**Priscila Silva Martins**

# Aprendizado de Máquina para Otimização de Parâmetros em Sistemas Baseados em Conhecimento.

**FLORIANÓPOLIS**
**2003**

**UNIVERSIDADE FEDERAL DE SANTA CATARINA**
**CURSO DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA**

# Aprendizado de Máquina para Otimização de Parâmetros em Sistemas Baseados em Conhecimento.

Dissertação submetida à
Universidade Federal de Santa Catarina
como parte dos requisitos para a
obtenção do grau de Mestre em Engenharia Elétrica.

**Priscila Silva Martins**

Florianópolis, Junho/2003.

**Aprendizado de Máquina para Otimização de Parâmetros em Sistemas Baseados em Conhecimento.**

Priscila Silva Martins

'Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Engenharia Elétrica, Área de Concentração em *Controle, Automação e Informática Industrial*, e aprovada em sua forma final pelo Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Santa Catarina.'

Guilherme Bittencourt, Dr. Rer. Nat.
Orientador

Edson Roberto De Pieri, Dr.
Coordenador do Programa de Pós-Graduação em Engenharia Elétrica

Banca Examinadora:

Guilherme Bittencourt. Dr. Rer. Nat.
Presidente

Eduardo Camponogara, Dr.

Geber L. Ramalho, Dr.

Marcelo Ricardo Stemmer, Dr. Rer. Nat.

*À Walburgues, Margarete*
*Gisele, Adir*
*Lucas e Matheus.*

Preciso de Alguém

Que me olhe nos olhos quando falo
Que ouça minhas tristezas e neuroses com paciência
E, ainda que não compreenda, respeite meus sentimentos.
Preciso de alguém, que venha brigar ao meu lado sem precisar ser
convocado; alguém Amigo o suficiente para dizer-me as verdades que
não quero ouvir, mesmo sabendo que posso odiá-lo por isso.
Nesse mundo de céticos, preciso de alguém que creia, nessa coisa misteriosa,
desacreditada, quase impossível: A Amizade.
Que teime em ser leal, simples e justo, que não vá embora se um dia
eu perder o meu ouro e não for mais a sensação da festa.
Preciso de um amigo que receba com gratidão o meu auxílio, a minha
mão estendida.
Mesmo que isso seja muito pouco para as suas necessidades.
Preciso de um Amigo que também seja companheiro, nas farras e pescarias,
nas guerras e alegrias, e que no meio da tempestade grite em coro comigo:
"Nós ainda vamos rir muito disso tudo"e ria.
Pois com uma amizade verdadeira a vida se torna mais simples, mais rica, mais bela ...

Charles Chapling

# AGRADECIMENTOS

Deus obrigada, meu Senhor. Por ter me dado força quando não mais tinha vontade de continuar, obrigada por sempre estar ao meu lado e por me guiar durante toda a vida. E obrigada por ter me presenteado com uma família tão maravilhosa.

Pai, Mãe, Gisele obrigada por vocês serem tão maravilhosos. Obrigada por me amarem tanto, e sempre terem me dado incentivo para buscar meus sonhos. Não haverá papel suficiente no mundo para que eu possa agradecer à tudo que vocês fizeram por mim. Obrigada pelas palavras de apoio, pelo ombro amigo, por estarem ao meu lado em cada tombo, e por me ensinarem que nunca devo desistir daquilo que eu quero, e que sou capaz de triunfar em cada empreitada. Amo muito vocês!!

Obrigada ao meu orientador Guilherme Bittencourt, pela oportunidade, e por nunca ter desistido do meu projeto, nas inúmeras adversidades, e por sempre ter acendido uma lanterna, nas vezes que cheguei em sua sala sem rumo. Obrigada a equipe do UFSC-Team, Augusto Loureiro, Eder Gonçalves e Luciano Rottava. Um agradecimento especial ao que se tornou o meu *co-co-orientador*, Eder valeu pela paciência e pela ajuda.

Já diz no livro de Eclesiastes: Um amigo fiel é um abrigo seguro; quem o achou descobriu um tesouro. A vocês meus amigos de longa data, não poderia faltar o meu agradecimento, gostaria de poder citar as inúmeras vezes que vocês me ampararam, apoiaram, foram pai e mãe para mim. A distância que separou algum de nós não diminui em nada a nossa amizade. Flávio, Ivana, Vilma, Eric, Raquel, Gyslene, Ricardo, Léo, Graça só posso dizer uma coisa Eu amo vocês, obrigada por tudo.

Uma grande conquista que veio junto a esse meu título de mestre, muito mais importante que ele próprio, são os amigos que adquiri durante esta jornada. Vocês foram muito importantes também nesta conquista. Não sei se preciso citar nomes, pois sempre fiz questão de sempre dizer pessoalmente o quanto eu adoro vocês. Mas para ninguém reclamar... Ana, Pati, Cris, Ju, Cássia, Michelle,Carminha, Tércio, Leandro, Ricardo, Fábios (Pinga, Favarim e Baiano), Emerson, Alysson, Paquito, Carlos, Sérgio obrigada amigos, por ótimos momentos nesta empreitada em Florianópolis.

Agradeço ao CNPq , Conselho Nacional de Desenvolvimento Científico e Tecnológico, pelo suporte financeiro, apoio que é fundamental para a comunidade acadêmica.

Se alguém foi esquecido de ser citado neste agradecimento, não foi de propósito. À todos aqueles que estiveram envolvidos nesta jornada: OBRIGADA!

Resumo da Dissertação apresentada à UFSC como parte dos requisitos necessários para obtenção do grau de Mestre em Engenharia Elétrica.

# Aprendizado de Máquina para Otimização de Parâmetros em Sistemas Baseados em Conhecimento.

**Priscila Silva Martins**

Junho/2003

Orientador: Guilherme Bittencourt
Área de Concentração: Controle, Automação e Informática Industrial
Palavras-chave: Inteligência Artificial, Sistemas Multiagentes, Aprendizado de Máquina
Número de Páginas: xii + 65

A RoboCup surgiu em 1995 como um novo desafio para os pesquisadores das áreas de Inteligência Artificial e Robótica Inteligente, com o objetivo de explorar novas teorias e arquiteturas como: princípios de projeto de agentes autônomos, colaboração multiagente, aquisiação estratégica de conhecimento, raciocínio em tempo real e robótica. Em particular, a liga de robôs simulados da RoboCup oferece um ambiente desafiador para Sistemas Multiagentes. Devido à complexidade do problem é necessário, durante o desenvolvimento de um time, adquirir conhecimento estratégico. Uma das formas de aquisição de conhecimento estratégico é utlizar o Aprendizado de Máquina (AM). O AM, possibilita ao agente aprender ações durante uma simulação de jogo, por exemplo seu posicionamento em campo, o momento certo de passar a bola, ou ainda como driblar um oponente, entre outras. Na primeira versão do UFSC-Team, as variáveis responsáveis pela geração das metas do agente e por concluir em qual situação encontra-se o jogo eram determinadas empiricamente, tendo seus valores baseados apenas na observação. Desta forma por não serem ideais, estes valores geravam ocasionalmente, inferências erradas sobre o estado do jogo. O uso de aquisição de conhecimento sobre estas variáveis possibilita a melhora na atribuição de seus valores. Este trabalho propõem-se a aplicar técnicas de aprendizado por reforço, para atribuir valores mais próximos aos valores ideais, gerando assim uma otimização dos agentes do UFSC-Team e consequentemente uma redução dos erros de inferência.

Abstract of Dissertation presented to UFSC as a partial fulfillment of the requirements for the degree of Master in Electrical Engineering.

# Machine Learning for Parameters Otimization in Based in Knowledge Systems.

**Priscila Silva Martins**

June/2003

Advisor: Guilherme Bittencourt
Area of Concentration: Control, Automation and Industrial Computing
Key words: Artificial INtelligence, Learning Machine, Reinforcement Learning
Number of Pages: xii + 65

The RoboCup appeared in 1995 as a new challenge for the researchers of the Artificial Intelligence and Intelligent Robotic areas. The objective is to explore new theories and architectures as: principles of independent agents project, multiagent co-operation, strategic knowledge acquisition, reasoning in real time and robotic. In particular, the RoboCup league of simulated robots offers a challenging environment for Multiagents Systems. Because of the problem complexity, strategic knowledge is required during the team development. One of the forms of strategic knowledge acquisition is the use the Machine Learning (ML). The ML, enables the agent learning during a game simulation; the robot positioning in the field, the precise moment to pass the ball or how to dribble an opponent, among others, are examples of the actions to be learned. In the first version of UFSC-Team, the variable responsible for the generation of agent goals and to infer the matches status was determined in an empirical manner, having its values established only based in observation. Then, for not being ideals, the values occasionally generated erroneous inferences about the match state. The use of knowledge acquisition on these variables makes possible the improvement in the determination of its values. This work proposes the application of techniques of reinforcement learning to attribute values closer to the ideal, thus generating an optimization of the UFSC-Team's agents and consequently the inference errors reduction.

# Sumário

# Lista de Figuras

# Lista de Tabelas

# Capítulo 1

# Introdução

O objetivo central da IA inclui a criação de teorias e modelos para a capacidade cognitiva, e também a construção de sistemas computacionais baseados nestes modelos. Uma forma encontrada para impulsionar a pesquisa em IA é a proposta de problemas padrão, que permitem que diferentes soluções e enfoques sejam comparados.

Com o intuito de promover avanços nas pesquisas em IA os pesquisadores Kitano, Asada e Kuniyoshi (Kitano et al., 1997) lançaram como um novo desafio uma partida de futebol, desta proposta surgiu a RoboCup. A RoboCup é uma proposta bastante ambiciosa, pois engloba várias tecnologias e áreas de pesquisas distintas, tais como: aprendizado de máquina, sistemas especialistas, algoritmos genéticos, lógica nebulosa, robótica, sistemas de tempo-real e controle de motores. A meta principal da RoboCup é bastante ambiciosa:.. *em 2050 ter uma equipe de robôs humanóides em condições de vencer a equipe campeã mundial de futebol*.

Paralelamente à RoboCup outra área que teve seu desenvolvimento influenciado por este desafio foi a área de Aprendizado de Máquina (AM). A metodologia de AM tem se desenvolvido de acordo com os principais interesses das pesquisas de IA. Em resposta às dificuldades de codificar volumes de conhecimento sempre crescentes em modernos sistemas de IA, muitos pesquisadores têm voltado sua atenção para AM como meio de vencer o gargalo da aquisição de conhecimento. Um sistema de aprendizado é um programa de computador que toma decisões baseado em experiências acumuladas por meio de soluções bem-sucedidas de problemas anteriores. O aprendizado de máquina tem sido bastante explorado no âmbito da RoboCup como forma de aquisição de conhecimento.

Atualmente o DAS possui uma implementação de um time de futebol de robôs para a categoria de robôs simulados, o *UFSC-Team*, onde cada sistema que implementa um dos jogadores deste time é baseado em um modelo de agente autônomo concorrente proposto em

(Bittencourt e da Costa, 2001). De acordo com este modelo cada um dos agentes apresenta um sistema decisório dividido em três níveis: reativo, instintivo e cognitivo. Onde cada um destes níveis, juntamente com o nível inferior pretende modelar um agente completo e a complexidade do comportamento do agente cresce a cada nível decisório.

Como o *UFSC-Team* trata-se de um laboratório para a realizações de experimentos na área de IAD, foi realizado um estudo sobre as técnicas de aprendizado mais utilizadas recentemente para a implementação de uma estrutura que fosse capaz de atuar na aquisição de conhecimento para os agentes do time. As técnicas estudadas foram: Árvores de Decisão, Redes Neuronais e Aprendizado por Reforço. Este último foi escolhido por se adequar melhor ao ambiente enfrentado em uma partida de futebol. Após o estudo mais apronfundado deste método foi efetuada a implementação de uma estrutura para aquisição de conhecimento através do uso de aprendizado de máquina dentro do UFSC-Team, utilizando o método de aprendizado por reforço. Esta estrutura permitirá o desenvolvimento de vários treinamentos necessários ao UFSC-Team, entre eles podemos citar: aprender a chutar ao gol, a driblar o adversário, a conduzir a bola, entre tantos outros movimentos realizados em uma partida de futebol.

Essa estrutura consiste no acréscimo de um processo chamado *RL* dentro da arquitetura existente do UFSC-Team. Este processo RL será responsável pelos aprendizados que serão implementados no UFSC-Team, ele provê uma troca de mensagens com os outros processos, pois as modificações necessárias foram adicionadas à biblioteca Expert-Coop++, a qual já era utilizada na implementação do UFSC-Team.

O estudo de caso apresentado neste trabalho foi realizado em relação a localização de um agente jogador do time UFSC-Team dentro do campo. Neste experimento o processo RL, recebe mensagens do nível instintivo e avalia o resultado do sistema especialista deste nível. Se a inferência da posição for incorreta, o processo RL irá alterar a base de regras do SE do nível instintivo, tentando corrigir o erro. Este processo de `inferência-avaliação-alteração` é realizado até atingir um nível de acerto aceitável. Embora este experimento seja de certa forma um aprendizado de baixa complexidade, ele pode melhorar o desempenho do time. E principalmente comprova a eficácia da estrutura de aprendizado montada.

Esta dissertação está dividida da seguinte maneira: o capítulo 2 apresenta um breve histórico da Inteligência Artificial e uma discussão sobre Resolução Distribuída de Problemas e Sistemas Multiagentes. O capítulo 3 descreve o histórico da RoboCup como um novo problema padrão para a Inteligência Artificial, a liga de robôs simulados e o simulador **Soccerserver**, o qual foi utilizado como base para a realização deste trabalho. No capítulo 4 estão descritos os níveis que compõe a arquitetura do agente UFSC-Team.

O capítulo 5 apresenta a área de Aprendizado de Máquina. Na seção 5.2 está descrito

o método do Aprendizado por Reforço, o método de aprendizado o qual foi aplicado neste trabalho. Os outros métodos estudados estão nos Anexos A e B.

O capítulo 6 descreve a estrutura de aprendizado implementada para a otimização dos parâmetros do nível instintivo do UFSC-Team, e também apresenta uma análise dos resultados obtidos com a implantação desta estrutura. Por fim o capítulo 7 traz as conclusões do trabalho e os trabalhos futuros.

# Capítulo 2

# Inteligência Artificial Distribuída

## 2.1 Introdução

A Inteligência Artificial (IA) nasceu em 1956, com uma conferência de verão no *Dartmouth College*, NH, USA. Os organizadores desta conferência foram John McCarthy, Marvin Minsky, Nathaniel Rochester e Claude Shannon. A proposta desta conferência teve a seguinte intenção: " ... dez homens, realizar um estudo durante dois meses, sobre o tópico *inteligência artificial*". Desde seus primórdios, a IA gerou polêmica, a começar pelo seu próprio nome, considerado presunçoso por alguns, até a definição de seus objetivos e metodologias. O desconhecimento dos princípios que fundamentam a inteligência, por um lado, e dos limites práticos da capacidade de processamento dos computadores, por outro, levaram periodicamente à promessas exageradas e à correspondentes decepções.

A Inteligência Artificial Distribuída (IAD) emergiu da integração entre as áreas de Inteligência Artificial (IA) e Sistemas Distribuídos (SD). Esta integração delineou a IAD não como uma sub-área da IA, mas sim como um ramo da Computação com características próprias e distintas. A IAD se diferencia da área de SD pois não enfoca questões relacionadas ao processamento distribuído, objetivando aumentar a eficiência da computação propriamente dita, mas procura desenvolver técnicas de cooperação entre entidades envolvidas em um sistema. A IAD se diferencia da IA na medida em que traz novas e mais abrangentes perspectivas sobre a representação do conhecimento, planejamento, resolução de problemas, coordenação, etc.

Pesquisas em IAD são freqüentemente em duas áreas: Resolução Distribuída de Problemas (RDP) e Sistema Multiagentes (SMA), em ambos os casos a teoria da IAD permite resolver problemas de forma distribuída e cooperativa, utilizando processos chamados de **agentes**.

## 2.2 Resolução Distribuída de Problemas

A RDP relaciona múltiplos agentes para resolver um problema específico, de maneira coerente e robusta. A partir do problema são especificados os agentes necessários para compor o ambiente e gerar a solução. Seu objetivo global é desenvolver ténicas de raciocínio e representação de conhecimento necessárias para que nodos, contendo solucionadores de problemas interligados em uma rede fracamente acoplada, cooperem efetivamente para solucionar um problema distribuído complexo (Bittencourt, 2001). Por outro lado, SMAs preocupam-se com o agente, suas propriedades internas e seu comportamento no ambiente. Assim, pode-se dizer que a diferença básica entre RDP e SMA está na generalidade do ambiente. Em RDP o ambiente é construído tendo-se um problema em mente, enquanto em SMA o mesmo ambiente pode servir de base para resolução de uma gama maior de problemas.

Alguns pontos importantes da abordagem RDP:

- apesar de trabalharem cooperativamente, não há necessidade dos agentes representarem explicitamente quais são suas habilidades e metas. Isto é implicitamente representado pelo projetista;
- a descrição e decomposição das tarefas na maioria dos casos é totalmente decidida pelo projetista. Isto é um caso extremo, mas mesmo havendo alguma decomposição dinâmica de tarefas, os métodos usados são fortemente dependentes do domínio da aplicação;
- na maioria dos casos, se a tarefa for dividida corretamente pelo projetista não ocorrerá nenhum conflito. Uma vez que, este é um caso crítico, mas se ocorrer algum conflito, ele é fortemente dependente do domínio da aplicação;
- mesmo que os agentes possam se comunicar, não há necessidade de uma conversação complexa para alcançar a meta;
- novos agentes não podem ser inseridos dinamicamente na sociedade. Isto significa que este tipo de sistema, não pode ser considerado um sistema aberto (Sichman et al., 1992).

