

Henrique de Castro Neto

**LS-DRAUGHTS – *Um Sistema de
Aprendizagem de jogos de Damas baseado
em Algoritmos Genéticos, Redes Neurais e
Diferenças Temporais***

Uberlândia - MG

2007

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Autor: **Henrique de Castro Neto**

Titulo: ***LS-DRAUGHTS*** – Um Sistema de Aprendizagem de jogos de Damas baseado em Algoritmos Genéticos, Redes Neurais e Diferenças Temporais

Faculdade: **Faculdade de Computação**

Copyright 2007

Fica garantido à Universidade Federal de Uberlândia o direito de circulação e impressão deste material para fins não comerciais, bem como o direito de distribuição por solicitação de qualquer pessoa ou instituição

Henrique de Castro Neto

**LS-DRAUGHTS – *Um Sistema de
Aprendizagem de jogos de Damas baseado
em Algoritmos Genéticos, Redes Neurais e
Diferenças Temporais***

Dissertação apresentada à Coordenação do
Mestrado em Ciência da Computação da
Universidade Federal de Uberlândia para a
obtenção do título de Mestre em Ciência da
Computação.

Orientadora:
Profa. Dra. Rita Maria da Silva Julia

MESTRADO EM CIÊNCIA DA COMPUTAÇÃO
FACULDADE DE COMPUTAÇÃO
UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Uberlândia – MG

Janeiro / 2007

Dados Internacionais de Catalogação na Publicação (CIP)

- C355L Castro Neto, Henrique de, 1981-
LS-DRAUGHTS – um sistema de aprendizagem de jogos de damas
baseado em algoritmos genéticos, redes neurais e diferenças temporais /
Henrique de Castro Neto. - 2007.
118 f. : il.
- Orientadora: Rita Maria da Silva Julia.
Dissertação (mestrado) – Universidade Federal de Uberlândia, Progra-
ma de Pós-Graduação em Ciência da Computação.
Inclui bibliografia.
1. Aprendizado do computador - Teses. 2. Algoritmos genéticos - Te-
ses. 4. Inteligência artificial - Teses. I. Julia, Rita Maria da Silva. II. Uni-
versidade Federal de Uberlândia. Programa de Pós-Graduação em Ciên-
cia da Computação. III. Título.

CDU: 681.3 : 007.52

Elaborado pelo Sistema de Bibliotecas da UFU / Setor de Catalogação e Classificação

Dissertação apresentada ao Programa de Pós-Graduação da Faculdade de Ciência da Computação da Universidade Federal de Uberlândia como requisito para obtenção do grau de Mestre em Ciência da Computação.:

Profa. Dra. Rita Maria da Silva Julia
Orientador

Prof. Dr. Geber Lisboa Ramalho
Universidade Federal de Pernambuco UFPE/PE

Profa. Dra. Gina Maira Barbosa de Oliveira
Universidade Federal de Uberlândia UFU/MG

*Dedico esta dissertação à minha família,
à Deus e à minha namorada Anallígia,
por tudo que representam em minha vida.*

Agradecimentos

Aos meus pais Hélio e Délia, à minha irmã Ana Flávia, ao meu irmão Hélio Júnior e à minha namorada Anallígia pelo apoio, estímulo e carinho em todos os momentos da minha vida.

À professora Rita Maria da Silva Julia na qualidade de orientadora científica, pelas contribuições para o desenvolvimento deste projeto, mas também pela confiança e motivação transmitidas. E ainda pelo espírito prático e capacidade de tornar fácil o que parece difícil.

Aos meus professores da Universidade Federal de Uberlândia que contribuíram para o meu aprendizado, em especial à professora Gina Maira Barbosa de Oliveira pelo apoio na elaboração deste trabalho.

Aos grandes amigos que fiz dentro do mestrado e que compartilharam comigo, direta ou indiretamente, este período de muito trabalho, alegrias e tristezas. Em especial aos amigos Gustavo Carmo, Alexandre Grings, Paulo Vidica Moises e Flávio Luis Duarte.

Finalmente, à Deus por tudo!

*“Há homens que lutam um dia e são bons.
Há outros que lutam um ano e são melhores.
Porém, há os que lutam toda a vida.
Esses são os imprescindíveis.”*

Bertolt Brecht

Resumo

O objetivo deste trabalho é propor um Sistema de Aprendizagem de Damas, *LS-Draughts*, que visa, por meio da técnica dos Algoritmos Genéticos (AGs), gerar, automaticamente, um conjunto de características mínimas necessárias e essenciais de um jogo de Damas, de forma a otimizar o treino de um agente jogador que aprende a jogar Damas. A aprendizagem deste agente consiste em aproximar uma rede neural MLP através do método de Aprendizagem por Reforço TD(λ) aliado com a busca *minimax*, com o mapeamento de tabuleiro *NET-FEATUREMAP* (feito a partir das características geradas pelo AG) e com a técnica de treinamento por *self-play* com clonagem. O objetivo da auto-aprendizagem do agente, sem ter que recorrer a uma base de jogos de especialistas, é permitir que um sistema inteligente aprenda a jogar Damas pela sua própria experiência. Tal processo de aprendizagem é análogo ao processo utilizado pelo sistema *NeuroDraughts* proposto por Mark Lynch. Contudo, o *LS-Draughts* expande o *NeuroDraughts* ao fazer a geração automática de um conjunto eficaz e resumido de características do mapeamento *NET-FEATUREMAP*, ao passo que, o último, utiliza um conjunto fixo e definido manualmente. Foi efetuado um torneio entre o melhor jogador obtido pelo *LS-Draughts* e o melhor jogador de Mark Lynch disponível. Os resultados do torneio, vencido pelo jogador do *LS-Draughts*, evidenciam o fato de o AG representar uma importante ferramenta de melhoria no desempenho geral desses jogadores automáticos.

Palavras-chave: Aprendizagem Automática; Aprendizagem de Máquina; Aprendizagem Incremental; Aprendizagem por Reforço; Computação Evolutiva; Algoritmo Genético; Redes Neurais; Busca *Minimax*; Aprendizagem por Diferenças Temporais; Teoria dos Jogos; Damas.

Abstract

This objective of this work is to propose a Learning System Draughts, *LS-Draughts*, that aims, through the Genetic Algorithm (GA), to generate automatically, a set of minimal features which are necessary and essential to a game of Draughts in order to optimize the training of a player agent that learns to play Draughts. The learning method of this agent consists in approaching a Neural Network MLP through Reinforcement Learning TD(λ) along with the minimax search, with the mapping of the game board NET-FEATUREMAP (done according to the features generated by the GA) and with the training process of self-play with cloning. The goal of the agent's self-learning, without any expert game analysis, is to allow an intelligent system to learn to play Draughts through its own experience. Such learning process is analogous to the process used by the *NeuroDraughts* system proposed by Mark Lynch. However, the *LS-Draughts* expands to *NeuroDraughts* as it generates automatically an effective and concise set of features of the NET-FEATUREMAP mapping, so that, the last one, uses a fixed and manually defined set of features. A tournament was promoted between the best player obtained by the *LS-Draughts* and the best available player of the *LS-Draughts*. The tournament's results, won by the player of the *LS-Draughts*, shows the fact that the GA represents an important improvement tool in these automatic players general performance.

Keywords: Automatic Learning; Machine Learning; Incremental Learning; Reinforcement Learning; Evolutionary Computation; Genetic Algorithm; Neural Network; Minimax Search; Temporal Difference Learning; Game Theory; Draughts; Checkers.

Sumário

Lista de Figuras

Lista de Tabelas

1	Introdução	p. 16
1.1	Introdução e Motivação	p. 16
1.2	O Princípio da Aprendizagem de Máquina	p. 18
1.3	Teoria dos Jogos	p. 19
1.3.1	O que é um jogo ?	p. 20
1.4	Estrutura da dissertação	p. 21
2	Referencial Teórico	p. 23
2.1	Agentes Inteligentes	p. 23
2.2	Estratégia de Busca Minimax	p. 24
2.3	Rede Neural Artificial - RNA	p. 26
2.3.1	O Neurônio biológico e sua ativação	p. 27
2.3.2	A modelagem matemática do neurônio	p. 28
2.3.3	Os tipos de RNAs	p. 31
2.3.3.1	A estrutura das redes	p. 32
2.3.3.2	O treinamento das redes	p. 32
2.3.4	O Perceptron simples ou de camada única	p. 33
2.3.5	O Perceptron Multicamada - MLP	p. 34
2.3.5.1	O algoritmo de retropropagação do erro	p. 35

2.4	Aprendizagem por Reforço	p. 40
2.4.1	Características da Aprendizagem por Reforço	p. 41
2.4.2	O Problema da Aprendizagem por Reforço	p. 42
2.4.3	Fundamentos da Modelagem Markoviana	p. 46
2.4.3.1	Propriedade de Markov	p. 46
2.4.3.2	Processo de Decisão de Markov - PDM	p. 47
2.4.4	Método de solução por Diferenças Temporais	p. 49
2.5	Computação Evolutiva	p. 52
2.5.1	Algoritmo Genético - AG	p. 53
2.5.1.1	População e codificação dos indivíduos	p. 55
2.5.1.2	Definição da população inicial	p. 56
2.5.1.3	Método de seleção dos indivíduos para próxima geração	p. 56
2.5.1.4	Operadores Genéticos	p. 57
2.5.1.5	Função de Avaliação ou <i>fitness</i>	p. 59
2.5.1.6	Critério de Parada	p. 59
2.5.2	Outras Técnicas Evolutivas	p. 60
2.6	Considerações Finais	p. 60
3	Estado da Arte	p. 62
3.1	Introdução	p. 62
3.2	Tipo de Treinamento	p. 63
3.3	Aprendizagem por Reforço	p. 64
3.3.1	O problema da atribuição de crédito	p. 65
3.4	Método das Diferenças Temporais	p. 67
3.4.1	O Sucesso de <i>TD-GAMMON</i>	p. 67
3.4.2	Uma proposta de sucesso para outros jogos	p. 69
3.5	Complexidade dos Jogos	p. 71

3.6	Representações do Estado	p. 71
3.7	Estratégias de Treinamento	p. 73
3.7.1	Ajuste dos parâmetros de treino	p. 75
3.7.2	Combinando Diferença Temporal com Busca Minimax	p. 76
3.8	Diferenças Temporais x Computação Evolutiva	p. 77
3.9	Diferenças Temporais em outros domínios	p. 79
3.10	Considerações Finais	p. 81
4	<i>LS-Draughts</i> – Um Sistema de Aprendizagem de Damas	p. 82
4.1	Jogador de Mark Lynch e o Processo de Treinamento por TD(λ)	p. 86
4.1.1	Mapeamento das características e representação do tabuleiro	p. 87
4.1.2	Cálculo das predições	p. 92
4.1.3	Escolha da melhor ação	p. 95
4.1.4	Processo de reajuste dos pesos da rede MLP	p. 98
4.1.5	Cálculo de reajuste dos pesos da rede MLP	p. 103
4.1.6	Estratégia de treino por <i>self-play</i> com clonagem	p. 106
4.2	O Processo Evolutivo do <i>LS-Draughts</i>	p. 108
4.2.1	População e codificação dos indivíduos no <i>LS-Draughts</i>	p. 109
4.2.2	Seleção dos indivíduos e aplicação dos operadores genéticos	p. 110
4.2.3	Treinamento da rede MLP acoplada aos indivíduos	p. 112
4.2.4	Função de avaliação e atualização da população para próxima geração	p. 115
4.3	Ferramenta utilizada na implementação do <i>LS-Draughts</i>	p. 116
4.4	Resultados Experimentais	p. 117
4.5	Considerações Finais	p. 124
5	Conclusões	p. 126
5.1	Perspectiva de Trabalhos Futuros	p. 127

Referências	p. 129
--------------------	--------

Anexo A	p. 134
----------------	--------

Partida de Damas entre o melhor jogador do <i>LS-Draughts</i> e o melhor jogador do <i>NeuroDraughts</i>	p. 134
---	--------

