad (8.27)$$

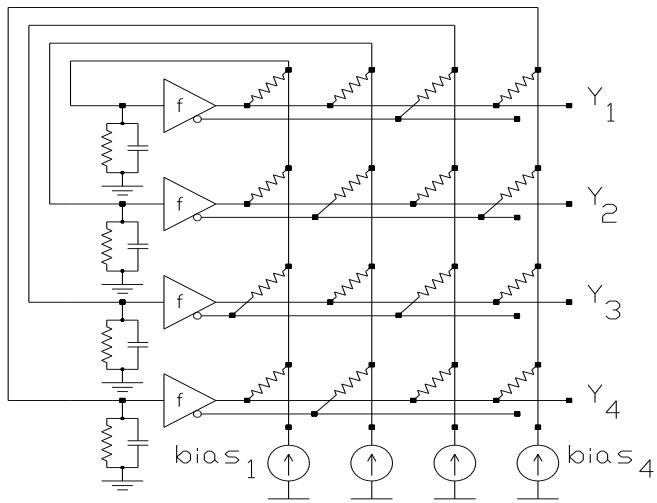


Figura 8-7. Rede de Hopfield com atualização contínua, implementado na forma de circuito elétrico.

A Figura 8-7 ilustra um circuito elétrico análogo para uma rede de Hopfield com 4 unidades e atualização contínua. Assumindo que a matriz de pesos W é simétrico e que as funções de ativação $f_i(\cdot)$ são inferior e superiormente limitadas e monotonicamente crescentes, a função energia que pode ser proposta é:

$$H(t) = -\frac{1}{2} Y^T W Y - \text{bias}^T Y + \sum_{i=1}^N G_i \int_0^{y_i} f_i^{-1}(\tau) d\tau \quad (8.28)$$

Aplicando-se a regra da cadeia e utilizando a simetria da matriz W , verifica-se que:

$$\frac{dH}{dt} = -\sum_{i=1}^N C_i \frac{df_i^{-1}(y_i)}{dy_i} \left(\frac{dy_i}{dt} \right)^2 \quad (8.29)$$

Notando-se que $f_i(\cdot)$ é monotonicamente crescente, tem-se que:

$$\frac{df_i^{-1}(y_i)}{dy_i} > 0 \quad (8.30)$$

e, portanto, $dH/dt \leq 0$, sendo que $dH/dt = 0$ se e somente se $dy_i/dt = 0$ para todas as unidades $i = 1, 2, \dots, N$. Logo, a função energia é não crescente com o tempo, e deixa de ser decrescente apenas quando $dy_i/dt = 0$, ou seja, y_i não mais varia no tempo.

Analogamente ao que ocorria no caso de redes de Hopfield de tempo discreto, a tendência da rede neural é evoluir dinamicamente, na direção em que há decréscimo da função energia, até que a saída se estabilize em um dado ponto Y^* . Pontos que exibem tal característica são conhecidos na literatura como; ponto de equilíbrio e, desde que a condição inicial Y^0 esteja na vizinhança de Y^* (ou seja, na região de atração de Y^*), $Y(t) \rightarrow Y^*$ à medida que $t \rightarrow \infty$.

Se em um problema de otimização, a função custo pode ser expressa da forma:

$$H(t) = -\frac{1}{2} Y^T W Y - \text{bias}^T Y \quad (8.31)$$

então, a rede de Hopfield pode ser utilizada para a sua minimização. De fato, considere-se funções de ativação f_i^λ com ganhos muito elevados:

$$f_i^\lambda(\text{net}_i) = f_i(\lambda \text{net}_i) \quad (8.32)$$

onde λ é um número positivo muito grande e $f_i(\text{net})$ é, por exemplo, do tipo $(1+\exp(-\text{net}))^{-1}$ ou $\tanh(\text{net})$. Neste caso, $H(t)$ pode ser escrito como:

$$H(t) = -\frac{1}{2} Y^T W Y - \text{bias}^T Y + \frac{1}{\lambda} \sum_{i=1}^N G_i \int_0^{y_i} f_i^{-1}(\tau) d\tau \quad (8.33)$$

em vista de $\text{net}_i = f_i^{-1}(y_i)/\lambda$.

Se λ assume valores elevados, o terceiro termo de (8.33) pode ser desconsiderado frente aos demais. Como a rede de Hopfield é tal que: $Y(t) \rightarrow Y^*$, que minimiza $H(t)$, é possível utilizá-la para resolver problemas de minimização (Hopfield e Tank, 1986).

Entre as diversas aplicações propostas na literatura estão:

- Conversão análogo-digital;
- Decomposição de sinais;
- Programação linear;
- Problema do caixeiro viajante.

8.2.2 Ajuste de Pesos Mediante Amostras de Pares Entrada-Saída

O primeiro problema a ser resolvido quando se treina uma RNA usando aprendizado supervisionado é selecionar o conjunto de dados a ser usado para treinar a rede. Tal conjunto de dados deve conter a relação que a rede deve adquirir. Dado que na maioria dos casos esta relação não é conhecida, tal seleção de dados para treinamento pode ser um problema não trivial.

8.2.2.1 Regra de Hebb

Em 1949, Donald Hebb, em seu livro *The Organization of Behavior*, propôs o que hoje é conhecido como a regra de Hebb:

"Quando um axônio do neurônio A está suficientemente próximo para excitar o neurônio B e repetidamente colabora para o seu disparo, acontece algum processo de crescimento ou mudança metabólica em um ou em ambos os neurônios de forma tal que a eficiência do neurônio A para disparar o neurônio B aumenta."

A regra de Hebb já foi apresentada no contexto de redes de Hopfield e pode ser expressa quantitativamente por:

$$\Delta w_{ij} = \eta y_i x_j \quad (8.34)$$

onde $X = [x_1 \ x_2 \ \dots \ x_p]^T$ é o vetor de entrada, $Y = [y_1 \ y_2 \ \dots \ y_q]^T$ é o vetor de saída, $\eta > 0$ é o passo de aprendizado e p e q são respectivamente o número de entradas e saídas da rede.

Um exemplo simples da aplicação da regra de Hebb envolve um modelo conhecido como Matriz Linear Associativa (*Linear Associative Matrix*, *LAM*, ou *Linear Associator*), introduzido por J. A. Anderson, 1968. Neste modelo a saída é uma função linear das entradas, ou seja, $Y = W X$. A Figura 8-8 ilustra tal rede, a qual é usada para associar um conjunto de vetores de entrada $X = [X^1 \ X^2 \ \dots \ X^M]$ a um conjunto de vetores de saída desejados $D = [D^1 \ D^2 \ \dots \ D^M]$, onde M é o número de associações desejadas (ou seja, quando se apresenta X^i à rede, deseja-se que $Y = D^i$).