A abordagem RDP pode ser representada da seguinte forma:

**(problema a ser resolvido)** $\Rightarrow$ **(projeto dos agentes)** $\Rightarrow$ **(problema sendo resolvido)** $\Rightarrow$ **(solução)**

## 2.3 Sistema Multiagentes

Não há ainda um concenso sobre a definição de *agente* na comunidade de IAD. Esta falta de concordância leva a definições contextuais, adequadas aos objetivos da aplicação ou ao ponto de vista de cada pesquisador. Por exemplo, Etzioni e Weld em (Etzioni e Weld, 1995) definem um agente como *"...um programa de computador que se comporta de forma análoga a um agente humano, tal como um agente de viagem ou um agente de seguro"*, enquanto Russel e Norvig (Russell e Norvig, 1996) definem o agente como *"...qualquer coisa que pode perceber seu ambiente através de sensores e agir neste ambiente através de atuadores"*.

Em SMA a preocupação inicial é com o comportamento e a interação de um grupo de agentes. A literatura para modelos de agentes é rica, mas há duas abordagens principais: **agentes reativos e agentes cogntivos**.

Os agentes reativos não possuem raciocínio simbólico complexo, estruturas de memória e nenhuma representação interna explícita do conhecimento. Com estas restrições um agente reativo somente percebe o ambiente externo e, baseado nos estímulos do ambiente, reage de uma forma pré-determinada pelo programador. Em uma sociedade de agentes reativos o comportamento inteligente do sistema advém da interação dos comportamentos básicos de cada agente. Este tipo de sociedade está baseado em modelos de organizações biológicas e etológicas, por exemplo, uma sociedade de formigas. Geralmente este tipo de sociedade dispõem de um alto número de agentes, centenas ou até milhares. Existem diversos trabalhos na literatura sobre SMA baseados em agentes reativos, por exemplo, a arquitetura de subsunção de Brooks (Brooks, 1986), e a primeira geração de sistemas especialistas (Marietto, 2000).

Os agentes cognitivos possuem uma representação explícita do ambiente e dos outros agentes, dispõem de memória, por isso, são capazes de planejar suas ações futuras, e um sistema desenvolvido de cooperação e comunicação. Uma sociedade de agentes cognitivos é formada por um pequeno número de indivíduos, geralmente no máximo de duas ou três dezenas (Marietto, 2000). Normalmente este tipo de sociedade utiliza da metáfora de grupos sociais humanos para a sua estruturação, onde times de especialistas podem solucionar problemas de forma cooperativa. Pontos importantes da abordagem SMA:

- a decomposição das tarefas é feita pelos agentes, e não pelo projetista. No máximo, pode haver uma reorganização dinâmica, isto é, os agentes podem decidir o que eles podem mudar no seu comportamento a fim de melhor realizar suas tarefas;
- os agentes são autônomos, isto é, eles podem ter suas próprias metas locais. Portanto, conflitos usualmente podem aparecer, devido a existência de metas locais e globais.

Além disso uma comunicação complexa deve ser organizada para estabelecer o papel de cada agente nas atividades para solucionar o problema;

- um agente pode entrar ou deixar a sociedade quando necessário. Se um novo agente é inserido na sociedade, os demais agentes irão incorporar em sua base de conhecimento suas capacidades e metas. Isto é feito, para manter uma representação explícita das metas e capacidades de todos os agentes.
- se o ambiente mudar os agentes devem incorporar estas mudanças em seus modelos de ambiente interno. Robôs móveis que navegam em lugares desconhecidos e diferentes, são um exemplo de agentes que têm que interagir com um mundo dinâmico (Sichman et al., 1992).

Em SMA os agentes coexistem em um ambiente comum, e há uma colaboração mútua para que a meta seja atingida. Esquematicamente:

**(agentes)** $\Rightarrow$ **(problema a ser resolvido)** $\Rightarrow$ **(problema sendo resolvido)** $\Rightarrow$ **(solução)**

Alguns pesquisadores têm discutido uma abordagem híbrida, isto é, a construção de um agente que não seja completmente cognitivo, nem completamente reativo. Esta abordagem seria composta por um agente com dois ou mais sub-sistemas: um cognitivo, o qual teria a representação simbólica do ambiente, que iria desenvolver planos e tomar decisões, e um reativo, que seria capaz de reagir aos eventos que ocorrem no ambiente sem possuir um raciocínio complexo.

## 2.4 Modelo Genérico

Em (Bittencourt e da Costa, 2001), uma arquitetura híbrida (ilustrada na Figura 2.1) foi apresentada. Este modelo genérico apresenta três níveis: reativo, instintivo e cognitivo, funcionalmente estes três níveis são similares aos três componentes da arquitetura adotada no agente de Sloman (Sloman, 1999). O modelo é duplamente comprometido com a abordagem evolucionária: o nível reativo é baseado em um mecanismo evolucionário, e o modelo é projetado de tal maneira que os componentes em cada nível evoluam um após o outro. A seguir será apresentado um breve sumário sobre cada nível.

O **nível reativo** tem por objetivo modelar um animal simples, como um inseto. Ele consiste em um ambiente evolucionário, cujo os elementos são: *padrões*, extraídos de uma

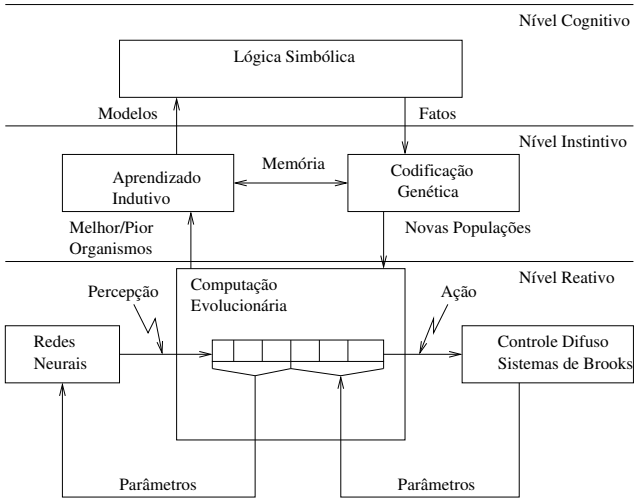


Figura 2.1: Modelo Genérico para agentes cognitivos.

informação sobre um ambiente externo, *controle causador* que produz ações no mesmo ambiente externo e um população de *agentes reativos* que unem percepção e ação. Este nível é caracterizado por uma alta atividade paralela que resulta em um ciclo rápido de percepção/ação. Ao final de cada ciclo, os melhores agentes na comunidade, de acordo com a *função de avaliação* (do inglês *fitness function*) estão aptos a *agir*.

O **nível instintivo** introduz uma *memória de longo termo* (do inglês *long term memory*) no modelo. Através do nível reativo é possível identificar as populações no ambiente que são responsáveis por uma melhor atuação em uma dada situação. Se dessas populações somente o melhor e o pior agentes forem selecionados, de acordo com uma função de avaliação (do inglês "*fitness*") é possível extrair suas propriedades, obtendo-se desta forma uma descrição geral de uma dada população, um tipo de reserva genética. Esta memória é composta essencialmente por estas descrições e o ato de "memorização" corresponde à introdução de uma nova população no nível reativo, da qual os agentes são codificados geneticamente de acordo com essas descrições gerais, geradas no nível reativo. O efeito a longo prazo desta memória é análogo a "criação" e "domesticação" da população de agente reativos. O nível instintivo é caracterizado por uma atividade de um ciclo mais longo que necessita algumas repetições da mesma situação para ser completa. O nível instintivo junto com o reativo têm a intenção de modelar animais mais complexos, como os mamíferos.

O **nível cognitivo** preocupa-se com a manipulação das descrições geradas no nível instintivo. Este nível é baseado em duas atividades complementares: *aprendizado das descrições das situações relevantes* e *geração de novas estratégias de ação*. A principal vantagem do nível cognitivo é o fato dele permitir o estabelecimento de um segundo canal de comunicação entre o ambiente e o agente: linguagem. Esta linguagem pode ser usada na descrição

de situações abstratas, pode permitir conceituar uma situação social e reconstruir as relações sociais relevantes em uma teoria interna coerente que possa apropriadamente ser simulada no nível reativo.

# Capítulo 3

# Robocup

## 3.1 Introdução

A idéia de utilizar uma partida de futebol de robôs como um problema de pesquisa foi introduzida por Mackworth mas esta idéia não foi a frente, não antes de ser readaptada por Kitano, Asada e Kuniyoshi (Kitano et al., 1997). Em 1995, estes pesquisadores lançaram um novo desafio para os pesquisadores das áreas de Inteligência Artificial e Robótica Inteligente: uma partida de futebol entre robôs autônomos. Desse desafio surgiram a RoboCup (originalmente chamada *Robot Worl Cup Initiative*) uma competição mundial entre times de robôs e a *RoboCup Federation* uma entidade internacional responsável pela organização do campeonato e pela manutenção de um fórum de discussão sobre os desafios e avanços científicos envolvidos com o tema.

A RoboCup difere de todas as pesquisas anteriores no campo de IA, pois foca-se em uma solução distribuída ao invés de uma centralizada, e também por não ser apenas um desafio para pesquisadores de IA, mas também de robótica, sociologia, tempo-real (Chen et al., 2001).

A principal meta da RoboCup é avançar por completo o nível tecnológico da sociedade, e como meta final foi proposto o seguinte:

> *Pelos meados do século 21, uma equipe de jogadores humanóides inteiramente autônoma deverá ganhar uma partida, de acordo com as regras oficiais da FIFA*[1]*, contra a equipe vencedora da Copa do Mundo mais recente (Rob, 2002).*

[1] *Federação Internacional das Associações de Futebol define as regras do futebol (fif, 2002)*

## 3.2 A RoboCup

A proposta de uma partida de futebol como problema não surgiu por acaso. A necessidade era de uma idéia que representasse um grande apelo à comunidade científica, e ao mesmo tempo que também se mostrasse como um extraordinário desafio, uma partida de futebol entre robôs atende a todos estes requisitos.

Utilizando a RoboCup como um problema padrão várias teorias, algoritmos, e arquiteturas podem ser avaliadas, tais como: princípios de projeto de agentes autônomos, colaboração multiagente, aquisição estratégica de conhecimento, raciocínio em tempo real, robótica, comportamento reativo, reconhecimento de contexto, visão e controle de motores.

Para atender a todos os tipos de tecnologias, e visando atingir os mais variados tipos de arquiteturas de robôs de uma forma justa e transparente, a RoboCup dividiu a competição em algumas categorias, baseadas no tamanho dos robôs e em seus métodos de movimento. São elas:

1. **RoboCupSoccer**
   - Liga de robôs simulados;
   - Liga de robôs pequenos com cinco robôs por equipe (F-180);
   - Liga de robôs pequenos com onze robôs por equipe (F-180);
   - Liga de robôs de tamanho médio (F-2000);
   - Liga de robôs Sony com pernas;
   - Liga de humanóides.
2. **RoboCupRescue**
   RoboCupRescue[2] é um novo domínio de pesquisa da RoboCup, cujos os alvos são: procurar e salvar pessoas em desastres de larga escala, como o terremoto que abateu Kobe, matando 5.000 pessoas. Este novo domínio foi escolhido por tratar-se de um problema social substancial, e possuir características em comum com uma partida de futebol. Assim como na RoboCup, a RoboCupRescue possui diferentes categorias, são elas:
   - Liga Simulada;
   - Liga de Robôs.

   O simulador para a RoboCupRescue ainda está em fase de desenvolvimento.

[2] http://www.robocup.org/games/36.html

3. **RoboCupJunior**
   A RoboCupJunior[3] é uma iniciativa de um projeto educacional orientado que patrocina eventos locias, regionais e internacionais com o intuito de introduzir a RoboCup para estudantes de escolas primárias e secundárias. O foco da liga júnior é puramente educacional. Para a RoboCupJunior três desafios foram desenvolvidos:

   - Futebol;
   - Dança;
   - Resgate.

## 3.3 Liga de robôs simulados

A competição simulada da RoboCup é um rico laboratório para a utilização de SMA. Ela é composta por um ambiente onde dois times jogam futebol, com todos os desafios que uma partida real oferece. Cada jogador (agente) tem seu próprio sistema de visão, percepção e ação. A comunicação dos agentes é permitida somente através de uma largura de banda estreita.

### 3.3.1 Soccerserver

Na liga simulada, dois times controlados por programas competidores jogam em um campo de futebol virtual simulado por um computador (Noda, 1995). A liga simulada está baseada em um simulador chamado **soccerserver**. O **soccerserver** foi desenvolvido para realizar disputas entre diversos agentes autônomos (independente da linguagem adotada) dispostos em um ambiente multiagente e de tempo-real. As vantagens deste simulador, está na abstração que é realizada, a qual livra os pesquisadores de terem que se preocupar com os problemas da robótica, tais como: reconhecimento de objetos, comunicação e a fabricação do hardware. Esta abstração permite que os pesquisadores se concentrem nas pesquisas de cooperação e aprendizado.

O **soccerserver** consiste em dois programas: **soccerserver** e o **soccermonitor**. A partida é disputada em um estilo cliente/servidor. Onde cada cliente representa um jogador em campo. O **soccerserver** é o programa que permite os movimentos dos jogadores e da bola, em uma partida. A comunicação entre os clientes, que é realizada via *socket* UDP/IP, e o controle do jogo também são assistidos pelo **soccerserver**.

[3] http://www.robocup.org/junior/index.html

O **soccermonitor** é uma ferramenta que permite a visualização do que acontece no servidor durante o jogo. As informações fornecidas pelo **soccermonitor** são: o placar, o nome dos times, a posição de todos os jogadores e a bola, a Figura 3.1 ilustra o que é mostrado pelo **soccermonitor**.

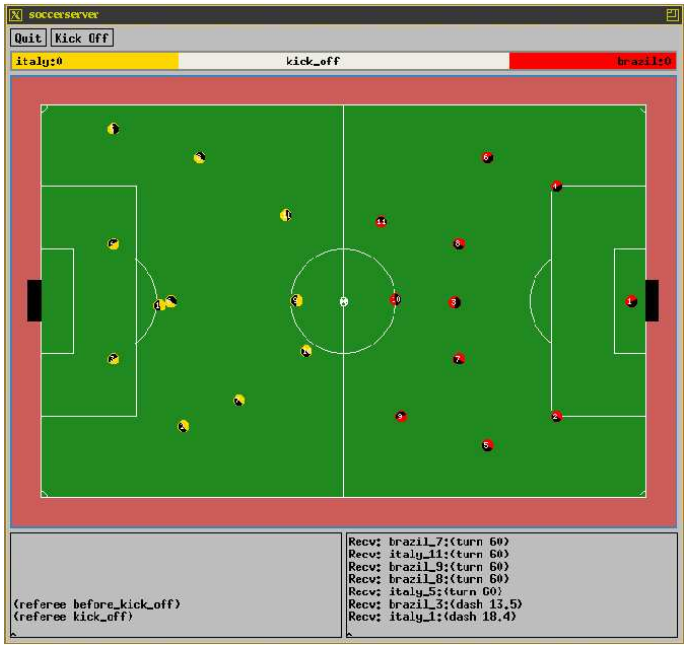


Figura 3.1: Modelo do Campo (Chen et al., 2001).

### 3.3.2 Como funciona o Soccerserver

Um cliente conecta-se com o **soccerserver** por um *socket* UDP. Através deste *socket*, o cliente enviará os comandos para controlar o jogador do respectivo cliente e receberá informações captadas dos sensores do mesmo, como mostrado na Figura 3.2. Ou seja, um programa cliente funciona como o cérebro do jogador: o cliente recebe informações dos sensores auditivos e visuais do servidor, e envia comandos de controle após o processamento destes. A Figura 3.2 apresenta a arquitetura do Soccer Server.

Uma exigência da comissão que regulamenta a RoboCup é que toda a comunicação entre clientes deve ser feita via **soccerserver**. Este compromisso dá-se no intuito de avaliar sistemas multiagentes, em que comunicação é um dos critérios.

Uma partida organizada pelo **soccerserver** é realizada obedecendo aos seguintes passos:

1. Cada cliente de cada time conecta-se com o servidor por meio do comando `init`.

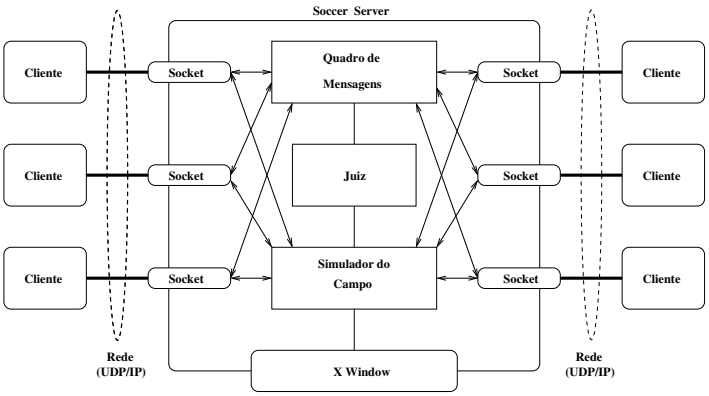


Figura 3.2: Arquitetura do Soccerserver (Chen et al., 2001).

2. Quando todos os clientes estão prontos para jogar, o juiz da partida (a pessoa que executa o servidor) começa o jogo pressionando o botão kick-off na janela do **soccermonitor**. Assim, começa o primeiro tempo.