Lista de Figuras

1	Modelo geral de um agente inteligente com capacidade de aprendizagem.	p. 24
2	A árvore de busca <i>minimax</i>	p. 26
3	Célula neural biológica (as setas largas indicam a seqüência de propagação de sinais pelos neurônios).	p. 28
4	Disparo de um neurônio (KOVACS, 1996).	p. 29
5	Célula neural artificial (MCCULLOCH; PITTS, 1943).	p. 30
6	Algumas funções de ativações conhecidas. (a) Função Rampa. (b) Função limite ríspido.	p. 30
7	Arquitetura de um perceptron simples.	p. 34
8	Problema do <i>AND</i> lógico. (a) Perceptron simples que resolve o problema do <i>AND</i> lógico. (b) Representação da função do <i>AND</i> lógico em um hiperplano.	p. 35
9	Redes neurais tipo <i>feedforward</i> com múltiplas camadas. (a) Arquitetura MLP. (b) Sentido de propagação do sinal funcional e do sinal de erro (HAYKIN, 2001).	p. 36
10	A interação agente-ambiente em Aprendizagem por Reforço.	p. 42
11	Exemplo do operador de <i>crossover</i> de ponto único.	p. 58
12	Exemplo do operador de mutação.	p. 59
13	Complexidade do espaço de estados e fator de ramificação de alguns jogos	p. 71
14	Processo de treinamento por <i>self-play</i> com clonagem	p. 75
15	Arquitetura geral do <i>LS-Draughts</i>	p. 83
16	Mapeamento do tabuleiro de Damas utilizado por Mark Lynch em (LYNCH, 1997)	p. 89
17	Rede neural MLP utilizada por Mark Lynch em (LYNCH, 1997)	p. 93

18	Função tangente hiperbólica e sua funcionalidade no domínio de Damas	p. 95
19	Árvore de busca minimax para o estado raiz S_0 com profundidade 4 . .	p. 97
20	Árvore de busca minimax para o estado raiz S_7 com profundidade 4 . .	p. 101
21	Árvore de busca minimax para o estado raiz S_{34} com profundidade 4 .	p. 102
22	Exemplos de superfície de erro. (a) Uma má superfície de erro, com muitos mínimos locais. (b) Uma boa superfície de erro, cujo mínimo ótimo pode ser facilmente obtido, por exemplo, por regressão linear. . .	p. 107
23	Representação dos 15 genes de um cromossomo vinculado a um determinado indivíduo da população	p. 109
24	Operação de <i>crossover</i> , com um único ponto de corte, aplicado a um par de indivíduos pais para gerar dois novos indivíduos.	p. 111
25	Operação de mutação de gene com uma taxa de 0.3 sobre o indivíduo K .	p. 111
26	Seleção das características ativas do indivíduo M para definição do mapeamento <i>NET-FEATUREMAP</i> que a rede acoplada estará utilizando no treinamento.	p. 112
27	Arquivo de tabuleiro para treinamento.	p. 115
28	Gráfico de evolução do melhor indivíduo em relação a média da população nas gerações 0, 4, 9, 14, 19, 24 e 29.	p. 118
29	Conjunto de características do melhor jogador da 8ª geração do <i>LS-Draughts</i>	p. 118
30	Conjunto de características do melhor jogador da 24ª geração do <i>LS-Draughts</i>	p. 120
31	Posição do tabuleiro de Damas em <i>loop</i> depois do 43º movimento do jogador vermelho.	p. 122
32	Posição do tabuleiro de Damas em <i>loop</i> depois do 44º movimento do jogador vermelho.	p. 123
33	Posição do tabuleiro depois do 6º movimento do jogador vermelho. . . .	p. 135
34	Posição do tabuleiro de Damas depois do 8º movimento do jogador vermelho.	p. 136

35	Posição do tabuleiro de Damas depois do 13º movimento do jogador vermelho.	p. 137
36	Posição do tabuleiro de Damas depois do 21º movimento do jogador vermelho.	p. 138
37	Posição do tabuleiro de Damas depois do 30º movimento do jogador vermelho.	p. 139
38	Posição do tabuleiro de Damas depois do 37º movimento do jogador vermelho.	p. 140
39	Posição do tabuleiro de Damas depois do 42º movimento do jogador vermelho.	p. 141
40	Posição do tabuleiro de Damas depois do 44º movimento do jogador preto (vitória do jogador do <i>LS-Draughts</i>).	p. 142

Lista de Tabelas

- 1 Tabela de probabilidades das transições de estados e retornos previstos para um PDM finito p. 49
- 2 Tabela com as 12 características implementadas por Mark Lynch em (LYNCH, 1997) p. 90
- 3 Relação das 15 características utilizadas na representação do indivíduo em *LS-Draughts*. p. 125

1 *Introdução*

1.1 Introdução e Motivação

O paradigma da Aprendizagem por Reforço tem sido de grande interesse na área da aprendizagem automática, uma vez que dispensa um “professor” inteligente para o fornecimento de exemplos de treino, fato que o torna particularmente adequado a domínios complexos em que a obtenção destes exemplos seja difícil ou até mesmo impossível (RUSSELL; NORVIG, 2004). Dentre os métodos de Aprendizagem por Reforço, destaca-se o método das Diferenças Temporais (TD), que é ampla e eficazmente utilizado, inclusive, na construção de agentes capazes de aprender a jogar Damas, Xadrez, Go, Gamão, Othello ou outros jogos (LYNCH; GRIFFITH, 1997; SCHAEFFER et al., 2001; SAMUEL, 1959, 1967; SCHRAUDOLPH; DAYAN; SEJNOWSKI, 2001; THRUN, 1995; TESAURO, 1994; LEUSKI, 1995; EPSTEIN, 2001). Tais agentes têm demonstrado que os jogos são, sem dúvida nenhuma, um domínio muito bom para se estudar as principais técnicas da aprendizagem automática.

Particularmente, cita-se como exemplo de bons jogadores automáticos o jogador de Damas de Mark Lynch (LYNCH, 1997), o qual implementa um jogador de Damas como uma rede neural que utiliza a busca *minimax* para a escolha da melhor jogada em função do estado do tabuleiro do jogo. Além disso, ele utiliza o método de aprendizagem por reforço TD(λ), aliado à estratégia de treino por *self-play* com clonagem, como ferramentas para atualizar os pesos da rede. Para tanto, o tabuleiro é representado por um conjunto de funções que descrevem as características do próprio jogo de Damas. No caso de Lynch, tais características são selecionadas manualmente.

Inspirado nos resultados de Pollack, Blair, Samuel, Lynch, Schaeffer e Fogel (POLLOCK; BLAIR, 1998; SAMUEL, 1959; LYNCH, 1997; SCHAEFFER et al., 2001; FOGEL; CHELLAPILLA, 2002; FOGEL et al., 2004), este trabalho visa estender o jogador de damas de Mark Lynch, *NeuroDraughts*, gerando, automaticamente, por meio da técnica dos Algoritmos Genéticos (AGs), as características mínimas necessárias e essenciais de um domínio de um jogo de Damas, de forma a tentar otimizar o processo de aprendizagem do agente

jogador de Lynch.

A escolha do jogo de Damas como um domínio de aplicação se deve ao fato de que ele apresenta significativas semelhanças com inúmeros problemas práticos e, por outro lado, apresenta uma complexidade que demanda a utilização dos recursos propiciados por técnicas poderosas como o método $TD(\lambda)$, busca *minimax*, Redes Neurais e Algoritmos Genéticos. Como exemplos desses problemas práticos, podem-se citar os seguintes:

- *Problema de navegação em que os mapas são obtidos autonomamente por um robô móvel:* a tarefa de aprendizagem parte de um ponto de referência inicial, onde o robô deve aprender uma trajetória de navegação de modo a atingir um ponto alvo, e ao mesmo tempo, desviar dos obstáculos do ambiente (RIBEIRO; MONTEIRO, 2003);
- *Problema de interação com humanos por meio de um diálogo:* cada vez mais, a vida moderna demanda agentes que dialogam com humanos (tais como os atendentes eletrônicos em empresas de prestação de serviços). Como exemplo de sistema que ataca esse problema, cita-se o sistema ELVIS (*Elvis Voice Interactive System*) de Walker (WALKER, 2000), que cria um agente que aprende a escolher uma ótima estratégia de diálogo por meio de suas experiências e interações com os usuários humanos;
- *Problema do controle de tráfego veicular urbano:* o objetivo é criar um agente capaz de controlar o número médio de veículos sobre uma rede urbana de forma a minimizar os congestionamentos e o tempo de viagem sob esta rede (WIERING, 2000).

Note que todos estes problemas apresentam dificuldades similares ao problema encontrado no domínio dos jogos, a saber:

- Aprender a se comportar em um ambiente onde o conhecimento adquirido é armazenado em uma função de avaliação;
- Escolha de um mínimo de atributos possíveis que melhor caracterizem o domínio e que sirva como um meio pelo qual a função de avaliação adquirirá novos conhecimentos (esta questão é de fundamental importância para se obterem agentes com alto nível de desempenho);
- Seleção da melhor ação para um determinado estado ou configuração do ambiente onde o agente está interagindo (problema de otimização);

- Estratégias de aprendizagem que facilitem a geração de um agente com ótimo nível de desempenho.

A seção seguinte descreve, sucintamente, o princípio da Aprendizagem de Máquina, apontando um dos trabalhos mais significativos na tarefa de mostrar as vantagens de se utilizarem os jogos como terreno de teste às técnicas de aprendizagem automática: o trabalho de Arthur Samuel além de impressionar pela época (1959), também serviu de base para as idéias que, ainda hoje, estão em voga.

1.2 O Princípio da Aprendizagem de Máquina

No ano de 1946 Arthur L. Samuel, como professor de Engenharia Eletroeletrônica da Universidade de Illinois, Chicago - USA, iniciou um projeto de elaboração de um programa que pudesse jogar Damas com o objetivo de desafiar o campeão mundial, derrotando-o. Além disso, o Jogo de Damas, geralmente considerado como sendo um jogo mais simples que o Xadrez, parecia ser o domínio perfeito para demonstrar o poder da computação através de um curto projeto de programação.

Os dois principais artigos (SAMUEL, 1959, 1967), resultante da sua investigação, foram provavelmente a pesquisa de Aprendizagem de Máquina mais antiga a obter sucesso. Embora fosse informal e tivesse várias falhas, esse trabalho continha a maior parte das idéias modernas em Aprendizagem por Reforço, inclusive a diferenciação temporal e a aproximação de funções. Com isto, Samuel abriu caminho para a idéia de atualização de avaliações, baseado nas predições sucessivas temporárias, em jogos de Damas: “...*we are attempting to make the score, calculated for the current board position, look like that calculated for the terminal board position of the chain of moves which most probably will occur during actual play. Of course, if one could develop a perfect system of this sort it would be the equivalent of always looking ahead to the end of the game.*”

Nos seus trabalhos, Samuel não foi apenas pioneiro das inúmeras técnicas de busca moderna, utilizadas em programas que jogam com alto desempenho, como os cortes *alpha-beta*, mas também inventou um vasto leque de técnicas de aprendizagem para melhorar o desempenho dos programas ao longo do tempo. Samuel considerou o domínio de Damas como sendo um domínio perfeito para o estudo das técnicas de aprendizagem automática porque nos jogos muitas das complicações que surgem nos problemas da vida real são simplificadas, permitindo que os investigadores se foquem nos problemas de aprendizagem propriamente dito (SAMUEL, 1959). Como resultado disso, muitas das técnicas que contribuíram para o sucesso da aprendizagem automática como ciência podem ser rela-

cionadas a Samuel e conseqüentemente muitas das idéias de Samuel para aprendizagem ainda são utilizadas hoje em dia, de uma forma ou de outra.

As áreas de jogos e de aprendizagem automática evoluíram muito desde os dias de Arthur Samuel. Apesar dessa evolução, muitas das novas técnicas desenvolvidas nestas duas áreas podem hoje serem relacionadas diretamente com algumas de suas idéias. O seu jogador de Damas ainda é considerado um dos trabalhos mais influentes nas duas áreas, sendo um exemplo perfeito de uma fusão frutífera.

1.3 Teoria dos Jogos

A Teoria dos Jogos é uma ferramenta matemática criada para melhor entender ou interpretar a maneira com que agentes que tomam decisões interagem entre si. Considere, por exemplo, um *jogo* onde vários *jogadores* estão interagindo entre si. Esses jogadores têm um conjunto de *decisões* (ou *ações*) passíveis de serem tomadas. As tomadas de decisões são baseadas nas preferências de cada jogador e na sua expectativa sobre as ações dos outros jogadores. É justamente nessa dinâmica que a Teoria dos Jogos foca seu estudo.

A Teoria dos Jogos tornou-se um ramo proeminente da matemática aplicada nos anos 30 do século XX, especialmente depois da publicação do clássico “*The Theory of Games and Economic Behavior*” de John Von Neumann e Oskar Morgenstern em 1944 (NEUMANN; MORGENSTERN, 1944). Desde então, ela despertou a atenção da Ciência da Computação que a vem utilizando em avanços na Inteligência Artificial e Cibernética.

