CAPÍTULO 5

TÓPICOS DE INTELIGÊNCIA COMPUTACIONAL APLICADOS EM TECNOLOGIAS ESPACIAIS

José Demisio Simões da Silva

RESUMO

Este minicurso apresenta técnicas de inteligência computacional que podem ser utilizadas no escopo de aplicações em ciências e tecnologias espaciais. São apresentados tópicos teóricos de diferentes abordagens e alguns exemplos de aplicação.

Preâmbulo

Nas seções que se seguem são apresentados de forma introdutória alguns tópicos de Inteligência Artificial (IA). O material apresentado não é exaustivo e nem cobre todos os tópicos e as diferentes facetas da área de IA. Os tópicos são apresentados na intenção de despertar a curiosidade do leitor para que este busque se aprofundar nas diferentes fontes de informação disponíveis sobre IA. Para estudos mais aprofundados dos tópicos 1 a 4, recomenda-se a leitura dos livros de Bittencourt (1998), Russell e Norvig (1995), Giarratano (1994) e Rich (1983). Para estudos mais aprofundados do tópico 5, sobre Lógica Nebulosa, recomenda-se a leitura dos livros Bittencourt (1998), Tsoukalas e Uhrig (1996), LIN e LEE (1996), Russell e Norvig (1995) e Giarratano (1994). Para o tópico 6, sobre Redes Neurais Artificiais, recomenda-se a leitura dos livros de Haykin (1999), Tsoukalas e Uhrig (1996), Hagan et al (1996), LIN e LEE (1996), Fausett (1994), Simpson (1990), Hecht-Nielsen (1990) e Rumelhart e Mcclelland (1990). Para o tópico 8 recomenda-se a leitura dos livros de Lacerda e Carvalho (1999), Bittencourt (1998), Heitkoetter e Beasley (1999), Michalewicz (1996), Lin e Lee (1996), Davis(1991) e Goldberg (1989).

5.1 Introdução

A inteligência é um processo dinâmico associado à habilidade de se alcançarem objetivos. Existe um número considerável de tipos e graus de inteligência, fazendo com que não exista uma definição formal única que possa ser usada para se identificar esta característica nos indivíduos (ou animais). Como um processo dinâmico, inteligência exige: aquisição, triagem, ordenação e interpretação de conhecimento do ambiente de contexto.

A Inteligência Artificial (IA) é uma subárea da Ciência da Computação que investiga e desenvolve técnicas, métodos, modelos, ferramentas e teorias para conceber sistemas ou máquinas (inteligentes) com capacidade de aprendizagem, buscando a adaptação e o desempenho computacional.

Em 1950, Alan Turing discutiu as condições para se considerar uma máquina como sendo inteligente, associando esse conceito essencialmente à natureza humana. Turing argumentava que uma máquina poderia ser considerada inteligente se ela tivesse sucesso em se passar por um humano diante de um observador com bastante conhecimento. Para o teste, Turing idealizou um observador interagindo com a máquina e com outro

humano através de terminais de maneira indireta. A máquina programada para se passar por humano tentaria enganar o observador.

Os sistemas que se baseiam em técnicas de IA, segundo Russel e Norvig (1995), podem ser classificados em: a) Sistemas que "pensam" como humanos (Abordagem de modelagem cognitiva); b) Sistemas que "pensam racionalmente" (Abordagem das leis do pensamento); c) Sistemas que "agem" como humanos (Abordagem de Turing); e d) Sistemas que "agem racionalmente" (Abordagem dos agentes racionais).

Em todas as classes anteriores observa-se que os sistemas que usam os princípios de IA baseiam-se em conhecimento, necessitando, portanto, de meios de aquisição e métodos de estruturação que permitam a recuperação fácil. Essa recuperação exige formas (em geral uma linguagem) para manipulação do conhecimento que permitam o tratamento de complexidade, incerteza e ambigüidade das informações disponíveis.

5.2 Representação do Conhecimento

Os sistemas que usam técnicas de IA são fortemente dependentes do conhecimento da área de aplicação envolvida. Para que eles possam atingir os objetivos estabelecidos, faz-se necessário que esse conhecimento seja modelado eficientemente, de maneira que o acesso para uso e manutenção das informações aconteça da forma mais completa possível. A modelagem desse conhecimento deve ser precedida da fase de engenharia do conhecimento, quando as informações necessárias são coletadas. Isso exige dos projetistas de sistemas que usam IA um entendimento aprofundado da aplicação a que se destina o sistema.

A representação do conhecimento exige primeiro que seja feito um levantamento e a especificação completa do que é necessário para abordar o problema a ser resolvido. A representação deve ser feita de maneira que facilite a inferência e a memorização. Existem diversos métodos de representação que foram concebidos ao longo do tempo. Entre eles, estão a Lógica (diversos tipos), as Regras de produção, as Redes semânticas, os Quadros (Frames) e as Redes Neurais Artificiais.

Um mecanismo de inferência permite a manipulação do conhecimento para síntese de conhecimento novo, que deve ser memorizado na base de conhecimento. Alguns problemas relacionados com o processo de representação do conhecimento são: a forma de medir o conhecimento que deve ser usado para resolver um problema; a forma de aquisição do conhecimento; e a forma de analisar a completude do conhecimento representado para resolver o problema em mãos.

5.2.1 Lógica Proposicional

Há muito tempo a lógica estuda matemática e filosoficamente a natureza do raciocínio e do conhecimento, e foi um dos primeiros esquemas de representação do conhecimento usados pelos pesquisadores da IA.

Na lógica proposicional as expressões são chamadas de proposições, que podem ser verdadeiras ou falsas. Exemplos de proposições: a) "Há cobertura de nuvens"; b) "Com nuvens o satélite produz imagens com nuvens".

As proposições simples podem formar proposições compostas quando combinadas através de conectores lógicos como "E" (&), "OU" (V), "NÃO" ($\neg$), "IMPLICAÇÃO" ($\rightarrow$)e "EQUIVALÊNCIA" ($\leftrightarrow$). Usando os símbolos A e B para representar os exemplos de proposições anteriores, é possível representar o conhecimento da seguinte forma:

A – "Há cobertura de nuvens"; B - "Com nuvens o satélite produz imagens com nuvens"

A descrição ficaria: A $\rightarrow$ B

5.2.2 Lógica dos Predicados ou Lógica de 1ª. Ordem

A lógica dos predicados não é suficiente para a IA. Uma representação dos fatos do mundo com formalismo exige não apenas as proposições verdadeiras ou falsas, mas também a capacidade de expressar ou descrever objetos e generalizações das classes de objetos.

A Lógica de Predicados satisfaz esses objetivos e apresenta facilidade de manipulação e dedução de novos fatos a partir da base de conhecimento existente, porém apresenta dificuldade para determinar os fatores relevantes durante um processo.

Exemplo: "Todo satélite produzido no INPE é rastreado pelo CCS".

$$\forall x, \text{é}(x,\text{satélite}) \ \& \ \text{produzido}(x,\text{INPE}) \rightarrow \text{rastreado}(x,\text{CCS})$$

5.2.3 Regras de Produção

As regras de produção formam a base para os Sistemas Especialistas. Neste método, os conhecimentos são representados através de regras de produção com "condição-ação". As regras de produção foram concebidas por Emil Post em 1943 como um modelo computacional geral de solução de problemas. Nos anos 1950 e 1960, as regras de produção foram usadas para representar a forma humana de resolver problemas (xadrez e criptoaritmética); nos anos 1970, os sistemas baseados em regras de produção foram usados para representar modelos mentais. Mas, recentemente, os sistemas baseados em regras de produção têm sido usados para descrever uma família de sistemas constituídos de regras.

Os elementos da regra de produção são: a) a Condição (lado esquerdo, antecedente, premissa, padrão) que determina a aplicabilidade da regras; b) a Ação (lado direito ou conseqüente) que descreve o que deve ser realizado se a regra é aplicável. Exemplos:

Regra: Farol vermelho

SE o farol está vermelho ENTAO pare

Regra: Farol verde

SE o farol está verde ENTAO atravesse

Há duas formas pelas quais as regras podem ser deduzidas em um sistema baseado em regras de produção: a) encadeamento para frente (*forward chaining*); e b) encadeamento para trás (*backward chaining*).

O encadeamento para frente parte das premissas (dados, fatos – conhecimento inicial) e aplica as regras para gerar um novo conhecimento até que se chegue a uma solução para o problema ou até que nenhuma inferência adicional possa ser feita. Neste caso, aplicar uma regra é comparar os fatos conhecidos com as condições especificadas.

O encadeamento para trás parte do objetivo que se quer provar, que é recursivamente particionado em sub-objetivos mais simples, até que uma solução seja encontrada ou todos os objetivos sejam particionados em componentes mais simples. Neste método, a aplicação de uma regra é a comparação de sua conclusão, que contém o que se quer provar, com os fatos conhecidos. A escolha do tipo de mecanismo de inferência depende

da aplicação. Para problemas de diagnósticos, pode-se utilizar o encadeamento para trás. Os problemas de monitoramento e controle requerem o encadeamento para frente. Uma regra é ativada ou instanciada se todos os padrões (premissas) forem satisfeitos. Se várias regras podem ser ativadas, o mecanismo de inferência deve escolher uma regra para ativar.

5.2.4 Redes Semânticas

As redes semânticas foram usadas por Rosi Quilan nos anos 1960 para representar modelos computacionais de memória humana (memória semântica). Originalmente, as redes semânticas foram usadas como suporte para representar o conhecimento no processamento da linguagem natural.

Uma rede semântica é um grafo orientado no qual os nós e arestas (arcos) possuem rotulação *mnemônica*, que traduz a semântica. Os nós (vértices) representam objetos, situações ou conceitos (elementos), e as arestas (arcos) relacionam os vértices. Ou seja, uma rede semântica descreve os objetos e suas inter-relações. As ligações podem ser quaisquer, mas devem transmitir a semântica da maneira mais fiel possível. Exemplos de tipos de rotulação mais comuns para os arcos são: "é-parte-de", "é-um" ou "é-uma". Estas inter-relações exprimem se um nó é subparte ou componente de outro ("é-parte-de") ou se um nó é uma subclasse representada por outro nó.

Uma rede semântica pode ser usada para descrever tipos de semântica, estabelecendo significados para as relações. Assim, pode-se observar: a) Semântica Descritiva, que tenta mostrar o quanto a descrição se aproxima da realidade representada; b) Semântica de Equivalência, que procura estabelecer equivalências entre a representação efetuada e outra forma de representação do conhecimento; e c) Semântica Procedural, que usa programas para exprimir o significado, exigindo, portanto, entendimento de uma linguagem de programação. Independentemente do tipo de semântica, uma representação com redes semânticas está sujeita à perda de informação, dificuldades de codificação das informações e ao esquecimento de inserir informações.

Com um exemplo de representação por uma rede semântica, considere os seguintes fatos: *"O computador é de Márcia. Ela o usa para escrever um livro importante. O livro é parte dos trabalhos de pesquisa que Márcia realiza na Universidade."*

Um exemplo de representação em rede semântica pode ser visto na Figura 5.1.

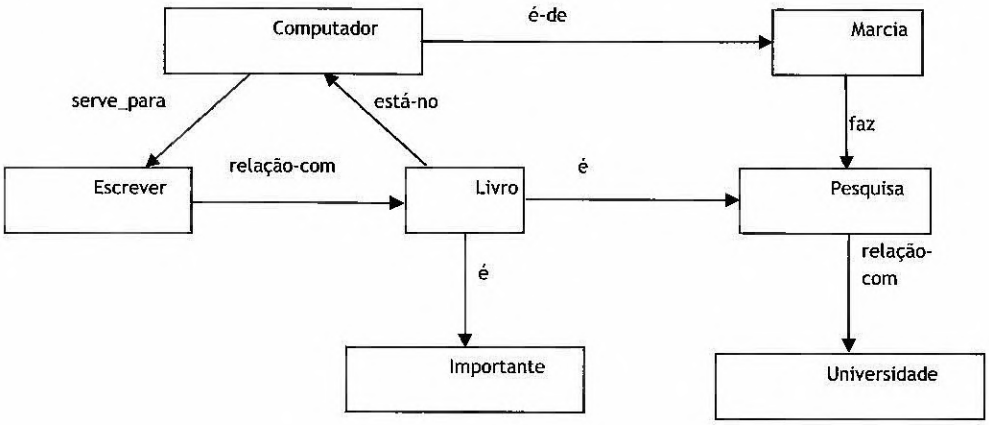


Figura 5.1 – Exemplo de rede semântica.