3. O primeiro tempo é de 5 minutos. Quando este termina, o servidor suspende a partida.

4. O intervalo é também de 5 minutos. Durante este tempo, os competidores podem trocar seus programas clientes.

5. Antes do reinício do jogo, cada cliente conecta-se com o servidor por um comando `reconnect`.

6. Quando todos os clientes estiverem prontos, o juiz reinicia a partida pressionando o botão kick-off.

7. O segundo tempo, como o primeiro, é de mais 5 minutos. Ao final, o servidor para o jogo.

8. No caso de empate, a prorrogação tem início. Esta termina no instante que o primeiro gol é marcado (morte súbita ou *golden goal*) (Gonçalves, 2001).

O sistema **soccerserver** apresenta um conjunto de parâmetros que visam o modelamento do ambiente de simulação, tanto do **soccerserver** como do **soccermonitor**. O conjunto de parâmetros está descrito em (Chen et al., 2001), e estes podem ser modificados através dos comandos `soccerserver` e `soccermonitor`.

### 3.3.3 A Partida

As partidas são disputadas em um campo de futebol virtual (108m x 68m), provido pelo simulador **soccerserver**, em dois intervalos de cinco minutos (3000 ciclos de simulação).

Cada um dos times é composto por onze jogadores.

Cada um dos clientes (jogadores), recebe periodicamente através da sua conexão com o servidor (via *socket*), uma mensagem contendo os objetos captados por uma câmera de vídeo que estaria situada no topo do respectivo robô jogador.

A informação visual, enviada para o agente, corresponde aos objetos indentificados pelo robô, no setor visível do jogador. Esse setor visível pode assumir três diferentes ângulos: normal $[-45, 45]$, amplo $[-90, 90]$ e direcionado $[-22.5, 22.5]$. Os valores assumidos pelo ângulo de visão e sua respectiva qualidade influenciam diretamente na taxa de atualização da informação visual enviada pelo simulador.

Além da informação visual, o agente recebe assincronamente (via *socket*), mensagens enviadas pelo juiz informando alterações no estado do jogo e mensagens, com um tamanho máximo de 512 caracteres, enviadas por outros jogadores em um raio de 50m.

A comunicação entre o **soccerserver** e os agentes envolve troca de mensagens síncronas e assíncronas. A sincronização entre o agente e o **soccerserver** é mais um dos desafios a serem enfrentados na implementação do agente. Os comandos enviados para o simulador possuem um tempo de resposta, e nada garante que na próxima informação visual o efeito do comando enviado seja percebido. A incorreta sincronização entre o agente e o **soccersever** pode levar o jogador a apresentar um comportamento completamente indesejado.

### 3.3.4 O Técnico

Assim como no futebol real, as equipes simuladas possuem o seu próprio técnico. Os técnicos são clientes privilegiados usados para auxiliar os jogadores. Existem dois tipos de técnicos: o técnico *online* e o treinador.

O treinador pode exercer mais controle sobre o jogo, por isso ele é útil durante o desenvolvimento para tarefas como: executar um aprendizado automatizado ou controlando jogos. Mas o treinador pode ser usado apenas no estágio de desenvolvimento do time. Já o técnico *online* é utilizado para fornecer conselhos e informações aos jogadores durante o jogo, pois somente ele pode ser conectado no jogo oficial.

O treinador tem como habilidades:

- Controlar o *status* da partida;
- Difundir mensagens auditivas;

- Mover os jogadores e a bola para qualquer localização no campo e controla suas direções e velocidades;
- Obter informações a respeito de qualquer objeto móvel no campo.

O técnico *online* tem a intenção de observar o jogo e prover a informação para os jogadores. Então, suas habilidades são de certa forma limitadas, podendo apenas:

- Comunicar-se com os jogadores;
- Obter informações sobre os objetos móveis.

Não é permitido ao técnico *online* conduzir as ações de um jogador passo a passo, mantendo desta forma o caráter autônomo do jogador. Assim como no futebol de humanos, a atuação do treinador deve limitar-se a observar a partida e corrigir o posicionamento dos jogadores.

### 3.3.5 O Juiz

Como apresentado na Figura 3.2, o módulo Juiz é responsável por arbitrar a partida de acordo com as regras da FIFA. Entretanto, quando as regras são passíveis de interpretação, este encargo fica por conta de um juiz humano.

#### Regras julgadas pelo Servidor

- GOL: quando uma equipe marca um gol, o árbitro anuncia o gol enviando uma mensagem para todos os clientes (*broadcast*), atualiza o placar, suspende a partida por cinco segundos, período em que os jogadores devem voltar a posição de início da partida, move a bola para o centro do campo e modifica o modo de partida (do inglês "*play mode*") do jogo para `kick_off`.
- INÍCIO DO JOGO: se algum jogador não estiver em seu campo de defesa, antes de começar o jogo, ou na reposição da bola depois de um gol, o juiz irá movê-lo de volta a seu campo para um posição escolhida arbitrariamente.
- REPOSIÇÃO DA BOLA: quando a bola sai do campo, o juiz move a bola para o respectivo lugar de reposição e modifica o modo da partida de acordo com o modo de reposição. Por exemplo: `kick_in` no caso de arremesso lateral, `corner_kick` para cobrança de escanteio, ou ainda `goal_kick` no caso de tiro de meta.

- DISTÂNCIAS: quando o modo de partida é `kick_off`, `kick_in`, ou `corner_kick`, o árbitro remove todos os jogadores de defesa localizados dentro de um círculo centrado na bola com raio de 9.15m.

  Quando o modo de partida vai para `offside`, todos os jogadores de ataque são movidos de volta para uma posição fora do impedimento. Os jogadores em posição de impedimento são postos a 9.15m da bola.

  Quando o modo de partida é `goal_kick`, todos os jogadores de ataque são movidos para fora da grande área. O modo de partida muda imediatamente para `play_on` após a bola sair da grande área.

- CONTROLE DO MODO DE PARTIDA: o árbitro tem o poder de mudar o modo da partida para `play_on` depois que a bola é reposta em jogo, por intermédio do comando `kick`.

- INTERVALO E FINAL DE JOGO: o árbitro suspende a partida quando o primeiro ou o segundo tempo terminam. Em caso de empate o jogo é extendido.

#### Regras jugadas pelo árbitro humano

O árbitro humano deve julgar violações não compreendidas pelo juiz do **soccerserver**, faltas como "obstrução" são difíceis de serem julgadas, pois trata-se de julgar a intenção do jogador. Algumas faltas onde o juiz humano atua são descritas abaixo:

- JOGO DESONESTO: vários jogadores bloquearem apenas um oponente;
- BLOQUEIO: bloquear o movimento de outros jogadores;
- DEMORA: retardar a reposição da bola em jogo;
- GOLEIRO: o goleiro não pode soltar e voltar a agarrar a bola.

# Capítulo 4

# O Agente UFSC-Team

## 4.1 Introdução

O UFSC-Team trata-se de uma arquitetura de agentes autônomos baseada no modelo genérico para agentes cognitivos proposto em (Bittencourt e da Costa, 2001), mas esta arquitetura foi simplificada, por razões de eficiência e também por restrições do simulador do **soccerserver**. Neste modelo cada agente cognitivo adotado apresenta três níveis decisórios: reativo, instintivo e cognitivo, conforme Figura 4.1. Cada um destes níveis, juntamente com o nível inferior pretende modelar um agente completo e cada nível decisório incrementa a complexidade do comportamento do agente.

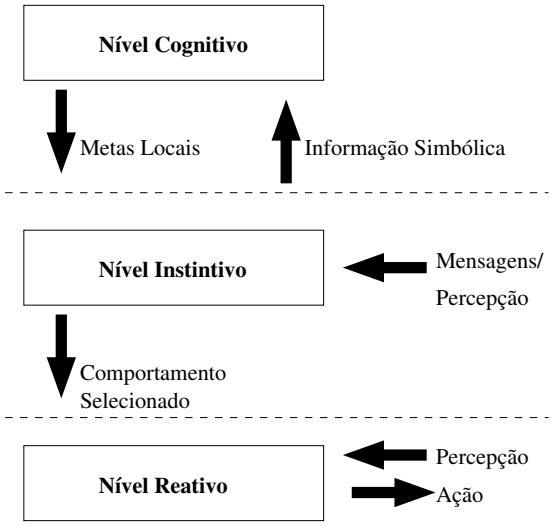


Figura 4.1: O fluxo de informação no agente.

O motor de inferência, que consiste em um controlador difuso, do nível reativo é responsável pela resposta em tempo real do agente, por exemplo por receber a informação visual do

**soccerserver** e por enviar os comandos de ação adequados. O motor de inferência do nível instintivo se responsabiliza por atualizar as variáveis simbólicas utilizadas no nível cognitivo e por escolher o comportamento mais adequado para a situação corrente, isto é, qual o controlador difuso deve ser utilizado no nível reativo para alcançar a meta local em vigor. Por fim, o nível cognitivo é quem irá determinar as metas locais e globais do agente. O nível cognitivo não interfere diretamente sobre o nível reativo, ele apenas determina a meta local e a envia para o nível instintivo. Essa meta tem efeito direto nas regras do nível instintivo, que seleciona o comportamento reativo adequado.

Os três processos que compõem o agente utilizam uma abordagem de programação *multi-thread*. Essa tecnologia permite particionar o processo e executar concorrentemente as partes resultantes. No caso do UFSC-Team, cada processo é composto por dois *threads*. O primeiro é responsável por manipular a interrupção SIGIO do Unix, usada para informar que uma nova mensagem foi recebida pelo *socket* e por colocar essa nova mensagem no *mailbox*. O outro *thread*, o principal, se responsabiliza pela execução das atividades do processo propriamente dito. A exclusão mútua entre os dois *threads* é feita utilizando semáforos. Essa implementação consiste em uma abordagem concorrente do clássico problema produtor/consumidor. Isso evita que o processo principal despenda um tempo precioso verificando se existe ou não numa nova mensagem no *socket*.

A implementação é escrita utilizando a linguagem de programação C++, e esta integra o ambiente para desenvolvimento de sistemas multiagentes cognitivos sob restrições de tempo real chamado Expert-Coop++ (da Costa, 1997). A seguir estes três níveis decisórios serão apresentados.

## 4.2 Nível Reativo

O nível reativo é composto por um *mailbox*, um conjunto de controladores difusos, um filtro de entrada e um filtro de saída (ver Figura 4.2). O *mailbox* é responsável pela recepção e pelo ordenamento das mensagens recebidas pelo processo. Todas as mensagens enviadas pelo **soccerserver** relativas à percepção (informação visual) serão armazenadas neste *mailbox*. As mensagens enviadas pelo juiz do jogo e pelos demais agentes são re-direcionadas para o nível instintivo.

Os controladores difusos são implementados utilizando-se a biblioteca *CNCL* escrita em C++ para auxiliar a implementação de sistemas especialistas difusos ou controladores (Junius e Stepple, 1997). Cada um desses controladores difusos é responsável por uma habilidade reativa do agente chamada Comportamento. O conjunto de controladores difusos

associado a cada agente do UFSC-Team depende do grupo ao qual esse agente pertence: goleiro, defensor, meio-campo, atacante, pois não faria sentido um goleiro possuir controlores para chutar a bola ao gol do adversário.

O filtro de entrada é responsável por extrair da informação visual os valores das variáveis lingüísticas utilizadas pelo controlador difuso ativo. O filtro de saída, por sua vez, é responsável por averiguar as saídas do controlador nebuloso ativo e combiná-las.

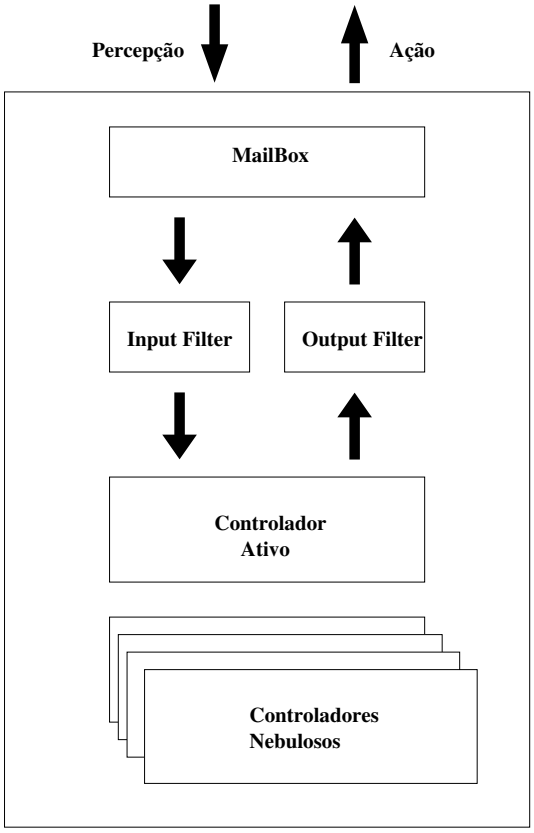


Figura 4.2: Nível Reativo.

Os controladores nebulosos possuem quatro tipos de saída: *kick-direction*, *kick-power*, *turn-moment* e *dash-power*. As entradas são um conjunto de variáveis lingüísticas, dependendo de qual comportamento está ativo. Cada controlador possui seu próprio conjunto de variáveis e o filtro de entrada se responsabiliza por extraí-las da infromação visual (percepção) recebida pelo **soccerserver** a cada novo ciclo de simulação.

A utilização de controladores difusos para implementação do comportamento reativo do agente possui algumas vantagens:

- A sincronização dos agentes é garantida, apenas ajustando a taxa entre entrada e saída, isto é, ajustando-se o ganho do controlador;

- As regras utilizadas para a implementação do controlador difuso podem ser escritas intuitivamente, evitando o dispêndio de um longo tempo para modelar um ambiente extremamente dinâmico;
- Pode-se assegurar que um dado controlador difuso estará sempre apto a satisfazer os requisitos de tempo real, pois estes são sistemas determinísticos.

Além disso, uma vez que o controlador difuso corresponde ao comportamento reativo mais apropriado para uma dada situação, os motores de inferência dos níveis instintivo e cognitivo podem dispor de mais tempo para realizar tarefas mais sofisticadas como extrair informação simbólica da percepção, planejar, estabelecer metas ou participar de processos de cooperação (da Costa, 2001).

## 4.3 Nível Instintivo

O motor de inferência deste processo é responsável pela execução das metas locais do agente e pela geração da informação simbólica para atualização da base de conhecimento do nível cognitivo.

Este nível consiste de um sistema especialista de um único ciclo de inferência que escolhe, a cada mudança de estado do jogo, o comportamento reativo mais adequado à meta local em vigor. Uma meta pode ser atingida por uma sequência de comportamentos que conduzem o agente a uma situação desejada. Cada estado do jogo é definido por um conjunto de condições que são monitoradas no nível instintivo. Os valores destas condições são determinados experimentalmente. Essas condições se referem à percepção e às mensagens enviadas pelo juiz, e são utilizadas como premissas das regras, análogas às do nível reativo. Mas no nível instintivo as implicações das regras consistem em variáveis simbólicas utilizadas para atualizar a base de conhecimento do nível cognitivo e/ou para selecionar um novo comportamento no nível reativo. A cada instante, o comportamento selecionado deve responder aos estímulos do ambiente buscando alcançar a meta, devendo também ter suas condições satisfeitas pelo estado do partida. Uma vez escolhido um comportamento, o nível instintivo se mantém monitorando os requisitos associados a este comportamento. Caso algum desses requisitos não mais se verifique, ele utiliza seu conjunto de regras para inferir um novo comportamento. Caso não seja possível, a meta corrente está comprometida, e uma nova meta deve ser especificada.

O nível instintivo também manipula as mensagens enviadas pelo juiz da partida informando uma mudança no *status* do jogo. Essas mudanças são tratadas de forma análoga a

das mudanças do estado da partida, levando o nível instintivo a escolher o comportamento adequado à nova situação.

As entradas do motor de inferência deste nível são a percepção, recebida pelo nível reativo e as mensagens enviadas pelo juiz da partida e pelos demais agentes do UFSC-Team. A percepção consiste na mesma informação visual recebida de forma síncrona pelo nível reativo e proveniente do **soccerserver** mas, diferentemente do nível reativo, o nível instintivo possui uma memória. Essa memória consiste em um *buffer*, onde a percepção é armazenada e cujo tamanho inicial é definido na implementação do agente. Isso torna possível escolher o montante de informação visual (percepção) usado no ciclo de inferência.

A percepção é armazenada no *buffer* para mensagens síncronas e as mensagens enviadas pelo juiz da partida e demais agentes do UFSC-Team armazenadas em outro *buffer*, destinado às mensagens assíncronas, conforme mostra a Figura 4.3.

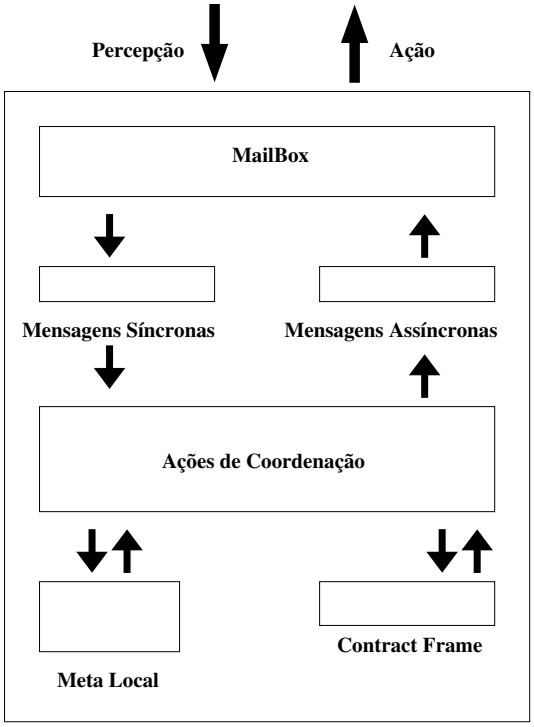


Figura 4.3: Nível Instintivo.