A Teoria dos Jogos procura encontrar estratégias racionais em situações onde o resultado não depende somente da estratégia própria de um agente, mas, também, das estratégias escolhidas por outros agentes que possivelmente têm estratégias ou objetivos comuns. Os resultados da Teoria dos Jogos tanto podem ser aplicados a simples jogos de entretenimento como a aspectos significativos da vida em sociedade. Um exemplo deste último tipo de aplicação pode ser visto na edição 175 da revista Superinteressante de Abril de 2002 (página 68 à 73). Esta edição mostra como a Teoria dos Jogos ajuda a entender o comportamento humano em situações de conflito (o que também vale para os jogos). A reportagem, intitulada “*A matemática explica*”, dá uma série de exemplos que ilustram tais situações:

- “*Você está estacionando o carro e... - crassshh - amassa o paralama daquele reluzente BMW ao lado. Ninguém viu. Você, um cara decente, pensa em deixar um bilhete se identificando e assumindo a responsabilidade. Mas, espera aí. É um*

BMW. O dono certamente tem dinheiro, e não estaria dirigindo um carro desses por aí se não tivesse seguro. Essa batidinha para ele não será nada, mas para você...”;

- *“Já é tarde da noite e você está na estação do metrô. Ninguém por perto. Por que não saltar a roleta e viajar sem pagar ? É claro que a companhia do metrô não vai quebrar se você fizer isso. Os trens circulam com ou sem passageiros. Por que não saltar a roleta ?”.*

Nestes dois exemplos vistos acima, note que há uma infinidade de situações em que o interesse individual se choca com o coletivo. No caso do carro em que você bateu, o seguro paga e repassa o custo para os prêmios que cobra. Não assumindo o prejuízo, você acaba penalizando gente que nada tem a ver com isso. O caso do metrô é idêntico: engrossando as estatísticas dos que não pagam, você contribui para o aumento das passagens dos que pagam. Esse é um dilema freqüente nas organizações - na família, nas empresas, entre nações. Ele surge de um impulso com o qual todo mundo lida em inúmeras circunstâncias: a tendência a satisfazer o interesse individual agindo de uma forma que, se todos imitassem, seria catastrófica para todos.

É neste sentido que a fascinante Teoria dos Jogos usa a solidez da matemática para compreender e antecipar o insólito e imprevisível comportamento humano, bem como dos agentes no mundo dos jogos. O objetivo da Teoria dos Jogos é lançar luz sobre *conflitos de interesse* e ajudar a responder ao seguinte: o que é preciso para haver colaboração ? Em quais circunstâncias o mais racional é não colaborar ? Que políticas devem ser adotadas para garantir a colaboração ?

1.3.1 O que é um jogo ?

A teoria dos jogos pode ser definida como a teoria dos modelos matemáticos que estuda a escolha de decisões ótimas sob condições de conflito. O elemento básico em um jogo é o conjunto de *jogadores* que dele participam. Cada jogador tem um conjunto de *estratégias*. Quando cada jogador escolhe sua estratégia, tem-se uma *situação* ou *perfil* no espaço de todas as situações (perfis) possíveis. Cada jogador tem interesse ou preferências para cada situação no jogo. Em termos matemáticos, cada jogador tem uma *função utilidade* que atribui um número real (o *ganho* ou *payoff* do jogador) a cada situação do jogo.

Mais especificamente, um jogo tem os seguintes elementos básicos: existe um conjunto finito de jogadores, representado por $G = \{g_1, g_2, \dots, g_n\}$. Cada jogador $g_i \in G$ possui um conjunto finito $S_i = \{s_{i1}, s_{i2}, \dots, s_{im_i}\}$ de opções, denominadas *estratégias puras* do jogador g_i ($m_i \geq 2$). O produto cartesiano $S = \prod_{i=1}^n S_i = S_1 * S_2 * \dots * S_n$ é denominado *espaço de*

estratégia pura do jogo e seus elementos de *perfis de estratégia pura*. Para cada jogador $g_i \in G$, existe uma *função utilidade* $u_i : S \rightarrow R$ que associa o *ganho* (*payoff*) $u_i(s)$ do jogador g_i a cada perfil de estratégia pura $s \in S$.

Possivelmente o exemplo mais conhecido na teoria dos jogos é o *dilema do prisioneiro* (<http://www.answers.com/topic/prisoner-s-dilemma>). Ele foi formulado por Albert W. Tucker em 1950, em um seminário para psicólogos na Universidade de Standford, para ilustrar a dificuldade de se analisar certos tipos de jogos. A situação é a seguinte: dois ladrões, Al e Bob, são capturados e acusados de um mesmo crime. Presos em salas separadas e sem poderem se comunicar entre si, o delegado de plantão faz a seguinte proposta: cada um pode escolher entre confessar ou negar o crime. Se nenhum deles confessar, ambos serão submetidos a uma pena de 1 ano. Se os dois confessarem, então ambos terão pena de 5 anos. Mas se um confessar e o outro negar, então o que confessou será libertado e o outro será condenado a 10 anos de prisão.

Neste contexto, têm-se:

$$G = \{Al, Bob\}, S_{Al} = \{confessar, negar\}, S_{Bob} = \{confessar, negar\}, \\ S = \{(confessar, confessar), (confessar, negar), (negar, confessar), (negar, negar)\},$$

onde em cada par ordenado (x, y) , “x” e “y” se referem às ações de AL e Bob, respectivamente.

As duas funções utilidade $u_{Al} : S \rightarrow R$ e $u_{Bob} : S \rightarrow R$ são dadas por:

$$u_{Al}(confessar, confessar) = -5, u_{Al}(confessar, negar) = 0, \\ u_{Al}(negar, confessar) = -10, u_{Al}(negar, negar) = -1,$$

(que apresentam os ganhos (*payoffs*) de Al) e

$$u_{Bob}(confessar, confessar) = -5, u_{Bob}(confessar, negar) = -10, \\ u_{Bob}(negar, confessar) = 0, u_{Bob}(negar, negar) = -1,$$

(que apresentam os ganhos (*payoffs*) de Bob).

1.4 Estrutura da dissertação

Os próximos capítulos estão organizados conforme disposto a seguir:

Capítulo 2. Apresentação da modelagem de um agente inteligente e dos fundamentos teóricos da busca *minimax*, Redes Neurais, Aprendizagem por Reforço, ilustrado por

um exemplo de modelagem PDM e apresentação de um método para resolução de problemas de Aprendizagem por Reforço conhecido como Diferenças Temporais. Além disso, uma apresentação teórica também é feita com relação a área da Computação Evolutiva, especificamente o Algoritmo Genético.

Capítulo 3. Um levantamento do estado da arte em programas que aprendem a jogar.

Capítulo 4. Descrição e resultados com a implementação do sistema *LS-Draughts*: um programa que visa, por meio da técnica dos AGs, gerar, automaticamente, um conjunto de características mínimas necessárias e essenciais de um domínio de um jogo de Damas, de forma a otimizar o treino do agente jogador de Lynch que utiliza o mapeamento *NET-FEATUREMAP*.

Capítulo 5. Conclusões e possíveis linhas de investigação para trabalhos futuros.

Anexo A. Partida de Damas entre o melhor jogador do sistema *LS-Draughts* e o melhor jogador de Lynch referente ao seu sistema *NeuroDraughts*.

2 *Referencial Teórico*

2.1 Agentes Inteligentes

Pode-se definir um *Agente Inteligente* como uma entidade que age em um mundo de acordo com seus objetivos, percepções e o estado atual do seu conhecimento. As ações de um agente são percebidas pela produção de eventos que correspondem às alterações no ambiente em que o mesmo está inserido. Em termos matemáticos, pode-se afirmar que o comportamento do agente é descrito pela função do agente que mapeia qualquer sequência de percepções específica para uma ação.

Um agente é uma entidade simples, capaz de executar tarefas cuja complexidade varia de acordo com sua construção. Para resolução de problemas mais complexos, faz-se necessária uma interação entre vários agentes, de forma ordenada. Os agentes podem combinar diferentes habilidades para solucionar diferentes problemas.

Segundo (RUSSELL; NORVIG, 2004), as propriedades básicas de uma entidade para ser considerada como um agente são:

- **Autonomia:** escolhe a ação a tomar baseado mais na própria experiência do que no conhecimento recebido inicialmente por seu projetista. Dessa forma, o agente deve aprender o que puder para compensar um conhecimento prévio parcial ou incorreto. As ações do agente não requerem interferência humana direta. Entretanto, pode acontecer situações onde seja necessária a interferência de um agente humano. Mas não se projeta um agente para ser dependente destas informações;
- **Reatividade:** reage aos estímulos do ambiente selecionando ações baseadas em sua percepção atual;
- **Proatividade:** capaz de, além de responder a estímulos do ambiente, exibir um comportamento orientado a objetivos. Ou seja, ser capaz de prever como atingir ou evitar um determinado estado ou objetivo. Pensar no futuro, antecipar, agindo em função de prever;

- **Comunicação:** troca informações com o ambiente e com os outros agentes;

De uma forma geral, os agentes inteligentes são entidades de *software* capazes de demonstrar comportamento autônomo, orientado a um objetivo, dentro de um ambiente computacional heterogêneo. A figura 1 a seguir mostra a estrutura de um agente inteligente com um *elemento de aprendizagem* e um *elemento de desempenho*. O elemento de aprendizagem é responsável pela execução de aperfeiçoamentos do elemento de desempenho (ou função do agente) e este, por sua vez, é responsável pela seleção de ações externas. O elemento de aprendizado utiliza a realimentação do crítico sobre como o agente está funcionando em relação a um padrão fixo de desempenho e determina de que maneira o elemento de desempenho deve ser modificado para funcionar melhor no futuro.

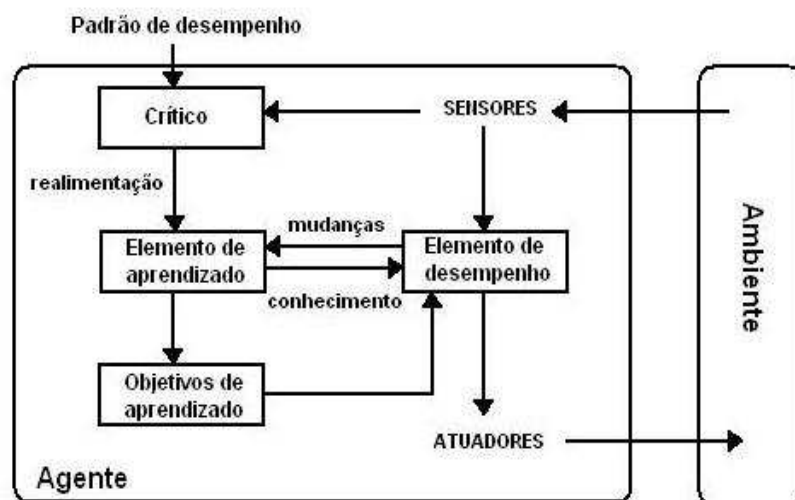


Figura 1: Modelo geral de um agente inteligente com capacidade de aprendizagem.

O agente inteligente que se pretende construir neste trabalho é um jogador de Damas automático que consiga jogar com alto nível de desempenho. A modelagem deste agente será discutida com mais detalhes no capítulo 4.

2.2 Estratégia de Busca Minimax

De forma genérica, as estratégias de busca tradicionais envolvem uma busca em uma árvore que descreve todos os estados possíveis a partir de um estado inicial dado. Formalmente, o espaço de busca é constituído por um conjunto de nós conectados através de arcos. A cada arco pode ou não estar associado um valor, que corresponde ao custo c

de transição de um nó a outro. A cada nó temos associada uma profundidade p , sendo que a mesma tem valor 0 no nó raiz e aumenta de uma unidade para um nó filho. A aridade a de um nó é a quantidade de filhos que o mesmo possui, e a aridade de uma árvore é definida como a maior aridade de qualquer um de seus nós. O objetivo da busca é encontrar um caminho (ótimo ou não) do estado inicial até um estado final, explorando sucessivamente os nós conectados aos nós já explorados, até a obtenção de uma solução para o problema.

Entretanto, em problemas onde se deseja planejar, com antecedência, ações a serem executadas por um agente em um ambiente no qual outros agentes estão fazendo planos contrários àquele, surge o chamado *problema de busca competitiva*. Nestes ambientes as metas dos agentes são mutuamente exclusivas. Os jogos são exemplos de ambientes que apresentam este tipo de problema de busca competitiva: o jogador não tem que se preocupar apenas em chegar ao objetivo final, mas também em evitar que algum oponente chegue antes dele, ou seja, vença o jogo. Desta maneira, o jogador deve se antecipar à jogada do seu adversário para poder fazer a sua jogada. Uma das maneiras de solucionar este tipo de problema é através do método de busca *minimax*.