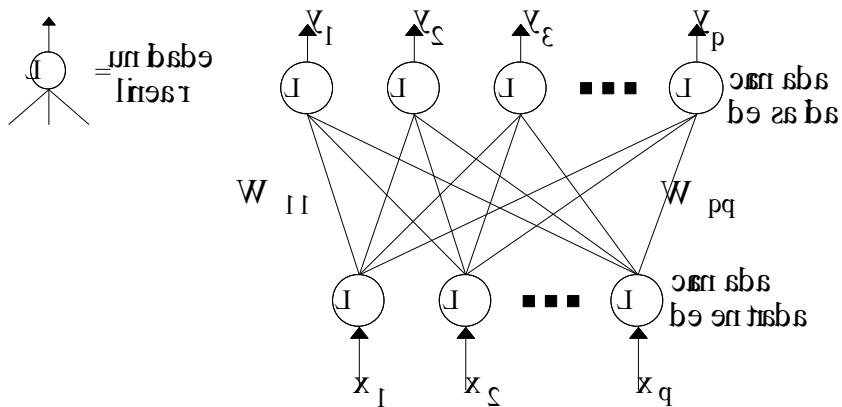


Figura 8-8. Esquema de uma matriz linear associativa.

Na fase de aprendizado, se os vetores X formarem uma base ortonormal, isto é, se eles forem ortogonais entre si e tiverem comprimento unitário, podemos inicializar a matriz de pesos W como uma matriz nula e usar $\eta = 1$. A correta associação entrada-saída será então obtida com apenas uma apresentação de cada par entrada-saída desejada, pois, fazendo:

$$\Delta W(k) = W(k) - W(k-1) = D^k [X^k]^T, \quad 1 \leq k \leq M \quad (8.35)$$

depois da apresentação dos M pares de vetores, a matriz de pesos W será:

$$W(M) = D^1 [X^1]^T + D^2 [X^2]^T + \dots + D^M [X^M]^T \quad (8.36)$$

Algumas limitações importantes deste modelo são:

- 1) O número de associações aprendidas (ou "armazenadas" na rede) é menor ou igual ao número de unidades de entrada ($M \times p$);
- 2) Os vetores, ou padrões, de entrada têm que ser ortogonais entre si;
- 3) Não é possível aprender relações entrada-saída não lineares.

8.2.2.2 O Perceptron de 1 camada e regra Delta

Em 1958, Frank Rosenblatt propôs o modelo *Perceptron*. O modelo do Perceptron de 1 camada de pesos consiste de 1 camada de entrada binárias e 1 camada de saídas também binárias. Não existem camadas escondidas, portanto, existe apenas 1 camada de pesos modificáveis. As unidades de saída utilizam a função bipolar (relé $\{-1, 1\}$ ou degrau unitário $\{0, 1\}$), como ilustrado na Figura 8-9.

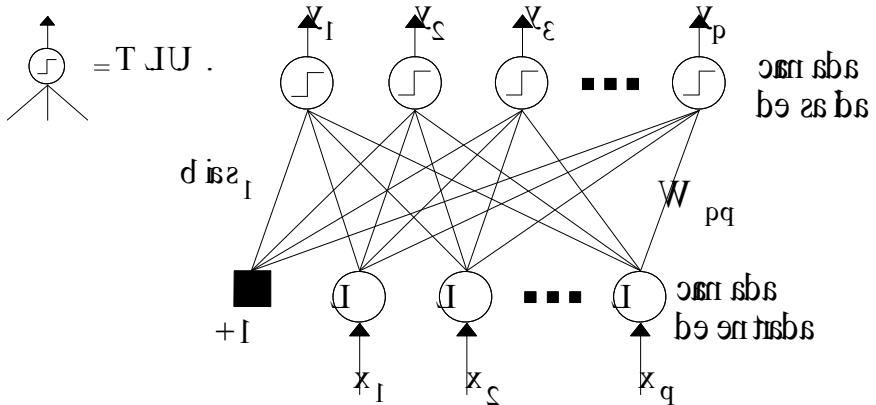


Figura 8-9. Perceptron de 1 camada.

Rosenblatt propôs o seguinte procedimento para o aprendizado supervisionado do Perceptron, hoje conhecido como a regra do Perceptron:

- Passo 1) Aplique um padrão de entrada e calcule a saída Y ;
- Passo 2) Se a saída for correta, vá ao passo 4;
- Passo 3) Se a saída for incorreta e igual a -1,
adicione o valor da entrada ao peso correspondente, $\Delta W_{ij} = X_j$;
Se for igual a +1, subtraia o valor da entrada do
peso correspondente, $\Delta W_{ij} = -X_j$;

Passo 4) Selecione outro padrão de entrada do conjunto de treinamento e retorne ao passo 1.

O passo 2 pode ser expresso alternativamente como:

$$\Delta W_{ij} = \frac{1}{2} [D_i - Y_i] X_j \quad (8.37)$$

Note que esta é uma regra baseada no sinal de erro.

Rosenblatt, 1962, provou que, se uma solução existe, isto é, se existe uma matriz W que fornece a classificação correta para o conjunto de padrões de treinamento, então o procedimento acima encontrará tal solução depois de número *finito* de iterações. Esta prova é conhecida hoje como o *Teorema de Convergência do Perceptron*. Portanto é muito importante entender em que casos tal solução existe.

Considere apenas 1 unidade de saída do Perceptron de 1 camada. Tal unidade divide o espaço de entrada em 2 regiões ou classes (uma região onde a saída é "alta", e outra região onde a saída é baixa, 0 ou -1). Estas regiões são separadas por um *hiperplano* (uma linha para o caso de 2 entradas e 1 plano para 3 entradas). O hiperplano é a *superfície de decisão* neste caso. A posição do hiperplano é determinada pelos pesos e bias recebidos pela unidade de saída. A equação do hiperplano da unidade i é:

$$\sum_{j=1}^p W_{ij} X_j + \text{bias}_i = 0 \quad (8.38)$$

Minsky e Papert, 1969 analisaram a capacidade e limitações do Perceptron de 1 camada e mostraram que este pode resolver problemas que são *linearmente separáveis*, isto é, problemas onde, para cada unidade de saída existe um hiperplano que divide o espaço de entrada nas 2 duas classes corretas. Infelizmente muitos problemas interessantes não são linearmente separáveis. Além disso, Peretto, 1992 mostra que o número de funções lógicas linearmente separáveis reduz a zero à medida que o número de entradas cresce.

A Figura 8-10 mostra as funções lógicas AND, OR e XOR para 2 entradas e 1 saída. Nesta figura, podemos ver que as funções AND e OR são linearmente separáveis. Entretanto a função XOR **não** é linearmente separável visto que não é possível posicionar 1 linha de tal forma a separar os pontos que devem produzir saída 0 (ou -1) dos pontos que devem produzir saída 1.