Cada vez que um desse dois *buffers* é atualizado, ou quando uma nova meta local é recebida do nível cognitivo, o sistema especialista é executado. Dado uma entrada, as regras estão aptas a reconhecer as mudanças no estado do jogo. O resultado da execução destas regras pode ser uma atualização da base de conhecimento do nível cognitivo e/ou a seleção de um novo controlador difuso para conduzir o agente do estado atual até a meta local.

## 4.4 Nível Cognitivo

O nível decisório cognitivo, consiste em um sistema baseado em conhecimento simbólico e orientado a objetos que manipula tanto as informações simbólicas recebidas do nível instintivo quanto as mensagens assíncronas recebidas dos demais agentes do UFSC-Team, gerando metas locais e globais. Esse sistema baseado em conhecimento possui três bases de conhecimento: *Dynamic KB, Static KB* e *Export KB*, conforme mostra a Figura 4.4.

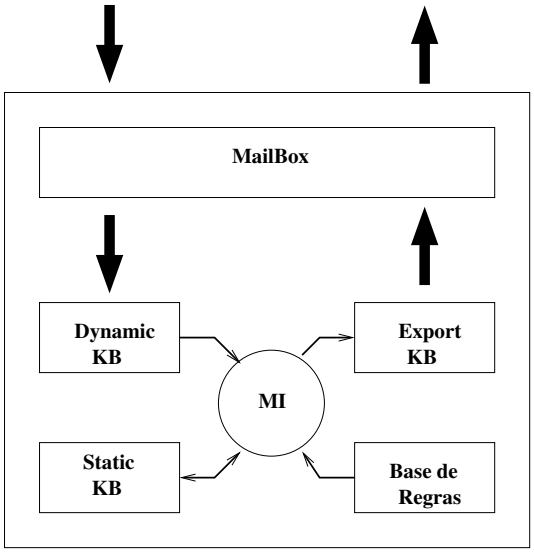


Figura 4.4: Nível Cognitivo.

*Dynamic KB* é utilizada para armazenar a informação simbólica gerada pelo nível instintivo e as mensagens assíncronas enviadas por outros agentes do UFSC-Team. A *Static KB* armazena o conhecimento que foi inferido no nível cognitivo sobre o jogo, o time, o oponente, os planos do agente, metas, etc. Ambas, *Dynamic KB* e *Static KB*, são usadas pelo motor de inferência para gerar as metas dos agentes. Os novos fatos sobre o jogo, os planos e metas são armazenados dentro do *Static KB*. *Export KB* é usada para armazenar a saída do nível cognitivo. Basicamente esta saída consiste de metas locais a serem enviadas ao nível instintivo, informação para ser usada em estratégias de cooperação, ou mensagens a serem enviadas a outros agentes do UFSC-Team.

Em resumo, o nível cognitivo é responsável pelo estabelecimento de metas locais e pela interação do agente com a comunidade estabelecendo metas globais através de processo de cooperação. Uma importante característica dessa arquitetura é que o nível cognitvo pode despender mais tempo com planejamento, estabelecimento de novas metas, entre outras coisas, uma vez que o nível reativo assim como, em algumas situações, o nível instintivo são responsáveis pela interação com o ambiente respeitando os requisitos de tempo-real.

## 4.5 O nível instintivo como um problema de aprendizado

*O nível instintivo foi escolhido como um problema para se aplicar um aprendizado, pois os valores de algumas de suas variáveis foram adotados de forma empírica. O que possibilita uma ótima proposta, pois já se conhecem alguns resultados, e sabe-se que ocasionalmente estes resultados não são os desejados. Nas próximas seções serão abordados alguns aspectos que devem ser levados em consideração para o entendimento do problema.*

### 4.5.1 A visão do agente

O sensor visual de um agente informa o objeto que está sendo visto por ele no momento. Esta informação visual chega para o agente através de uma mensagem enviada pelo servidor que possui o formato demonstrado no exemplo 4.1, e é enviada para o agente a cada intervalo definido pelo `sense_step` que possui normalmente valor de 150ms.

**Exemplo 4.1** *(**see** ObjName Distance Direction DistChng DirChng BodyDir HeadDir)*

*ObjName ::=* *(player Teamname UniformNumber)*
| *(goal [l|r])*
| *(ball)*
| *(flag c)*
| *(flag [l|c|r] [t|b])*
| *(flag p [l|r] [t|c|b])*
| *(flag g [l|r] [t|b])*
| *(flag [l|r|t|b] 0)*
| *(flag [t|b] [l|r] [10|20|30|40|50])*
| *(flag [l|r] [t|b] [10|20|30])*
| *(line [l|r|t|b])*

onde, o objeto (**goal r**) é interpretado como o centro do gol posicionado do lado direito do campo, (**flag c**) é um flag virtual que situa-se no centro do campo, (**flag 1 b**) é o flag que se localiza no canto inferior esquerdo do campo. (**flag p 1 b**) é um flag virtual localizado no canto inferior direito da grande área posicionada do lado esquerdo do campo, entre outros. Atualmente existem 55 flags e 4 linhas de limitação do campo possíveis de serem visualizadas pelo jogador, elas podem ser visualidas na Figura 4.5.

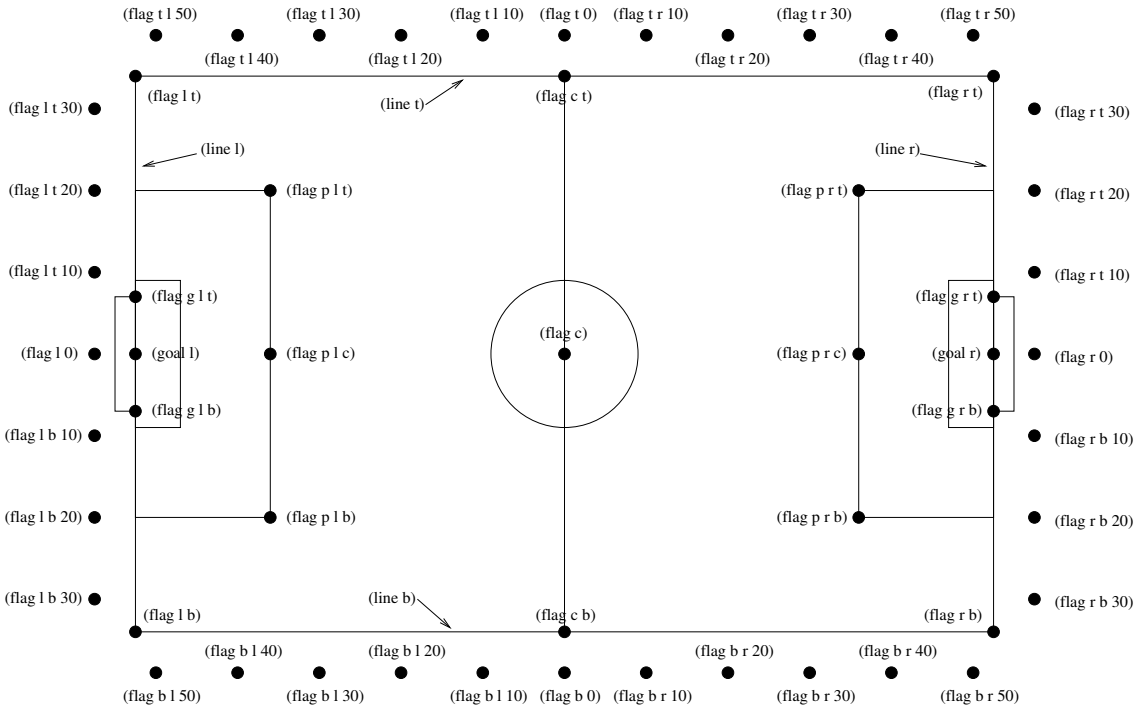


Figura 4.5: Localização dos flags no campo

O campo de visão de um jogador depende de vários fatores, primeiro de tudo tem-se os parâmetros `sense_step` e `visible_angle` que determinam o tempo de envio da informação visual e quantos graus o cone de visão do jogador irá ter. A qualidade e a frequência da informação visual pode ser influenciada pelo jogador, para isto basta-se modificar o valor das variáveis: `ViewWidth` e `ViewQuality`.

O jogador pode "ver" um objeto se esse encontra-se a `visible_distance` metros do jogador. Portanto os objetos **b** e **g** na Figura 4.6 não podem ser visto, pois encontram-se fora do cone de visão.

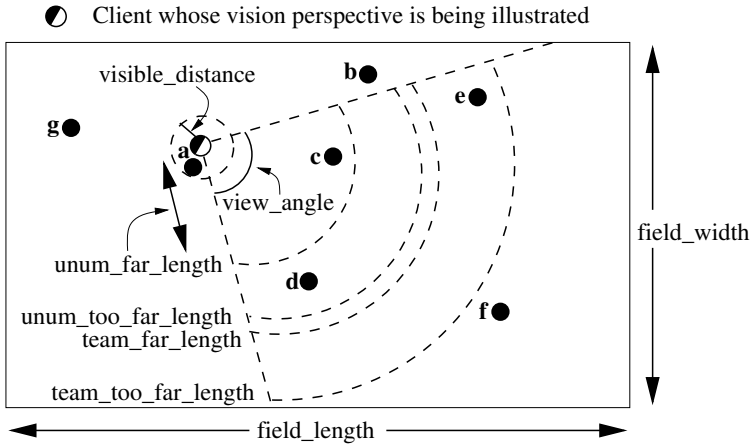


Figura 4.6: Campo de visão de um jogador

A Figura 4.6 também ilustra a quantidade de informação que um jogador obtém sobre outro que está próximo a ele. Na tabela 4.1 algumas distâncias são consideradas e suas respectivas faixas de visão, sendo `dist` a distância entre os jogadores.

| **Parâmetro** | **Visibilidade** |
|---|---|
| `dist` $\leq$ **unum_far_length** | tanto o número do jogador quanto o nome do time são visíveis |
| **unum_far_lenght** $<$ `dist` $<$ **unum_too_far_length** | o nome do time sempre é visível, mas a probabilidade do número do uniforme ser visível é inversamente proporcional a `dist`. |
| `dist` $\geq$ **unum_too_far_length** | o número do uniforme não é visível. |
| `dist` $\leq$ **team_far_length** | o nome do time é visível. |
| **team_far_lenght** $<$ `dist` $<$ **team_too_far_length** | a probalidade do nome do time ser visível é inversamente proporcional a `dist`. |
| `dist` $\geq$ **team_too_far_length** | o nome do time não é visível. |

Tabela 4.1: Faixa de Visão de um agente.

Por exemplo, na Figura 4.6, assume-se que os pontos **c**, **d**, **e**, **f** são jogadores. Então o jogador **c** seria identificado tanto pelo número de seu uniforme, quanto pelo seu time; o jogador **d** pelo nome de seu time, e com uma chance de 50% de ser pelo número de seu uniforme. O jogador **e** teria 20% de chance de ser reconhecido apenas pelo nome de seu time, e por fim o jogador **f** seria identificado apenas como um jogador anônimo.

### 4.5.2 A base de regras

O motor de inferência do nível instintivo pode utilizar as informações visuais do agente, para inferir a regra que será utilizada. O exemplo 4.2 apresenta a estrutura de uma regra utilizada no nível instintivo.

**Exemplo 4.2** (rule_001
(if (logic ( game state before_kick_off ))
(logic ( reactive_behavior status ?x1 )))
(filter ( != ?x1 reactive_disable ))
(then (logic ( reactive_behavior status reactive_disable ))
(message ((to Interface) (from Coord) (deadline 0) (grade 0.0) (alpha 0.0) (round 0.0)
(body (INFORM ((logic ( reactive_disable )))))))
(message ((to RL) (from Coord) (deadline 0) (grade 0.0) (alpha 0.0) (round 0.0)
(body (INFORM ((logic ( reactive_disable ))))))))

A regra exposta no Exemplo 4.2 verifica se o estado da partida é igual a `before_kick_off`, ou seja, a partida encontra-se parada aguardando o início da partida. Esta regra garante que caso a partida esteja para iniciar ou momentaneamente parada, nenhum controlador está ativo. Até o momento ela atendeu seu propósito, no entanto, tal regra pode se tornar inconveniente se durante a execução da partida for necessário executar algum controlador.

Esta certificação de que nenhum controlador esta ativo é descrita na linha: `(filter ( != ?x1 reactive_disable ))`, onde o valor da variável `x1` atribuído na linha `(logic ( reactive_behavior status ?x1 ))` é igual a `reactive_disable`, ou seja todo o comportamento reativo está inativo.

# Capítulo 5

# Aprendizado de Máquina

*Aprendizado é a essência da inteligência!*

## 5.1 Introdução

Aprendizado de Máquina (AM) é uma área de IA cujo objetivo é o desenvolvimento de modelos computacionais dos processos de aprendizado, bem como a construção de sistemas capazes de adquirir conhecimento de forma automática. Um sistema de aprendizado é um programa de computador que toma decisões baseado em experiências acumuladas por meio de soluções bem-sucedidas de problemas anteriores. Os diversos sistemas de Aprendizado de Máquina possuem características particulares e comuns que possibilitam sua classificação quanto à linguagem de descrição, modo, paradigma e forma de aprendizado utilizado (Monard e Baranauskas, 2003).

Os processos de aprendizado incluem a aquisição de novos conhecimentos, o desenvolvimento das habilidades motoras e cognitivas através de instrução e prática, a organização de novos conhecimentos, o desenvolvimento de formas efetivas de representar o conhecimento, e a descoberta de novos fatos e teorias por meio de observações e experimentações. Aprendizado é a chave da superioridade da inteligência humana. O homem é capaz de aprender habilidades motoras e cognitivas. A visão é o principal meio para adquirir habilidades motoras, mas mesmo com a falta desta, o homem é capaz de aprender as mesmas habilidades motoras.

O estudo e a modelagem computacional dos processos de aprendizado em suas múltiplas manifestações constituem o principal objetivo do AM. Um aumento significativo da inteligência da máquina será resultado do aumento das capacidades de aprendizado da mesma. É

difícil encontrar uma definição geral sobre aprendizado, mas (Monard et al., 2000) define o aprendizado da seguinte maneira:

*"Aprendizado é a habilidade de aperfeiçoar-se em uma determinada questão."*

Por exemplo, esta questão pode ser reconhecer uma determinada fisionomia. Essa definição é comportamental na medida em que julga o desempenho do ator que aperfeiçoa-se a partir de uma ponto de vista exterior, tratando-o como uma caixa preta. Por exemplo, o conhecimento necessário para reconhecer fisionomias pode ser expresso em termos de regras de classificação, listando as características de uma fisionomia específica; a habilidade de reconhecer fisionomias pode ser aperfeiçoada melhorando as regras de classificação. Nesse ponto de vista, aprendizado é a habilidade de adquirir novos conhecimentos. Existem diversos paradigmas de AM, tais como:

- **Simbólico:** Os sistemas de aprendizado simbólico buscam aprender construindo representações simbólicas de um conceito através da análise de exemplos e contra-exemplos desse conceito. As representações simbólicas tomam tipicamente a forma de uma expressão lógica, árvore de decisão, conjunto de regras ou rede semântica.
- **Estatístico:** Pesquisadores em estatística têm criado diversos métodos de classificação, muitos deles semelhantes aos métodos posteriormente desenvolvidos pela comunidade de AM. A idéia geral consiste em utilizar modelos estatísticos para encontrar uma boa aproximação do conceito a ser aprendido. Entre os métodos estatísticos, destaca-se o aprendizado Bayesiano, que utiliza um modelos probabilístico baseado em conhecimento prévio do problema.
- **Conexionista:** Redes Neurais são construções matemáticas inspiradas em modelos biológicos do sistema nervoso. A metáfora biológica com as conexões neurais do sistema nervoso tem interessado muitos pesquisadores, e as analogias com a biologia têm levado muitos pesquisadores a acreditar que as redes neurais possuem um grande potencial na resolução de problemas que requerem intenso processamento sensorial humano.

Um ponto interessante sobre os seres humanos está relacionado à sua habilidade de fazer generalizações precisas a partir de fatos. O ser humano é capaz de encontrar padrões apenas observando um processo do mundo real. Na computação isso pode ser obtido a partir de um conjunto de exemplos, fornecido pelo usuário ou por um processo do mundo real, e por meio da inferência indutiva, a qual, mesmo sendo um dos recursos mais utilizado pelo cérebro na produção de conhecimento novo, deve ser utilizada cuidadosamente.

### 5.1.1 Aprendizado Indutivo

Indução é a forma de inferência lógica que permite que conclusões gerais sejam obtidas de exemplos particulares. É caracterizada como o raciocínio que parte do específico para o geral, do particular para o universal, da parte para o todo. No Aprendizado por Indução, o aprendiz adquire um conceito fazendo inferências indutivas sobre os fatos apresentados. Hipóteses geradas pela inferência indutiva podem ou não preservar a verdade.

O processo que usa observações a fim de descobrir regras e procedimentos é denominado indução. O processo de indução é indispensável ao ser humano, pois é um dos principais meios de criar novos conhecimentos e prever eventos futuros.

Foi através de induções que Kepler descobriu as leis do movimento planetário, que Mendel descobriu as leis da genética e que Arquimedes descobriu o princípio da alavanca. Pode-se ousar em afirmar que a indução é o recurso mais utilizado pelos seres humanos para obter novos conhecimentos. Apesar disto, esse recurso deve ser utilizado com os devidos cuidados pois, se o número de observações for insuficiente ou se os dados relevantes forem mal escolhidos, as regras obtidas podem ser de pouco ou nenhum valor.

Existem duas formas de aprendizado por indução. No Aprendizado por Exemplos, o aprendiz induz a descrição de um conceito formulando uma regra geral a partir dos exemplos e dos contra-exemplos fornecidos pelo professor ou pelo ambiente. O professor já tem o conhecimento do conceito e, assim, ele pode ajudar o aprendiz selecionando exemplos relevantes para aprender um determinado conceito. A tarefa do aprendiz é determinar a descrição geral de um conceito, analisando exemplos individuais a ele fornecidos. Essa estratégia também é conhecida como aprendizado supervisionado.