O *minimax* (RUSSELL; NORVIG, 2004) é uma técnica de busca para determinar a estratégia ótima em um cenário de jogo com dois jogadores. O objetivo dessa estratégia ótima é decidir a melhor jogada para um dado estado do jogo. Há dois jogadores no *minimax*: o MAX e o MIN. Uma busca em profundidade é feita a partir de uma árvore onde a raiz é a posição corrente do jogo. As folhas dessa árvore são avaliadas pela ótica do jogador MAX, e os valores dos nós internos são atribuídos de baixo para cima com essas avaliações. As folhas do nível minimizar são preenchidas com o menor valor de todos os seus nós filhos, e o nível de maximizar são preenchidos com o maior valor de todos os nós filhos. Como a quantidade de busca cresce exponencialmente com o aumento da profundidade de análise do algoritmo *minimax*, é necessário, para reduzir o tempo de busca, que nenhum tempo seja perdido analisando jogadas que obviamente não serão boas para o jogador. Um modo significativo de se conseguir isso é através do corte alfa-beta no algoritmo *minimax*. Segue abaixo um versão do algoritmo de busca *minimax* sem corte alfa-beta.

Algoritmo de busca *minimax*

FUNÇÃO MINIMAX(N)

{

```

SE  $N$  é uma folha ENTÃO
    RETORNE A AVALIAÇÃO DESTA FOLHA
SENÃO
{
    GERE TODOS OS NÓS  $N_1, N_2, \dots, N_M$  SUCESSORES DE  $N$ 
    SE  $N$  é um nó MIN ENTÃO
        RETORNE  $\min(\text{MINIMAX}(N_1), \text{MINIMAX}(N_2), \dots, \text{MINIMAX}(N_M))$ 
    SENÃO
        RETORNE  $\max(\text{MINIMAX}(N_1), \text{MINIMAX}(N_2), \dots, \text{MINIMAX}(N_M))$ 
}
}

```

A figura 2 mostra um exemplo de aplicação do algoritmo *minimax* que gera a árvore de busca do jogo para um determinado estado e profundidade 2.

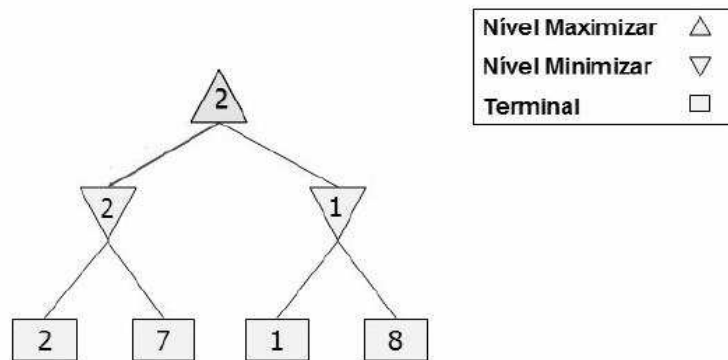


Figura 2: A árvore de busca *minimax*.

A busca *minimax* implementada no sistema *LS-Draughts* é utilizada para seleção da melhor ação a ser executada pelo agente jogador de Damas em função do estado do tabuleiro do jogo. Este tópico será discutido com detalhes na subseção 4.1.3.

2.3 Rede Neural Artificial - RNA

A idéia que se segue ao se tentar utilizar redes neurais como função de avaliação em agentes inteligentes é dado por Rich e Knight em (RICH; KNIGHT, 1992): “Na nossa

busca de construir máquinas inteligentes, só temos um modelo que ocorre naturalmente: o cérebro humano. Uma idéia óbvia para IA, então, é simular o funcionamento do cérebro diretamente em um computador”.

O trabalho em redes neurais artificiais, usualmente denominadas “redes neurais”, tem sido motivado desde o começo pelo reconhecimento de que o cérebro humano processa informações de uma forma inteiramente diferente do computador digital convencional. O cérebro é um computador altamente complexo, não-linear e paralelo. Ele tem a capacidade de organizar seus constituintes estruturais, conhecidos por neurônios, de forma a realizar certos processamentos muito mais rapidamente que o mais rápido computador digital hoje existente (HAYKIN, 2001).

As seções que se seguem abordam os principais aspectos dessa abordagem, apresentando: **i)** os aspectos biológicos, **ii)** a modelagem matemática do neurônio, **iii)** a arquitetura de RNA, e **iv)** o famoso treinamento supervisionado através da retropropagação do erro pela regra delta generalizada.

Destaca-se entretanto que, devido à grande diversidade de arquiteturas encontradas na literatura, apenas as de maior importância ou de alguma forma relevantes ao trabalho proposto serão abordadas, em especial o perceptron multicamada. Um panorama geral de outras arquiteturas pode ser encontrado em algumas referências bibliográficas da área (HAYKIN, 2001; FAUSETT, 1994; KOVACS, 1996).

2.3.1 O Neurônio biológico e sua ativação

Com o avanço dos estudos fisiológicos do sistema nervoso iniciado no século 19, o neurônio foi reconhecido como o dispositivo elementar do processamento cerebral e este passou a ser alvo de extensivos estudos, culminando nos dias atuais em que seu funcionamento é conhecido com alguma profundidade.

A anatomia da célula nervosa (figura 3) caracteriza-se pela presença de um corpo celular (*soma*) - responsável pelos processos metabólicos da célula - e pelas projeções deste corpo (*dendritos* e o *axônio*). Tal célula destaca-se pela capacidade de transmissão de impulsos elétricos, possível graças às propriedades peculiares de sua membrana celular.

As entradas neurais provêm das conexões sinápticas, efetuadas pelas ramificações dos dendritos de um neurônio em contato com os axônios de outros neurônios. O neurônio, após receber tais entradas (aqui referidas como impulsos nervosos), processa-as, de alguma forma, e produz uma saída em seu axônio.

O impulso nervoso propaga-se segundo o processo biológico conhecido como a *bomba*

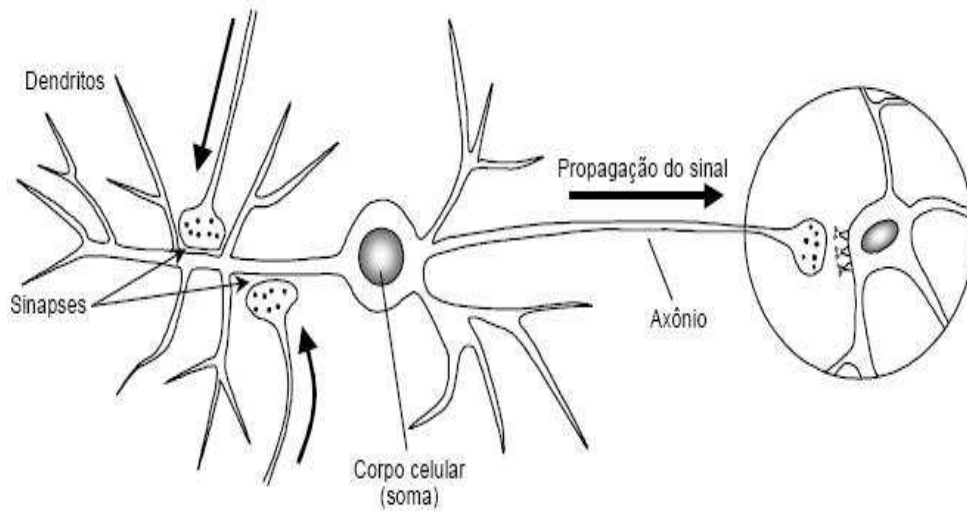


Figura 3: Célula neural biológica (as setas largas indicam a sequência de propagação de sinais pelos neurônios).

de sódio e potássio. A difusão de íons pela membrana possibilita que esta apresente uma polarização entre suas regiões interna e externa da ordem de alguns milivolts. Nesses termos, pode-se caracterizar o impulso nervoso como sendo a propagação de uma polarização, durante algum tempo, ao longo da membrana.

Em (KOVACS, 1996), Kovács apresenta resumidamente a caracterização de ativação e propagação do impulso nervoso de um neurônio. Segundo ele, a ativação ocorre sempre que a membrana recebe estímulos suficientes para cruzar um *limiar de disparo*. Ao receber tal estímulo, a célula se excitará por tipicamente alguns microsegundos (T_n), emitindo nesse período forte descarga; depois entrará em um período de *refração absoluta* (T_R), em que a célula não produzirá novo disparo, chegando a um período de *refração relativa* (T_r), em que um novo disparo será produzido somente sob uma excitação bastante maior que o valor do limiar de disparo para, finalmente, retornar ao seu estado de repouso. Este processo pode ser visto na figura 4.

2.3.2 A modelagem matemática do neurônio

Com a base descrita acima, pode-se entender um neurônio como um dispositivo que tem muitas entradas e apenas uma saída. O primeiro modelo matemático de um neurônio artificial foi proposto em 1943, por McCulloch e Pitts (MCCULLOCH; PITTS, 1943). Neste modelo (figura 5), as conexões entre os neurônios artificiais procuram simular as conexões sinápticas biológicas fazendo uso de uma variável chamada *peso*. O *peso* é um atributo

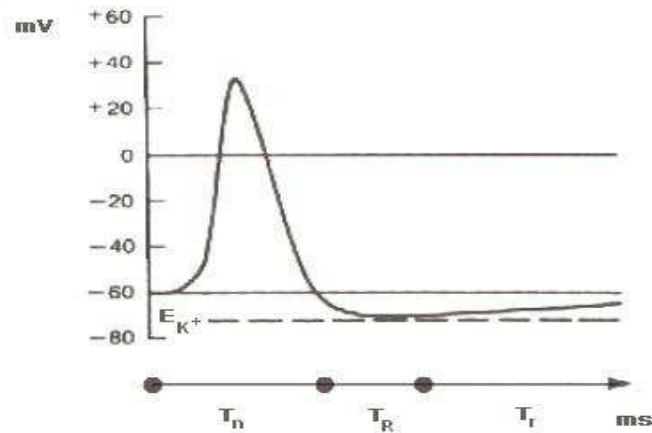


Figura 4: Disparo de um neurônio (KOVACS, 1996).

importantíssimo do neurônio, podendo ser comparado com os dendritos realizando as suas sinapses em outros neurônios. Representados por w , são valores que indicam o grau de importância que determinada entrada possui em relação àquele determinado neurônio.

Já o corpo celular biológico é representado por uma *função de soma*, Σ , e uma *função de ativação*, g . A função de soma acumula os dados recebidos (estímulos) de outros neurônios a fim de que a função de ativação possa processá-los, transformando-os. Em outras palavras, depois de acumulado o valor somado dos produtos ocorridos entre as entradas e os pesos, a função de ativação compara este valor com um limiar ou *bias* (um valor estipulado), e, atingindo-o, o valor é então passado adiante através da saída (caso contrário, o sinal não é transferido adiante). Em ambos os casos da rede, com ou sem sinal, a resposta é significativa, pois afetará diretamente, ou a resposta final da rede, ou os neurônios da próxima camada. A lógica neural expõe, dessa forma, que a intensidade dos sinais de entrada dispara, ou não, o sinal do neurônio, fazendo com que este estimule o neurônio seguinte.

No modelo da figura 5, a ativação de saída do neurônio j é dada por: $a_j = g(in_j) = g(\sum_{i=0}^n w_{ij}.a_i)$ onde a_i é a ativação de saída da unidade (ou neurônio) i conectado a j e w_{ij} é o peso no vínculo da unidade i a essa unidade j . Note que o neurônio j da figura 5 apresenta um peso de desvio w_{0j} conectado a uma entrada fixa $a_0 = -1$. Este peso define o limite real para a unidade j , no sentido de que a unidade é ativada quando a soma ponderada de entradas reais $\sum_{i=1}^n w_{ij}.a_i$ excede $w_{0j}.a_0$.

A função de ativação g ou *camada de processamento de limiares* é projetada para atender a duas aspirações:

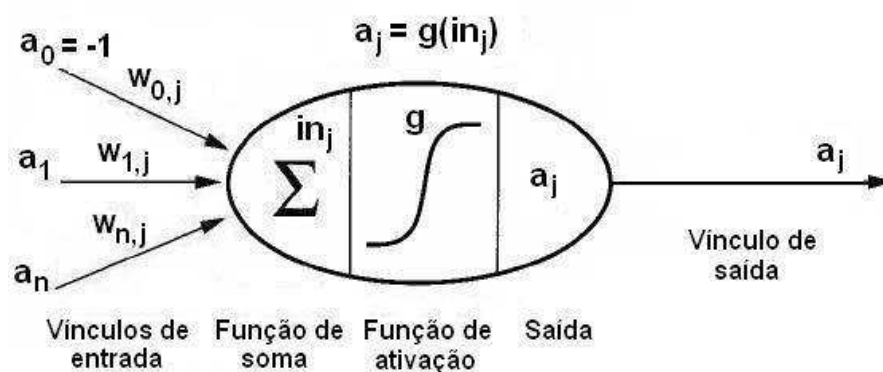


Figura 5: Célula neural artificial (MCCULLOCH; PITTS, 1943).