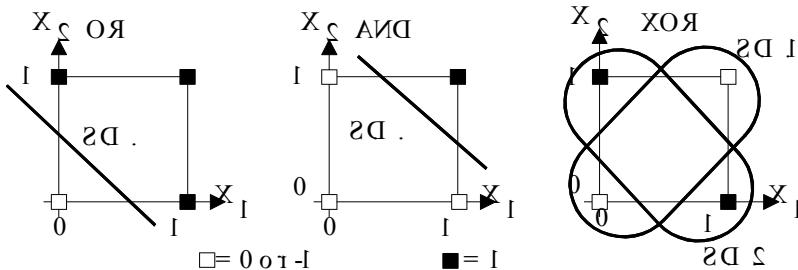


Figura 8-10. Superfícies de Decisão (DS) para os casos OR, AND e XOR.

Outra maneira de ilustrar que o Perceptron de 1 camada não pode resolver o problema do XOR é escrever as 4 inequações que devem ser satisfeitas pelos pesos e bias. Estas inequações são:

- 1) $0 W_1 + 0 W_2 < \text{bias} \Rightarrow \text{bias} > 0$
- 2) $1 W_1 + 0 W_2 > \text{bias} \Rightarrow W_1 > \text{bias}$
- 3) $0 W_1 + 1 W_2 > \text{bias} \Rightarrow W_2 > \text{bias}$
- 4) $1 W_1 + 1 W_2 < \text{bias} \Rightarrow W_1 + W_2 < \text{bias}$

Estas inequações não podem ser satisfeitas simultaneamente, pois os pesos W_1 e W_2 não podem ser positivos e maiores que o bias, ao mesmo tempo em que sua soma seja menor que o bias.

É importante notar que neste caso do Perceptron de 1 camada, não existe uma solução para ser encontrada, isto é, trata-se de uma limitação devido ao *problema de representação* (onde estamos interessados em saber se existe pelo menos 1 solução) e não um *problema de aprendizado* (onde já sabemos que existe pelo menos 1 solução e se deseja encontrar 1 das soluções).

Uma maneira de superar a limitação da separação linear é usar RNAs com mais de 1 camada, tais como o *Perceptron de Múltiplas Camadas (Multi-Layer Perceptron - MLP)*, que introduz camadas extras de unidades (as chamadas camadas escondidas) entre as camadas de unidades de entrada e saída. É possível mostrar que isto é equivalente a definir novas formas para as superfícies de decisão, uma consequência da combinação seqüencial de vários hiperplanos. O problema passa a ser então como determinar os pesos, ou seja, como treinar tal rede com várias camadas.

Comparando a regra do Perceptron e a regra Delta, pode-se perceber que essas duas regras são praticamente idênticas, com a única diferença significativa sendo a omissão da função de limiar (*threshold function*) durante o treinamento no caso da regra Delta. Entretanto elas são baseadas em princípios diferentes: a regra do Perceptron é baseada no princípio de posicionar um hiperplano e a regra Delta é baseada no princípio de minimizar o erro quadrado médio do erro de saída.

Outro ponto interessante é que se a matriz linear associativa for treinada usando a regra Delta ao invés da regra de Hebb, os vetores de entrada não precisam ser mais ortogonais entre si. Basta apenas que sejam linearmente independentes. Entretanto, para p unidades de entrada a rede ainda poderá armazenar apenas até p associações lineares, dado que não é possível ter mais que p vetores linearmente independentes em um espaço com dimensão p .

No caso particular onde:

- 1) η é suficientemente pequeno;
- 2) Todos os padrões de treinamento são apresentados com a mesma probabilidade;
- 3) Existem p padrões de entrada e p unidades de entrada; e,
- 4) Os padrões de entrada formam um conjunto linearmente independente;

mediante o emprego da regra Delta, a matriz de pesos W irá convergir para a solução ótima W^* onde:

$$W^* = \left[D^1 \ D^2 \ \dots \ D^p \right] \left[X^1 \ X^2 \ \dots \ X^p \right]^{-1} \quad (8.39)$$

8.2.2.3 ADALINE e a regra delta

Em 1960 Widrow e Hoff propuseram um neurônio denominado de ADALINE, inicialmente uma abreviação para ADAptive LInear NEuron e mais tarde, ADAptive LINear Element. O ADALINE é uma TLU (Threshold Linear Unit = Unidade Linear de Limiar) com uma saída bipolar (-1 e 1) e múltiplas entradas bipolares (-1 e 1).

Para treinar o ADALINE, Widrow e Hoff, 1960, propuseram a regra Delta, também conhecida como regra de Widrow-Hoff ou algoritmo LMS (Least-Mean-Square). A regra delta é também uma regra baseada no sinal de erro e, portanto, uma regra de aprendizado supervisionado.

O princípio básico da regra Delta é alterar, em cada apresentação de um par entrada-saída desejada do conjunto de treinamento, os pesos da rede na direção que diminui o valor do quadrado do erro da saída, definido como:

$$E^{pat} = \sum_{i=1}^q E_i^{pat} = \sum_{i=1}^q \frac{1}{2} [D_i - Y_i]^2 \quad (8.40)$$

Em outras palavras, a regra Delta é um procedimento de otimização que usa a direção do gradiente executado a cada iteração. Repetindo-se o procedimento acima em todos os pares do conjunto de treinamento, o erro *médio* E^{av} é minimizado onde:

$$E^{av} = \frac{1}{M} \sum_{pat=1}^M E^{pat} \quad (8.41)$$

E, consequentemente, temos:

$$\begin{aligned} \Delta W_{ij} &= -\eta \frac{\partial E^{pat}}{\partial W_{ij}} \\ &= -\eta \frac{\partial E^{pat}}{\partial Y_i} \frac{\partial Y_i}{\partial W_{ij}} \\ &= \eta [D_i - Y_i] \frac{\partial Y_i}{\partial W_{ij}} \end{aligned} \quad (8.42)$$

Porém a função de ativação na saída não é contínua, portanto não é diferenciável e em princípio a equação acima não poderia ser usada. Na realidade o que Widrow e Hoff propuseram foi que, durante o treinamento, fosse usada uma função de ativação linear $Y=W X + bias$. Tal modificação acelera a aprendizagem porque os pesos são alterados mesmo quando a classificação da saída é quase correta, em contraste com a regra do Perceptron que altera os pesos somente quando ocorre um erro de classificação grosso. Outra modificação importante é o uso de entradas bipolares (-1 e 1) ao invés de entradas binárias (0 e 1) onde quando a entrada é 0, os pesos associados com tal entrada não mudam. Usando uma entrada bipolar, os pesos são alterados mesmo quando a entrada está inativa (-1 neste caso).