No Aprendizado por Observação, o aprendiz analisa entidades fornecidas ou observadas e tenta determinar se alguns subconjuntos dessas entidades podem ser agrupados em certas classes (i.e. conceitos) de maneira útil. Como não há um professor que já tenha o conhecimento do conceito para fornecer exemplos significativos ao conceito a ser aprendido, essa estratégia é também chamada de aprendizado não supervisionado.

## 5.2 Aprendizado por Reforço

O uso da interação com o ambiente para o aprendizado de certas tarefas é a primeira idéia que surge ao se tratar de aprendizado natural. Durante toda a vida, indubitavelmente a maior fonte de conhecimento dos seres humanos é a sua interação com o ambiente onde

está inserido. O aprendizado através da interação é a idéia fundamental que sustenta algumas teorias de aprendizado e inteligência.

Aprendizado por Reforço (RL, do inglês *Reinforcement Learning*) é sinônimo de aprendizado por interação, uma vez que o agente aprende diretamente da interação com o ambiente onde está inserido (Shabani et al., 2003). Neste processo, não é dito ao aprendiz qual ação deve ser tomada, ao invés disso ele deve descobrir qual ação irá retornar uma melhor recompensa.

Segundo (Kaelbling et al., 1996) o modelo padrão de aprendizado por reforço consiste de: um conjunto de estados do ambiente ($S$), um conjunto de ações possíveis ao agente ($A$) e um conjunto de sinais escalares de reforço, comumente $\{0,1\}$. Neste modelo, um agente é conectado ao ambiente via percepção ($p$) e ação ($a$), como demonstra a Figura 5.1. A cada interação o agente recebe uma informação ($i$) que indica o estado corrente ($s$) do ambiente. Após o agente saber qual o estado corrente, realiza uma ação que irá modificar o estado do ambiente, e o valor desta transição de estado é informada ao agente por um valor de reforço $r$, chamado recompensa.

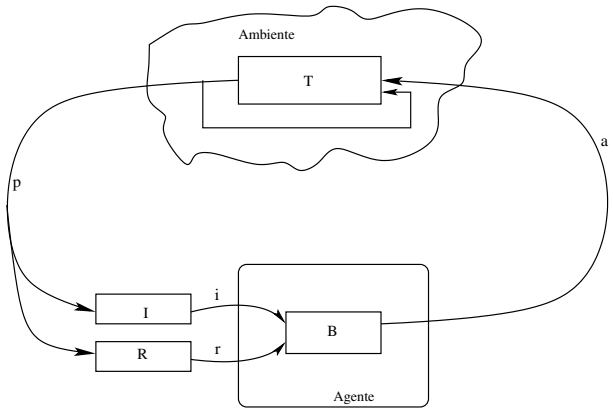


Figura 5.1: Modelo padrão de um Aprendizado por Reforço.

O aprendizado por reforço é definido não caracterizando algoritmos de aprendizado, mas sim o problema a ser aprendido. Qualquer algoritmo que satisfaça a resolução de um problema, pode ser considerado um algoritmo de aprendizado por reforço (Sutton e Barto, 1998). A idéia básica é simplesmente capturar os aspectos mais importantes do problema real que um agente de aprendizado enfrenta durante a interação com o ambiente para alcançar uma meta.

O RL difere do aprendizado supervisionado, pois este último consiste em um aprendizado através de exemplos e estes dependem de algum supervisor externo para serem introduzidos no sistema. Esse é um importante tipo de aprendizado, mas por si só não é adequado a

um aprendizado por interação. Nos problemas de interação às vezes se torna impraticável obter exemplos, que sejam ao mesmo tempo corretos e que representem todos os tipos de situações. Em muitos domínios o aprendizado por reforço é o único meio praticável para treinar um programa que envolva um alto nível de complexidade.

### 5.2.1 Processos de Decisão de Markov

Uma grande parte dos trabalhos realizados em aprendizado por reforço assume que a interação do agente com o ambiente pode ser modelada como um processo de decisão de Markov (MDP), desta forma tem-se que um passo de um agente acontece da seguinte maneira:

No tempo $t$, o agente está em um estado qualquer, $s \in S$, e escolhe uma ação, $a \in A_{(s)}$, de acordo com uma política de aprendizado $\pi$. Após ter realizado esta ação, o agente atinge um outro estado, $s^{'}$, em $t = t+1$ com uma probabilidade de $P^{a}_{ss^{'}}$, probabilidade de atingir o estado $s^{'}$, estando no estado $s$ e tomando uma ação $a$. Para realizar esta transição o agente recebe uma recompensa, $r_{t+1}$, dada por $R^{a}_{ss^{'}}$, probabilidade de receber uma recompensa $r$ ao atingir o estado $s^{'}$, partindo do estado $s$, tomando uma ação $a$ .

Um processo finito e discreto de Markov no aprendizado de reforço consiste de:

- Um conjunto finito de estados S;
- Um conjunto finito de ações A;
- Um função de transição de estado, $P^{a}_{ss^{'}} = Pr(s_{t+1} = s^{'} | s_t = s, a_t = a)$, isto é, a probabilidade de alcançar $s^{'}$ no tempo = $t+1$ dada uma ação $a$ que foi tomada no estado $s$ no tempo $t$.
- Uma função de recompensa que, a partir da tripla $(s, a, s^{'})$ gera uma valor numérico para o agente, através da $R^{a}_{ss^{'}}$.
- Um relógio global, $t = 1, 2, ...T$, onde T é infinito.

Um processo é dito ser um processo de Markov, se obedece a Propriedade de Markov, ou seja, se:

$$Pr(s_{t+1} \mid s_t, a_t) = Pr(s_{t+1} \mid s_t, a_t, s_{t-1}, a_{t-1}, ...) \tag{5.1}$$

for verdade. Isto é, a distribuição da probabilidade dos estados inseridos em $t+1$ é condicionalmente independente dos eventos anteriores de $(s_t, a_t)$ - sabendo-se o estado corrente e

a ação tomada, são fatores suficientes para se definir o que irá acontecer no próximo passo. No aprendizado por reforço esta condição também é utilizada para a função de recompensa,

$$Pr(r_{t+1} \mid s_t, a_t) = Pr(r_{t+1} \mid s_t, a_t, s_{t-1}, a_{t-1}, \ldots). \tag{5.2}$$

### 5.2.2 Elementos do Aprendizado por Reforço

Além do agente e do ambiente, pode-se identificar quatro sub-elementos importantes para o aprendizado por reforço:

1. uma política de aprendizado,
2. uma função de recompensa,
3. uma função de avaliação e
4. um modelo do ambiente.

#### Política de Aprendizado

Uma **política de aprendizado** $\pi$ é o mecanismo utilizado para a seleção das ações a serem tomadas em um determinado estado (ten Hagen e Kröse, 1997). Em outras palavras, uma política $\pi$ é a organização dos estados, $s \in S$, e das ações, $a \in A(s)$, para a probabilidade $\pi(s,a)$ de se tomar uma ação $a$ enquanto se estiver no estado $s$. Esta política é o que a psicologia chama de conjunto de regras "estímulo-resposta" ou associações (Sutton e Barto, 1998). Para se otimizar a interação do agente com o ambiente, as conseqüências futuras de uma política de aprendizado devem ser conhecidas.

#### Função de Recompensa

Num processo de aprendizado por reforço, a meta é definida através de uma **função de recompensa**. A meta de um agente é maximizar o total de recompensas recebidas (Shabani et al., 2003). A grosso modo, a função faz o mapeamento dos estados percebidos do ambiente para um único valor, normalmente numérico, chamado de recompensa. Esta função depende do ambiente e também de alguns parâmetros, tais como as ações tomadas pelo agente e os estados que o ambiente já atingiu. A função de recompensa define os eventos que estão ou não corretos para o agente. Uma recompensa imediata define o problema enfrentado pelo agente, por isso deve ser necessariamente uma função pré-determinada.

**Função de Avaliação**

Enquanto a função de recompensa indica se o agente executou ou não uma boa ação logo após esta ter sido executada, a **função de avaliação** de um estado, $V(s)$, descreve o comportamento deste a longo prazo, ou seja, o valor de um estado corresponde ao total de recompensas que um agente estima acumular a partir do estado corrente até alcançar um estado final.

A maioria dos algoritmos de aprendizado por reforço são baseados na estimativa das funções de avaliação. Uma função de avaliação é uma função que estima o quanto é bom para o agente estar em um determinado estado, se aplicada uma política *pi*. As funções de avaliação são definidas respeitando uma política de aprendizado particular.

O valor de um estado $s$ sujeito a uma política $\pi$, $V^{\pi}(s)$, é o valor esperado quando se começa em um estado $s$ e desde então segue-se uma política $\pi$ . $V^{\pi}(s)$ segundo (Reynolds, 2002) pode ser definido como:

$$V^{\pi}(s) = \sum_{a} \pi(s,a) \sum_{s'} P^{a}_{ss'} [R^{a}_{ss'} + \gamma V^{\pi}(s')] \tag{5.3}$$

onde $\gamma \in [0,1]$ é o fator de desconto que é utilizado para ponderar futuras recompensas. Se $\gamma = 1$ corresponde a recompensa imediata, caso contrário, se $\gamma = 0$, então corresponde a soma esperada das futuras recompensas. A Equação (5.3) é conhecida como a Equação de Bellman para $V^{\pi}$, ela expressa a relação entre o valor do estado corrente com os seus sucessores. A equação de Bellman simplesmente calcula a média sobre todas as possibilidades, pesando cada estado de acordo com a sua probabilidade de ser atingido. O valor do estado inicial $s$ deve ser igual ao valor esperado do seu sucessor, mais a recompensa esperada ao longo do caminho. $V^{\pi}$ é a única solução para esta equação de Bellman (5.3) (Sutton e Barto, 1998).

**Modelo do Ambiente**

O quarto elemento de um sistema de aprendizado por reforço é um **modelo do ambiente**. Estes modelos são utilizados para o planejar, aquilo que pretende ser um modo de decisão do percurso de uma ação, considerando situações futuras possíveis, antes delas serem realmente experimentadas. A junção dos modelos e do planejamento nos sistemas de aprendizado por reforço é um desenvolvimento relativamente novo.

Em um problema geral de aprendizado por reforço, as ações do agente determinam não apenas a recompensa imediata, mas também (através de probabilidade) o próximo estado do

ambiente. Em alguns casos o agente tem que aprender a partir de recompensas atrasadas, isto é, ele tem que tomar uma seqüência longa de ações, recebendo recompensas insignificantes, problemas com este tipo de recompensa são bem modelados como Processos de Decisão de Markov (MDP).

### 5.2.3 Achando uma Política de aprendizado dado um Ambiente

Esta seção irá explorar técnicas para determinar uma política de aprendizado ótima, conhecendo o modelo do ambiente onde o agente está inserido. O modelo consiste em se conhecer as probabilidades das transições de estados $P^a_{ss'}$ e da função de recompensa $R^a_{ss'}$. Para se encontrar uma política ótima, primeiro tem-se que achar uma função de avaliação ótima. A função de avaliação ótima é única e pode ser definida como:

$$V^*(s) = max_a \sum_{s' \in S} P^a_{ss'} \left( R^a_{ss'} + \gamma V^*(s') \right), \forall s \in S, \tag{5.4}$$

o que define o valor da função de avaliação ótima de um estado $s$ é a recompensa instantânea somado ao valor do próximo estado, acreditando-se ter realizado a melhor ação possível. Dado a função de avaliação ótima de um estado, pode-se especificar a política ótima da seguinte maneira:

$$\pi^*(s) = arg\,max_a \left( \sum_{s' \in S} P^a_{ss'} \left( R^a_{ss'} + \gamma V^*(s') \right) \right) \tag{5.5}$$

#### Iteração de Avaliações

Uma maneira de se encontrar uma política de aprendizado ótima, é achar uma função de avaliação ótima. Esta pode ser determinada usando um algoritmo simples chamado de **Iteração de Avaliações** (Algoritmo 1), que demonstra como converger a valores corretos de $V^*$:

**Algoritmo 1** Iteração de Avaliações

initialize $V(s)$ arbitrarily
**repeat**
  **for** $s \in S$ e $a \in A$ **do**
    $V(s) = max_a \left( \sum_{s' \in S} P^a_{ss'} \left( R^a_{ss'} + \gamma V(s') \right) \right)$
  **end for**
**until** policy good enough

#### Iteração de Políticas

Um outro algoritmo utilizado para se encontrar uma política de aprendizado (Algoritmo 2), trata diretamente a política de aprendizado, para depois encontrar indiretamente uma função de avaliação ótima. Uma vez que se conhece o valor de uma avaliação de um determinado estado que está sujeito a uma política de aprendizado, pode se considerar que este valor pode ser melhorado mudando-se a primeira ação tomada. Se isto for possível, muda-se a política de aprendizado para se tomar a nova ação. Este passo garante o desempenho da nova política. Quando a ação não pode ser melhorada, isto quer dizer que tem-se uma política de aprendizado ótima.

**Algoritmo 2** Iteração de Políticas

choose an arbitrary policy $\pi$
**repeat**
  $\pi := \pi'$
  compute de value function of policy $\pi$
  solve the linear equations
  $V(s) := \sum_{s' \in S} P^{\pi(s)}_{ss'} \left( R^{\pi(s)}_{ss'} + \gamma V(s') \right)$
  **for** $s \in S$ improve the policy **do**
    $\pi'(s) := arg\,max_a \left( \sum_{s' \in S} P^{\pi(s)}_{ss'} \left( R^{\pi(s)}_{ss'} + \gamma V_\pi(s') \right) \right)$
  **end for**
**until** $\pi = \pi'$

### 5.2.4 Aprendendo uma Política: sem conhecer o ambiente

Quando o modelo do ambiente não é conhecido previamento, a preocupação do aprendizado por reforço está voltada a obter uma política de aprendizagem ótima. Para isso o agente tem que interagir diretamente com o ambiente para obter informações que serão processadas

por um determinado algoritmo para produzir uma política ótima. Existe uma classe de algoritmos chamados de *Métodos de diferença temporal*, estes algoritmos utilizam *"insights"* do algoritmo 1 para ajustar os valores estimados das funções de avaliação de um determinado estado baseado na recompensa imediata e o valor estimado do próximo estado. Os algoritmos que se enquadram nesta classe serão abordados abaixo. A partir deste ponto quando se falar em valor de um estado, este valor refere-se ao valor da função de avaliação deste estado.

**Crítico Heurístico Adaptável e TD(λ)**

O algoritmo Crítico Heurístico Adaptável é uma adaptação do algoritmo 2, que utiliza o algoritmo TD$(0)$ ao invés de equações lineares para calcular o valor da função de avaliação de um deteminado estado. Para o crítico aprender o valor de uma política, foi definido que $\langle s,a,r,s^{'}\rangle$ é uma *experience tuple* que resume uma única transição de estados no ambiente. O valor da política é aprendido usando o algoritmo de Sutton TD$(0)$ (Sutton e Barto, 1998) que utiliza a regra de atualização:

$$V(s) = V(s) + \alpha\left(r + \gamma V(s^{'}) - V(s)\right) \tag{5.6}$$

onde $s$ é o estado visitado, seu valor estimado é atualizado para se aproximadar de $r+\gamma V(s^{'})$, onde $r$ é o valor da recompensa instantânea recebida e $V(s^{'})$ é o valor estimado do próximo estado. A idéia principal é que $r+\gamma V(s^{'})$ é uma amostra do valor de $V(s)$, e está mais próximo ao valor correto porque incorpora a recompensa real $r$. Se a taxa de aprendizado $\alpha$ for bem ajustada e a política for mantida fixa, TD$(0)$ é uma garantia para convergir a uma ótima função de avaliação.

**Q-Learning**

O algoritmo Q-learning (Watkins, 1989) é fácil de ser implementado. Para a compreenssão deste algoritmo algumas equações têm de ser apresentadas. Primeiro a Equação 5.7 define o valor de uma função de avaliação estado-ação ou Q-valor. Esta função leva em consideração o valor de uma ação $a$ ótima, efetuada no estado $s$.

$$Q^{\pi}(s,a) = \sum_{s^{'}} P^{a}_{ss^{'}}\left(R^{a}_{ss^{'}} + \gamma V^{\pi}(s^{'})\right) \tag{5.7}$$

Segundo a Equação 5.4 que pode ser reduzida a: $V^{*}(s) = max_{a}Q^{*}(s,a)$. A partir destas duas equações surge uma outra, que irá tratar a Equação 5.7 de uma forma recursiva:

$$\begin{aligned} Q^*(s,a) &= max_\pi Q^\pi(s,a) \\ &= \sum_{s'} P^a_{ss'} \left( R^a_{ss'} + \gamma max_a Q^*(s,a) \right) \end{aligned} \tag{5.8}$$

Os Q-valores podem ser estimados usando um método igual ao do TD(0) (Equação 5.6), e também podem ser utilizados para definir a política. Pois uma ação pode ser escolhida apenas tomando a ação que tenha o maior valor entre os Q-valores para o estado corrente. Em (Faria e Romero, 1999) este tipo de atualização foi utilizada na implementação do algoritmo Q-Learning. A regra de atualização dos Q-valores torna-se:

$$Q(s,a) = Q(s,a) + \alpha(r + \gamma max_{a'} Q(s^{'}, a^{'}) - Q(s,a) \tag{5.9}$$

### 5.2.5 Exploration X Exploitation

Um dos dilemas enfretados quando se trata de algoritmos de aprendizado por reforço, é o paradigma da *Exploration X Exploitation*. Em sua tradução literal, as duas palavras significam a mesma coisa, mas no aprendizado por reforço, uma abordagem de *Exploration* significa que o agente irá tomar uma postura de agente explorador do ambiente, ou seja, irá sempre estar tentando explorar caminhos que ainda não foram experimentados. Já a abordagem *Exploitation* trata-se de um comportamento mais voltado a maximimar os valores de sua função de avaliação, ou seja, o agente sempre irá tomar a ação que lhe retorne uma melhor recompensa.