5.2.5 Quadros (*Frames*)

Um quadro é um arcabouço para a representação simbólica do conhecimento concebida por Marvin Minsk nos anos de 1970. Essencialmente, é uma forma de representação do conhecimento muito semelhante às redes semânticas, que se baseia na hipótese de que os seres humanos usam conhecimentos adquiridos em experiências prévias quando submetidos a novas situações. Ou seja, o ser humano dispõe de um conjunto extenso de estruturas, representando as experiências anteriores com objetos, situações e pessoas, e, na análise de uma nova experiência, ele tenta selecionar na memória uma dessas estruturas armazenadas (um quadro – "*frame*"), que é relembrada e adaptada à realidade (os detalhes de acordo com a necessidade do evento). Um quadro é uma estrutura adequada para representar situações estereotipadas como, por exemplo: a) Ir ao supermercado; ou b) Objetos do mundo real como cadeira, sala, etc.

As informações são reunidas na estrutura de quadros e armazenadas nos terminais (*slots*, atributos). As propriedades dos atributos (facetas) são associadas ao tipo de valor e às restrições de número de cada atributo. Os tipos de informações que podem ser armazenadas são: a) experiências e situações anteriores, que criam expectativas do que possa acontecer a seguir; b) orientações sobre o que fazer se as expectativas não forem confirmadas.

Os atributos (facetas) podem ser do tipo *<valor>*, quando se trata de uma atribuição confiável, ou do tipo *<default>*, quando se trata de uma atribuição (*a priori*) baseada em generalizações e estereótipos, ou seja, seguindo um raciocínio guiado por expectativas.

A Figura 5.2a mostra um exemplo de quadro genérico de um estádio de futebol e a Figura 5.2b mostra um exemplo de quadro de um estádio específico.

Frame: Estádio_de_Futebol		Frame: Estádio Maracanã	
Nome :	<valor> = sequência_de_caracteres	Nome :	<valor> = Mário Filho
Proprietário :	<valor> = particular_ou_estado	Proprietário :	<valor> = Estado
Capacidade :	<valor> = um_inteiro	Capacidade :	<valor> = 200.000 pessoas
Dimensões :	<default> = oficiais	Dimensões :	<valor> = oficiais

a) b)

Figura 5.2 – Exemplos de Quadros a) Genéricos e b) Específico.

Observa-se na Figura 5.2b que a faceta "Dimensões" é do tipo *<valor>*, pois o *frame* representa um estádio de futebol do qual se tem a informação correta de que as "Dimensões" são oficiais, o que não ocorre na Figura 5.2a, onde a faceta "Dimensões" é do tipo <default>, significando uma informação baseada em generalizações.

Uma estrutura de quadros é uma rede de nós e relações com níveis mais altos (topo) fixos, usados para representar tudo que é sempre verdadeiro sobre uma situação suposta. Os níveis mais baixos têm vários atributos (*slots*) preenchidos por instâncias específicas ou dados. Um *slot* pode especificar condições que seus valores devem satisfazer, ou pode ser um sub-quadro.

A representação de condições simples é feita por marcadores com atribuição terminal, como, por exemplo, Pessoa, Objeto ou Ponteiro para um sub-quadro de um determinado tipo. Já a representação de condições mais complexas pode especificar relações entre o que foi atribuído a vários atributos (*slots*).

Os Sistemas de Quadros são conjuntos de quadros relacionados (ligados) uns aos outros que podem ser usados para: a) Facilitar alguns tipos de cálculo; b) Representar mudanças de ênfase e atenção; ou c) Explicar as propriedades de imagens. O sucesso de uma representação usando a teoria de quadros depende da inclusão de expectativas e outras suposições.

Os sistemas de quadros formam uma rede de recuperação de informação que provê um quadro substituto quando um determinado *frame* não se adapta à realidade, ou seja, quando não é possível encontrar atribuições terminais que correspondam adequadamente às condições do marcador terminal.

As estruturas inter-quadros permitem outras formas de representação do conhecimento sobre fatos, analogias e outras informações úteis ao entendimento. Uma vez escolhido um quadro para representar uma situação, um processo de correspondência tenta atribuir valores aos terminais de cada quadro, consistentes com os marcadores em cada local. Esse processo de correspondência é parcialmente controlado pela informação associada ao quadro (incluindo informação de como tratar as surpresas) e pelo conhecimento sobre as metas do sistema.

Os sistemas de quadros ajudam a explicar alguns fenômenos da inteligência humana e são: a) apropriados para a interpretação de uma seqüência específica observada; b) úteis para prever a ocorrência de certos acontecimentos que não tenham sido mencionados anteriormente; c) úteis para domínios onde a forma e o conteúdo do dado desempenham um papel importante na solução do problema (por exemplo, interpretação de cenas, entendimento da fala).

Os sistemas de quadros são formalismos de representação do conhecimento (basicamente uma *estrutura de armazenamento de dados*) que necessitam de uma forma efetiva de implementação. Diversos recursos podem ser utilizados para uma boa implementação, mas a maneira mais efetiva é dispor de uma linguagem de quadros (linguagem de *frames*), implementada para permitir a navegação em todo o conhecimento representado. A linguagem deve ser composta de primitivas para:

- *Inicializar* o sistema de quadros
- *Criar* novos quadros
- *Atribuir* valores ou funções aos atributos dos quadros
- *Eliminar* atributos de quadros
- *Recuperar* informação dos quadros
- *Visualizar* mensagens para o usuário

A Figura 5.3 apresenta uma gramática para definição de uma linguagem de quadros disponível em Bittencourt (1998), e a Figura 5.4 mostra um exemplo de uma representação de um cômodo utilizando a teoria de quadros, também disponível em Bittencourt (1998).

```
<command>      ::= (frame-query <query>) |
                   (frame-store <store>) |
                   (frame-list)
<pattern>      ::= (frame{<query>}) |
                   (frame{<store>})
<query>        ::= (<frame> [ <antecedent>]{<slot query>})
<frame>        ::= <variable> | <symbol >
<antecedent>   ::= <variable> | <symbol >
<slot query>   ::= (<slot><value >)
<slot>         ::= <variable> | <symbol >
<value>        ::= qualquer S-expressão
<store>        ::= (<frame><antecedent> {<slot definition>})|
                   (off(<frame>)|
                   (off(<frame> (<slot>)+))
<slot definition> ::= (<slot><value>) |
                      (<slot><function definition><demon type>)|
<demon type>   ::= if-create | if-modify | if-necessary | if-delete
```

Figura 5.3 – Gramática para definição de linguagem de programação. Adaptado de Bittencourt (1998).

Frame: Cômodo	Super-Frame: Lugar-coberto		
Atributos	Default	Tipo	Se-necessário
Número de paredes	4	número	
Formato	retangular	símbolo	
Altura	3	número(m)	
Área		número(m^2)	
Volume		número(m^3)	(Área*Altura)

é-um

Frame: sala	Super-frame : Comodo	
Atributos	Default	Tipo
Mobiliário	(sofa, mesa, cadeiras)	Lista de símbolos
Finalidade	Convivência	símbolo

Figura 5.4 - Representação de um cômodo e uma sala em um sistema de quadros. Adaptado de Bittencourt (1998).

5.3 Sistemas Especialistas

Os Sistemas Especialistas (SE) foram concebidos nos anos 1970. Um SE é um programa que se comporta como um especialista ("*expert*") com conhecimento específico em algum domínio restrito de aplicação, sendo então capaz de resolver problemas que requerem este conhecimento especializado. Um SE também é comumente conhecido como um sistema baseado em conhecimento (mas nem todo sistema baseado em conhecimento é um SE).

Um SE deve ser capaz de gerar explicações de seu comportamento e de suas decisões para que os usuários (em alguns domínios) aumentem a confiança nos conselhos e decisões das máquinas ou para que problemas de raciocínio possam ser corrigidos. Além disso, um SE deve ser capaz de tratar incertezas e informações não completas advindas de informações incompletas ou não-confiáveis de algum problema. Em alguns casos, as relações no domínio do problema podem ser aproximadas requerendo raciocínio probabilístico, como exemplo:

- *Um médico pode não estar completamente certo de um sintoma em um paciente em um diagnóstico.*
- *Um dado medido na localização de defeitos em certos equipamentos pode estar errado.*
- *Um remédio pode causar problemas.*

No desenvolvimento de um SE é necessário considerar as funções desejadas para se atingirem os objetivos: a) Função de resolução do problema, que exige conhecimento específico do domínio, podendo inclusive requerer o tratamento de incertezas; b) Função de interação com o usuário, que inclui explicações das intenções e decisões do sistema durante e depois do processo de solução do problema.

A implementação destas funções exige o projeto da representação do conhecimento sobre o domínio, e um mecanismo de inferência para navegação pelo conhecimento na busca pela síntese de novos conhecimentos. A Figura 5.5 mostra uma estrutura genérica de um SE, enfatizando os constituintes principais.

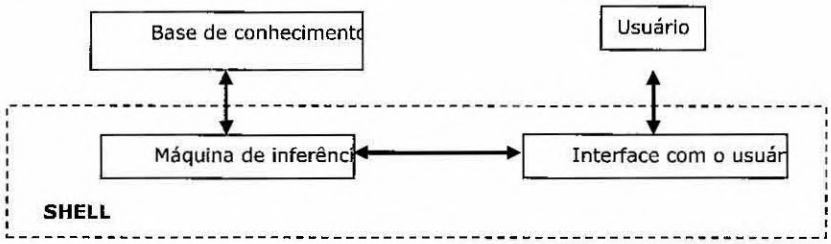


Figura 5.5 – Estrutura genérica de um Sistema Especialista.

Na Figura 5.5 a Base de conhecimento contém todo o conhecimento sobre o domínio de aplicação. O SHELL consiste da parte do SE que engloba o mecanismo de inferência e a interface com o usuário. Estes dois subsistemas são fixos e, teoricamente, independentes do domínio de aplicação. Portanto, um único SHELL pode ser utilizado para diferentes aplicações de SE. A Interface com o usuário pode ser adaptada para domínios diferentes. O Usuário pode ser um especialista da área de aplicação ou um usuário comum que não possui conhecimento específico do domínio considerado. Idealmente o SE deve ser capaz de auxiliar a ambos os tipos de usuários.

A Máquina de inferência define como o conhecimento será manipulado, se de maneira formal (seguindo formalismos com a lógica) ou heuristicamente através de métodos definidos dependentes do tipo de

representação de conhecimento adotada (no caso da representação por Quadros, a heurística está relacionada com a linguagem de Quadros que manipula o conhecimento).

Os Sistemas Especialistas baseado em regras de produção resultam em sistemas mais simples e diretos, com propriedades que permitem robustez nas aplicações. O uso de regras facilita a geração de respostas às perguntas: a) *Como*? (como se chegou a esta conclusão?); e b) *Por quê*? (Por que o interesse nesta informação?).