O procedimento para treinamento do perceptron de 1 camada usando a regra Delta pode ser esquematizado da seguinte maneira:

Passo 1: Inicialize a matriz de pesos W e o vetor bias como pequenos números aleatórios;

Passo 2: Selecione do conjunto de treinamento um par de vetores entrada-saída desejada;

Passo 3: Calcule a saída da rede como: $Y = W X + \text{bias}$

Passo 4: Altera a matriz de pesos W e o vetor bias usando:

$$\Delta W_{ij} = -\eta [D_i - Y_i] X_j \quad (8.43)$$

$$\Delta \text{bias}_i = -\eta [D_i - Y_i] \quad (8.44)$$

Passo 5) Se o vetor do erro de saída $D-Y$ não for suficientemente pequeno para todos os vetores de entrada do conjunto de treinamento, retornar ao Passo 2.

Depois do treinamento, a saída da rede Y é calculada para um vetor de entrada X qualquer, em dois passos:

Passo 1) Calcule o vetor net (de *net input*) como:

$$\text{net} = W X + \text{bias} \quad (8.45)$$

Passo 2) Calcular a saída da rede:

$$Y_i = \begin{cases} +1 & \text{se } \text{net}_i \geq 0 \\ -1 & \text{se } \text{net}_i < 0 \end{cases} \quad (8.46)$$

Esta fase é conhecida como a fase de evocação (*recall phase*). O procedimento acima minimiza a média da diferença entre a saída desejada e a *net input* de cada unidade de saída (o que Widrow e Hoff chamam de *erro medido*). É possível mostrar que, indiretamente, também se está minimizando a média do erro de saída (o que Widrow e Hoff chamam de *erro do neurônio*).

Em relação à capacidade da rede, Widrow e Lehr, 1990 mostram que, em média, um ADALINE com p entradas pode armazenar até 2^p padrões de entrada aleatórios com suas saídas desejadas binárias também aleatórias. O valor 2^p é o limite superior alcançado apenas quando $p \rightarrow \infty$.

8.2.2.4 O papel das unidades escondidas dos Perceptrons

A Figura 8-11 mostra, esquematicamente, um Perceptron de múltiplas camadas. É necessário pelo menos 1 camada de unidades escondidas com uma função de ativação não linear, pois 1 RNA com unidades escondidas lineares é equivalente a uma RNA sem camada escondida. As unidades de saída podem ter funções de ativação linear ou não linear. É também possível a existência de conexões diretas entre as unidades de entrada e saída.

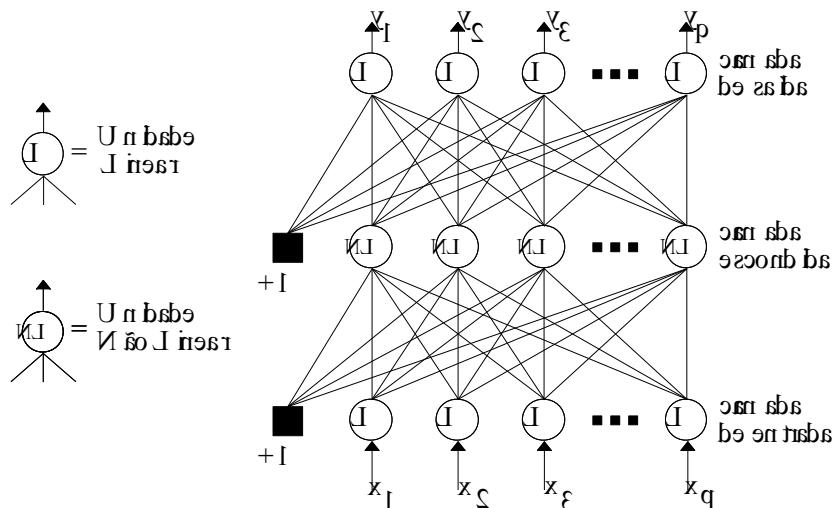


Figura 8-11. Perceptron de Múltiplas Camadas.

O uso de unidades escondidas torna possível a recodificação dos padrões de entrada, criando uma representação diferente. Cada camada escondida executa a recodificação do seu sinal de entrada. Alguns autores referem-se a isto como as unidades escondidas criando *representações internas* ou extraíndo os *atributos escondidos* dos dados fornecidos.

Dependendo do número de unidades escondidas, é possível encontrar uma nova representação onde os vetores de saída da camada escondida são linearmente separáveis apesar dos vetores de entrada não serem.

Quando se tem apenas 1 camada escondida com um número pequeno de unidades, não suficiente para que seja possível esta recodificação dos vetores de entrada, uma possível solução é adicionar uma segunda camada de unidades escondidas. Assim, entre outras coisas, o projetista da rede tem que decidir entre: 1) usar apenas 1 camada escondida com muitas unidades; ou 2) usar 2 ou mais camadas escondidas com poucas unidades em cada camada. Normalmente não mais de 2 camadas escondidas são usadas por 2 razões: 1) provavelmente o poder de representação de uma rede com até 2 camadas escondidas será suficiente para resolver o problema; 2) as simulações tem mostrado que para a maioria dos algoritmos usados para treinamento de RNA disponíveis atualmente, o tempo de treinamento da rede aumenta rapidamente com o número de camadas escondidas.

O poder de um algoritmo que consegue adaptar os pesos de um Perceptron de múltiplas camadas vem do fato que tal algoritmo poderá encontrar automaticamente a recodificação interna necessária para resolver o problema usando os exemplos do mapeamento entrada-saída desejado. É possível interpretar tal recodificação interna, ou representação interna, como *um conjunto de regras* (ou *micro-regras* como alguns autores preferem). Assim, usando-se uma analogia com *expert systems*, tal algoritmo "extraí" as regras ou atributos de um conjunto de exemplos. Alguns autores referem-se a isto como a propriedade de executar a *extração dos atributos* do conjunto de dados.

As Figuras 8.12, 8.13 e 8.14 ilustram 3 diferentes soluções para o problema do XOR usando unidades tipo TLU nas camadas escondida e de saída. Observe que nestas soluções a unidade de saída pode também ser linear sem bias, isto é, respectivamente $y = x_3 + x_4$, $y = x_4 - x_3$, e $y = x_1 + x_2 - 2x_3$. Nas figuras 8.12 e 8.13 as duas unidades escondidas recodificam as variáveis de entrada x_1 e x_2 como as variáveis x_3 e x_4 . Os 4 pontos de entrada são mapeados para 3 pontos no espaço $[x_3, x_4]$. Estes 3 pontos são linearmente separáveis como as figuras ilustram. Observe que a solução ilustrada na Figura 8.12 é uma combinação das funções AND e OR.