Na abordagem *Exploration* o agente age de forma randômica, podendo desta forma explorar todo o ambiente onde está inserido, já a *Exploitation* pode acabar encontrando um caminho que atinga a meta final do agente, mas que não seja ótimo, mas por não atuar de maneira exploradora nunca irá conhecer todos os caminhos possíveis.

Uma abordagem ideal para um agente seria uma que estivesse entre as duas. O agente poderia agir de uma maneira mais exploradora quando tivesse apenas noção do ambiente onde está inserido, e a partir do momento que tivesse o modelo do ambiente formulado, agiria de uma forma a maximizar a sua função de avaliação.

### 5.2.6 Aplicacações do Aprendizado por Reforço

Uma das razões do aprendizado por reforço ser tão popular, é por ele ser tratado como uma ferramenta teórica para estudar os princípios de um agente aprendendo a agir. Mas ele também tem sido usado por inúmeros pesquisadores como uma ferramenta computacional para a construção de sistemas autônomos que melhorem a partir de suas próprias experiências. As aplicações deste aprendizado atinge diversas áreas como robótica, manufatura industrial e jogos de computador.

#### RoboCup

O aprendizado por reforço tem sido vastamente explorado na RoboCup. Em (Riedmiller et al., ) traz uma versão de um time de um grupo de pesquisadores que aprendeu movimentos básicos através do aprendizado por reforço. Entre os movimentos aprendidos pode-se destacar:

1. *Chute*: o jogador pode chutar a bola para que ela atinga uma velocidade qualquer, entre 0 e 2.5m/s, em uma determinada direção;
2. *Interceptação da bola*: o jogador aprende a interceptar a bola em movimento, levando em consideração o ambiente estocástico da RoboCup;
3. *Drible*: o jogador aprende a correr sem perder o controle sobre a bola;
4. *Para a bola*: o jogador aprende a para a bola que estejam em alta velocida, isto pode ser traduzido para o "futebolês" como "matar a bola".

Uma outra aplicação do aprendizado na RoboCup trata-se deste trabalho que utilizou o aprendizado por reforço para que o jogador aprendesse a sua posição em campo. Esta aplicação será apresentada na seção 6.3 do capítulo 6.

# Capítulo 6

# Otimização dos Parâmetros do Nível Instintivo

## 6.1 Introdução

No capítulo 4 a arquitetura do UFSC-Team foi apresentada a sua totalidade. Como apresentado na seção 4.3 o nível instintivo é responsável pela execução das metas locais do agente e pela geração da informação simbólica para atualização da base de conhecimento do nível cognitivo. A Figura 6.1 demonstra a arquitetura de um Sistema Especialista (SE).

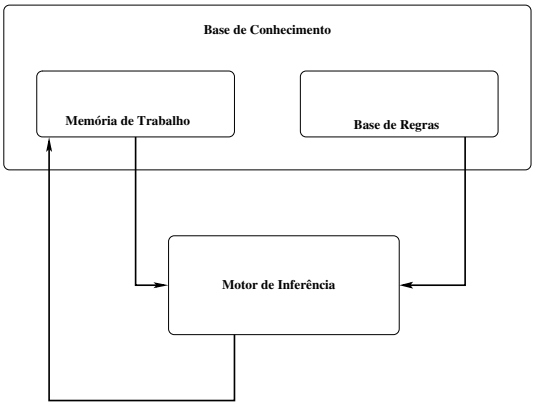


Figura 6.1: Arquitetura de um Sistema Especialista

No SE do nível instintivo, o motor de inferência é capaz de manipular fatos lógicos, frames contendo as informações visuais e mensagens no formato da linguagem Parla. Durante o processo de inferência os fatos da `visual_kb`, as mensagens recebidas do nível cognitivo e

do juiz e o conjunto de regras que determinam o comportamento reativo do agente. O resultado da inferência no SE desse nível é responsável pela escolha do comportamento reativo mais adequado a cada mudança de estado. Um estado do jogo é determinado de acordo com as condições impostas nas regras do SE do nível instintivo.

Este capítulo tratará especificamente da otimização através do aprendizado de máquina destas condições. As próximas seções trazem uma descrição do problema a ser tratado, da estrutura que foi montada para a execução deste aprendizado, um estudo de caso e a análise dos resultados obtidos neste estudo.

## 6.2 Estrutura de Aprendizado

A estrutura de aprendizado montada para que o UFSC-Team pudesse realizar treinamentos de jogadas dentro do futebol foi desenvolvida de uma forma modular. Um novo processo chamado de **RL** foi inserido na estrutura original do agente UFSC-Team. Nesta nova arquitetura o fluxo de informações entre os níveis decisórios não foi alterado. Simplesmente nesta nova arquitetura o nível instintivo além de enviar mensagens para o nível reativo, também envia estas mensagens para o processo RL, como demonstra a Figura 6.2

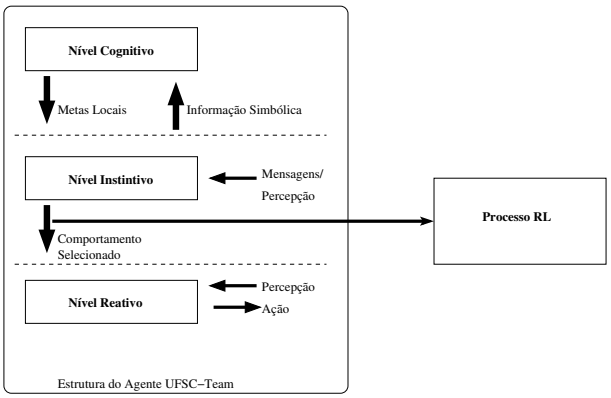


Figura 6.2: Arquitetura da estrutura de aprendizagem

Esta inserção não foi realizada de uma maneira simples, foi preciso antes um estudo completo da comunicação entre os processos existentes no UFSC-Team para que essa comunicação pudesse ser efetivada, o que acarretou alguns meses de estudos e tentativas frustadas. Já que por vezes, a comunicação não podia ser realizada, e o motivo não era tão claro, pois o processo RL utiliza a mesma abordagem de programação *multi-thread* utilizada nos níveis decisórios do UFSC-Team. A implementação foi feita utilizando a linguagem de programação C++, e integrando o ambiente para desenvolvimento de sistemas multiagentes

cognitivos sob restrições de tempo real chamado Expert-Coop++ (da Costa, 1997).

### 6.2.1 O Processo RL

Como já dito anteriormente, este processo utiliza uma abordagem de programação *multi-thread*, essa tecnologia permite particionar o processo e executar concorrentemente as partes resultantes. O processo RL é composto por dois *threads*. O primeiro responsável por manipular a interrupção SIGIO do UNIX, usada para informar que uma nova mensagem foi recebida pelo *socket* e por colocar essa nova mensagem no *mailbox* o outro *thread*, o principal, se responsabiliza pela execução de um um algoritmo de aprendizado (ver Figura 6.3). A exclusão mútua entre os dois *threads* é feita utilizando semáforos, evitando desta forma que o processo principal despenda tempo verificando a existência ou não de mensagens no socket.

Uma vez que a comunicação entre os processos antigos e o processo RL foi estabelecida este *mailbox* ficou responsável pela recepção e pelo ordenamento das mensagens recebidas do nível instintivo. Estas mensagens contêm as regras que foram inferidas pelo motor de inferência do nível instintivo. A partir disto, as informações contidas nestas regras (como por exemplo: o estado do jogo, qual regra foi inferida pelo sistema especialista, as distâncias do jogador em relação as *flags* do campo, entre outras) serão utilizadas ou como entrada, ou como padrões para a realização de um determinado aprendizado.

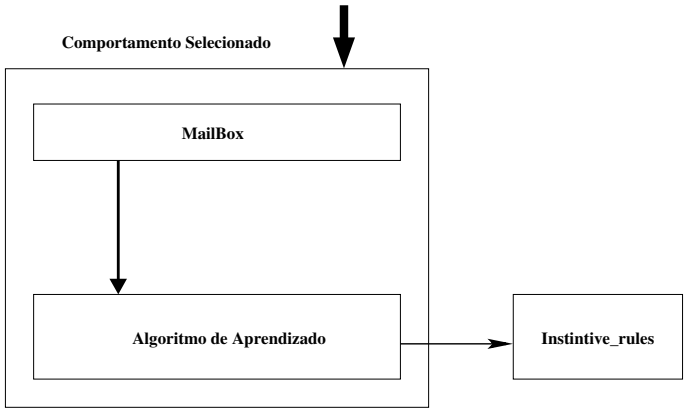


Figura 6.3: Processo RL

Com esta estrutura de aprendizado o UFSC-Team passa a ter uma autonomia maior, pois vários tipos de algoritmos podem ser utilizados para treinar várias situações, como por exemplo: chutes ao gol, defesas de pênaltis. Para isto basta que um problema seja escolhido, e um algoritmo que resolva este problema seja implementado nesta estrutura.

## 6.3 Estudo de caso

Como um exemplo do funcionamento desta estrutura, foi escolhido um problema dentro do UFSC-Team: aprendizado da localização do agente em campo. Na primeira versão do UFSC-Team este problema é tratado através de regras que serão julgadas pelo SE do nível instintivo, não cabe a este trabalho inovar a forma de como esta localização é feita, mas sim apresentar uma forma de melhorar esta implementada. Este problema a primeira vista, parece um problema simples, mas envolve uma série de fatores. Primeiro será exposto a maneira de como esta localização é feita.

Já na primeira versão do UFSC-Team, para facilitar a localização do agente no campo virtual, este foi divido em 12 sub-áreas como demonstra a Figura 6.4 (Gonçalves, 2001). Cada sub-área possui no mínimo 1 flag (descritas na seção 4.5.1) que auxilia o agente na sua localização.

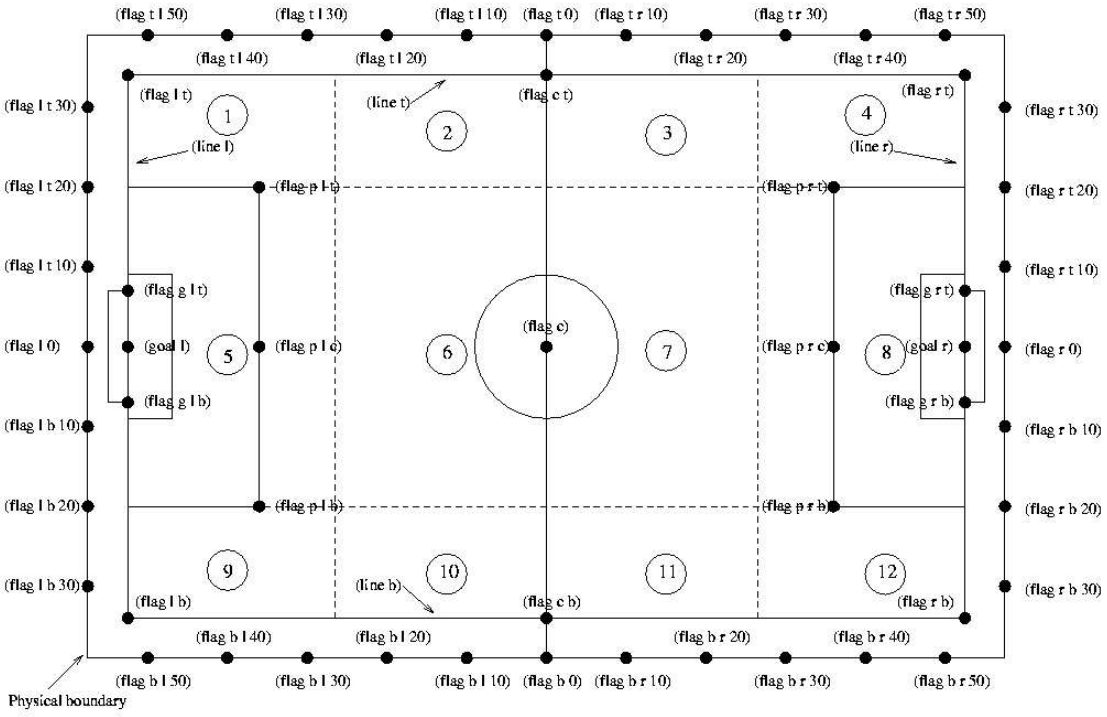


Figura 6.4: Divisão do Campo

A localização do campo é implementada através de regras no nível instintivo. O agente através da sua informação visual, identifica uma das flags, a partir disto, ele irá calcular sua distância desta flag, e esta informação irá disparar uma das regras do motor de inferência do nível instintivo. O Exemplo 6.1 traz uma regra que irá inferir que o jogador se encontra

na posição `attack_midfield_center`. As variáveis expostas no campo `filter` é que irão definir qual a posição do agente.

O valor desta distância foi definida na primeira versão do UFSC-Team apenas pela observação visual da divisão do campo, ocasionando as vezes, uma inferência errada sobre a posição do jogador. Seja $x$ = distância do jogador do objeto (flag p r c), e $y$ = distância do jogador do objeto (flag p r b). Se $10m < x < 37m$ e $10m < y < 37m$, então o jogador está na posição `attack_midfield_center` .

**Exemplo 6.1** (rule_004
(if (logic ( player position?x1 ))
( frame ( ?zy1 ((flag p r c)(distance ?x2 )))
( frame ( ?zy2 ((flag p r b) (distance ?x3 )))
(filter ( != ?x1 attack_midfield_center )
( < ?x2 37.0)
( > ?x2 10.0)
( < ?x3 37.0)
( > ?x3 10.0))
(then (logic ( player position attack_midfield_center))
(message ((to Expert) (from Coord) (deadline 0) (grade 0.0) (alpha 0.0) (round 0.0)
(body (INFORM ((logic (player position attack_midfield_center )))))))
(message ((to RL) (from Coord) (deadline 0) (grade 0.0) (alpha 0.0) (round 0.0)
(body (INFORM ((logic (attack_midfield_center ))))))))

Utilizando a estrutura de aprendizado mostrada na seção 6.2, foi implementado um algoritmo, baseado no aprendizado por reforço, ou seja, tentativa-erro, para que essa distância do agente até uma `flag` visualizada fosse um valor mais próximo do ideal, ou seja, valores que retornem o menor número erro de inferências possível. Como o aprendizado é similar para todas as sub-áreas, as sub-áreas foram "aprendidas" separadamente. Por este problema ser um processo não-markoviano, pois não possui uma função de transição entre os estados, os algoritmos convencionais empregados na aprendizagem por reforço não puderam ser utilizados. Desta forma um novo algoritmo foi idealizado neste trabalho para tratar do problema.

O algoritmo implementado para este aprendizado funciona da seguinte maneira, primeiro o processo RL irá verificar se existe uma mensagem no seu *mailbox*, se houver, ele irá processar esta mensagem (Exemplo 6.2). A informação que interessa ao processo RL nesta mensagem está em seu `body`, ou seja, a regra que foi inferida pelo motor de inferência. Após o processo RL receber a informação de qual regra foi disparada, ele vai descobrir qual a posição do campo esta regra diz respeito, a partir de agora o processo RL já tem conhecimento de qual posição foi inferida e qual regra foi disparada.

**Exemplo 6.2** (message ((to RL) (from Coord) (deadline 0) (grade 0.0) (alpha 0.0) (round 0.0)
(body (INFORM ((logic ( `posicao_inferida` ))))))))

O próximo passo é saber qual a posição real do jogador, isto é feito através da leitura de um arquivo externo que contem as coordenadas (x,y) que determinam esta posição. Estas coordenadas são gravadas neste arquivo pelo técnico, elas são geradas de uma forma aleatória, levando em consideração os limites de cada sub-área a ser treinada. Tendo conhecimento da posição inferida e da posição real, o algoritmo realiza uma comparação simples entre os dois valores, se este for correto, simplesmente o processo recebe uma recompensa positiva e incrementa o número de posições corretas. Caso contrário, o algoritmo irá receber uma recompensa negativa, re-calcular esses valores do `filter` e por fim alterar o arquivo `instintive_rules`, como demonstrado na Figura 6.3. O fluxo de informação do processo RL, está descrito abaixo:

**Algoritmo 3** Otimização das variáveis do nível instintivo

**Entrada:** comportamento selecionado no nível instintivo
**Saída:** Valores aproximados do ideal
**while** $mailbox > 0$ **do**
  Ler mensagem enviada pelo nível instintivo
  Ler posicao real (arquivo externo)
  **if** $posicao\_inferida = posicao\_real$ **then**
    recompensa++; FIM
  **else**
    recompensa–;
    recalcula-se as distâncias do jogador em relação ao objeto observado;
    altera-se o arquivo instintive_rules; FIM
  **end if**
**end while**

## 6.4 Resultados

Alguns testes foram efetuados depois da implementação do estudo de caso apresentado. O primeiro teste leva em consideração a área total do campo, e foi feito somente para atestar a necessidade de um aprendizado automático para a inferência das posições. A soma dos resultados corretos e incorretos pode ser maior que o número total de iterações, pois cada iteração equivale a um ciclo de simulação, onde podem ocorrer mais de uma inferência sobre a posição do agente em campo.