As regras de produção em um SE definem relações lógicas entre conceitos no domínio do problema. As relações puramente lógicas podem ser caracterizadas como pertencentes ao conhecimento categórico, no sentido de que elas são sempre verdadeiras. Alguns exemplos de regras são:

- *Se precondição P então conclusão C*
- *Se situação S então ação A*
- *Se condições C1 e C2 acontecem então condição C não acontece*

Entretanto, há domínios nos quais o conhecimento é probabilístico ou parcial, no sentido de que regularidades empíricas são válidas somente até certo grau. Nesses casos são acrescentados fatores de certeza (crença) às interpretações lógicas:

- *If condição A então conclusão B acontece com fator de certeza (crença) F*

As principais características (ou propriedades) de SE baseados em regras de produção são: a) Modular; b) Incremental; c) Modificável; e d) Transparente. A propriedade Modular está associada à possibilidade de existência de grupos de regras em subdomínios do domínio da aplicação, sem modificação da base de conhecimento existente. A propriedade Incremental se caracteriza pela possibilidade de inserção de novas regras na base. A propriedade Modificável significa que a base de conhecimento pode ser manipulada em busca da completude, integridade e coerência da base de regras, no sentido de eliminação de regras redundantes e conflitantes. A Transparência está associada à habilidade do sistema em explicar suas intenções e decisões.

Um exemplo de SE simples para o problema de diagnóstico para descobrir um vazamento em um apartamento pode conter as seguintes regras:

- *R1: O vazamento pode aparecer no banheiro ou na cozinha*
- *R2: O vazamento mancha a parede no corredor.*
- *R3: O vazamento só acontece em um ambiente ou no outro (cozinha ou banheiro).*

A definição de como o SE deve agir consiste no estabelecimento de um grafo de inferência constituído em nós com operações "&"(E) e "V" (OU). A Figura 5.6 ilustra um grafo de inferência para o pequeno SE composto das regras 1, 2 e 3 anteriores.

A inferência sobre o grafo da Figura 5.6 pode ser feita utilizando raciocínio por encadeamento para frente ou por encadeamento para trás. Em geral a escolha do tipo de raciocínio depende da inteligibilidade para o usuário.

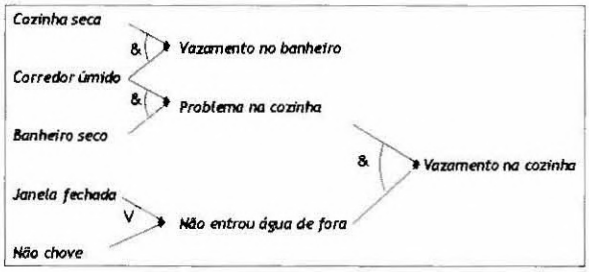


Figura 5.6 – Exemplo de grafo de inferência simples de um sistema especialista.

No procedimento de encadeamento para trás, pode-se partir da hipótese de que há *Vazamento na cozinha* e tentar confirmar os fatos *Problema na cozinha* e *Não entrou água de fora*. O fato *Problema na cozinha* se confirma se os fatos *Corredor úmido* e *Banheiro úmido* são confirmados. O fato *Não entrou água de fora* se confirma se (por exemplo) *Janela fechada* for confirmada.

No procedimento de encadeamento para frente, inicia-se com os fatos confirmados. Assim, confirmando os fatos *Problema na cozinha* e *Não entrou água de fora*, conclui-se o fato *Vazamento na cozinha*.

A geração de explicação para a decisão ou conclusão procura responder as perguntas *Como*? e *Por quê*?. Para isso, a maneira mais simples é mostrar os caminhos seguidos durante o processo de inferência.

O exemplo com conclusão *Vazamento na cozinha* pode ser explicado pelos fatos: a) Há um *Problema na cozinha*, que foi concluído a partir do fato do *corredor úmido* e do *Banheiro seco*; e b) *Não entrou água de fora*, que foi concluído a partir do fato da janela fechada. Esse tipo de explicação é uma *árvore de prova*, onde as conclusões seguem das regras e fatos na base de conhecimento.

Um exemplo de árvore de prova de uma proposição pode ser visto abaixo.

(1) Se *P* é um fato então a árvore de prova é *P*.

(2) Se *P* foi gerado usando a regra "*Se Condicao então P*"

A árvore de prova é: *P* <= *ProvaCondicao*, onde *ProvaCondicao* é árvore de prova de *Condicao*

Considerando *P1* e *P2* duas proposições, cujas árvores de provas são *Prova1* e *Prova2*.

- Se *P* é *P1* & *P2* → a árvore de prova é *Prova1* & *Prova2*.
- Se *P* é *P1* V *P2* → a árvore de prova é *Prova1* V *Prova2*.

5.4 Agentes

De acordo com a definição em Russell e Norvig (1995), um agente em IA é um conceito que está associado a qualquer coisa que tenha capacidade de sentir o ambiente – através de sensores – e agir sobre ele – através de atuadores. Essa definição é bascada em um modelo de agente humano e na sua forma de interagir com seu ambiente. Entretanto, os agentes robóticos e os agentes de software também satisfazem o modelo de interação com o ambiente.

Genericamente, portanto, um agente interage com seu ambiente através de monitoramento do ambiente por um conjunto de sensores e um conjunto de atuadores, como ilustrado na Figura 5.7. Assim, é possível afirmar que

um agente é um sistema com capacidade de agir de forma autônoma através de *cooperação*, *coordenação* e *negociação* com o ambiente. Isso significa que no projeto e construção de agentes é necessário observar sua interação com o ambiente e a adequação de suas ações sobre este. Nesse caso, o agente é considerado racional se realiza seu trabalho segundo o princípio da racionalidade: *Dada uma seqüência percebida, o agente escolhe, segundo seus conhecimentos, as ações para atingir seu objetivo.*

Essa racionalidade pode ser verificada por meio de uma medida de desempenho. Por exemplo, imaginando um agente robótico para controlar uma antena de rastreio de satélite, a eficiência poderia estar associada à maximização do tempo de visada da antena para o satélite.

Mesmo considerando os objetivos da aplicação, ainda persistem problemas associados a *Como* e *Quando* realizar a medida. O *Como*, em geral, é tratado tomando-se a medida de desempenho como o objetivo da aplicação com o agente. O *Quando* significa o instante em que a avaliação deve ser feita. Isso implica que se poderia pensar na avaliação focada na conclusão da tarefa com avaliação final, não importando o tempo total da ação, em detrimento de eficiência. Por outro lado, medir só quantidade de tempo pode penalizar a consistência das ações.

Apesar da dificuldade para medir o desempenho, a cada seqüência percebida, um agente racional *ideal* deve fazer qualquer ação esperada que maximize sua medida de desempenho, com base na evidência dada pela seqüência percebida e pelo conhecimento que o agente tem.

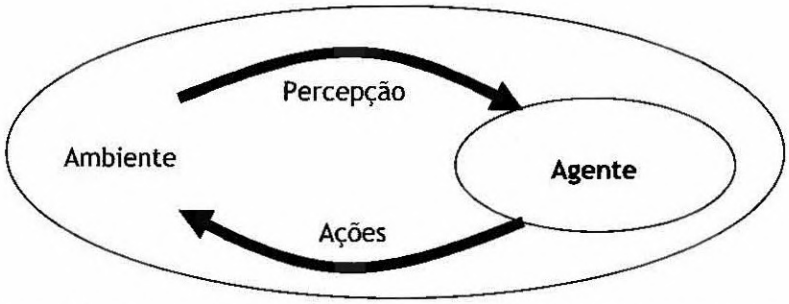


Figura 5.7 - Diagrama de um Agente Genérico em um ambiente não-estruturado.

A associação entre as percepções (P^*) e as ações (A) que o agente deve tomar é realizada através de um mapeamento (uma função) $f: P^* \rightarrow A$. A forma mais simples de implementação do mapeamento é através de uma tabela com a estrutura (P^*,A). Mas, dependendo da aplicação, o número de registros da tabela pode se tornar não-enumerável.

A implementação de um agente necessita de uma arquitetura (computador, sensores e atuadores) e um programa (função que implementa o mapeamento). Os requisitos para se projetar um agente compreendem: a) *O que será percebido*; b) *Ações*; c) *Os objetivos ou a medida de desempenho*; d) *O tipo de ambiente*. A Figura 5.8 ilustra um exemplo de um programa agente.

```
function AGENTE(percepçao) returns ação
        static memoria, memória do agente sobre o mundo
        memoria ← Atualiza-memoria(memoria,percepçao)
        ação ← Escolhe-melhor-ação(memoria)
        memoria ← Atualiza-memoria(memoria,ação)
        return ação
```

Existem diferentes formas de classificação de agentes. Em Russell e Norvig (1995), os agentes são classificados em: a) *Reflexivos* (*Reativos*) – escolhem as ações com base nas percepções; b) *Agentes que seguem o mundo* (*Baseados em modelo*) – são reflexivos e consideram também um histórico de percepções que reflete o estado atual do ambiente; c) *Agentes baseados em objetivos* – são baseados em modelo e requerem informação sobre objetivos associados a situações desejáveis; d) *Agentes baseados em utilidade* – são baseados em objetivos e buscam satisfazer um critério de utilidade de suas ações para atingi-los; e) *Agentes com aprendizagem* – incorporam conhecimento novo adquirido durante o processo de interação com o ambiente.

Os agentes com aprendizagem têm sido preferidos para o desenvolvimento de aplicações em IA, por diminuírem a necessidade de especificação *a priori* de todo o conhecimento envolvido em uma aplicação. Construindo agentes com essa característica, é possível deixá-los interagirem com ambientes inicialmente desconhecidos para aumentar sua competência. A Figura 5.9 ilustra o modelo geral de um agente com aprendizagem.

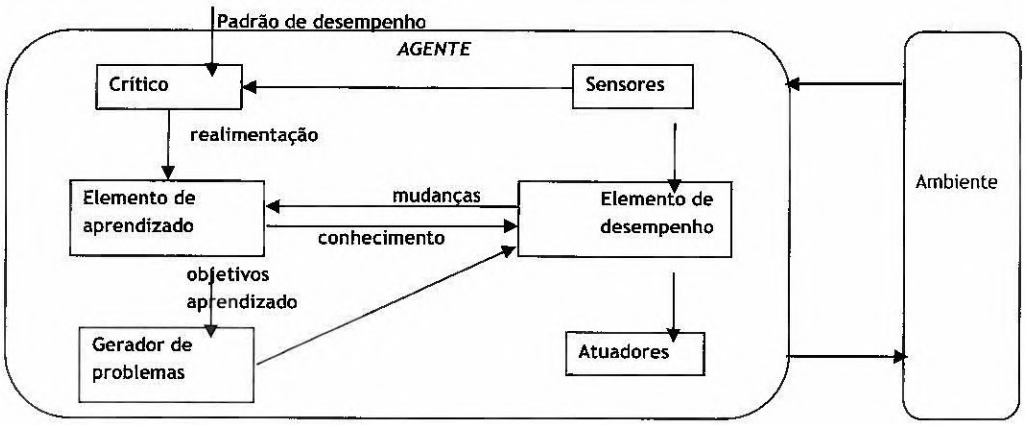


Figura 5.9 – Modelo geral de um agente com aprendizagem. Adaptado de Russell e Norvig (1995).

O *Elemento de desempenho* é responsável pela seleção das ações a partir das percepções. O *Elemento de aprendizado* faz a incorporação de novos conhecimentos derivados das alterações conseqüentes das ações tomadas pelo agente, a partir das observações do *Crítico*, que pode ser pensado como um supervisor. O *Gerador de problemas* gera cenários com ações que podem gerar novas experiências e informações.

Pelas definições apresentadas, fica evidente que o projeto de um agente é complexo e se baseia em conhecimento, que deve ser o mais completo possível para que o agente seja eficiente. Portanto, as formas de representação de conhecimento escolhidas podem fazer a diferença para a eficiência de um agente.

5.5 Introdução à Lógica Nebulosa

A Lógica Nebulosa (ou Lógica Difusa – "*Fuzzy Logic*") trabalha com o conceito de verdade parcial (ou seja, imprecisão). Na lógica clássica as declarações podem ser apenas verdadeiras (V) ou falsas (F). Na Lógica Nebulosa, introduzida formalmente por Lotfi Zadeh em 1965, é usado o conceito de grau de pertinência. Assim, uma declaração na Lógica Nebulosa pode ter graus de pertinências associados a verdadeiro e falso simultaneamente.