- A unidade deverá ser ativa quando as entradas recebidas forem “corretas” e inativa quando as entradas recebidas forem “erradas”;
- A ativação precisa ser não-linear, caso contrário a rede neural inteira entrará em colapso, tornando-se uma função linear simples.

A figura 6 mostra alguns exemplos de funções de ativações conhecidas.

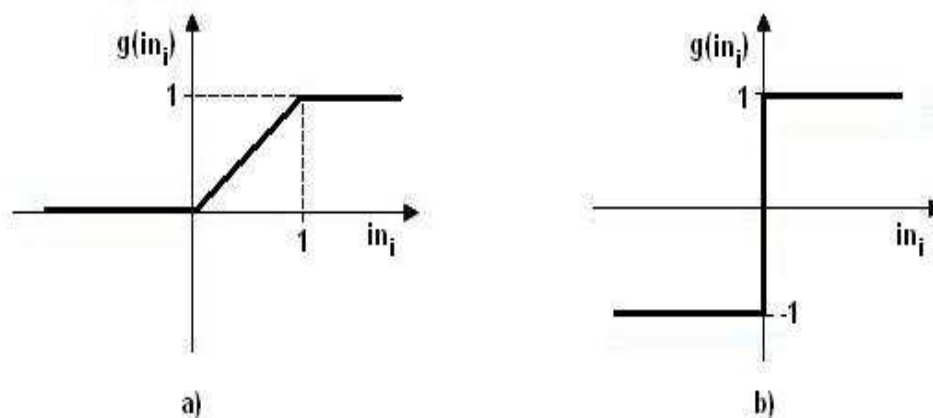


Figura 6: Algumas funções de ativações conhecidas. (a) Função Rampa. (b) Função limite ríspido.

O ajuste sináptico entre os neurônios de uma RNA representa o aprendizado em cada neurônio do fato apresentado, isto é, cada neurônio, conjuntamente com todos os outros, representa a informação que atravessou pela rede. Nenhum neurônio guarda em si todo o conhecimento, mas faz parte de uma malha que retém a informação graças a todos os seus

neurônios. Dessa forma, o conhecimento dos neurônios e, conseqüentemente, da própria rede neural, reside nos pesos sinápticos.

Dessa forma, pode-se dizer que as redes neurais artificiais têm sido desenvolvidas como generalizações de modelos matemáticos de cognição humana ou neurobiologia, assumindo que:

- O processamento da informação ocorre com o auxílio de vários elementos chamados *neurônios*;
- Os sinais são propagados de um elemento a outro através de *conexões*;
- Cada conexão possui um *peso* associado, que, em uma rede neural típica, pondera o sinal transmitido; e
- Cada neurônio (ou unidade) possui uma *função de ativação* (geralmente não-linear), que tem como argumento a soma ponderada dos sinais de entrada, a qual determina sua saída.

2.3.3 Os tipos de RNAs

A abordagem conexionista das RNAs abre um amplo leque de formas de conexão entre as unidades de processamento, isto é, os neurônios. Isso abrange o número de camadas presentes na rede, a forma de conexão entre tais unidades, a forma de treinamento, as funções de ativação presentes em cada camada etc.

Devido à grande bibliografia disponível sobre o tema redes neurais artificiais, até mesmo sua classificação gera algumas discussões. Fausett (FAUSETT, 1994), por exemplo, define *arquitetura de uma rede neural* como a disposição dos neurônios em camadas e as conexões entre as camadas. Em um sentido mais amplo, outros pesquisadores utilizam a notação arquitetura na denominação de todo um conjunto de características de uma rede, englobando sua forma de treinamento, finalidade etc.

Em (XING; PHAM, 1995), Duc Pham define dois critérios básicos para a classificação das RNAs:

- Quanto à estrutura;
- Quanto à forma de treinamento.

2.3.3.1 A estrutura das redes

Com relação às estruturas, uma rede neural pode ser definida por duas formas:

- **Redes acíclicas ou redes de alimentação direta (*feedforward*):** a propagação do processamento neural é feita em camadas sucessivas, ou seja, neurônios dispostos em camadas terão seus sinais propagados sequencialmente da primeira à última camada, de forma unidirecional. Um exemplo típico desse tipo de rede seria o Perceptron Simples ou o Perceptron Multicamadas (MLP);
- **Redes cíclicas ou redes recorrentes (*recurrent*):** as saídas de um (ou todos) os neurônios podem ser realimentadas a neurônios de camadas precedentes (tipicamente da primeira camada). Esse tipo de rede é classificado como *memória dinâmica*. Um exemplo típico dessa rede é a rede de Hopfield (HOPFIELD, 1982).

2.3.3.2 O treinamento das redes

Haykin propõe a seguinte definição para o aprendizado no contexto de redes neurais: “(...) é um processo pelo qual os parâmetros livres de uma rede neural são adaptados através de um processo de estimulação pelo ambiente no qual a rede está inserida. O tipo de aprendizagem é determinado pela maneira pela qual a modificação dos parâmetros ocorre” (HAYKIN, 2001).

Esta definição de Haykin sobre o processo de aprendizagem implica na seguinte sequência de eventos:

1. A rede neural é *estimulada* por um ambiente;
2. A rede neural *sofre* modificações nos seus parâmetros livres como resultado desta estimulação;
3. A rede neural *responde de uma maneira nova* ao ambiente, devido às modificações ocorridas na sua estrutura interna.

A forma de treinamento, então, diz respeito a como são atualizados os valores dos pesos sinápticos durante o aprendizado da rede. Neste contexto, pode-se destacar:

1. Redes com treinamento supervisionado;
2. Redes com treinamento não-supervisionado;

3. Redes com aprendizado por reforço.

Nas *redes com treinamento supervisionado*, tipicamente, uma sequência de padrões de entrada associados a padrões de saída é apresentada à rede. Esta utiliza as comparações entre a sua classificação para o padrão de entrada e a classificação correta dos exemplos para ajustar seus pesos. Enquadram-se nesse contexto a maioria das redes utilizadas, como o Perceptron Simples e o Perceptron Multicamadas (MLP).

Nas *redes com treinamento não-supervisionado* não existe a apresentação de mapeamentos entrada-saída à rede: caberá exclusivamente a ela a tarefa de realizar a classificação correta, com base na informação de número de classes (padrões de saída) e topologia da rede. Para este tipo de treinamento não se usa um conjunto de exemplos previamente conhecidos. Uma medida da qualidade da representação do ambiente pela rede é estabelecida e os parâmetros são modificados de modo a otimizar esta medida.

O *aprendizado por reforço* refere-se à utilização de alguma função heurística (definida a priori) que descreva o quão boa é a resposta da rede a uma dada entrada, e da busca da maximização de tal função (XING; PHAM, 1995). Em outras palavras, não é fornecido à rede o mapeamento direto entrada-saída, mas sim uma recompensa (ou penalização) decorrente da saída gerada pela rede à entrada apresentada. Tal reforço é utilizado no ajuste dos pesos da rede.

2.3.4 O Perceptron simples ou de camada única

Os perceptrons de única camada são o tipo mais antigo de redes neurais, as quais são formadas por uma camada única de neurônios de saída que estão conectados às entradas $x_i(n)$ através dos pesos $w_{ij}(n)$, onde $x_i(n)$ representa o i -ésimo elemento do vetor padrão de entrada na iteração n ; e, $w_{ij}(n)$ representa o peso sináptico conectando a entrada $x_i(n)$ à entrada do neurônio de saída j na iteração n (veja figura 7).

A soma do produto entre pesos e entradas alimenta cada neurônio de saída, e se o resultado desta operação exceder um certo limiar (bias), o neurônio de saída deverá ser ativado, caso contrário, será desativado. A figura (8a) apresenta um exemplo de utilização de um perceptron simples para resolver o problema do *AND* lógico. Neste caso, a unidade a_j será ativada (obtendo o valor 1) sempre que a soma ponderada do produto das entradas pelos pesos exceder w_0 , isto é, $\sum_{i=0}^2 w_{ij}.x_i > 0$. Caso contrário, a unidade a_j será desativada (obtendo o valor 0). Neurônios com esse comportamento são chamados de *neurônios de McCulloch-Pitts* ou neurônios com limiar (MCCULLOCH; PITTS, 1943). Na literatura

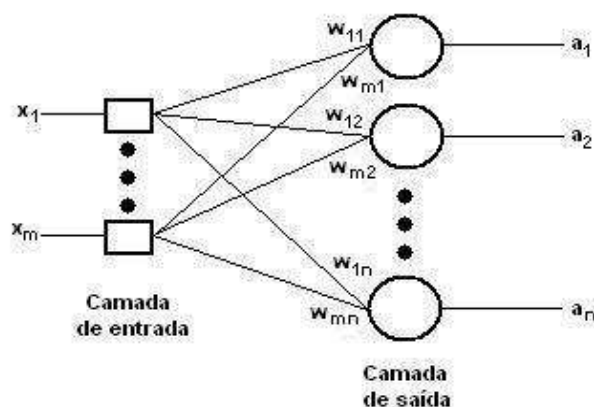


Figura 7: Arquitetura de um perceptron simples.

técnica, o termo *perceptron* diz respeito a redes com apenas um desses neurônios.

Perceptrons podem ser treinados por um algoritmo de aprendizagem simples, conhecido por *retropropagação do erro*. Este algoritmo será visto com mais detalhe nas próximas seções, onde o mesmo será aplicado ao perceptron multicamada.

O modelo do perceptron de camada única consegue aprender apenas problemas linearmente separáveis, isto é, problemas com comportamento linear que podem ser graficamente separados por meio de uma reta em um hiperplano. A figura (8b) mostra, graficamente, a separação linear de classificação do problema do *AND* lógico para as entradas X_1 e X_2 . O perceptron simples pode representar esta função *AND* porque existe uma linha que separa todos os pontos brancos de todos os pontos pretos do gráfico. Tal função é chamada de *função linearmente separável*. Por outro lado, o perceptron simples é incapaz de resolver problemas cujas funções não são linearmente separáveis, isto é, problemas que apresentam características de comportamento não linear. Como exemplo, pode ser citado o problema do *XOR*. A solução para este tipo de problema é acrescentar uma camada intermediária ao modelo do perceptron de camada única, obtendo-se, assim, o Perceptron Multicamada ou MLP.

2.3.5 O Perceptron Multicamada - MLP

Os perceptrons multicamadas ou MLPs se caracterizam pela presença de uma ou mais camadas intermediárias ou escondidas (camadas em que os neurônios são efetivamente unidades processadoras, mas não correspondem à camada de saída). Adicionando-se uma ou mais camadas intermediárias, aumenta-se o poder computacional de processamento não-linear e armazenagem da rede. Em uma única camada oculta, suficientemente grande,

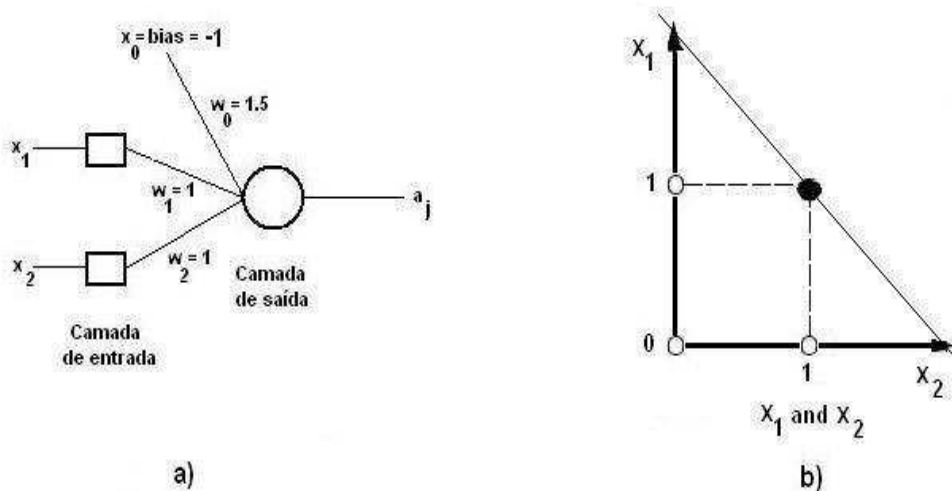


Figura 8: Problema do *AND* lógico. (a) Perceptron simples que resolve o problema do *AND* lógico. (b) Representação da função do *AND* lógico em um hiperplano.