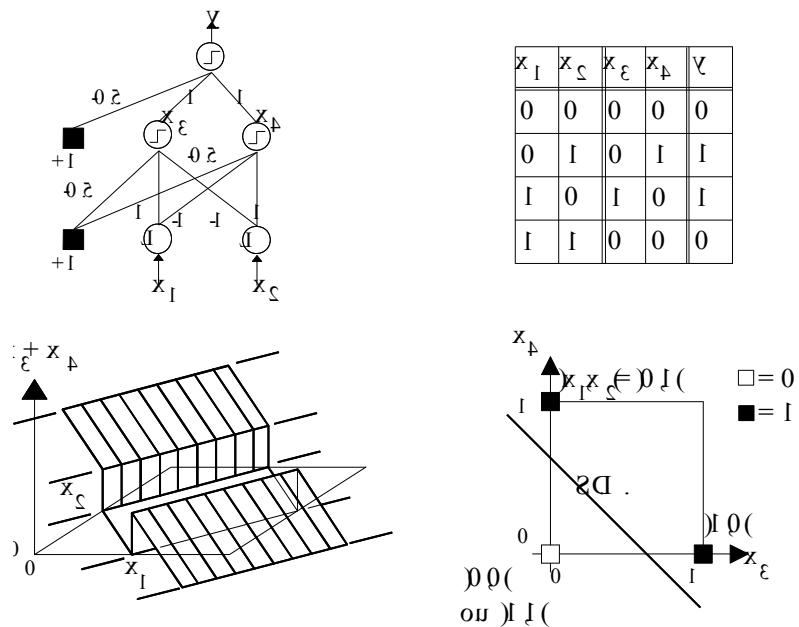


Figura 8.12. Primeira solução para o problema XOR.

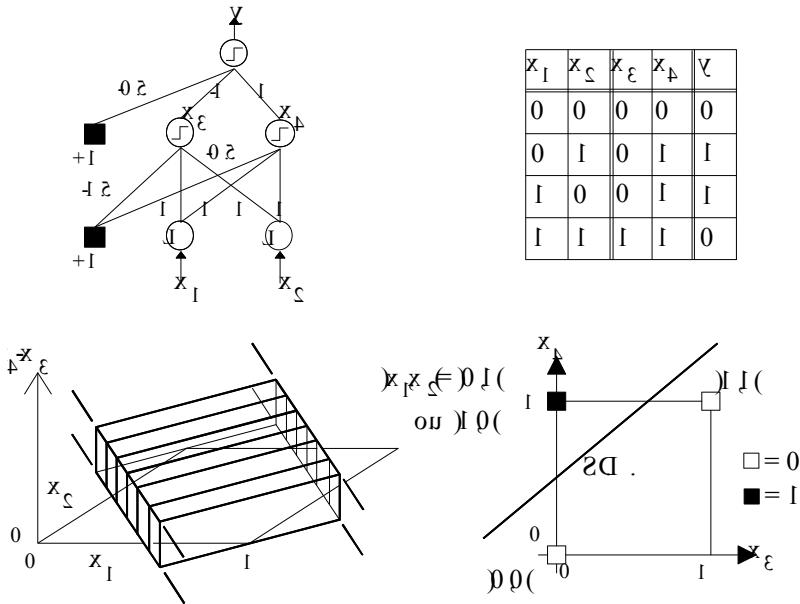


Figura 8-13. Segunda solução para o problema XOR.

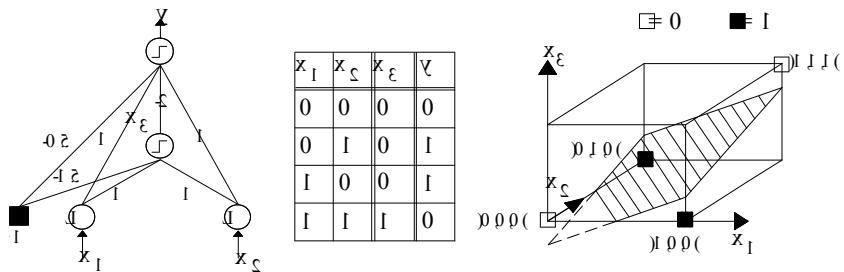


Figura 8-14. Terceira solução para o problema XOR.

A Figura 8-14 mostra que, se as conexões das unidades de entrada para as unidades de saída forem usadas, o problema do XOR pode ser resolvido usando apenas 1 unidade escondida que implementa a função AND. Então se considera o espaço $[x_1, x_2, x_3]$, os 4 padrões são linearmente separáveis pois é possível encontrar um plano que separa os pontos que devem produzir a saída "0" dos pontos que devem produzir a saída "1". Se a unidade de saída for uma unidade TLU, a superfície de decisão no espaço $[x_1, x_2]$ muda de uma reta para uma elipse, como ilustrado na Figura 8-10.

Como a função AND pode ser definida para variáveis de entrada como o produto dessas variáveis. Da Figura 8-14 mostra que se fosse disponível uma entrada extra para a rede com o valor de $x_1 \cdot x_2$ então apenas 1 camada de pesos seria necessária para resolver o problema e não haveria a necessidade de unidades escondidas. Generalizando esta idéia, quando uma unidade utiliza os produtos de suas variáveis de entrada, ela é chamada de *higher-order unit* e a rede de *higher-order ANN*. Em geral, *higher-order units* implementam a função:

$$y_i = f(\text{bias} + \sum_j w_{ij}^{(1)} x_j + \sum_j \sum_k w_{ijk}^{(2)} x_j x_k + \sum_j \sum_k \sum_l w_{ijkl}^{(3)} x_j x_k x_l + \dots) \quad (8.47)$$

Da definição acima o Perceptron é uma RNA de primeira ordem pois utiliza apenas o primeiro termo da equação acima. Widrow chama tais unidades como unidades que possuem *Polynomial Discriminant Functions*. O problema com *higher-order ANN* é a explosão do número de pesos à medida que o número de entradas aumenta. Entretanto recentemente tais redes tem sido utilizadas com sucesso para a classificação de imagens, independentemente da posição, rotação ou tamanho das imagens.

8.2.3 O Algoritmo Back-Propagation

Foi visto anteriormente que a vantagem em usar as chamadas unidades escondidas é que a RNA pode então implementar superfícies de decisão mais complexas, isto é, o poder de representação da rede é bastante aumentado. A desvantagem em usar as unidades escondidas é que o aprendizado fica muito mais difícil, pois o método de aprendizado tem que decidir que atributos devem ser extraídos dos dados de treinamento. Basicamente, do ponto de vista de otimização, a dimensão do problema é também bastante aumentada, pois se requer a determinação de um número bem maior de pesos.