Um exemplo para elucidar os conceitos considera o universo de todas as *pessoas* que trabalham em uma empresa, do qual é preciso selecionar todas as *pessoas altas*. Para formar, então, o conjunto desejado, poder-se-ia estabelecer a seguinte regra: *Qualquer pessoa que tenha no mínimo 1,72 m de altura é considerada alta.* Essa regra claramente excluiria uma pessoa com 1,71 *m* do conjunto de interesse, apesar de muitos a considerarem uma pessoa alta. Se o limiar inferior fosse alterado para 1,70 *m*, muitas pessoas seriam incluídas no conjunto de interesse, mas que, ao mesmo tempo, também poderiam pertencer ao conjunto *pessoas não-altas* (ou outro tipo de conjunto nebuloso definido).

Na Lógica Nebulosa, a verdade de uma declaração é tratada como um grau de pertencimento (pertinência), significando que a declaração pode tanto ser verdadeira como falsa ao mesmo tempo, mas com graus diferentes. Esse conceito de grau de pertencimento poderia ser usado para resolver o problema do conjunto de pessoas altas anterior. Assim, considerando 1,72 metros como o limiar para o conjunto, a afirmação de que uma pessoa com 1,71 *m* é alta poderia ser considerada como verdadeira com grau de 90% e como falsa com grau de 10%, por exemplo.

O grau de pertencimento de um objeto a um conjunto nebuloso é declarado através de funções de pertencimento (ou de pertinência). Estas funções podem ser de diversas naturezas e estão associadas à semântica desejada para um conceito. A implementação das funções pode ser realizada através de equações que definem triângulos, trapézios, Gaussianas, forma de S, sinos, etc. Considerando um conjunto nebuloso A, sua função de pertinência é definida como: $\mu_A(x): X \rightarrow [0,1], \quad x \in X$, onde $\mu_A(x)$ representa a função de pertinência do elemento x ao conjunto A; X é o Universo de Discurso de todos os elementos, ou seja, o escopo da variável. O processo de atribuição de um grau de pertinência à variável x, $\mu_A(x)$, é denominado de "*fuzificação*" ("*fuzzyfication*").

Um conjunto nebuloso discreto pode ser representado associando-se cada elemento x ao seu grau de pertinência segundo a notação μ/x, significando que x é um elemento do conjunto nebuloso A com grau μ. Na equação (5.1), tem-se um exemplo dessa notação para um conjunto nebuloso discreto A e, na equação (5.2), para um conjunto nebuloso contínuo A.

$$A = \mu_A^1/x_1 + \mu_A^2/x_2 + \cdots + \mu_A^n/x_n = \sum_{i=1,n} \mu_A^i/x_i \qquad (5.1)$$

$$A = \int_X \mu_A(x)/x \qquad (5.2)$$

A forma da função de pertinência controla várias características de um sistema baseado em definições de conjuntos nebulosos (a nebulosidade, a tolerância, a precisão, etc.). A Figura 5.10 ilustra uma definição de conjuntos nebulosos usando funções trapezoidais.

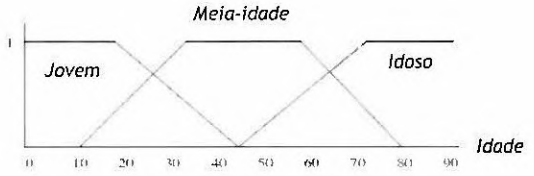


Figura 5.10 – Exemplo de uma definição de conjuntos nebulosos.

A lógica clássica utiliza os operadores lógicos "E" (& ou $\wedge$), "OU" ($\vee$) e "NÃO" ($\neg$), que também se aplicam à lógica nebulosa. Na realidade, se os graus de pertinência estão nos extremos (1 ou 0), os operadores lógicos padrão se aplicam diretamente, porque a lógica nebulosa pode ser considerada como um superconjunto da lógica clássica. Mas, normalmente, o operador "E" (& ou $\wedge$) é implementado através de uma função de cálculo de valor mínimo; o operador "OU" ($\vee$) é implementado por uma função de cálculo de valor máximo; e o operador "NÃO" ($\neg$) usa o complemento aditivo, ou seja:

$$\mu_A(x) \wedge \mu_B(x) = min(\mu_A(x),\mu_B(x)) \quad (5.3)$$

$$\mu_A(x) \vee \mu_B(x) = max(\mu_A(x),\mu_B(x)) \quad (5.4)$$

$$\mu_{\neg A}(x) = 1 - \mu_A(x) \quad (5.5)$$

Estas são as formas mais simples de definir os três operadores, mas eles também podem ser definidos de outras formas usando *T-normas*, para uma classe geral de operadores de intersecção, ou *T-conormas*, para uma classe geral de operadores de agregação para a união de conjuntos nebulosos.

É possível também construir sistemas baseados em regras utilizando a lógica nebulosa. A interpretação das regras estabelecidas envolve:

a) "Fuzificação" das entradas. São atribuídos valores de pertinência no intervalo [0,1] a todas as declarações nebulosas nos antecedentes das regras. Caso haja um único antecedente, seu grau de pertinência é considerado também como o grau de certeza (suporte) da regra.

b) Aplicação de operadores nebulosos aos múltiplos antecedentes, o que resultará em um antecedente único com valor de pertinência no intervalo [0,1], que também será considerado como o grau de suporte da regra.

c) Aplicação do método de implicação, que usa o grau de suporte de toda regra para obter o conjunto nebuloso de saída, ou seja, realiza a dedução. O conseqüente de uma regra nebulosa atribui um conjunto nebuloso à saída, representado por uma função de pertinência escolhida para indicar as qualidades do conseqüente. Se o antecedente é apenas parcialmente verdade (grau de pertinência < 1), então o conjunto nebuloso de saída é truncado de acordo com o método de implicação. Na lógica nebulosa a implicação faz uso de uma versão nebulosa de *modus ponens*, originalmente usado para obter B a partir de $A \wedge (A \rightarrow B)$. Tomando-se o fato de que $A \wedge (A \rightarrow B) \leftrightarrow (A \wedge B)$, a versão nebulosa poderia ser: $\mu_R(x,y) = min(\mu_A(x),\mu_B(y))$, conhecida como a Regra de Mamdani. Com esta caracterização de implicação, pode-se obter a inferência nebulosa (ou *modus ponens generalizado*):

Se *x é A* Então *y é B*

Tem-se A^*

Então B^*

Quando existem várias regras que envolvem a mesma variável, é necessário usar um método de agregação das conclusões implicadas por cada regra.

Se *x é* A^1 Então *y é* B^1

Se *x é* A^2 Então *y é* B^2

...

Se *x é* A^n Então *y é* B^n

Existem diferentes técnicas de agregação (Mamdani, Larsen, Lukasiewics etc.), que produzem resultados ligeiramente diferentes. A escolha do método depende do comportamento desejado no sistema.

Após o processo de agregação, cada variável de saída é representada por um conjunto nebuloso. Nas aplicações, em geral, o sistema precisa fornecer uma saída na forma de um escalar. O processo de se obter esse escalar é denominado "desfuzificação" (adaptado do inglês "Defuzzification"). Várias técnicas podem ser usadas para a "desfuzificação" de um conjunto nebuloso, sendo as mais usadas conhecidas como "centro de gravidade", "centro de possibilidade máxima" e "o maior máximo". Este processo sempre implica em perda de informação.

5.6 Introdução às Redes Neurais Artificiais

As Redes Neurais Artificiais (RNA), também conhecidas como métodos conexionistas, são inspiradas no funcionamento dos neurônios biológicos do cérebro. A pesquisa e o desenvolvimento de aplicações desses sistemas deram origem à área de computação neural, também conhecida como neurocomputação.

Em 1943, McCulloch (neurofisiologista) e Pitts (matemático) desenvolveram um modelo computacional que descreve o funcionamento de um neurônio biológico. Entretanto, a aprendizagem em RNA só foi abordada posteriormente, com base nos estudos do psicólogo Donald Hebb, que propôs a base do aprendizado com a formulação explícita de uma regra de aprendizagem fisiológica para a modificação sináptica, teorizando que aprendizagem em redes neurais se dá pela variação dos pesos na entrada do neurônio (Haykin, 1999). Segundo a teoria de Hebb, o aprendizado baseia-se no reforço das ligações sinápticas (pesos) entre neurônios.

Somente em 1958, Frank Rosenblatt desenvolveu o *Perceptron* com sinapses ajustáveis que, seguindo o postulado de Hebb, podiam ser treinadas para classificar certos tipos de padrões. Rosenblatt propôs uma estrutura de ligação entre os neurônios, além de um algoritmo de treinamento.

Em 1960, Widrow e Hoff propuseram o *Adaline – Elemento Linear Adaptável* (*Adaptive Linear Element*), que consiste em um modelo similar ao *Perceptron*, mas com ajustes dos pesos sinápticos dado pelo algoritmo dos mínimos quadráticos (*LMS - Least Mean Squared*). O modelo *Adaline* foi sucedido pelos modelos *Madaline*, que consistiam de *Adaline* em múltiplas camadas (Fausett, 1994).

Em 1970, surgiram os mapas auto-organizáveis, que utilizam aprendizagem competitiva. Willshaw e Maslburg (1986) publicaram o primeiro artigo sobre a formação de mapas auto-organizáveis, e em 1980 Grossberg, baseado no seu trabalho sobre aprendizagem competitiva, estabeleceu um princípio de auto-organização conhecido como a *Teoria da Ressonância Adaptável* (*ART, Adaptative Resonance Theory*). Também em 1982, outro modelo de rede neural foi proposto por Teuvo Kohonen na categoria dos mapas auto-organizáveis, utilizando o aprendizado competitivo (SOM - *self-organizing maps*).

Depois de um período de estagnação, provocado por críticas sobre a incapacidade das redes para resolver problemas não lineares, as pesquisas em redes neurais foram retomadas após a publicação, em 1982, do artigo de John Hopfield, que propunha um novo modelo de computação executada por redes recorrentes simétricas, com base na minimização de uma função de energia. Hopfield estabeleceu um isomorfismo entre o modelo de rede recorrente proposto e um modelo físico utilizado em física estatística, que despertou um novo interesse dos pesquisadores pelas RNA (Haykin, 1999). Outros modelos similares ao de Hopfield surgiram pouco depois: a máquina de Boltzmann, proposta por Hinton em 1985, e a BAM (*Bidirectional Associative Memory*), proposta por Kosko em 1987.

Nos anos 1980, as RNA baseadas em Perceptrons de múltiplas camadas foram viabilizadas por um novo algoritmo de aprendizado chamado *Error Back-propagation* (retropropagação do erro), que permitiu a estas RNA resolverem problemas não lineares (Haykin, 1999).

Existem diversas arquiteturas de RNA, sendo as mais comuns (ver Figura 5.11): a) RNA de uma camada de neurônios; b) RNA com múltiplas camadas de neurônios; c) reticulados de neurônios; d) RNA de uma camada com conexões recorrentes.