Os valores da tabela 6.1 comprovam o que já fora dito anteriormente, algumas inferências

| Área total do campo | | |
|---|---|---|
| Iterações | Acertos | Erros |
| 20 | 16 | 23 |
| 50 | 24 | 90 |
| 100 | 65 | 150 |

Tabela 6.1: Resultados obtidos antes do aprendizado

estavam erradas. Porém a grande maioria destes erros ocorriam em fronteiras de sub-áreas, somente algumas inferências estavam realmente erradas, como por exemplo, foi observado que o jogador se encontrava na posição `back_midfield_center` e foi inferido que ele se encontra na posição `atack_area_center`.

A tabela 6.2 traz os resultados antes da realização do na sub-área `attack_midfield_center`, esta sub-área foi escolhida pois trata-se de uma área onde os jogadores estão mais presente.

| Sub-Área attack_midfield_center | | |
|---|---|---|
| Iterações | Acertos | Erros |
| 20 | 10 | 20 |
| 50 | 12 | 30 |
| 100 | 20 | 35 |

Tabela 6.2: Resultados obtidos antes do aprendizado da sub-área `attack_midfield_center`

A tabela 6.3 traz os números após o aprendizado ter sido aplicado, verificou-se uma grande melhora no número de acertos, e nestas iterações não foi encontrada nenhum tipo de inferência totalmente errada, como observado antes do aprendizado ser empregado.

| Sub-Área attack_midfield_center | | |
|---|---|---|
| Iterações | Acertos | Erros |
| 20 | 15 | 7 |
| 50 | 20 | 9 |
| 100 | 40 | 17 |

Tabela 6.3: Resultados obtidos após o aprendizado na sub-área `attack_midfield_center`

# Capítulo 7

# Conclusão

Com o intuito de otimizar os parâmetros do nível instintivo do agente jogador do time de robôs UFSC-Team, este projeto realizou a implementação de uma estrutura de aprendizado que vai permitir que esses parâmetros sejam otimizados até um valor o mais próximo do ideal possível. Esta estrutura é composta por um *mailbox* que recebe as mensagens oriundas do nível instintivo, e um algoritmo que efetua o aprendizado sobre os parâmetros citados acima.

De modo a concretizar a implementação desta estrutura de aprendizado, primeiro foi necessário um estudo sobre técnicas de aprendizado de máquina, em especial árvores de decisão, redes neuronais e aprendizado por reforço, sendo que a técnica selecionada foi esta última, pois esta trata melhor os processos onde não se tem um conhecimento do ambiente onde o agente está inserido. O problema escolhido para verificação da funcionalidade da estrutura de aprendizado foi o do aprendizado da localização do jogador do UFSC-TEAM dentro do campo, por tratar-se de um problema onde os resultados podem influenciar diretamente no funcionamento do time.

A implementação do aprendizado da localização do agente se deu através de um algoritmo do aprendizado baseado em reforço, uma vez que a literatura (Stolzenburg et al., 2002), (Stone e McAllester, 2000), (Stone e Sutton, ) tem citado com um dos recursos que tem apresentado bons resultados no âmbito da RoboCup.

O nível instintivo é responsável tanto pela execução das metas locais do agente como pela geração da informação simbólica que atualiza a base de conhecimento do nível cognitivo. É implementado através de um sistema especialista de um ciclo de inferência único que escolhe, a cada vez que o estado do jogo muda, o comportamento reativo mais adequado dada a atual meta local. Neste trabalho o sistema especialista foi responsável simplesmente por identificar a posição do campo onde o jogador se encontra, e a cada inferência errada, o

algoritmo de aprendizado implementado dentro do processo RL alterava a base de regras a fim de otimizar os valores do filtro de cada regra.

Após realizados alguns testes, esta otimização foi verificada através de simulações que após o aprendizado ter sido efetuado, o número de inferências incorretas foi reduzido, com exposto nas tabelas 6.2 e 6.3 do capítulo 6. A sub-área `attack_midfield_center` foi a única a ser treinada, mas o aprendizado para as outras sub-áreas pode ser efetuado facilmente, para isto basta modificar no processo `tecnico` os valores que delimitam as sub-áreas a serem treinadas.

## 7.1 Trabalhos Futuros

Como propostas de trabalhos futuros, pode-se citar:

- Um trabalho que pode ser realizado é o treinamento de todas as sub-áreas.
- Utilizar outras técnicas de aprendizado, como por exemplo, redes bayesianas + Q-Learning para resolver alguns problemas como: drilbe, chute, conduzir a bola (Tuyls et al., ).
- Utilizar uma abordagem numérica para aprender o ponto exato para uma interceptação de bola, como exposto no trabalho de (Stolzenburg et al., 2002).
- Utilizar técnicas de aprendizado por refoço para o treinamento de goleiros, e para jogadores que irão ter que chutar ao gol (pênalt).

# Apêndice A

# Árvores de Decisão

Árvore de decisão é um dos algoritmos de aprendizado mais simples e apesar disso é considerado o mais próspero algoritmo de aprendizado. Uma árvore de decisão toma como entrada situações descritas por conjuntos de atributos. Como saída têm-se funções booleanas, isto é, SIM ou NÃO. Cada nó interno da árvore corresponde ao valor do teste de uma atributo. Os galhos correspondem aos possíveis valores dos testes e as folhas são as saídas. A Figura A.1 ilustra uma árvore de decisão para o problema de se esperar por uma mesa em um restaurante.

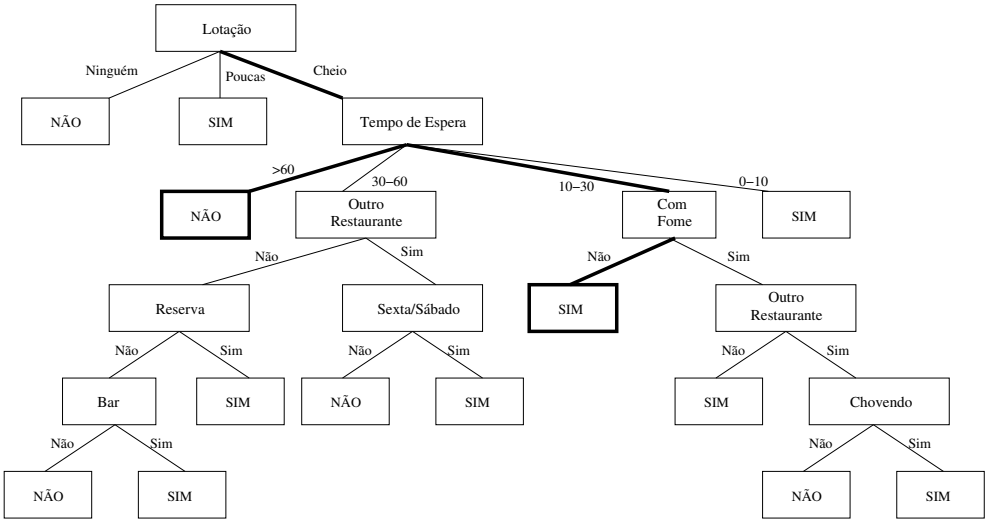


Figura A.1: Ilustração de uma árvore de decisão

Neste exemplo, a meta a se aprender é: se a pessoa irá ou não esperar por uma mesa. Pode-se sugerir a seguinte lista de atributos:

1. Outro Restaurante: se existe um outro restaurante pelas redondezas;

2. Bar: se o restaurante possui um bar, onde se possa esperar por uma mesa;
3. Sexta/Sábado: se o dia da semana for Sexta ou Sábado;
4. Com Fome: se a pessoa está com fome;
5. Lotação: quantas pessoas estão no restaurante;
6. Preço: o preço médio que o restaurante cobra;
7. Chovendo: se está chovendo;
8. Reserva: se a pessoa fez reserva;
9. Tipo: qual o tipo de comida que o restaurante serve;
10. Tempo de Espera: o tempo de espera estimado pelo anfitrião.

Como observado, nem todos os atributos foram utilizados na construção da árvore, isto se deve a irrelevância destes atributos diante da meta especificada pelo exemplo. As árvores de decisão podem também ser expressas de forma lógica ou como um conjunto de regras. Por exemplo, na árvore mostrada na Figura A.1 os caminhos destacados podem ser lidos da seguinte maneira na forma lógica:

**Caminho 1** $\forall r\ \text{Lotação}(r, Cheia) \wedge \text{TempodeEspera}(r, 10-30) \wedge \text{ComFome}(r, N) \Rightarrow \text{Espera}(r)$

**Caminho 2** $\forall r\ \text{Lotação}(r, Cheia) \wedge \text{TempodeEspera}(r, > 60) \Rightarrow \text{Não Espera}(r)$

Na representação como conjunto de regras, cada regra teria seu início na raiz da árvore e caminharia até uma de suas folhas, ou uma forma alternativa seria representar cada nó separadamente. Segue abaixo a representação dos caminhos na forma de conjunto de regras:

**Caminho 1**

**se** Lotação = cheio **e** Tempo de Espera = 10 - 30 **e** Com Fome = Não **então**
saída = Sim

**Caminho 2**

**se** Lotação = cheio **e** Tempo de Espera = > 60 **então**
saída = Não

Apesar da simplicidade e da fácil implementação, árvores de decisão não são utilizadas para todo tipo de aprendizado, isto porque uma árvore de decisão trata apenas de um objeto. Não se pode representar um teste que se refira a dois ou mais objetos diferentes. Por exemplo, se no caso do restaurante, fosse utilizada a meta: **Restaurante Perto e Barato** (se existe um restaurante perto e barato), teria-se de adicionar um outro atributo **RestPertoBarato** para tratar o caso, mas é impraticável adicionar todos os tipos de atributos. As árvores de decisão são totalmente expressivas dentro de uma classe de linguagem proposicional, isto é, qualquer função booleana pode ser tratada com árvores de decisão, basta representar cada "disputa" na tabela da verdade para a função correspondente ao caminho da árvore.

A idéia básica através do algoritmo de aprendizado através de árvores de decisão é testar o atributo mais significativo primeiro. Quando se fala em atributo mais significativo, diz-se respeito ao atributo que faz a maior distinção a meta a ser atingida em um exemplo. Algumas possibilidades para escolher este atributo, segundo (Monard e Baranauskas, 2003) são:

- aleatória: seleciona qualquer atributo aleatoriamente;
- menos valores: seleciona o atributo com a menor quantidade de valores;
- ganho máximo: seleciona o atributo que possui o maior ganho de informação esperado.

Um algoritmo de aprendizado é bom quando ele prediz corretamente a classificação de exemplos não vistos. Para avaliar o desempenho de um algoritmo de aprendizado, pode seguir a metodologia dada por (Russell e Norvig, 1996):

1. Reunir um grande conjunto de exemplos;
2. Dividir este conjunto em dois sub-conjuntos: o **conjunto de treinamento** e o **conjunto de teste**;
3. Aplicar o algoritmo de aprendizado nos exemplos do conjunto de treinamento para gerar a hipótese $H$;
4. Medir a porcentagem de exemplos no conjunto de teste que são classificadas corretamente pela hipótese $H$;
5. Repetir os passos 1-4 para diferentes tamanhos de conjuntos de treinamento.

## A.1 Aplicações de Árvores de Decisão

- **Gasoil - um equipamento para plataformas de petróleo**

Em 1986, BP[1] desenvolveu um sistema inteligente chamado *Gasoil*, para projetar sistemas de separação de gás e óleo em plataformas de óleo. A separação do óleo e do gás é feita na raiz por um sistema de separação muito grande, complexo e caro. O projeto do sistema depende de vários atributos. O *Gasoil* é um dos maiores sistemas comerciais no mundo, ele contêm aproximadamente 2500 regras.

- **Simulador de vôo**

Em (Sammut et al., 1992) foi desenvolvido um simulador de vôo de um avião modelo Cessna. Os dados do sistema foram gerados observando 30 execuções de um mesmo plano de vôo, realizado por três experientes pilotos. Cada vez que um piloto tomava uma ação de controle, como propulsão ou ajuste dos *flaps*, um programa filtrava os registros de cada ação tomada, gerando arquivos de entrada para um programa de indução. Assim, a cada ação tomada um novo exemplo de treinamento era criado. Ao todo, 90.000 exemplos de treinamento foram obtidos, cada um descrito por 20 variáveis de estado e rotulados pela ação que havia sido tomada. Após esses dados serem processados, foram submetidos ao sistema C4.5 (Quinlan, 1987) para que a partir de cada exemplo uma árvore de decisão fosse construída. As regras construídas no C4.5 são puramente reativas. Após a construção de uma árvore de decisão, esta é convertida em código C e é inserida no controle do simulador de vôo.

Os resultados deste projeto foram surpreendentes: o simulador desenvolvido por Sammut et al. não apenas aprende a voar, como também aprende a voar de certa forma "melhor" que seus professores. Isto porque no processo de generalização são extingüidos quaisquer erros ocasionais que possam ser cometidos por humanos.

[1] A BP é uma companhia internacional, que opera em 70 países. Suas produção está voltada para a exploração e produção de gás e óleo; no refinamento, no marketing e na produção de produtos de petroquímicos. Mais informações em http://www.bp.com

# Apêndice B

# Redes Neuronais

Redes Neuronais é uma das duas grandes linhas de pesquisa da IA e tem por objetivo investigar a possibilidade de simulação de comportamentos inteligentes através de modelos baseados na estrutura e funcionamento do cérebro humano. Os primeiros trabalhos desenvolvidos na área datam de 1943, quando o neurofisiologista, filósofo e poeta americano Warren McCulloch, e o lógico Walter Pitts desenvolveram o primeiro modelo matemático de um neurônio (Bittencourt, 2001).

Redes Neuronais são sistemas paralelos distribuídos compostos por unidades de processamento simples (nodos) que computam determinadas funções matemáticas. Tais unidades são dispostas em uma ou mais camadas e interligadas por um grande número de conexões, geralmente unidirecionais. Na maioria dos modelos estas conexões estão associadas a pesos, os quais armazenam o conhecimento representado no modelo e servem para ponderar as entradas recebidas por cada neurônio da rede. O funcionamento destas redes é inspirado em uma estrutura física natural: o cérebro humano.

## B.1 O modelo biológico e o modelo de McCulloch e Pitts

Um neurônio é uma célula biológica especial que processa informações. Um neurônio recebe sinais (impulsos) provenientes de outros neurônios através de seus dendritos (receptores) e transmite os sinais gerados pelo corpo celular através do axônio (transmissor), o qual eventualmente se ramifica. No final destas ramificações estão as *sinapses*. Uma sinapse é uma estrutura elementar e unidade funcional entre dois neurônios. Quando o impulso alcança o terminal da sinapse certos elementos químicos chamados neurotransmissores são liderados, eles se difundem através do espaço sináptico para fortalecer ou inibir, dependendo

do tipo de sinapse, a tendência intríseca do neurônio receptor em emitir impulsos elétricos. As sinapses tem um papel fundamental na memorização da informação e são principalmente as do córtex cerebral e algumas vezes de partes mais profundas do cérebro que armazenam esta informação (Barreto, 1999)

A estrutura do neurônio artificial proposto por McCulloch e Pitts (McCulloch e Pitts, 1943) é baseada no neurônio biológico. Este neurônio matemático computa uma soma ponderada de seus $n$ sinais de entrada, $x_j$, $j = 1,2,...,n$ e gera uma saída com valor 1 se a soma estiver acima de um determinado limiar $f(x)$. Caso contrário obtém-se uma saída igual a 0. Matematicamente tem-se:

$$y = f(x)\left(\sum_{j=1}^{n} w_j x_j - \alpha\right) \tag{B.1}$$

onde $y$ é uma função pulso unitário em 0, $w_j$ é o peso associado a $j$-ésima entrada. McCulloch e Pitts provaram que, em princípio, com uma escolha apropriada de pesos, um arranjo síncrono de tais neurônios realiza computações universais. Formalmente, este funcionamento pode ser descrito da seguinte maneira. Considere a $i$-ésimo neurônio de uma rede neuronal com $n$ neurônios. Este neurônio é caracterizado pelo valor $x_i$, chamado *atividade* do neurônio (que correponde à taxa média de disparos dos *potenciais* de *ação* do neurônio biológico) e pelo valor $\sigma_i$, chamado *nível de ativação* do neurônio. No modelo de McCulloch e Pitts, o nível de ativação é definido da seguinte maneira:

$$\sigma_i = \sum_{j=1}^{n} w_{ij} x_j \tag{B.2}$$

onde $w_{ij} \in \Re$ é o peso atribuído àquela entrada do neurônio $i$ cuja origem é a atividade do neurônio $j$. Assim como no caso biológico, também para as redes neuronais artificiais há dois tipos de sinapses: as excitadoras e a inibidoras. Pesos positivos correspondem a sinapses *excitadoras*, enquanto que pesos negativos indicam sinapses *inibidoras*. A atividade de um neurônio $i$ é dada por $x_i = f(\sigma_i)$. A função $f$, chamada *função de ativação* ou de *transferência*, adotada no modelo, é a *função degrau*:

$$f(x) = \begin{cases} 0 & \text{se} \quad x \leq \alpha \\ 1 & \text{se} \quad x > \alpha. \end{cases} \tag{B.3}$$

onde $\alpha$ é o limite de disparo. A maior limitação do modelo de neurônio de McCulloch e Pitts é sua natureza binária.

Mesmo com este modelo rudimentar de neurônio, McCulloch e Pitts foram capazes de provar que uma rede neuronal é equivalente a uma máquina de Turing e, logo, capaz de calcular qualquer função computável (Bittencourt, 2001).