é possível representar, com exatidão, qualquer função contínua das entradas. O conjunto de saídas dos neurônios de cada camada da rede é utilizada como entrada para a camada seguinte. A figura (9a) ilustra uma rede MLP *feedforward* com duas camadas ocultas.

As redes *feedforwards* de múltiplas camadas são geralmente treinadas usando o algoritmo de retropropagação do erro (*error backpropagation*), embora existam outros algoritmos de treinamento. Este algoritmo requer a propagação direta (*feedforward*) do sinal de entrada através da rede, e a retropropagação (propagação reversa, ou *backpropagation*) do sinal de erro, como ilustrado na figura (9b).

2.3.5.1 O algoritmo de retropropagação do erro

O algoritmo de retropropagação do erro (do inglês *error backpropagation*) é o principal algoritmo de treinamento para redes MLPs e certamente o algoritmo de treinamento mais conhecido dentro da comunidade de redes neurais. Também chamado de *regra delta generalizada*, o algoritmo é derivado do famoso método da regra delta de Widrow e Hoff (WIDROW; HOFF, 1960) e consiste em uma otimização não-linear baseada em gradiente descendente.

Basicamente, o algoritmo de retropropagação do erro é composto por duas etapas fundamentais:

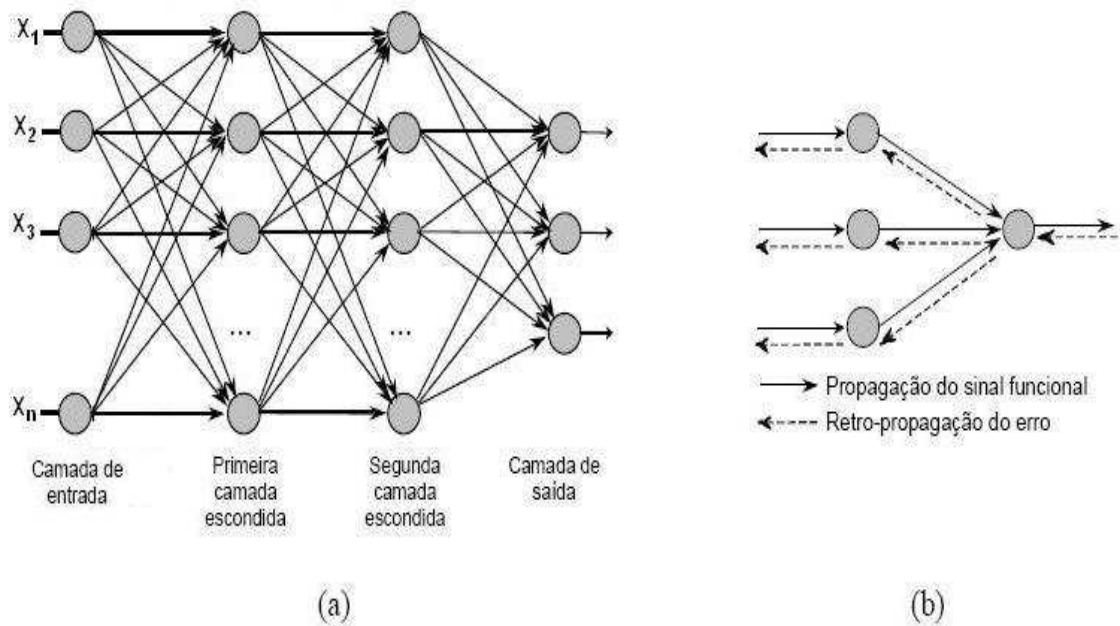


Figura 9: Redes neurais tipo *feedforward* com múltiplas camadas. (a) Arquitetura MLP. (b) Sentido de propagação do sinal funcional e do sinal de erro (HAYKIN, 2001).

1. **Etapa da propagação direta ou *forward*:** um padrão é apresentado às unidades da camada de entrada da rede neural e, a partir desta camada, as unidades calculam sua resposta a ser produzida na camada de saída, obtendo um erro em comparação com a saída esperada;
2. **Etapa da propagação reversa ou *backward*:** o erro obtido na etapa anterior é propagado a partir da camada de saída até a camada de entrada, e os pesos das conexões das unidades das camadas internas vão sendo modificadas utilizando a regra delta generalizada.

A seguir, uma lista de notações utilizada no algoritmo *error backpropagation* será apresentada a fim de melhor entender o funcionamento do mesmo (HAYKIN, 2001):

- Os índices i , j e k se referem a neurônios diferentes na rede; com os sinais se propagando através da rede da esquerda para a direita, o neurônio j se encontra em uma camada à direita do neurônio i , e o neurônio k se encontra em uma camada à direita do neurônio j , quando o neurônio j é uma unidade oculta;
- Na iteração n , o n -ésimo padrão de treinamento (exemplo) é apresentado à rede neural;

- O símbolo $e_j(n)$ se refere ao sinal de erro na saída do neurônio j , para a iteração n ;
- O símbolo $d_j(n)$ se refere à resposta desejada para o neurônio j (é usada para calcular $e_j(n)$);
- O símbolo $a_j(n)$ se refere ao sinal funcional que aparece na saída do neurônio j , na iteração n ;
- O símbolo $w_{ij}(n)$ representa o peso sináptico conectando a saída do neurônio i à entrada do neurônio j , na iteração n . A correção aplicada a este peso na iteração n é representada por $\Delta w_{ij}(n)$;
- O campo local induzido, isto é, a soma ponderada de todas as entradas sinápticas acrescida do bias) do neurônio j na iteração n é representado por $in_j(n)$; constitui o sinal aplicado à função de ativação associada ao neurônio j ;
- A função de ativação, que descreve a relação funcional de entrada-saída da não-linearidade associada ao neurônio j , é representada por $g_j(n)$;
- O bias aplicado ao neurônio j é representado por $b_j = +1$; o seu efeito é representado por uma sinapse de peso w_{0j} conectada a esta entrada fixa b_j ;
- O i -ésimo elemento do vetor padrão de entrada na n -ésima iteração é representado por $x_i(n)$;
- O k -ésimo elemento do vetor padrão de saída global na n -ésima iteração é representado por $o_k(n)$;
- O parâmetro da taxa de aprendizagem é representado por α ;
- O símbolo m_l representa o tamanho (ou número de neurônios) da camada l do perceptron de múlti-camadas; $l = 0, 1, \dots, L$ onde L é a profundidade da rede. Assim, m_0 representa o tamanho da camada de entrada, m_1 representa o tamanho da primeira camada oculta e m_L representa o tamanho da camada de saída.

Dado uma amostra de treinamento $\{x(n), d(n)\}_{n=1}^N$, o modo de execução do algoritmo de retropropagação do erro é definido por:

1. **Inicialização dos pesos:** carregue os pesos iniciais da rede a fim de iniciar o processo de treinamento;

2. **Apresentação dos exemplos de treinamento:** para cada exemplo do conjunto de treinamento realize a seqüência de propagação direta e reversa descritas nos pontos 3 e 4, abaixo, respectivamente;
3. **Propagação direta do padrão de entrada:** suponha que um exemplo de treinamento seja representado por $(x(n), d(n))$, com o vetor de entrada $x(n)$ aplicado à camada de entrada de neurônios sensoriais e o vetor resposta desejada $d(n)$ apresentado à camada de saída. Calcule os campos locais induzidos e os sinais funcionais da rede prosseguindo para frente através da rede, camada por camada. O campo local induzido $in_j^{(l)}(n)$ para o neurônio j na camada l , para $1 \leq l \leq L$, é definido por:

$$in_j^{(l)}(n) = \sum_{i=0}^{m_{(l-1)}} w_{ij}^{(l-1)}(n) \cdot a_i^{(l-1)}(n)$$

onde $a_i^{(l-1)}(n)$ é o sinal de saída do neurônio i na camada anterior $l-1$, na iteração n , e $w_{ij}^{(l-1)}(n)$ é o peso sináptico conectando a saída do neurônio i da camada $(l-1)$ à entrada do neurônio j da camada l , na iteração n . Para $i = 0$, tem-se que $a_0^{(l-1)}(n) = b_j = +1$ e $w_{0j}^{(l-1)}(n)$ é o peso do bias b_j aplicado ao neurônio j na camada l . Obtido o campo local induzido, o sinal de saída do neurônio j na camada l , para $1 \leq l \leq L$, é dado por:

$$a_j^{(l)}(n) = g_j(in_j^{(l)}(n))$$

Para o neurônio j que está na camada de entrada, isto é, $l = 0$, faça:

$$a_j^{(0)}(n) = x_j(n)$$

onde $x_j(n)$ é o j -ésimo elemento do vetor de entrada $x(n)$.

Para o neurônio j que está na camada de saída, isto é, $l = L$, onde L é denominado a profundidade da rede, faça:

$$a_j^{(L)}(n) = o_j(n)$$

Calcule o sinal de erro:

$$e_j(n) = d_j(n) - o_j(n)$$

onde $d_j(n)$ é o j -ésimo elemento do vetor resposta desejada $d(n)$;

4. **Propagação reversa ou retropropagação do erro:** calcule os gradientes locais da rede, isto é, δ s. O gradiente local δ é definido, para $1 \leq l \leq L$, por:

$$\delta_j^{(l)}(n) = \begin{cases} e_j^{(L)}(n) \cdot g'_j(in_j^{(L)}(n)), & \text{para neurônio } j \text{ da camada de saída } L \\ g'_j(in_j^{(l)}(n)) \sum_k \delta_k^{(l+1)}(n) \cdot w_{jk}^{(l)}(n), & \text{para neurônio } j \text{ na camada oculta } l \end{cases}$$

onde $g'_j(\cdot)$ representa a diferenciação em relação ao argumento. Após a obtenção do gradiente local $\delta_j^{(l)}(n)$, os pesos sinápticos da rede na camada l , para $0 \leq l \leq (L-1)$, são ajustados de acordo com a regra delta generalizada:

$$w_{ij}^{(l)}(n+1) = w_{ij}^{(l)}(n) + \alpha \cdot \delta_j^{(l+1)}(n) \cdot a_i^{(l)}(n) + \mu [\Delta w_{ij}^{(l)}(n-1)] \quad (2.1)$$

onde α é o parâmetro da taxa de aprendizagem e μ é a constante de momento a serem tratadas abaixo.

5. **Iteração:** itere as propagações direta e reversa nos pontos 3 e 4, apresentando novos exemplos de treinamento para a rede, até que seja satisfeito o critério de parada.

O algoritmo de retropropagação do erro (ou *error backpropagation*) fornece uma “aproximação” para a trajetória no espaço de pesos calculada pelo método da descida mais íngreme. Quanto menor for o parâmetro da taxa de aprendizagem α , menor serão as variações dos pesos sinápticos da rede, de uma iteração para a outra, e mais suave será a trajetória no espaço de pesos. Esta melhoria, entretanto, é obtida à custa de uma taxa de aprendizagem lenta. Por outro lado, se o parâmetro da taxa de aprendizagem α for muito grande, a fim de acelerar a aprendizagem, grandes modificações nos pesos sinápticos resultantes podem tornar a rede instável, isto é, oscilatória. Um método simples de aumentar a taxa de aprendizagem, evitando no entanto o perigo de instabilidade, é a inclusão do termo momento μ , como mostrado na equação (2.1).

Em (HAYKIN, 2001), Haykin comenta que a inclusão do termo momento no algoritmo de retropropagação tem as seguintes vantagens:

- Acelerar a aprendizagem da rede através de descidas em direções no espaço de pesos onde a declividade na superfície de erro é constante;
- Diminuir a aprendizagem mantendo um efeito estabilizador nas direções do espaço de pesos que oscilam em sinal;
- Evitar que o processo de aprendizagem termine em um mínimo local raso na superfície de erro.

A rede neural implementada no *LS-Draughts* é uma rede MLP que representa o próprio agente jogador de Damas. Maiores detalhes sobre sua implementação serão abordados nas seções 4.1 e 4.2.

2.4 Aprendizagem por Reforço

Dentro do paradigma de Aprendizagem de Máquina, as abordagens tradicionais que utilizam aprendizagem supervisionada trabalham com sistemas que aprendem através de exemplos de pares de entrada e saída. Tais pares fornecem aos sistemas indicativos de como se comportar para tentar aprender uma determinada função que “poderia” gerá-los. Formalmente, isto significa que, dados exemplos de pares $(x_i, f(x_i))$, onde x_i é a entrada e $f(x_i)$ é a saída da função aplicada a x_i , então a tarefa é encontrar, dentre uma coleção de exemplos de f , uma função h que mais se aproxime de f . Estes métodos são apropriados quando existe alguma espécie de “professor” fornecendo os valores corretos para a saída da função de avaliação. Entretanto, se não houver nenhum “professor” fornecendo exemplos, o que o agente poderá fazer? Experimentando movimentos aleatórios em seu ambiente, o agente terá que ter uma total liberdade e ser capaz de aprender, com base em recompensas ou reforços fornecidos por um “crítico” ou pelo próprio agente, através da observação das transições de estado que ele provoca no ambiente (RUSSELL; NORVIG, 2004).