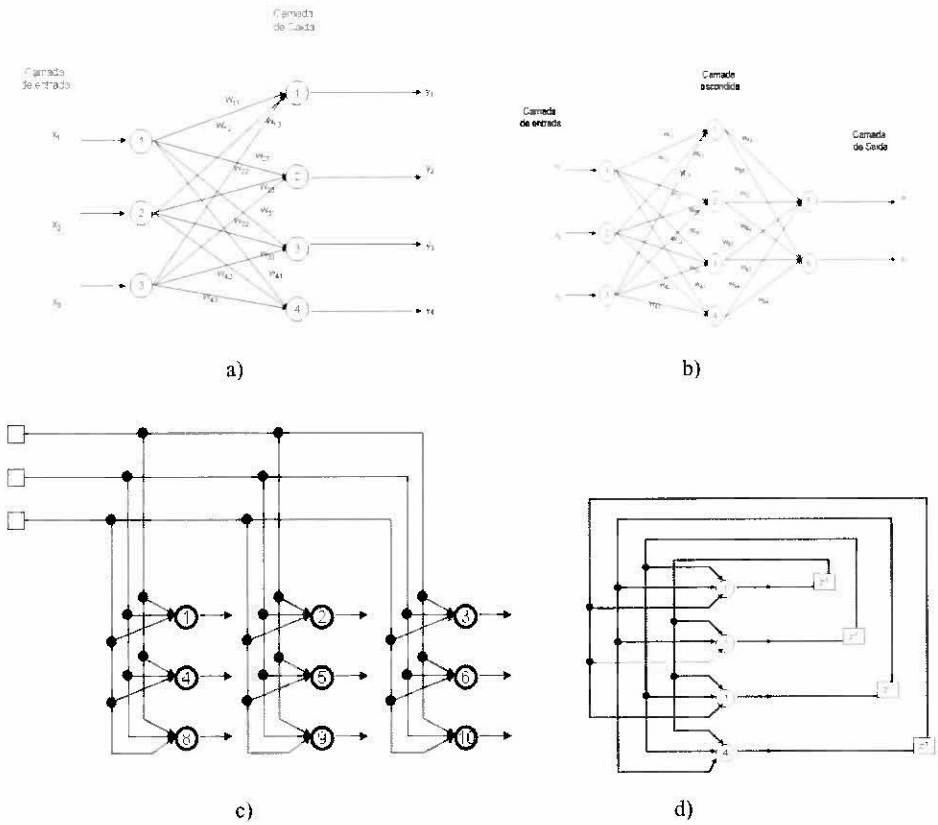


Figura 11 – Exemplos de Arquiteturas de RNA. a) Rede de uma camada; b) Rede com múltiplas camadas; c) Reticulado de neurônios; d) Rede com uma camada com conexões recorrentes. Adaptado de Haykin (1999).

É importante notar que as informações são distribuídas para todas as unidades em uma mesma camada, o que fazem as RNA serem totalmente distribuídas. Os neurônios em uma mesma camada executam, simultaneamente, determinadas funções matemáticas, normalmente não-lineares, fazendo com que as RNA sejam paralelas. Os pesos entre neurônios de mesma camada ou de camadas diferentes armazenam o conhecimento do problema em função das entradas apresentadas para cada neurônio da rede. Este conhecimento é adquirido devido à capacidade de aprender através de exemplos e de generalizar a informação das RNA.

As RNA, por serem modelos altamente distribuídos, apresentam duas características fundamentais para o desenvolvimento de aplicações: a) são tolerantes a falhas; e b) tomam decisões com dados incompletos.

As arquiteturas na Figura 5.11 utilizam um elemento básico de processamento (ou neurônio artificial) cujo funcionamento é descrito através de uma agregação ponderada de informação e da execução de uma função de ativação, modelando o funcionamento de um neurônio biológico. A Figura 5.12 ilustra a estrutura de um neurônio biológico. A Figura 5.13 ilustra o modelo matemático proposto por McCulloch e Pitts.

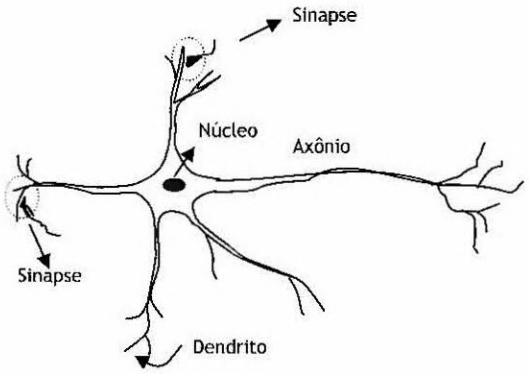


Figura 5.12 – Estrutura de um neurônio biológico.

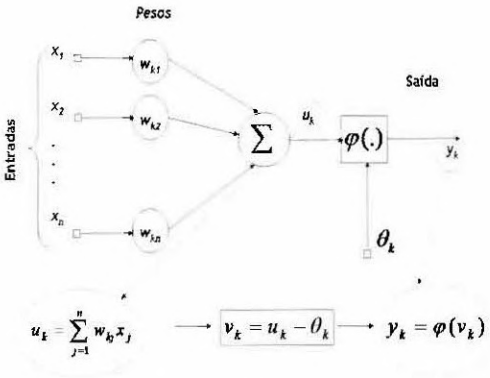


Figura 5.13 - Modelo de McCulloch e Pitts.

O neurônio biológico representado na Figura 5.12 ilustra os elementos básicos associados ao modelo matemático na Figura 5.13. As informações provenientes de sensores ou de outras camadas de neurônios são comunicadas a um neurônio através das sinapses. Os dendritos são responsáveis pela transmissão das informações que passam pelas sinapses para o núcleo da célula, onde ocorre a agregação. Na realidade, esses processos são fluxos de íons que terminam alterando o potencial na membrana celular. Quando esse potencial atinge um limiar (40 mV), a célula dispara um pulso pelo axônio e volta ao repouso (com potencial na membrana em torno de -70mV).

Na Figura 5.13, os x_j representam as informações que chegam ao neurônio k. Os w_{kj} são os respectivos pesos que ponderam as entradas, simulando as sinapses. A informação transmitida é agregada através de uma combinação linear, dada pela expressão u_k. O resultado da agregação é comparado com o limiar θ_k, resultando em v_k, que define se o neurônio dispara ($y_k = 1$) ou não ($y_k = 0$). A função $\varphi(.)$ define a forma de ativação do neurônio, podendo ser, entre outras: a) a função limiar; b) a função sinal; c) a função logística sigmoidal; d) a função linear. A Figura 5.14 ilustra os quatro tipos de funções mencionados. O modelo de neurônio artificial na Figura 5.13 utiliza a função limiar. Entretanto, as funções sigmoidais (Figura 5.14c) são as mais utilizadas, por serem monotonicamente crescentes e suaves, características importantes para a implementação de alguns algoritmos de aprendizagem.

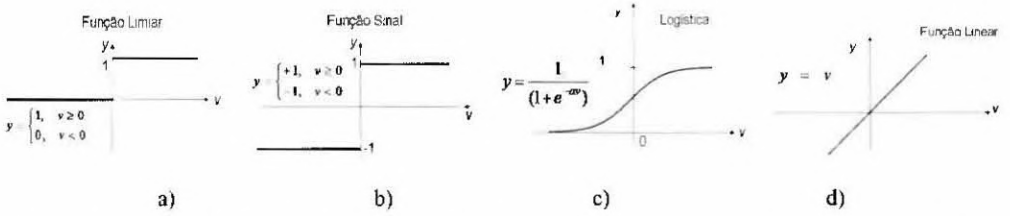


Figura 5.14 – Tipos de funções de ativação. a) Função Limiar; b) Função Sinal; c) Função Logística; d) Função Linear.

5.6.1 Aprendizagem em Redes Neurais Artificiais

A aprendizagem (ou treinamento) em RNA consiste em adaptar os pesos sinápticos que ligam os neurônios às entradas e a outros neurônios da rede. O processo pode ser caracterizado formalmente pela expressão (6.1), ou seja, enquanto houver alterações do peso sináptico que conecta o neurônio k ao neurônio (ou entrada) j, para todos e quaisquer k e j, a RNA se encontra em processo de aprendizagem.

$$Aprendizagem = \frac{dw_{kj}}{dt} \neq 0 \qquad (6.1)$$

Existem dois métodos básicos de aprendizagem: a) supervisionado; e b) não-supervisionado. O método supervisionado precisa do auxílio de um "professor", que supervisiona o desempenho da rede. A rede é exposta simultaneamente a um conjunto de vetores de dados de entrada e a um conjunto de vetores de dados de saída que se espera a rede reproduza. Os pesos são adaptados em função da proximidade dos vetores de saída produzidos pela rede e os vetores de referência fornecidos com as entradas. A aprendizagem termina quando a rede atinge um limiar de tolerância desejado. Nos métodos não-supervisionados, as redes adaptam seus pesos sinápticos em função das regularidades estatísticas presentes nos dados, ou seja, o método não-supervisionado tem conotação de aprendizagem "autodidata". Nesse caso, em geral, o processo de aprendizagem se repete e a rede monitora critérios de estabilidade dos pesos.

Existem diferentes algoritmos de aprendizagem que se enquadram nos métodos citados anteriormente, muitos deles diretamente associados aos modelos de RNA. Como este material é introdutório, não exaurindo todos os aspectos sobre RNA, serão abordados três tipos de algoritmos neste texto: a) Aprendizagem por correção do erro; b) Aprendizagem Hebbiana; e c) Aprendizagem competitiva.

Na aprendizagem por correção do erro, o ajuste dos pesos é proporcional ao erro observado na saída da rede. A RNA é estimulada na entrada,a informação é processada até a saída da rede, e a saída obtida é comparada com a resposta esperada para o vetor de entrada fornecido. A comparação gera um sinal de erro, utilizado para corrigir os pesos sinápticos. Considerando $\overline{x}(n)$ a entrada e $\overline{d}(n)$ a saída desejada, quando estimulado, o neurônio produz uma saída $\overline{y}(n)$. O sinal de erro gerado em cada neurônio k é:

$$e_k(n) = d_k(n) - y_k(n) \qquad (6.2)$$

O ajuste aplicado sobre as conexões de um neurônio k será dado por:

$$\Delta w_{kj}(n) = \eta e_k(n) x_j(n) \qquad (6.3)$$

onde $\eta > 0$ é um valor de proporcionalidade positivo; $e_k(n)$ é o erro observado no neurônio k para a entrada fornecida no instante discreto de tempo n; e $x_j(n)$ é a componente j do vetor de entrada. Com isso, o novo valor do peso no instante n+1 será:

$$w_{kj}(n+1) = w_{kj}(n) + \Delta w_{kj}(n) \tag{6.4}$$

Na aprendizagem Hebbiana, o ajuste dos pesos ocorre proporcionalmente à correlação entre as atividades pré e pós sinápticas, ou seja, entre a entrada e a saída do neurônio, conforme a equação (6.5). O ajuste do peso é dado pela equação (6.4).

$$\Delta w_{kj}(n) = \eta x_j(n) y_k(n) \tag{6.5}$$

Na aprendizagem competitiva, os neurônios de uma camada competem entre si pelo direito de ativarem para um determinado vetor apresentado na entrada da rede. Este tipo de aprendizagem envolve um conjunto de neurônios do mesmo tipo, um limite imposto à robustez de cada neurônio e um mecanismo de competição entre os neurônios. O neurônio vencedor leva tudo ("*winner-takes-all*"), no sentido de que só ele terá a saída ativada e os pesos ajustados segundo a equação (6.6).

$$\Delta w_{kj}(n) = \begin{cases} \eta[x_j(n) - w_{kj}(n)], & se \quad k \quad é \quad vencedor \\ 0, & caso \quad contrário \end{cases} \tag{6.6}$$

5.6.2 O Percetpron

O Perceptron foi proposto por Rosenblatt em 1958 como um modelo de rede neural com aprendizagem supervisionada para resolver problemas de classificação com duas classes linearmente separáveis. O algoritmo de aprendizagem do Perceptron converge e posiciona a superfície de decisão (um hiperplano) entre classes. A prova de convergência do algoritmo é conhecida como o teorema de convergência do Perceptron. Um único Perceptron classifica apenas duas classes. A classificação de mais de duas classes exige o uso de mais Perceptrons. A capacidade de classificação é dada por 2^N (onde N é o número de Perceptrons).