O algoritmo BP usa o mesmo princípio da regra Delta, isto é, minimizar a soma dos quadrados dos erros de saída, considerando-se sua média sobre o conjunto de treinamento, usando uma busca baseada na direção do gradiente. Por esta razão o algoritmo BP é também chamado de Regra Delta Generalizada (*Generalized Delta Rule*). A modificação crucial foi substituir as unidades TLU por unidades com funções de ativação contínuas e suaves. Isto possibilitou a aplicação de uma busca tipo *gradient-descent*, mesmo nas unidades escondidas. A função de ativação padrão para as unidades escondidas são as funções chamadas *squashing* ou *S-shaped*, tais como a função sigmóide,

$$\text{sig}(x) = \frac{1}{1 + e^{-x}} \quad (8.48)$$

e a função tangente hiperbólica,

$$\tanh(x) = 2 * \text{sig}(x) - 1 \quad (8.49)$$

Algumas vezes a classe geral de funções tipo *squashing* é denominada de *funções sigmoidais*. A função sigmóide cresce monotonicamente de 0 até 1, enquanto que a função tangente hiperbólica cresce de -1 até 1. Observe que a função sigmóide pode ser visto como uma aproximação suave da função bipolar {0,1}, enquanto que a função tangente hiperbólica pode ser vista como uma aproximação de uma função bipolar {-1,1}, como no ADALINE de Widrow.

A dificuldade em treinar um Perceptron de múltiplas camadas é que não existe um erro pré-definido para as unidades escondidas. Como o algoritmo BP é uma regra de aprendizado supervisionado, nós temos o valor que as unidades de saída devem seguir, mas não para as unidades escondidas.

Seja uma RNA tipo *feedforward* com N_{inp} unidades de entrada e N_{out} unidades de saída. Assumindo, apenas por uma questão de simplicidade na notação, uma RNA com apenas uma camada escondida com N_{hid} unidades conforme ilustrado na Figura 8-11, tem-se:

$$\text{net}_k^H(j) = \sum_{l=1}^{N_{inp}+1} w_k^H(j,l) u_k(l) \quad (8.50)$$

$$y_k^H(j) = f^H(\text{net}_k^H(j)) \quad (8.51)$$

onde $1 \leq j \leq N_{hid}$,

$$\text{net}_k^S(i) = \sum_{j=1}^{N_{hid}+1} w_k^S(i,j) y_k^H(j) \quad (8.52)$$

$$y_k^S(i) = f^S(\text{net}_k^S(i)) \quad (8.53)$$

onde $1 \leq i \leq N_{out}$, e:

u_k = vetor de entradas da RNA com dimensão $(N_{inp} + 1) \times 1$, sendo que se assume $u_k(N_{inp} + 1) = 1$,

$w_k^H(j,l)$ = peso entre a unidade escondida j e a unidade de entrada l ,

$\text{net}_k^H(j)$ = entrada total para a unidade escondida j ,

$y_k^H(j)$ = saída da unidade escondida j , sendo assumido:

$$y_k^H(N_{hid} + 1) = 1$$

$f^H(.)$ = função de ativação usada nas unidades escondidas,

$w_k^S(i,j)$ = peso entre a unidade de saída i e a unidade escondida j ,

$\text{net}_k^S(i)$ = entrada total para a unidade de saída i,
 $y_k^S(i)$ = saída da unidade de saída i,
 $f^S(\cdot)$ = função de ativação usada na unidade de saída i.

O algoritmo Back-Propagation propõe que a variação dos pesos ocorra na direção de máximo decréscimo do erro, apontado pelo sentido contrário ao gradiente $\partial E_k / \partial w_k$:

$$\Delta w_k(i, j) = w_{k+1}(i, j) - w_k(i, j) = -\eta \frac{\partial E_k}{\partial w_k(i, j)} \quad (8.54)$$

onde, E_k é definido como o erro quadrático calculado sobre as unidades de saída quando o padrão u_k é aplicado na entrada da RNA, ou seja:

$$E_k = \frac{1}{2} \sum_{l=1}^{N_{\text{out}}} \left(d_k^S(i) - y_k^S(i) \right)^2 = \frac{1}{2} \sum_{l=1}^{N_{\text{out}}} \left(e_k^S(i) \right)^2 \quad (8.55)$$

onde o erro e_k^S para a unidade de saída i ($e_k^S(i)$) é simplesmente definido como o escalar:

$$e_k^S(i) \triangleq d_k^S(i) - y_k^S(i) \quad (8.56)$$

onde, $d_k^S(i)$ e $y_k^S(i)$ denotam respectivamente a saída desejada e a saída calculada para a unidade de saída i, quando o padrão u_k é aplicado na entrada da RNA.

O "pseudo-erro" e_k^H para a unidade escondida j, denotado ($e_k^H(j)$) é definido como o escalar:

$$e_k^H(j) \triangleq \sum_{i=1}^{N_{\text{out}}} e_k^S(i) \frac{\partial e_k^S(i)}{\partial y_k^H(j)}, \quad 1 \leq j \leq N_{\text{hid}} \quad (8.57)$$

isto é, o *pseudo-erro* de uma dada unidade escondida é calculado como a soma dos erros de cada unidade de saída multiplicado pela influência da unidade escondida no erro desta unidade de saída. Donde, pode-se escrever:

$$\frac{\partial e_k^S(i)}{\partial y_k^H(j)} = \frac{\partial e_k^S(i)}{\partial y_k^S(i)} \frac{d y_k^S(i)}{d \text{net}_k^S(i)} \frac{\partial \text{net}_k^S(i)}{\partial y_k^H(j)} \quad (8.58)$$

$$= -\bar{f}_k^S(i) w_k^H(i, j)$$

onde, $\bar{f}_k^S(i)$ denota a derivada da função $f^S(z)$ em relação à z calculada no ponto $z = \text{net}_k^S(i)$.

E assim tem-se que:

$$e_k^H(j) = - \sum_{i=1}^{N_{\text{out}}} e_k^S(i) \bar{f}_k^S(i) w_k^H(i, j), \quad 1 \leq j \leq N_{\text{hid}} \quad (8.59)$$

Portanto, pode-se calcular agora a variação do termo E_k em (8.27), em relação a w_k através da aplicação da regra da cadeia:

$$\frac{\partial E_k}{\partial w_k(i, j)} = \frac{\partial E_k}{\partial y_k(i)} \frac{d y_k(i)}{d \text{net}_k(i)} \frac{\partial \text{net}_k(i)}{\partial w_k(i, j)} \quad (8.60)$$

Se a unidade i é uma unidade da camada de saída:

$$\frac{\partial E_k}{\partial y_k^S(i)} = -e_k^S(i) \quad (8.61)$$

Se a unidade i é uma unidade da camada escondida:

$$\frac{\partial E_k}{\partial y_k^H(i)} = \sum_{l=1}^{N_{\text{out}}} \left(\frac{\partial E_k}{\partial y_k^S(l)} \frac{d y_k^S(l)}{d \text{net}_k^S(l)} \frac{\partial \text{net}_k^S(l)}{\partial y_k^H(i)} \right) \quad (8.62)$$

A expressão (8.62) pode ser reescrita como:

$$\frac{\partial E_k}{\partial y_k^H(i)} = - \sum_{l=1}^{N_{out}} e_k^S(l) \bar{f}_k^S(i) w_k^S(l,i) \quad (8.63)$$

onde, $\bar{f}_k^S(z)$ denota a derivada de $f_k^S(z)$, em relação à variável z , no ponto $z = net_k^S(i)$.