## B.2 Função de Ativação

Em um modelo genérico de um neurônio, a função de ativação $f$, que determina a atividade de um neurônio, é generalizada e passa a ser uma função limitada qualquer. É interessante que esta função seja não linear, pois neste caso as restrições do modelo binário de McCulloch e Pitts desaparece. É introduzido um valor de polarização $\theta \in \Re$, de modo que a atividade de um neurônio passa a ser calculada por

$$x_i = f(\sigma_i + \theta). \tag{B.4}$$

Na maioria dos modelos, a função $f$ é, da mesma maneira que no modelo de McCulloch e Pitts, simplesmente a soma ponderada, embora existam modelos onde é utilizado o produto, o mínimo ou o máximo. As funções $f$ mais utilizadas, além da função degrau, são:

- *função semi-linear*:

$$f(x) = \begin{cases} 0 & \text{se} \quad x < \alpha_{min} \\ mx + l & \text{se} \quad \alpha_{min} \leq x \leq \alpha_{max} \\ f_{max} & \text{se} x > \alpha_{max}. \end{cases} \tag{B.5}$$

- *função sigmoidal*:

$$f(x) = \frac{f_{max}}{1 + e^{-x}} \tag{B.6}$$

## B.3 Aprendizado em Redes Neuronais

O aprendizado conexionista é em geral um processo gradual e iterativo. Diversos métodos para treinamento de redes foram desenvolvidos, podendo estes serem agrupados em dois paradigmas principais. O *Aprendizado Supervisionado* é o mais comum no treinamento das RNAs, recebe este nome pois durante o aprendizado, a saída desejada para um dado exemplo é fornecida por um supervisor (professor) externo. O objetivo é ajustar os parâmetros da

rede, de forma a encontrar uma ligação entre os pares de entrada e saída fornecidos. A Figura B.1 ilustra o mecanismo de aprendizado supervisionado. O professor indica, explicitamente, um comportamento bom ou ruim para a rede, visando direcionar o processo de treinamento. A rede tem sua saída corrente (calculada) comparada com a saída desejada, recebendo informações do supervisor sobre o erro da resposta atual. A cada padrão de entrada submetido à rede compara-se a resposta desejada (que representa uma ação ótima a ser realizada pela rede) com a resposta calculada, e os pesos das conexões são ajustados para minimizar o erro. A minimização da diferença é incremental, já que pequenos ajustes são feitos nos pesos à cada etapa de treinamento, de tal forma que estes caminhem, se possível, para uma solução. A desvantagen do aprendizado supervisionado é que, na ausência do professor, a rede não conseguirá aprender novas estratégias para situações não cobertas pelos exemplos do treinamento da rede.

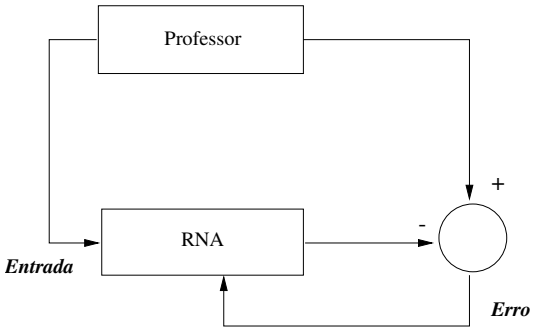


Figura B.1: Aprendizado Supervisionado.

No *Aprendizado Não Supervisionado*, como o próprio nome sugere, não há um professor ou supervisor para acompanhar o processo de aprendizado. Este método é ilustrado na Figura B.2. Para este tipo de aprendizado, somente os padrões de entrada estão disponíveis para a rede, ao contrário do aprendizado supervisionado, cujo conjunto de treinamento possui pares de entrada e saída. Este tipo de aprendizado, só se torna possível, quando existe redundância nos dados de entrada. Sem redundância seria impossível encontrar quaisquer padrões ou características dos dados de entrada. A desvantagem deste método, está intimamente ligada ao conjunto de dados para treinamento, que deve ser redundante para que a rede consiga abstrair características em seu treinamento.

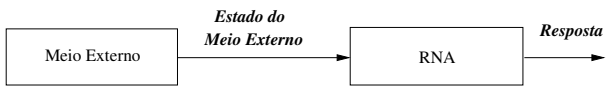


Figura B.2: Aprendizado Não Supervisionado.

## B.4 Principais Modelos de Redes Neuronais

Uma rede neuronal é caracterizada pela topologia da rede, pelo valor do peso da conexão entre os pares de neurônios, pelas propriedades dos nós, e pelas regras de atualização de estado. A definição da arquitetura de uma RNA é um parâmetro importante na sua concepção, uma vez que ela restringe o tipo de problema que pode ser tratado pela rede. Fazem parte da definição da arquitetura os seguintes parâmetros: número de camadas da rede (redes de camada única ou redes de múltiplas camadas), número de nodos em cada camada, tipo de conexão entre os nodos (acíclica ou cíclica). As redes também podem ser classificadas de acordo com a conectividade (rede fracamente ou fortemente conectada) Existem muitos modelos conexionistas, e uma quantidade razoável de publicações que se dedicam a classificá-los. Dentre este modelos serão expostos a seguir os 5 principais.

### B.4.1 Perceptron

Este modelo foi proposto por F. Rosenblatt em 1975. O perceptron de duas camadas que pode ser usado com valores contínuos foi o primeiro modelo conexionista desenvolvido. Suas principais características são:

- Classe de tarefas: reconhecimento de padrões;
- Propriedade dos neurônios: possuem entradas binárias e saídas que assumem os valores +1 ou -1;
- Função de ativação: função degrau;
- Propriedades da rede: rede acíclica de duas camadas;
- Aprendizado: Aprendizado por Reforço.

Esta rede gerou muito interesse pela habilidade de aprender a reconhecer padrões linearmente separáveis. Contudo, como a grande maioria dos problemas práticos relevantes não são linearmente separáveis, o perceptron não tem uso generalizado. O aprendizado por reforço aprendizado por reforço o desempenho é baseado em qualquer medida que possa ser fornecida ao sistema. No aprendizado por reforço, a única informação de realimentação fornecida à rede é se uma determinada saída está correta ou não, isto é, não é fornecida a rede a resposta correta para o padrão de entrada. O Aprendizado por Reforço é ilustrado na Figura B.3.

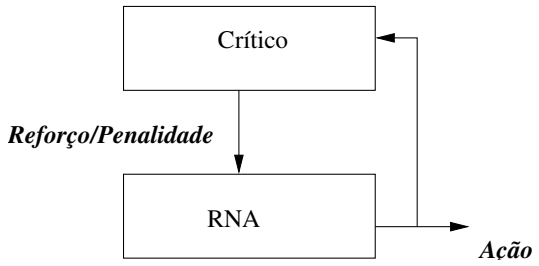


Figura B.3: Aprendizado por Reforço.

### B.4.2 Perceptrons Multi-Camadas

Foram desenvolvidos no começo da década de 70, são redes acíclicas com uma ou mais camadas de neurônios intermediários entre as camadas de entrada e saída. Um algoritmo capaz de treinar os perceptrons multi-camadas é o *backpropagation* (Rumelhart et al., 1986). Suas principais características são:

- Classe de tarefas: reconhecimento de padrões;
- Propriedades dos neurônios: são do mesmo tipo utilizado no perceptron e possuem valores contínuos;
- Função de Ativação: função sigmóide;
- Propriedades das redes: rede acíclica de três camadas (no mínimo);
- Aprendizado: utiliza a técnica de correção de erros (aprendizado supervisionado que ajusta os pesos das conexões entre nós, na proporção da diferença entre os valores desejados e computados de cada neurônio da camada de saída).

O algoritmo *backpropagation* foi testado em uma série de problemas clássicos, e em problemas relacionados com reconhecimento de padrões visuais. Na maioria dos casos, ele encontrou boas soluções para os problemas propostos.

### B.4.3 Classificador de Carpenter-Grossbert (Sistema ART)

No final da década de 70, Carpenter e Grossberg projetaram uma rede capaz de formar aglomerados de informações, e de ser treinada sem supervisão, chamada de sistema ART. Suas principais características são:

- Classe de tarefas: reconhecimento de padrões e processamento de imagens;

- Propriedades dos neurônios: possuem entradas binárias, podendo assumir valores contínuos;
- Função de ativação: função sigmóide;
- Propriedades das redes: rede acíclica de três camadas;
- Aprendizado: utiliza técnica de aprendizado por competição;

A idéia deste aprendizado é, dado um padrão de entrada, fazer com que as unidades de saída disputem entre si para serem ativadas. Existe, portanto, uma competição entre as unidades de saída para decidir qual delas será a vencedora e, consequentemente, terá a sua saída ativada e seus pesos atualizados no treinamento. As unidades de entrada são diretamente conectadas às unidades de saída, sendo que estas últimas também podem estar ligadas entre si via conexões laterais inibitórias, ou negativas. A unidade de saída com maior ativação inicial terá mais chance de vencer a disputa com as outras unidades, que perderão o poder de inibição ao longo do tempo sobre as unidades de maior ativação. A unidade mais forte fica ainda mais forte e seu efeito inibidor sobre as outras unidades de saída torna-se dominante. Com o tempo, todas as outras unidades de saída ficarão completamente inativas, exceto a vencedora.

### B.4.4 Rede de Kohonen

No começo da década de 80 este tipo de rede, proposta por Kohonen em 1982, colaborou com os estudos teóricos sobre a organização dos caminhos de sensoriamento na mente. Segundo a teoria de Kohonen, o cérebro humano é uma coleção estruturada de neurônios. Com isto, foi admitida uma ordem espacial das unidades de processamento que permitiu elaborar uma rede neuronal dotada de mecanismos que permitem formar representações estruturadas dos estímulos de entrada (Kohonen, 1982). A seguir são apresentadas suas principais características.

- Classe de tarefas: reconhecimento de padrões e aprendizado da distribuição de probabilidades dos dados;
- Propriedades dos neurônios: possuem entradas contínuas;
- Função de ativação: função sigmóide;
- Propriedades das redes: rede cíclica de duas camadas;
- Aprendizado: utiliza técnica de SCA, que foi introduzida pelo próprio Kohonen.

### B.4.5 Rede de Hopfield

Apresentada por Hopfield em 1982 (Hopfield, 1982), estas redes são mais indicadas quando representações binárias permitem modelar a situação desejada. Por exemplo, imagens branco e preto, onde os elementos de entrada podem ser representados pelos valores de cada ponto da imagem, 0 para branco e 1 para preto. Suas características são:

- Classe de tarefas: reconhecimento de padrões e memória associativa;
- Propriedades dos neurônios: possuem entradas binárias e saídas que assumem valores +1 ou -1.
- Função de ativação: função sigmóide;
- Propriedades das redes: rece cíclica de uma camada;
- Aprendizado: os padrões são armazenados no começo.

Pelo fato da rede de Hopfield ser do tipo binária, a primeira atitude a ser tomada antes do aprendizado e também da fase de reconhecimento da rede, é converter os valores binários (0,1) em bipolares (-1,1), para que o valor 0 não cause problemas quanto o cálculo das saídas.

# Referências Bibliográficas


(2002). *The goals of RoboCup*. RoboCup Federation, http://www.robocup.org/overview/22.html. Acessado em 20 de Janeiro de 2003.

(2002). *Laws of the game*. FIFA, http://www.fifa.com. Acessado em 20 de Janeiro de 2003.

Barreto, J. M. (1999). Inteligência artificial no limiar do século xxi. Impresso pr Duplic - Prestação de Serviços. 2ª edição.

Bittencourt, G. (2001). *Inteligência Artificial: ferramentas e teorias*. Editora da UFSC, Florianópolis, 2ª edition.

Bittencourt, G. e da Costa, A. C. P. L. (2001). Hybrid cognitive model. In *Workshop on Cognitive Agents and Multi-Agent Interaction, at The Third International Conference on Cognitive Science (ICCS'2001)*, http://www.das.ufsc.br/ gb.

Brooks, R. (1986). A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1):435–453.

Chen, M., Foroughi, E., Heintz, F., e Huang, Z. (2001). *RoboCup Soccer Server*. The RoboCup Federation, http://sserver.sourceforge.net, 1.1 edition. Users Manual for Soccer Server Version 7.06 or later.

da Costa, A. C. P. L. (1997). Um ambiente para desenvolvimento de sistemas multi-agentes cognitivos. Master's thesis, Universidade Federal de Santa Catarina.

da Costa, A. C. P. L. (2001). *Conhecimento Social Dinâmico: Uma Estratégia de Cooperação para Sistemas Multiagentes Cognitivos*. PhD thesis, Universidade Federal de Santa Catarina.

Etzioni, O. e Weld, D. S. (1995). Intelligent agents on the internet: Fact, fiction, and forecast. *IEEE Expert*, 10(3):44–49.

Faria, G. e Romero, R. F. (1999). Explorando o potencial de algoritmos de aprenzado com reforço em robôs móveis. In *Proceedings of the IV Brazilian Conference on Neural Networks*, pages 237–242, ITA - São José dos Campos - SP.

Gonçalves, E. M. N. (2001). Otimização de controladores nebulosos e sistemas especialistas reativos utilizando algoritmos genéticos. Master's thesis, Universidade Federal de Santa Catarina.

Hopfield, J. (1982). Neural networks and physical systems with emergent collectives computational abilities. In *Proceedings of the National Academy of Sciences 79*, pages 2554–2558, USA.

Junius, M. e Stepple, M. (1997). *CNCL Reference Manual*. ComNets, http://www.comnets.rwth-aachen.de. Acessado em 30 de Janeiro de 2003.

Kaelbling, L. P., Littman, M. L., e Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence*, Research 4:237–285.

Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., e Osawa, E. (1997). Robocup: The robot world cup initiative. In Johnson, W. L. e Hayes-Roth, B., editors, *Proceedings of the First International Conference on Autonomous Agents (Agents'97)*, pages 340–347, New York. ACM Press.

Kohonen, T. (1982). Clustering taxonomy and topological maps of patterns. In *Sixth International Conference on Pattern Recognition*, pages 114–128, Munich, Germany.

Marietto, M. G. B. (2000). *Definição Dinâmica de Estratégias Instrucionais em Sistemas de Tutoria Inteligente: Uma Abordagem Multiagentes na WWW.* Tese de doutorado, Intituto Tecnológico da Aeronáutica - ITA.

McCulloch, W. S. e Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. In *Bulletin of Mathematical Biophysics*, volume 5, pages 115–133, Oxford, UK.

Monard, M. C. e Baranauskas, J. A. (2003). *Sistemas Inteligentes: Fundamentos e Aplicações*, chapter 4, pages 89–139. Editora Manole, 1 edition. Organizado por: Solange Oliveira Rezende.

Monard, M. C., de Almeida Prado Alves Batista, G. E., Kawamoto, S., e Pugliesi, J. B. (2000). Uma introdução ao aprendizado simbólico de máquina por exemplos. http://labic.icmc.sc.usp.br/didatico/PostScript/ML.html. acessada em 25/03/2003.

Noda, I. (1995). Soccer server: a simulator of robocup. In *In Proceedings of Artificial Intelligent Symposium '95*, pages 29–34, Japan. Japanese Society for Artificial Intelligence. citeseer.nj.nec.com/noda95soccer.html.

Quinlan, J. R. (1987). Simplifying decision trees. *International Journal of Man-Machine Studies*, 27(03):221–234.

Reynolds, S. I. (2002). *Reinforcement Learning with Exploration*. Pd. d thesis, School of Computer Science. The University of Birmingham, United Kingdom.

Riedmiller, M., Merke, A., Meier, D., Hoffmann, A., Sinner, A., Thate, O., e Ehrmann, R. Karlsruhe brainstormers - a reinforcement learning approach to robotic soccer.

Rumelhart, D. E., Hinton, G. E., e Williams, R. J. (1986). Learning internal representations by error propagation. In *Parallel Distributed Processing*, volume 1, pages 318–362. MIT Press.

Russell, S. e Norvig, P. (1996). *Artificial Intelligence: A Modern Approach*. Prentice Hall, 1 edition.

Sammut, C., Hurst, S., Kedzier, D., e Michie, D. (1992). Learning to fly. In *Proceedings of the Ninth International Conference on Machine Learning*, Aberdeen. Morgan Kaufmann.

Shabani, J., Honarvar, A., Morovati, M., Abdollahian, G., e Mahmoodi, M. (2003). Reinforcement learning in soccer simulation. In *Workshop on Adaptability in Multi-Agent Systems and The First RoboCup Australian Open 2003 (AORC-2003)*, Sidney. Commonwealth Scientific and Industrial Research Organization.

Sichman, J. S., Demazeau, Y., e Boissier, O. (1992). When can knowledge-based systems be called agents? In *IX Simpósio Brasileiro de Inteligência Artificial (SBIA)*, pages 172–185, Rio de Janeiro.

Sloman, A. (1999). What sort of architecture is required for a human-like agent. In Wooldridge, M. e Rao, A., editors, *Foundations of rational agency*, pages 35–53. Kluwer Academic Publishers, http://www.citeseer.nj.nec.com/sloman98what.html.

Stolzenburg, F., Obst, O., e Murray, J. (2002). Qualitative velocity and ball interception. Poster on Spatial Cognition III. Tutzing, Germany.

Stone, P. e McAllester, D. (2000). An architecture for action selectin in robotic soccer. In *Fifth International Conference on Autonomous Agents*.

Stone, P. e Sutton, R. Keepaway soccer: a machine learning testbed. AT & T Labs - Research.

Sutton, R. e Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. MIT Press, Cambrigde, MA. http://www-anw.cs.umass.edu/ rich/book/the-book.html.

ten Hagen, S. e Kröse, B. (1997). A short introduction to reinforcement learning. In Daelemans, W., Flach, P., e van den Bosch, A., editors, *Proc. of the 7th Belgian-Dutch Conf. on Machine Learning*, pages 7–12, Tilburg.

Tuyls, K., Maes, S., e Manderick, B. Q-learning in simulated robotic soccer large state spaces and incomplete information. http://www.citeseer.nj.nec.com/531464.html. Departament of Computer Science, Vrije Universiteit Brussell.

Watkins, C. J. C. H. (1989). *Learning from Delayed Rewards*. Ph.d thesis, King's College, Cambrigde, UK.