Considere um Perceptron com um único neurônio e N entradas, cada qual fornecendo uma informação $x_j(n)$ do vetor de entrada $\bar{x}(n)$ ponderada por um peso w_j do vetor de pesos associado $\bar{w} = (w_1, w_2, \cdots w_N)$. O algoritmo de aprendizagem do Perceptron consiste em obter o valor do incremento $\Delta\bar{w}(n)$ a ser aplicado ao vetor de pesos $\bar{w}(n)$, de forma que o valor atualizado $\bar{w}(n+1)$ faça o Perceptron gerar uma saída mais próxima da saída desejada associada ao vetor $\bar{x}(n)$. A saída desejada é 1 se o vetor $\bar{x}(n) \in C_1$ (a entrada é da classe 1) ou -1 se $\bar{x}(n) \in C_2$ (a entrada é da classe 2). O erro $e(n)$ na saída do Perceptron é calculado conforme a equação (6.2), e o vetor de incremento, $\Delta\bar{w}(n)$, é calculado pela equação (6.7), onde η é a taxa de aprendizagem, como na equação (6.3).

$$\Delta\bar{w}(n) = \eta\bar{x}(n)e(n) \tag{6.7}$$

5.6.3 Perceptron de Múltiplas Camadas

O Perceptron de Múltiplas Camadas é usado para resolver problemas complexos não lineares, utilizando aprendizagem supervisionada pelo o algoritmo de retropropagação do erro ("*Error back-propagation*"), apresentado por Rumelhart et al (1986), tendo sido um passo importante para a retomada das pesquisas em redes neurais artificiais. Outros algoritmos para Perceptrons de Múltiplas Camadas foram pesquisados por Werbos (1988) e Parker (1987).

A Figura 5.11b ilustra a topologia de um Perceptron de Múltiplas Camadas para uma RNA com uma camada (escondida) entre as camadas de entrada e de saída. O número de camadas escondidas pode variar com a complexidade do problema e, assim, outras topologias podem ter L camadas. O algoritmo de aprendizagem por retropropagação do erro consiste em dois passos: a) um passo para frente, que propaga as informações apresentadas na entrada, camada por camada, até a geração de saída; e b) um passo para trás, que retropropaga o erro através do cálculos dos gradientes locais de cada camada.

Na fase de propagação, cada neurônio k em uma camada agrega as informações que lhes são apresentadas, começando pela primeira camada escondida após a camada de entrada, segundo a equação (6.8), onde m representa o número de entradas para o neurônio. O número de neurônios em cada camada escondida varia de acordo com a aplicação. O resultado da agregação passa por uma função de ativação do tipo sigmoidal (por exemplo, a função logística da Figura 5.14c, gerando o sinal na saída $y_k(n) = \varphi(v_k(n))$.

$$v_k(n) = \sum_{j=0}^{m} x_j(n) w_{kj}(n) \tag{6.8}$$

As saídas dos neurônios em uma camada escondida são as entradas dos neurônios na camada seguinte. O processo de ativação dos neurônios acontece em todas as camadas e se encerra quando os neurônios da camada de saída são ativados. Na propagação das informações pelas camadas, os pesos sinápticos da rede são mantidos fixos.

Os pesos sinápticos são ajustados de acordo com uma regra de correção do erro, com base no algoritmo LMS. O passo de retropropagação inicia-se com o cálculo do erro para cada neurônio na camada de saída (equação (6.2)). Esse sinal de erro é então propagado para trás, através das camadas da RNA, pelo cálculo dos gradientes locais dos neurônios de cada camada. Como o número de neurônios na camada de saída pode ser superior a 1, é calculado o valor instantâneo da soma dos erros quadráticos na saída conforme a equação (6.9), onde $\varepsilon(n)$ é o valor instantâneo da soma dos erros quadráticos dos q neurônios na camada de saída. Os novos valores dos pesos do neurônio k em cada camada l são calculados pela equação (6.10).

$$\varepsilon(n) = \frac{1}{2} \sum_{k=1}^{q} e_k^2(n) \tag{6.9}$$

$$w_{kj}^{l}(n+1) = w_{kj}^{l}(n) + \eta \delta_k^l(n) y_j^{l-1}(n) \tag{6.10}$$

Na equação (6.10), $y_j^{l-1}(n)$ é saída do neurônio j na camada anterior l-1 na iteração n; $w_{kj}^{l}(n)$ é o peso sináptico do neurônio k da camada l; η é a taxa de aprendizagem; e $\delta_k^l(n)$ é o gradiente local do neurônio k da camada l. Se k é um neurônio na camada de saída, o seu gradiente é obtido pela equação (6.11).

$$\delta_k^l(n) = -\frac{\partial \varepsilon(n)}{\partial v_k^l(n)} = -e_k \varphi'(v_k^l(n)) \tag{6.11}$$

Se k é um neurônio de uma camada escondida (l), o seu gradiente é dado pela equação (6.12), ou seja, pelo produto da derivada de sua função de ativação pela soma dos gradientes dos neurônios da camada seguinte (l+1) ponderados pelos pesos que conectam o neurônio k (na camada l) a cada neurônio j na camada seguinte (l+1).

$$\delta_k^l(n) = \varphi'(v_k^l(n)) \sum_j \delta_j^{l+1}(n) w_{jk}^{l+1}(n) \tag{6.12}$$

O ajuste dos pesos sinápticos faz com que a resposta da rede se mova para mais perto da resposta desejada. O processo de aprendizado termina quando se atinge um erro desejado ou quando é realizado um número máximo de épocas de aprendizado.

5.6.4 Mapa Auto-organizável de Kohonen

O SOM (*Self Organizing Map*), também conhecido como "Mapa de Kohonen", pertence à classe de RNA não-supervisionadas que usam aprendizagem competitiva, em que apenas um neurônio ou grupo local de neurônios é ativado para um vetor de entradas $\bar{x}(n)$ no instante n. Cada neurônio calcula o nível de similaridade entre o vetor de entrada e o seu vetor de pesos, utilizando uma medida de distância Euclidiana entre os dois vetores. A distância Euclidiana D_k entre o vetor de pesos $\overline{w_k}(n)$ e vetor de entrada é dada pela equação (6.13).

$$D_k = \sum_{j=1}^{m} [x_j(n) - w_{kj}(n)]^2 \tag{6.13}$$

O processo de competição escolhe o neurônio para o qual a distância entre seu vetor de pesos e o vetor de entradas é mínima quando comparada com a mesma distância calculada para qualquer outro neurônio no mapa. Esse processo é dado pela equação (6.14).

$$k^v = \arg \min_{k=1,\ldots,N} (\{D_k\}) \tag{6.14}$$

O neurônio vencedor k^v e seus vizinhos terão seus pesos ajustados pelo algoritmo de aprendizagem segundo a equação (6.6), sendo que o ajuste dos neurônios vizinhos cai com a distância para neurônio k^v, significando a influência de um sobre o outro. A equação (6.6) é modificada para permitir esse tipo de decaimento no ajuste através da introdução da função não linear h_{k,k^v}, que deve gerar o valor 1 para o neurônio vencedor k^v e um valor menor que 1 em função da distância do neurônio k para o vencedor k^v.

$$w_{kj}(n+1) = w_{kj}(n) + \Delta w_{kj}(n) = w_{kj}(n) + \eta h_{k,k^v} [x_j(n) - w_{kj}(n)] \tag{6.15}$$

A vizinhança de cada neurônio pode ser definida de acordo com a forma geométrica usada para representar o reticulado de neurônios. A Figura 5.15 ilustra dois exemplos de representação propostas por Kohonen, retangular e hexagonal.

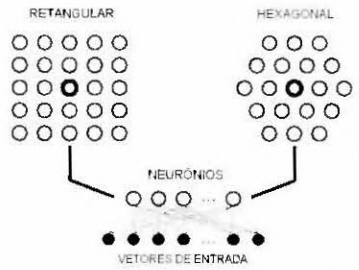


Figura 5.15 – Tipos de arranjos de neurônios para Mapas auto-organizáveis.

As RNA do tipo mapas auto-organizáveis usam o princípio de que padrões que compartilham características comuns devem ser agrupados, com cada grupo de padrões representando apenas uma classe (embora uma mesma classe possa ser representada por mais de um agrupamento). Tanto a vizinhança como a taxa de aprendizagem decaem com o número de iterações (épocas) do algoritmo.

Durante o processo de aprendizagem, o SOM constrói um mapa topológico, onde os nós topologicamente próximos respondem de forma semelhante a padrões de entrada semelhantes. A segunda fase é a de classificação, onde a rede SOM utiliza o mapa organizado para identificar a classe mais próxima à entrada.

No final do processo de aprendizagem, cada neurônio ou grupo de neurônios vizinhos representará um padrão distinto dentro do conjunto de padrões fornecidos como entrada.

5.7 Introdução aos Algoritmos Evolutivos

Os Algoritmos Evolutivos (AE) são técnicas heurísticas de otimização e de busca estocásticas que se baseiam na teoria da evolução das espécies de Charles Darwin, ou seja, na teoria de que, em uma população de indivíduos, apenas os que satisfazem alguns critérios de seleção (aptidão – *"fitness"*) estão aptos para a reprodução, enquanto os demais são extintos.

Com a repetição do processo de reprodução no tempo, a população converge para indivíduos que melhor satisfazem aos critérios de seleção, ou seja, com melhor aptidão (*"fitness"*). Entretanto, pode haver reprodução com indivíduos que não satisfazem ou satisfazem parcialmente os critérios de seleção (menos *aptos*), ou a alteração dos indivíduos resultantes da reprodução. Isso faz com que a população explore outras regiões do espaço de busca, havendo, portanto, uma probabilidade de que haja indivíduos fora do núcleo da população em convergência que satisfaçam melhor os critérios de seleção, redirecionando a população para outro núcleo de indivíduos.

Na resolução de um problema de otimização utilizando AE, os indivíduos da população representam as possíveis soluções, e o critério de seleção é ajustado de acordo com a qualidade da solução desejada para o problema.

De forma geral, o funcionamento dos AE segue os passos do algoritmo abaixo na busca por uma solução mais adaptada ao problema.

Gera a população P aleatoriamente com *N* indivíduos

Repete

Avalia a aptidão de todos os indivíduos em P

Seleciona os indivíduos mais aptos para reprodução

Reproduz entre os mais aptos gerando novos indivíduos

Atualiza a população. Insere os novos indivíduos e retira

alguns indivíduos menos aptos

Até critério de parada ser satisfeito

Uma propriedade importante dos AE para resolver problemas é que eles iniciam a busca a partir de uma população de soluções possíveis que trocam atributos importantes entre si durante o processo de reprodução. Apesar das características importantes dos AE, não há garantia de que as buscas utilizando AE convirjam para uma solução ótima global.

Os AE são usados em problemas de otimização considerados difíceis, quando não existe outra técnica para se obter uma solução. Alguns exemplos de aplicação são: Otimização de funções numéricas em geral; Otimização combinatória (Problema do caixeiro viajante, Problema de empacotamento, Alocação de recursos); Aprendizado de Máquina; Problemas Inversos; etc.

Existem diferentes algoritmos baseados em estratégias evolutivas. Entretanto, o mais popular é o Algoritmo Genético (AG), concebido por John Holland e seus alunos (Holland, 1975) como uma metáfora da teoria da evolução de Charles Darwin. Apesar das pesquisas em desenvolvimento nos anos 1950 e 1960, somente nos anos 1970 foram publicadas as pesquisas de Holland. Mesmo assim, a popularização dos AG só ocorreu com a publicação do livro de David Goldberg (1989).

Um AG é um algoritmo estocástico que trabalha simultaneamente com uma população de soluções para um problema, utilizando apenas informações de custo (aptidão das soluções) e sem requerer qualquer outra informação auxiliar (por exemplo, informação de gradiente). Os AG são de fácil implementação em computadores seqüenciais, totalmente adaptados para implementações em computadores paralelos, podem ser hibridizados com outras técnicas de busca e funcionam com parâmetros contínuos ou discretos.

Durante a evolução, os indivíduos são combinados pela operação de cruzamento ("*crossover*"), produzindo novos indivíduos que podem sofrer ou não mutação ("*mutation*"). A população evolui através de sucessivas gerações até a obtenção de uma solução ótima. O funcionamento de um AG simples é descrito segundo o algoritmo que segue.