Assim, para a unidade i da camada de saída:

$$\Delta w_k^S(i,j) = \eta e_k^S(i) \bar{f}_k^S(i) y_k^H(j) \quad (8.64)$$

onde $1 \leq i \leq N_{out}$ e $1 \leq j \leq N_{hid}+1$.

Para a unidade i da camada escondida:

$$\Delta w_k^H(i,j) = \eta \left[\sum_{l=1}^{N_{out}} e_k^S(l) \bar{f}_k^S(l) w_k^H(l,i) \right] \bar{f}_k^H(i) u_k(j) \quad (8.65)$$

onde, $1 \leq i \leq N_{hid}$, $1 \leq j \leq N_{inp}+1$. Utilizando a eq. (8.33) pode-se escrever então que para as unidades escondidas:

$$\Delta w_k^H(i,j) = -\eta e_k^H(i) \bar{f}_k^H(i) u_k(j) \quad (8.66)$$

onde, e_k^H e \bar{f}_k^H são definidas de forma análoga a e_k^S e \bar{f}_k^S , respectivamente.

Observe que na derivação do algoritmo BP, apenas as seguintes restrições foram usadas:

- 1) A rede é do tipo feedforward;
- 2) Todas as unidades possuem funções de ativação diferenciáveis; e
- 3) A função combinação (usada para calcular o valor da *net input*) foi definida como (em notação vetorial): $net = W \text{out} + \text{bias}$.

Os passos do algoritmo BP são:

Passo 1: Geração do conjunto inicial de pesos sinápticos para todas as unidades da RNA;

Passo 2: Inicialização das matrizes de covariância P e escalares r para cada unidade da RNA;

Passo 3: Escolha do vetor de entrada a ser treinado u_k e da saída desejada d_k associada a esta entrada;

Passo 4: Cálculo das saídas de todas as unidades da RNA, ou seja, do vetor:

$$\begin{bmatrix} y_k^H(1) & \dots & y_k^H(N_{hid}) & y_k^S(1) & \dots & y_k^S(N_{out}) \end{bmatrix};$$

Passo 5: Cálculo do erro e_k^S para cada unidade de saída;

Passo 6: Cálculo do erro e_k^H para cada unidade na camada escondida;

Passo 7: Cálculo do vetor de saída h_k para cada unidade da RNA (h_k^H e h_k^S);

Passo 8: Atualização dos pesos w_k utilizando as expressões (8.63) ou (8.65) (dependendo de se trata de Δw_k^S ou Δw_k^H);

Passo 9: Voltar ao passo 3 até o término do conjunto de padrões usados para o treinamento;

Passo 10: Voltar ao passo 3 até o término do número máximo de iterações permitido.

Alguns casos possíveis são o uso de diferentes funções de ativação na camada escondida, uso de várias camadas escondidas e redes tipo *feedforward*, mas não tipo estritamente *feedforward*.

Uma vantagem de se usar as funções sigmóide ou tangente hiperbólica em uma rede de múltiplas camadas é que as suas derivadas podem ser calculadas simplesmente usando o valor da saída da unidade sem a necessidade de cálculos mais complicados, pois:

$$\frac{dsig(x)}{dx} = sig(x)[1 - sig(x)] \quad (8.67)$$

e:

$$\frac{d \tanh(x)}{dx} = [1 + \tanh(x)][1 - \tanh(x)] \quad (8.68)$$

Tais propriedades são muito úteis, pois reduzem o número total de cálculos necessários para treinar a rede.

Em relação à inicialização dos pesos, Rumelhart e McClelland, 1988, sugerem usar pequenos números aleatórios. Em relação à taxa de aprendizado (η), ressalta-se que, embora um valor grande possa resultar em um aprendizado mais rápido, poderão também ocorrer oscilações. Uma maneira de reduzir o risco de provocar oscilações é incluir um *termo de momento*, modificando a equação de atualização dos pesos para:

$$\Delta w_{ij}(k+1) = -\eta \delta_i \text{out}_j + \alpha \Delta w_{ij}(k) \quad (8.69)$$

onde, o índice k indica o número da apresentação e α é uma constante positiva com valor pequeno selecionado pelo usuário. Tal modificação filtra as oscilações de alta frequência nas mudanças dos pesos, pois tende a cancelar as mudanças que tem direções opostas e reforça a direção predominante.

No caso da regra Delta, quando aplicada para treinar redes sem camadas escondidas e com unidades de saída lineares, a superfície de erro tem sempre a forma de uma taça, isto é, uma superfície convexa e consequentemente os pontos de mínimo locais são também pontos de mínimo globais. Se a taxa de aprendizado η for suficientemente pequena, a regra Delta irá convergir para um destes pontos de mínimo. No caso do Perceptron de múltiplas camadas a superfície de erro pode ser muito mais complexa com muitos pontos de mínimo locais. Como o algoritmo BP é um procedimento tipo *gradient-descent*, existe a possibilidade que o algoritmo fique preso em um destes pontos de mínimo locais e, portanto, não converja para a *melhor* solução, o ponto de mínimo global.

8.3 Aplicações de Redes Neurais Artificiais em Controle

As redes neurais podem ser utilizadas em uma vasta gama de aplicações, podendo-se citar como exemplos, processamento de sinais (cancelamento de ruídos, reconhecimento de caracteres, codificação e outros), controle de sistemas dinâmicos (manipuladores mecânicos, processos industriais, veículos auto-guiados e outros), sistemas de decisão (terapêutica médica, análise financeira e outras) e numerosas possibilidades descritas na literatura especializada.

Em particular, neste capítulo, são enfatizadas as aplicações de redes neurais artificiais empregadas como controladores clássicos, gerando um sinal $u(\cdot)$ a ser enviado para o atuador, a partir do sinal de erro $e(\cdot)$ entre a saída e a referência.

Um dos problemas encontrados na utilização de redes neurais no controle de sistemas é o das informações necessárias para o treinamento. De fato, na Figura 8-15 nota-se que, de início, os sinais $u(\cdot)$ que originam a resposta $y(\cdot)$ adequada pode não ser conhecida. Neste caso, utilizando-se, por exemplo, o algoritmo *backpropagation* juntamente com um perceptron multicamadas, necessita-se de um mecanismo para gerar $u^t(\cdot)$ que poderia, então, compor os pares $(e(\cdot), u^t(\cdot))$ para treinamento da rede.