Segundo Sutton e Barto (SUTTON; BARTO, 1998), Aprendizagem por Reforço (AR) nada mais é do que a aplicação dos conceitos básicos de Aprendizagem de Máquina: um indivíduo deve aprender a partir da sua interação com o ambiente onde ele se encontra, através do conhecimento do seu próprio estado no ambiente, das ações efetuadas no ambiente e das mudanças de estado que aconteceram depois de efetuadas as ações.

A importância de utilizar Aprendizagem por Reforço como uma técnica de aprendizagem está diretamente ligada ao fato de se tentar obter uma política ótima de ações. Tal política é representada pelo comportamento que o agente segue para alcançar o objetivo e pela maximização de alguma medida de reforço a longo prazo (globais), nos casos em que não se conhece, a priori, a função que modela esta política (função do agente-aprendiz).

Nas próximas seções, serão abordados os conteúdos voltados à Aprendizagem por Reforço, apresentando suas características, seus problemas, sua formulação matemática em torno do *Modelo de Markov* e o método de resolução pelo $TD(\lambda)$.

2.4.1 Características da Aprendizagem por Reforço

Dentre as diversas características da técnica de Aprendizagem por Reforço, existem cinco abordagens fundamentais que merecem destaque (RUSSELL; NORVIG, 2004):

- **Aprendizado pela Interação:** esta é a característica principal que define o próprio problema de Aprendizagem por Reforço. O agente atua sobre o ambiente e aguarda pelo valor de reforço que o mesmo deve informar como resposta à ação tomada. Assim, as próximas tomadas de decisões levam em conta este aprendizado obtido pela interação;
- **Retorno Atrasado:** um máximo valor de reforço que o ambiente envia para o agente não quer dizer necessariamente que a ação tomada pelo agente foi a melhor. Uma ação é produto de uma decisão local no ambiente, sendo seu efeito imediato de natureza local, enquanto, em um sistema de Aprendizagem por Reforço, busca-se alcançar objetivos globais no ambiente. Assim, as ações tomadas devem levar a maximizar o retorno total, isto é, a qualidade das ações tomadas é vista pelas soluções encontradas à longo prazo;
- **Filtragem:** a estrutura de um agente de Aprendizagem por Reforço deve saber identificar a recompensa ou reforço retornado pelo ambiente como uma *parte* da percepção de entrada, e não apenas como outra entrada sensorial. Além disso, o agente também deve ter um dispositivo para saber extrair informações relevantes de um conjunto de dados de entrada ruidosos;
- **Orientado ao Objeto-Ambiente:** em Aprendizagem por Reforço, o problema tratado é considerado como um ambiente que dá respostas a ações efetuadas, não sendo necessário conhecer detalhes da modelagem desse ambiente. Dessa forma, o agente age dentro do ambiente desconhecido tentando alcançar um objetivo. O objetivo é, geralmente, otimizar algum comportamento dentro do ambiente;
- **Investigação x Exploração:** este é um problema crucial no contexto da Aprendizagem por Reforço, pois, para que um sistema seja realmente autônomo o agente deve saber decidir quando aprender ou não em relação ao ambiente, levando em conta a informação já obtida até o momento. A decisão é fundamentalmente uma escolha entre agir baseado na melhor informação de que o agente dispõe no momento (investigação) ou agir para obter novas informações sobre o ambiente, explorando

ações ainda não executadas ou regiões pouco visitadas do espaço de estados (exploração). Como ambas as formas trazem, em momentos específicos, benefícios à solução dos problemas, uma boa estratégia é então mesclar estas duas formas.

2.4.2 O Problema da Aprendizagem por Reforço

Um sistema típico de Aprendizagem por Reforço (AR) constitui-se, basicamente, de um agente interagindo em um ambiente via percepção e ação. Em outras palavras, o agente percebe as situações dadas no ambiente, pelo menos parcialmente, e, baseado nessas medições, seleciona uma ação a tomar no ambiente. A ação tomada muda de alguma forma o ambiente, afetando o estado na tentativa de alcançar o objetivo relacionado, e as mudanças são comunicadas ao agente através de um sinal de reforço. A figura 10 a seguir mostra a estrutura de um problema de Aprendizagem por Reforço (SUTTON; BARTO, 1998).

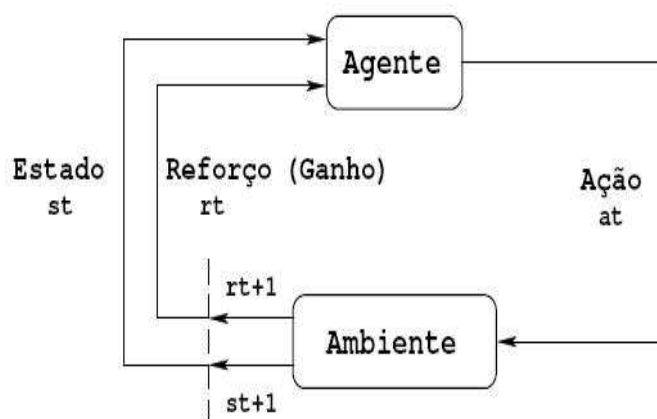


Figura 10: A interação agente-ambiente em Aprendizagem por Reforço.

Como os efeitos das ações não podem ser perfeitamente antecipados, o agente deve monitorar o ambiente freqüentemente e reagir apropriadamente. Formalmente, o modelo de um sistema AR consiste em (SUTTON; BARTO, 1998):

1. um conjunto de variáveis de estado percebidas pelo agente, onde o conjunto das combinações de valores dessas variáveis forma o conjunto de estados discretos do agente (S);
2. um conjunto de ações discretas, que escolhidas por um agente muda o estado do ambiente ($A(s)$, onde $s \in S$); e,

3. valor das transições de estados, que é passado ao agente através de um sinal de reforço, denominado ganho (valores tipicamente entre $[0,1]$).

O objetivo do método de Aprendizagem por Reforço é levar o agente a escolher a sequência de ações que tendem a aumentar a soma de valores de reforço, ou seja, é encontrar a política π , definida como o mapeamento de estados em ações que maximize as medidas do reforço acumuladas no tempo.

Definido o problema e o objetivo central da Aprendizagem por Reforço, a seguir descrevem-se, sucintamente, com base em (SUTTON; BARTO, 1998), os cinco componentes elementares deste contexto:

1. **Ambiente:** todo sistema de AR aprende um mapeamento de situações e ações por experimentação em um ambiente dinâmico. O ambiente no qual está inserido o sistema, em geral, deve ser, pelo menos parcialmente observável através de sensores, o que permite que toda informação relevante do mesmo esteja perfeitamente disponível para que o agente possa escolher ações baseadas em estados reais desse ambiente;
2. **Política:** uma política expressa pelo termo π representa o comportamento que o sistema AR segue para alcançar o objetivo. Em outras palavras, uma política π é um mapeamento de estados s e ações a em um valor $\pi(s, a)$. Assim, se um agente AR muda a sua política, então as probabilidades de seleção de ações sofrem mudanças e conseqüentemente, o comportamento do sistema apresenta variações à medida que o agente vai acumulando experiência a partir das interações com o ambiente. Portanto, o processo de aprendizado no sistema AR pode ser expresso em termos da convergência até uma política ótima ($\pi^*(s, a)$) que conduza à solução do problema de forma ótima;
3. **Reforço e Retorno:** o reforço é um sinal do tipo escalar (r_{t+1}) que é devolvido pelo ambiente ao agente assim que uma ação tenha sido efetuada e uma transição de estado ($s_t \rightarrow s_{t+1}$) tenha ocorrido. Existem diferentes formas de definir o reforço para cada transição no ambiente, gerando-se funções de reforço que, intrinsecamente, expressam o objetivo que o sistema AR deve alcançar. Assim, o agente deve maximizar a quantidade total de reforços recebidos, o que nem sempre implica em maximizar o reforço imediato (r_{t+1}) a receber, uma vez que pode corresponder, também, a maximizar o reforço acumulado durante a execução total. Tal acúmulo de reforços recebidos pelo ambiente, referente a uma determinada execução, é chamado de re-

torno. É justamente este retorno que o agente busca maximizar.

De modo geral, pode-se dizer que o sistema AR busca maximizar o valor esperado de retorno, que pode ser definido como uma função da seqüência de valores de reforço até um tempo T final. No caso mais simples é um somatório como aparece na equação seguinte:

$$R_T = r_{t+1} + r_{t+2} + r_{t+3} + \dots + r_T \quad (2.2)$$

Em muitos casos a interação entre agente e ambiente não termina naturalmente em um episódio (seqüência de estados que chegam até o estado final), mas continua sem limite, como por exemplo, em tarefas de controle contínuo. Para essas tarefas a formulação do retorno é um problema, pois $T = \infty$ e o retorno que se deseja obter também tende ao infinito ($R_T = \infty$).

Para estes problemas, é criada uma taxa de amortização (γ), a qual determina o grau de influência que têm os valores dos reforços futuros sobre o reforço total (ou retorno). Assim, aplicando a taxa de amortização à expressão do retorno da equação (2.2), têm-se a seguinte equação:

$$R_T = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \quad (2.3)$$

onde, $0 < \gamma < 1$. Se $\gamma \rightarrow 0$, o agente tem uma visão míope dos reforços, maximizando apenas os reforços imediatos, e se $\gamma \rightarrow 1$, a visão do reforço abrange todos os estados futuros dando maior importância ao estado final, desde que a seqüência R_T seja limitada.

Dessa forma, um sistema de AR faz um mapeamento de estados em ações baseado nos reforços recebidos. Com isso, o objetivo do AR é definido usando-se o conceito de função de reforço, a qual é uma função dos reforços futuros que o agente procura maximizar. Ao maximizar essa função, o objetivo será alcançado de forma ótima. É a função de reforço que define quais são os bons e maus eventos para os agentes;

4. **Função de Reforço:** as funções de reforço podem ser bastante complicadas, porém existem pelo menos três classes de problemas frequentemente usadas para criar funções adequadas a cada tipo de problema:
 - *Reforço só no estado final:* nesta classe, as recompensas são todas zero, exceto no estado final, em que o agente recebe uma recompensa (ex: +1) ou uma penalidade (ex: -1). Como o objetivo é maximizar o reforço, o agente irá aprender que os estados que levaram a uma recompensa são bons e os que levaram a uma penalidade devem ser evitados;

- *Tempo mínimo ao objetivo:* nesta classe as funções de reforço fazem com que o agente realize ações que produzam o caminho ou trajetória mais curta para alcançar um estado objetivo. Toda ação tem penalidade (-1), sendo que o estado final é (0). Como o agente tenta maximizar valores de reforço, ele aprende a escolher ações que minimizam o tempo que leva para alcançar o estado final;
- *Minimizar reforços:* em certas ocasiões o agente nem sempre precisa ou deve tentar maximizar a função de reforço, podendo também aprender a minimizá-la. Isto é útil quando o reforço é uma função para recursos limitados e o agente deve aprender a conservá-los ao mesmo tempo em que alcança o objetivo.

5. **Funções de Valor-Estado:** define-se uma função valor-estado como o mapeamento do estado, ou par estado-ação em um valor que é obtido a partir do reforço atual e dos reforços futuros.

Se a função valor-estado considera só o estado s é denotada como $V(s)$, enquanto se é considerado o par estado-ação (s, a) , então a função valor-estado é denotada como função valor-ação $Q(s, a)$.

a) **Função valor-estado $V(s)$:** uma vez que os reforços futuros mantêm dependências das ações futuras, as funções valor dependem também da política π que o AR segue. Em um *Processo de Decisão Markoviano* (ver subseção 2.4.3.2), define-se uma função valor-estado $V^\pi(s)$ dependente da política π como a equação:

$$V^\pi(s) = E_\pi[R_t | s_t = s] = E_\pi\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s\right] \quad (2.4)$$

onde a função $V^\pi(s)$ é o valor esperado do retorno para o estado $s_t = s$, isto é, o somatório dos reforços aplicando a taxa de amortização γ .

b) **Função valor-ação $Q(s, a)$:** se considerar o par estado-ação, a equação para a função valor-estado $Q^\pi(s, a)$ será a seguinte:

$$Q^\pi(s, a) = E_\pi[R_t | s_t = s, a_t = a] = E_\pi\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a\right] \quad (2.5)$$

que é semelhante à equação (2.4), só que considerando o reforço esperado para um estado $s_t = s$ e uma ação $a_t = a$.