Gera a população inicial

Repete

Avalia cada indivíduo da população (cálculo da aptidão)

Seleciona os indivíduos mais aptos

Cruzamento de indivíduos selecionados (operador de "*crossover*")

Mutação dos novos indivíduos (operador de mutação)

Atualiza a população. Insere os novos indivíduos e retira-se alguns

indivíduos menos aptos

Até critério de parada ser satisfeito

Nos AG cada solução do problema é representada por uma estrutura de dados denominada cromossomo. Os parâmetros do problema de otimização são representados por cadeias de valores. Alguns exemplos de

cromossomos são: vetores de número reais (2.345, 4.3454, 5.1, 3.4); vetores ("*strings*") de bits (111011011); vetores de números inteiros (1, 4, 2, 5, 2, 8); ou alguma outra estrutura de dados adequada para a representação da solução.

A Aptidão é uma nota associada ao indivíduo que avalia quão boa é a solução por ele representada. O tipo de aptidão pode ser a própria função objetivo, resultar do escalonamento da função objetivo ou ser baseado no ranking do indivíduo na população.

A Seleção imita o processo de seleção natural, em que os melhores indivíduos (com maior aptidão) são selecionados para gerar novos indivíduos através das operações de cruzamento e mutação. Este processo dirige o AG para as melhores regiões do espaço de busca. Os tipos mais comuns de seleção são: o proporcional à aptidão (os mais aptos têm maior probabilidade de ser selecionados) e o torneio, em que n (tipicamente 2) indivíduos são escolhidos aleatoriamente da população e o melhor é selecionado.

A operação de cruzamento ("*crossover*"), essencial para a geração de soluções cada vez mais aptas, consiste na troca de atributos entre os indivíduos da população selecionados. Existem diferentes estratégias para realização dessa operação. A mais simples consiste no cruzamento em um ponto escolhido aleatoriamente, conforme ilustrado na Figura 5.16a, em que os indivíduos 1 e 2 trocam os últimos 3 bits de seus cromossomos.

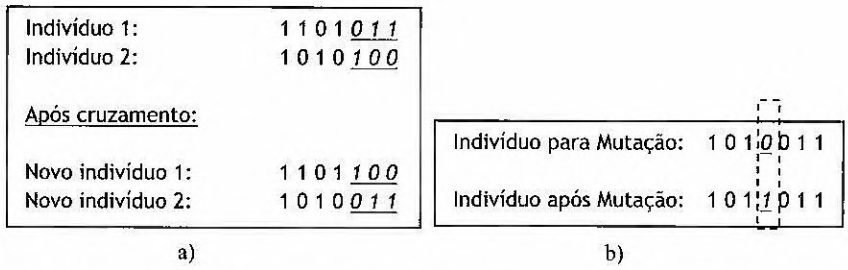


Figura 5.16 – a) Cruzamento em um ponto; b) Mutação de um indivíduo.

A Figura 5.16b ilustra a operação de mutação sobre um indivíduo, em que o *bit* que sofre a mutação é escolhido também aleatoriamente satisfazendo uma probabilidade pré-fixada. O AG evolui por um número de gerações até satisfazer um critério de parada escolhido, podendo ser um número de gerações pré-fixado, a convergência da população ou a melhor solução satisfaça ao critério de aptidão que determina a solução ótima. Outros critérios podem ser definidos em função dos objetivos do problema em mãos.

5.8 Exemplos de Aplicação de Inteligência Computacional

Nesta seção, serão apresentados dois exemplos de aplicação utilizando redes neurais artificiais em processamento de imagens de satélite.

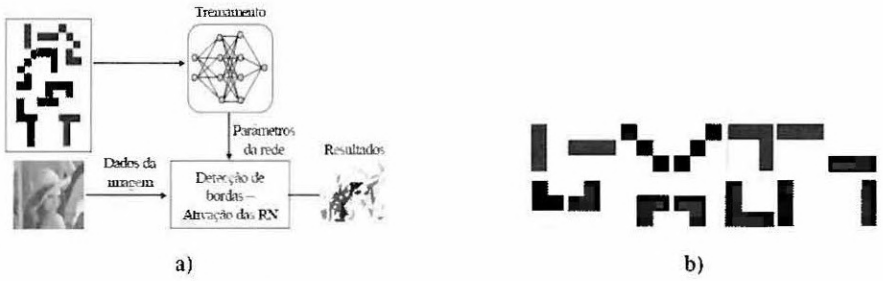
Aplicação 1 – Uso de redes neurais em detecção de bordas.

A detecção de bordas é uma técnica de segmentação de imagens que visa a localizar e realçar os pontos de uma imagem em que há transições significativas entre as intensidades luminosas de pixels (elementos de uma imagem) vizinhos. Dois problemas estão associados à detecção de bordas: a) a intensidade; e b) a orientação das bordas. Existem inúmeras técnicas reportadas na literatura concebidas para detectar bordas em imagens,

sendo a mais popular a concebida por Canny (1986), que utiliza o cálculo de gradientes unidirecionais para calcular as intensidades das bordas. O trabalho de Castro (2003) traz uma boa revisão sobre as técnicas de detecção de bordas.

O trabalho de Castro e Silva (2004) trata da aplicação de RNA supervisionadas para detectar bordas em imagens. Será descrita a estratégia adotada para treinar uma RNA do tipo Perceptron de Múltiplas Camadas para detectar bordas em imagens.

A idéia consiste em treinar a RNA com diferentes elementos de bordas (segmentos e cantos), associando a cada um a sua orientação, e depois submeter uma imagem para a RNA detectar as bordas. A Figura 5.17 ilustra a idéia do uso de uma RNA na detecção de bordas em imagens.



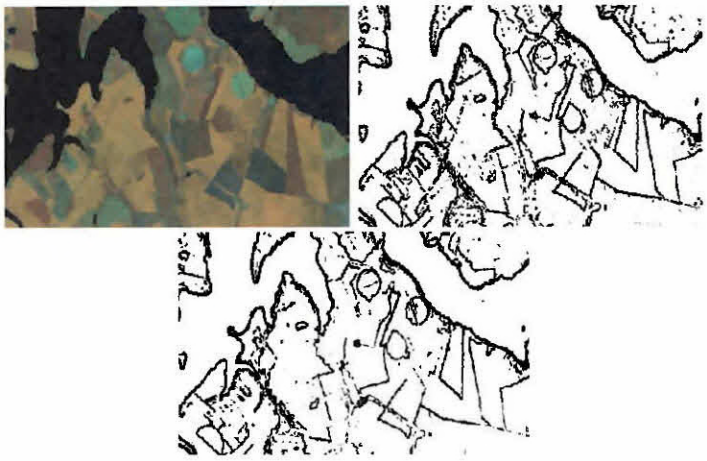
a) b)

Figura 5.17 – a) Idéia para treinamento e detecção de bordas em imagem usando uma RNA; b) Padrões de bordas utilizados para o treinamento da rede.

Os elementos de bordas são pequenas matrizes com dimensão 3x3 preenchidas com 0 e 1, segundo a forma do elemento de borda desejado. O processo de detecção é complementado com a pré-fixação de um limiar de variância local para a aplicação da rede. As matrizes 3x3 são linearizadas em vetores formados pela concatenação das linhas de cada matriz, gerando vetores com dimensão 1x9. Cada matriz é associada a um identificador relacionado com sua orientação (ou tipo de borda).

A idéia fundamental é a universalização do treinamento da rede, de forma que ele independa de informações da imagem.

A rede treinada foi submetida à detecção de bordas em diferentes tipos de imagens. A Figura 5.18 ilustra em (a) um trecho de uma imagem de satélite com área agrícola; em b) o resultado da detecção de bordas com o algoritmo de Canny; e em c) o resultado da detecção de bordas com a RNA treinada.



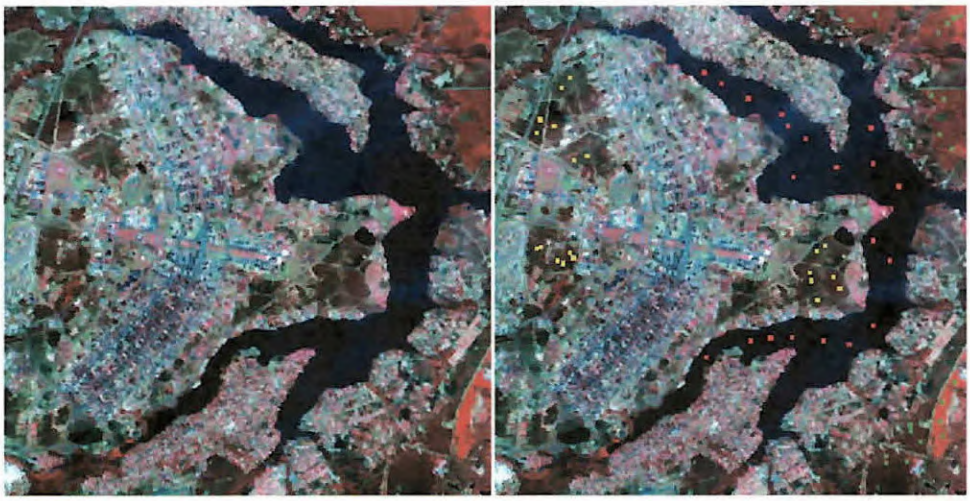
a) b) c)

Figura 5.18 – a) Trecho de uma imagem do satélite LANDSAT VII com área agrícola, cedido por Gleriani (2004); b) Imagem de bordas obtida com o operador de Canny; c) Imagem de bordas obtida com a RNA.

A análise visual dos resultados nas Figuras 5.18b e 5.18c mostra que ambos os operadores geraram imagens de bordas semelhantes. O operador de Canny detectou 33% dos pontos da imagem como pontos de bordas, e o operador neural, 28,5%. Observa-se que o resultado gerado pela RNA apresenta bordas mais finas e menos ruídos nas áreas internas em alguns contornos.

Aplicação 2 - Uso de RNA na classificação de imagens de satélite.

Na Figura 5.19a, está representada uma imagem do satélite LANDSAT V, composta por 512 x 512 pixels, obtida sobre a cidade de Brasília nos canais 4, 3 e 2, correspondentes às faixas espectrais do infravermelho, vermelho e verde. Neste exemplo, a idéia consiste em tomar amostras de áreas conhecidas e treinar uma rede neural para realizar a classificação da imagem.



a) b)

Figura 5.19 – a) Imagem LANDSAT V de Brasília (disponível em http://www.dgi.inpe.br/html/gal-1.htm); b) Amostras das classes consideradas. Quadrados em *Vermelho*: *Água*; *Verde*: *Vegetação densa*; *Amarelo*: *Solo exposto*.

Como o objetivo nesta descrição é acadêmico, mostrando uma forma de usar uma RNA para classificação de imagens, não se pretende fazer uma classificação exaustiva de todos os alvos na imagem. Assim, são consideradas três classes: água, vegetação densa e solo exposto. A Figura 5.19b ilustra os alvos de interesse com as respectivas amostras selecionadas pelo usuário. As amostras foram selecionadas com base em informações *ad hoc*, sem haver um trabalho de campo para comprovação da verdade terrestre. As rotulações estabelecidas baseiam-se nas informações de resposta espectral das faixas de freqüências dos sensores do satélite LANDSAT V. A representação das amostras está em cores: *Vermelho*: *Água*; *Verde*: *Vegetação densa*; *Amarelo*: *Solo exposto*.

De cada classe, foram tomadas 20 amostras. Cada amostra é composta pelos pontos dentro de uma janela com dimensão 3x3 em torno de um ponto selecionado visualmente pelo usuário. Uma vez selecionadas essas janelas, cada uma é caracterizada pelos seus valores de *desvio padrão* e *média* para cada uma das faixas espectrais. Assim, cada janela gera um vetor de atributos com 6 valores correspondendo a 3 pares de medidas de *desvio padrão* e *média*. Os conjuntos de amostras das classes foram divididos em dois subconjuntos: de treinamento (com 10 amostras) e de teste (com 10 amostras). Assim, as três classes somam 30 vetores para treinamento e 30 vetores para teste. A rede utilizada é um Perceptron de Múltiplas Camadas (Figura 5.11b) com 3 neurônios em uma camada escondida e um neurônio na saída. Cada vetor de entrada é associado à respectiva saída desejada para a rede, consistindo de um código que identifica a classe: 1 para *água*; 5 para *vegetação densa*; 9 para *solo exposto*.