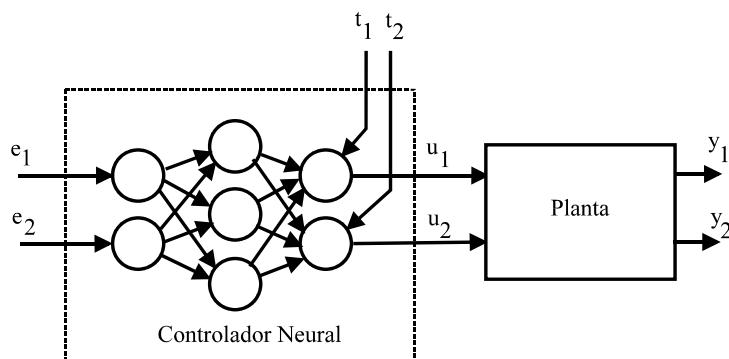


Figura 8-15. Utilização de uma rede neural como controlador.

Uma alternativa para gerar $u^t(\cdot)$ é utilizar um controlador convencional de forma que a rede neural possa “copiar” as suas características. A Figura 8-16 ilustra uma forma de obter os pares

entrada-saída do controlador que são utilizados no treinamento da rede neural. Este controlador pode ser, por exemplo, um operador humano.

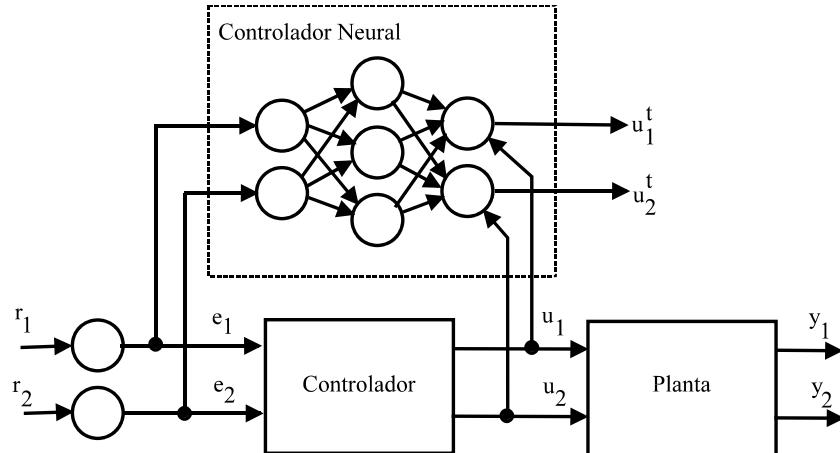


Figura 8-16. A rede neural pode ser utilizada para adquirir as características de um controlador convencional.

Outra alternativa, seria realizar uma identificação da planta a ser controlada. A identificação poderia conduzir a modelos diretos como o obtida por um esquema do tipo Figura 8-17 ou inverso como as provenientes de arranjos como o da Figura 8-18.

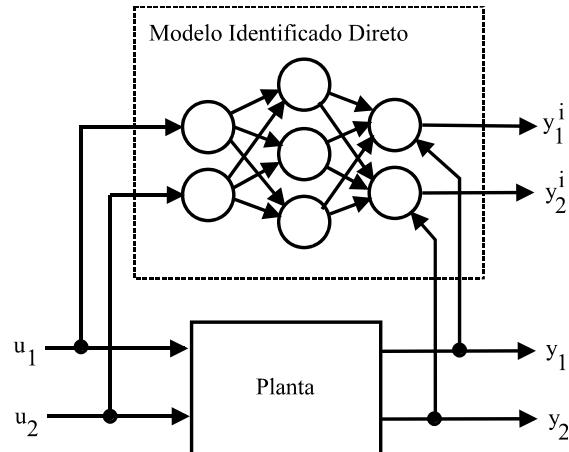


Figura 8-17. Identificação de modelo direto através da utilização de redes neurais.

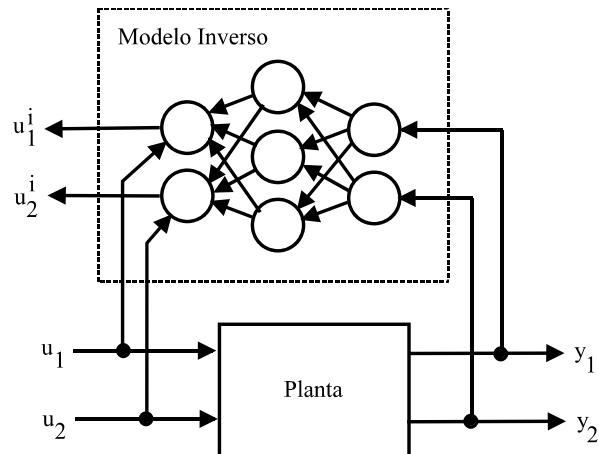


Figura 8-18. Identificação de modelo inverso através da utilização de redes neurais.

Uma outra possibilidade, é utilizar o conceito de aprendizado por reforço, onde a rede gera ações $u(\cdot)$ aleatórias e o resultado, avaliado por um crítico, é realimentado para ajuste dos pesos da rede, em um mecanismo de punição ou recompensa (figura 13.5)

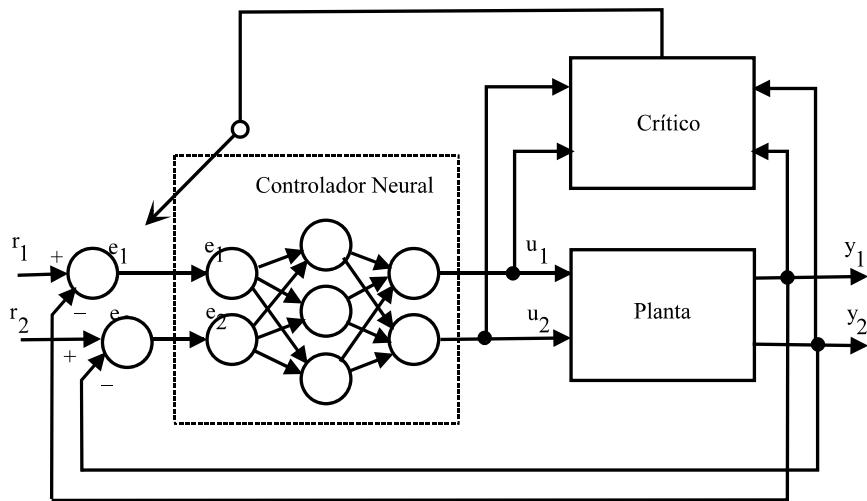


Figura 8-19. Esquema para utilização de rede neural como controlador treinado por mecanismo de punição ou recompensa.

Estas são as principais arquiteturas de redes neurais em aplicações de controle de sistemas dinâmicos em malha fechada, porém é possível encontrar outras variantes.