Na fase de aprendizagem da rede, é utilizado o conjunto de treinamento com 30 vetores, sendo 10 vetores por cada classe. A rede é treinada até atingir um desempenho satisfatório. Para testar o poder de generalização do que a rede aprende no treinamento, utiliza-se o conjunto de teste, que também consiste de 30 vetores, sendo 10 vetores por classe.

A matriz na Tabela 5.1 abaixo mostra a confusão da rede na classificação das amostras de teste das classes.

Tabela 5.1 – Matriz de confusão da generalização da rede sobre conjunto de teste.

	C_1	C_2	C_3	
C_1	8	2	0	10
C_2	0	9	1	10
C_3	0	1	9	10
Total				30

Na Tabela 5.1, pode-se observar que 20% das amostras da classe 1 foram confundidas com as amostras da classe 2; 10% das amostras da classe 2 foram confundidas com as amostras da classe 3; e 10% das amostras da classe 3 foram confundidas com as amostras da classe 2. Ou seja, considerando as 3 classes, o índice de acerto da RNA é de 86%. Evidentemente, o desempenho geral deste classificador pode ser melhorado através da realização de um pré-processamento sobre as amostras, bem como utilizando meios mais criteriosos para extração das amostras.

Aplicação 3 – Uso de lógica nebulosa na navegação robótica autônoma.

Essa é uma aplicação inspirada no uso de robôs autônomos para a exploração em aplicações espaciais, em águas profundas ou em ambientes inóspitos. Essencialmente a idéia é desenvolver uma estratégia inteligente para dotar robôs (de baixo custo ou não) com o máximo de autonomia para navegação, para que eles sejam explorados nas aplicações. Assim, os princípios de controle aplicados a um caso podem ser estendidos para outros.

Um dos objetivos a se guiar um automóvel em uma rodovia é manter-se na pista de rolagem compatível com a velocidade de deslocamento. Essa é uma tarefa relativamente trivial para os humanos quando estão guiando, principalmente quando a rodovia é sinalizada horizontalmente com faixas que acentuam os limites da faixa de rolagem. A Figura 5.20 ilustra a idéia de deslocamento em uma faixa de rolagem delimitada por faixas brancas.

Da Figura 5.20, é intuitivo que o veículo seja conduzido em função da direção das faixas laterais. Quando guiando os humanos não fazem cálculos precisos para tomada de decisão sobre a direção a ser tomada. Ao invés disso, seu instinto reflexivo pode ser modelado como "vai para lá", "para direita", "um pouquinho para direita", "vem, vem, vem!", etc. Os trabalho de Castro et al (2001) e de Hoffmann et al (2004) usam esse conhecimento de senso comum para modelar um sistema de controle inteligente, baseado em lógica nebulosa e em redes neurais, para a navegação autônoma de um robô de pequeno porte, utilizando imagens da pista que são capturadas por uma câmera a bordo.

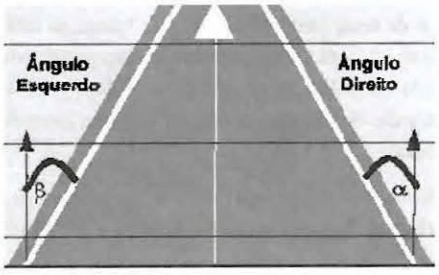
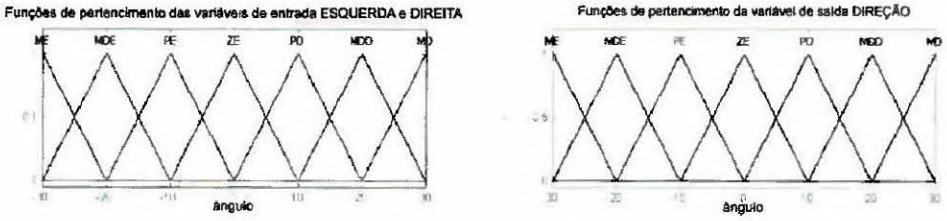


Figura 5.20 – Ilustração de situação de pista com faixas laterais (Fonte: Castro et al., 2001).

O sistema de software extrai a informação da direção das pistas (ângulos α e β na Figura 5.20) em diferentes regiões da imagem. A direção das faixas é determinada como uma direção de tendência (presença de incerteza). As faixas são detectadas por operadores de detecção de bordas como descritos nos trabalhos de Castro e Silva (2004) e de Hoffmann et al (2004). A informação de direção média é passada então por um processo de "fuzificação" para criar a informação com variáveis lingüísticas obtendo-se a informação da direção das faixas como "muito para esquerda", "pouco para esquerda", "muito para direita", "pouco para a direita", etc.

Essa informação é passada para o sistema de controle, com inferência nebulosa, gerar uma saída também como uma variável lingüística para o deslocamento do robô do tipo: "para frente", "pouco para esquerda", "muito para a direita", etc. Usando funções de pertencimento do tipo triangular, as definições nebulosas para a direção das faixas da direita e da esquerda estão na Figura 5.21a e para a direção de saída a ser tomada pelo robô na Figura 5.21b.



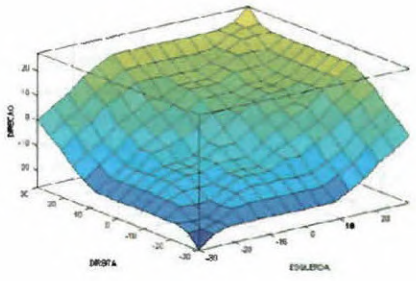
a) b)

Figura 5.21 – a) Definição nebulosa do ângulo das faixas da pista; b) Definição nebulosa da direção a ser tomada pelo robô. Adaptado de Hoffmann et al (2004).

A variável de saída é determinada utilizando o método de inferência de Mamdani (Seção 5). A Figura 5.22a exibe o conjunto de regras nebulosas para a determinação da direção, a partir da informação de direção das faixas, e a Figura 5.22b mostra a superfície de decisão para a direção.

DIREITA / ESQUERDA	ME	MDE	PE	ZE	PD	MDD	MD
ME	ME	MDE	MDE	MDE	MDE	PE	ZE
MDE	MDE	MDE	PE	PE	PE	ZE	PD
PE	MDE	PE	PE	PE	ZE	PD	MDD
ZE	MDE	PE	PE	ZE	PD	PD	MDD
PD	MDE	PE	ZE	PD	PD	PD	MDD
MDD	PE	ZE	PD	PD	PD	MDD	MDD
MD	ZE	PD	MDD	MDD	MDD	MDD	MD

a)



b)

Figura 5.22 – a) Conjunto de regras nebulosas para geração da direção a ser seguida pelo robô; b) Superfície de decisão utilizando a regra de Mamdani. Adaptado de Hoffmann (2004).

O sistema nebuloso apresentado foi utilizado sobre uma plataforma robótica com deslocamento através de controle remoto. As imagens fornecidas pela câmera a bordo eram transmitidas via radio para uma placa de captura que as digitalizavam e as submetiam aos operadores de extração da informação. Após a fuzificação das direções das faixas da pista, era gerada a nova direção a ser seguida pelo robô. A Figura 5.23a ilustra a

plataforma robótica utilizada nos experimentos conduzidos, a Figura 5.23b ilustra o ambiente artificial criado para os experimentos e a Figura 5.23c ilustra uma imagem obtida pelo robô com a informação de bordas das faixas extraídas.

a) b) c)

Figura 5.23 – a) Plataforma robótica utilizada; b) Ambiente artificial para realização dos experimentos; c) Exemplo de imagem obtida pela câmera a bordo do robô com a informação de bordas extraídas. Adaptada de Hoffmann et al (2004).

Referências

BITTENCOURT, G., Inteligência Artificial Ferramentas e Teorias, Ed. DAUFSC, 1998.

CANNY, J. *A Computational Approach to Edge Detection*, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 8, No. 6, Nov. 1986.

CASTRO, A. P. A., SILVA, J.D.S. Redes Neurais Supervisionadas para Detecção de Bordas. In: WORKSHOP DOS CURSOS DE COMPUTAÇÃO APLICADA DO INPE, 4. (WORCAP), 2004, São José dos Campos. **Anais...** São José dos Campos: INPE, 2004. CD-ROM, On-line. Disponível em: <http://urlib.net/lac.inpe.br/worcap/2004/09.25.18.11>. Acesso em: 06 jan. 2008.

CASTRO, A. P. A., SILVA, J. D. S., and SIMONI, P. O. Image based autonomous navigation fuzzy logic control. In *IJCNN International Joint Conference on Neural Networks*, volume 1, pg 2200–2205, Washington, 2001.

DAVIS, L. Handbook of genetic algorithms. New York: Van Nostrand Reinhold, 1991.

FAUSETT, L., Fundamentals of Neural Networks. Prentice Hall, 1994.

GLERIANI, J.M. Redes Neurais Artificiais para Classificação Espectro-Temporal de Culturas Agrícolas. Tese de Doutorado do Curso de Pós-Graduação em Sensoriamento Remoto, Instituto Nacional de Pesquisas Espaciais – INPE. 2004.

GOLDBERG, D. E. Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, 1989.

HAGAN, M.T., DEMUTH, H.B., BEALE, M., Neural Network Design. PWS Publishing Company, 1996.

HAYKIN, S., Neural Networks: A Comprehensive Foundation. 2a. Edição, MacMillan, 1999.

HECHT-NIELSEN, R., Neurocomputing. Addison Wesley Publ., 1990.

HEITKOETTER, J., BEASLEY, D. The hitch-hiker's guide to evolutionary computation: a list of frequently asked questions (FAQ). [online] <rtfm.mit.edu:/pub/usenet/news.answers/ai-faq/genetic>. Mar. 1999.

HOFFMANN, L. T. ; CASTRO, A. P. A. ; SILVA, J. D. S. . Controle inteligente de um robô móvel utilizando imagens. In: XXIV Congresso da Sociedade Brasileira de Computação - Encontro de Robótica Inteligente, 2004, Salvador. Anais do SBC 2004, 2004. v. 1. p. 1832-1841.

HOLLAND, J. Adaptation in Natural and Artificial Systems, Ann Arbor, MI: The University of Michigan Press, 1975.

LACERDA, E.G.M., CARVALHO, A.C.P.L.F. Introdução aos algoritmos genéticos. In: SBC'99 - Congresso Nacional da Sociedade Brasileira de Computação 19., Rio de Janeiro, Jul. 1999, **Anais.** v. 2, p. 51–126.

LIN, Chin-Tseng, LEE, C.S.G., Neural Fuzzy Systems: A Neural-Fuzzy Synergism to Intelligent Systems. Prentice Hall, 1996.

MICHALEWICZ, Z. Genetic algorithms + Data structures = Evolution programs, Springer, 1996

PARKER, D. B. Optimal algorithms for adaptive networks: second order back propagation, second order direct propagation, and second order hebbian learning. *Proceedings of the IEEE Conferenceon Neural Networks*, 2:593–600, 1987.

RICH, E., Artificial intelligence. New York, NY, McGraw, 1983.

RUSSELL, S., NORVIG, P., Artificial Intelligence A Modern Approach. Prentice Hall, 1995.

RUMELHART, D., HINTON, G., WILLIAMS, R.. Learning internal representations by error propagation, In: RUMELHART, D. MCCLELLAND, J., Parallel Distributed Processing: Explorations in the microstructure of cognition. Volume 1: Foundations, 318-363. MIT Press, Cambridge, MA, 1986

SIMPSON, P.K., Artificial neural systems. Pergamon Press. 1990.

TSOUKALAS, L.H., UHRIG, R.E., Fuzzy and Neural Approaches in Engineering. John Wiley & Sons, 1997.

WATERMAN, D.A., A guide to expert systems. Reading, MA, Addison, 1986.

WERBOS, P. J. Generalization of back propagation with applications to a recurrent gas market model. *Neural Networks*, 1:339-356, 1988.