As equações (2.4) e (2.5) apresentam as funções *valor-estado* e *valor-ação* respectivamente, que dependem exatamente dos valores de reforço, o qual implica o conheci-

mento completo da dinâmica do ambiente como um *Processo de Decisão Markoviano* (PDM). Na próxima seção resumem-se o processo PDM e suas propriedades.

2.4.3 Fundamentos da Modelagem Markoviana

Existem dois conceitos que devem ser conhecidos para facilitar a modelagem de um problema como um sistema de Aprendizagem por Reforço. A seguir, será apresentado uma breve descrição destes conceitos, baseando-se em (SUTTON; BARTO, 1998).

2.4.3.1 Propriedade de Markov

Quando a probabilidade de transição de um estado s para um estado s' depende apenas do estado s e da ação a adotada em s , isso significa que o estado corrente fornece informação suficiente para o sistema de aprendizado decidir que ação a ser tomada. Quando o sistema possui esta característica diz-se que ele satisfaz a *Propriedade de Markov* (BELLMAN, 1957).

No caso mais geral, se a resposta em $t + 1$ para uma ação efetuada em t depende de todo o histórico de ações até o momento atual, a dinâmica do ambiente é definida pela especificação completa da distribuição de probabilidades, como mostra a equação abaixo:

$$Pr\{s_{t+1} = s', r_{t+1} = r | s_t, a_t, r_t, s_{t-1}, a_{t-1}, \dots, r_1, s_0, a_0\} \quad (2.6)$$

onde a probabilidade (Pr) do próximo estado s_{t+1} ser o estado s' e o reforço r_{t+1} ser r é uma função que depende de todos os estados, ações e reforços passados.

Logo, se a resposta do ambiente à ação a_t depende apenas dos estados e reforços em t , então, a probabilidade da transição para o estado s' em $t + 1$ pode ser obtida pela fórmula a seguir (que corresponde a uma simplificação da equação 2.6 do caso geral):

$$P_{s,s'}^a = Pr\{s_{t+1} = s', r_{t+1} = r | s_t, a_t, r_t\} \quad (2.7)$$

A probabilidade de transição satisfaz às seguintes condições:

- 1) $P_{s,s'}^a \geq 0, \forall s, s' \in S, \forall a \in A(s)$; e,
- 2) $\sum_{s' \in S} P_{s,s'}^a = 1, \forall s \in S, \forall a \in A(s)$.

A *Propriedade de Markov* é de fundamental importância na AR, uma vez que tanto as decisões como os valores são funções apenas do estado atual, abrindo a possibilidade de

métodos de soluções incrementais, onde podem-se obter soluções a partir do estado atual e dos estados futuros, como é feito no método das Diferenças Temporais (ver subseção 2.4.4).

2.4.3.2 Processo de Decisão de Markov - PDM

Segundo Sutton e Barto em (SUTTON; BARTO, 1998), um *Processo de Decisão Markoviano* é definido como um conjunto de estados S , um conjunto de ações $A(s)$, $\forall s \in S$, um conjunto de transições entre estados associadas com as ações e um conjunto de probabilidades P sobre o conjunto S que representa uma modelagem das transições entre os estados. Assim, dado um par de estado e ação, a probabilidade do estado s passar para um estado s' é:

$$P_{s,s'}^a = Pr\{s_{t+1} = s' | s_t = s, a_t = a\}$$

onde Pr é a probabilidade de transição, isto é, Pr representa a probabilidade do estado s_{t+1} ser s' , sempre que o estado s_t for igual a s e a ação a_t for igual a a . Desta forma, a dependência que o estado seguinte s_{t+1} seja o estado s' está relacionada a tomar a ação a no instante t .

Similarmente, dados um estado e ação atuais e um estado seguinte s' , o valor esperado do retorno é:

$$R_{s,s'}^a = E\{r_{t+1} | s_t = s, a_t = a, s_{t+1} = s'\}$$

onde E é o valor esperado do reforço r_{t+1} , sempre que o estado s_t no instante t passe a ser o estado s' no instante $t + 1$.

Os valores de probabilidade $P_{s,s'}^a$ e retorno esperado $R_{s,s'}^a$ determinam os aspectos mais importantes da dinâmica de um PDM finito:

- 1) um ambiente evolui probabilisticamente baseado em um conjunto finito e discreto de estados;
- 2) para cada estado do ambiente existe um conjunto finito de ações possíveis;
- 3) cada passo que o sistema de aprendizado executar uma ação, é verificado um custo positivo ou negativo para o ambiente em relação à ação executada; e,
- 4) estados são observados, ações são executadas e reforços são relacionados.

Assim, para quase todos os problemas de Aprendizagem por Reforço é suposto que o ambiente tenha a forma de um *Processo de Decisão de Markov*, desde que seja satisfeita

a *Propriedade de Markov* no ambiente. Nem todos os algoritmos de AR necessitam uma modelagem PDM inteira do ambiente, mas é necessário ter-se pelo menos a visão do ambiente como um conjunto de estados e ações (SUTTON; BARTO, 1998).

A seguir, ilustrar-se-ão os conceitos vistos até o momento, através de um exemplo simples apresentado na seção 3.6 do livro “*Reinforcement Learning: An Introduction*” de Sutton e Barto (SUTTON; BARTO, 1998).

Exemplo do Robô Reciclador PDM:

O robô funciona a bateria e tem como objetivo coletar o maior número de latas possível, gastando o mínimo necessário de energia. Suas ações são definidas por um agente de Aprendizagem por Reforço que decide, periodicamente ou sempre que determinados eventos ocorram, o que o robô fará, considerando, para tanto, o seguinte conjunto de ações:

1. procurar ativamente por latas em um determinado período de tempo;
2. permanecer parado a espera de que alguém possa trazer-lhe as latas; ou,
3. voltar para a base para recarregar as suas baterias.

O agente toma suas decisões com base no nível de energia da bateria. Podem-se distinguir dois níveis, representados em (SUTTON; BARTO, 1998) por (alto, baixo), de forma que o espaço de estados é dado por $S = \{alto, baixo\}$ e suas possíveis ações são dadas por (*busca*, *espera*, *recarregar*). Dessa forma, a variável de estado do *Processo de Decisão de Markov* para o problema do robô reciclador é definida como sendo o nível de energia do robô e os valores possíveis para esta variável de estado são alto e baixo.

Sabendo que o agente se baseia no nível de energia para tomar suas ações, quando este encontrar-se no nível *alto*, tomar a ação *recarregar* não seria sensato. Assim, não se define tal ação para esse estado e o conjunto de ações do agente passa a ser: $A(alto) = busca, espera$ e $A(baixo) = busca, espera, recarregar$.

Em relação aos reforços, Sutton e Barto definiram que, para cada lata coletada, é adicionado uma recompensa (+1) e caso o robô fique sem carga o mesmo leva uma punição pesada de (-3). Como o objetivo do robô é coletar o maior número possível de latas adotou-se um retorno $R^{busca} \geq R^{espera}$, visto que terá um melhor resultado, e adotou-se, também, que nenhuma lata poderá ser coletada durante os períodos de recarga e quando a bateria estiver esgotada.

Devido ao fato de este sistema ser um típico PDM finito, então podem-se apresentar as probabilidades de transição e os ganhos previstos, como mostra a tabela 1.

$s = s_t$	$s' = s_{t+1}$	$a = a_t$	$P_{s,s'}^a$	$R_{s,s'}^a$
alto	alto	busca	α	R^{busca}
alto	baixo	busca	$1 - \alpha$	R^{busca}
baixo	alto	busca	$1 - \beta$	-3
baixo	baixo	busca	β	R^{busca}
alto	alto	espera	1	R^{espera}
alto	baixo	espera	0	R^{espera}
baixo	alto	espera	0	R^{espera}
baixo	baixo	espera	1	R^{espera}
baixo	alto	recarregar	1	0
baixo	baixo	recarregar	0	0

Tabela 1: Tabela de probabilidades das transições de estados e retornos previstos para um PDM finito

De acordo com os dados acima, observa-se, por exemplo, se o robô executar a ação de busca a partir de um estado em que a carga da bateria está alta, as probabilidades de, uma vez executada a ação, a bateria permanecer com carga alta ou passar para carga baixa são de α e $(1 - \alpha)$, respectivamente. Caso esteja no nível *baixo* e execute a ação *busca* tem-se duas possibilidades: uma com probabilidade (β) de continuar no mesmo nível, e uma probabilidade $(1 - \beta)$ de esgotar a bateria, neste caso o robô terá que ser salvo para recarregar a sua bateria. Pelo objetivo proposto, o robô não deve ficar sem energia. Caso isso ocorra, ele é severamente punido. Quando se escolhe a opção *espera* não há gasto de energia, ficando o robô no mesmo estado. Desta forma, aquelas opções em que há mudanças de estado têm probabilidade 0 de ocorrer. No caso da escolha da ação *recarregar*, o próximo estado será de bateria *alto*, não havendo outra possibilidade.

Observe que este exemplo é um típico problema de Aprendizagem por Reforço que segue a modelagem de um *Processo de Decisão de Markov*, onde estados são observados, ações são executadas e reforços são relacionados.

2.4.4 Método de solução por Diferenças Temporais

Dentre todos os algoritmos existentes para solucionar o problema da Aprendizagem por Reforço, como *Programação Dinâmica Adaptativa* (PDA), *Monte Carlo* (MC), *Q-learning* e *Sarsa*, aqui será enfocado o algoritmo de Diferenças Temporais TD(λ) de Sutton, descrito com mais detalhes em (SUTTON; BARTO, 1998).

Os métodos de Diferenças Temporais não exigem um modelo exato do sistema e permitem ser incrementais na busca de soluções para problemas de predições. Essa vantagem

de não exigir um modelo exato do sistema não é aplicado para todos os algoritmos de Aprendizagem por Reforço. Por exemplo, a Programação Dinâmica Adaptativa (PDA) requer conhecimento da distribuição de probabilidade completa para todas as possíveis transições de estado do ambiente (SUTTON; BARTO, 1998).

As Diferenças Temporais são capazes de utilizar o conhecimento anterior em ambientes parcialmente conhecidos para prever o comportamento futuro. Aprender a prever é uma das formas mais básicas e predominantes em aprendizagem. Através de um certo conhecimento, alguém poderia aprender a prever, por exemplo:

- se uma determinada disposição de peças no tabuleiro de xadrez conduzirá a vitória;
- se uma determinada formação de nuvens acarretará em chuva;
- se para uma determinada condição econômica de um país, isto implicará em um aumento ou diminuição na bolsa de valores.

Os métodos de Diferenças Temporais são guiados pelo erro ou diferença entre previsões sucessivas temporárias de estados sequenciais experimentados por um agente em um domínio. Assim, o aprendizado do agente pelo método TD é extraído de forma incremental, diretamente da experiência deste sobre o domínio de atuação, atualizando as estimativas da função valor-estado sem a necessidade de ter que alcançar o estado final de um episódio (o episódio pode ser definido como sendo um único estado ou uma sequência de estados de um domínio) para efetuar tais atualizações. Neste caso, a avaliação de uma política que defina o comportamento do agente sobre um ambiente, determinando que ação este deve executar em cada estado, é abordada como um problema de previsão, isto é, estimar a função valor-estado V^π sob a política π . A seguir, uma análise comparativa entre a equação de atualização da função valor-estado V^π do método TD e do método *Monte Carlo* será realizada a fim de verificar as vantagens de se utilizar o método TD.

Avaliação da Política - Previsão TD

Tanto TD quanto MC utilizam a experiência para resolver o problema da previsão. Dada certa experiência sob a política π , se é visitado um estado intermediário s_t , ambos os métodos atualizam suas estimativas da função valor-estado $V^\pi(s_t)$ baseando-se no que acontece depois de visitado o estado. Sendo que o método de *Monte Carlo* espera até que o retorno total de um episódio seja conhecido e usa esse retorno como objetivo para

a atualização de $V^\pi(s_t)$, tal como é mostrado na equação abaixo:

$$V^\pi(s_t) = V^\pi(s_t) + \alpha[R_t - V^\pi(s_t)], \quad (2.8)$$

onde R_t representa o retorno atual no instante t e o símbolo α representa a constante de atualização ou taxa de aprendizagem, sendo que $\alpha \in [0, 1]$.

Por outro lado, os métodos de Diferenças Temporais não necessitam alcançar o estado final de um episódio, e, sim, o estado seguinte no instante $t + 1$. Em TD são utilizados o valor de reforço imediato r_{t+1} e a função de valor estimada $V^\pi(s_{t+1})$ para o próximo estado ao invés do valor real de retorno R_t , como no método de *Monte Carlo*, isto é, em TD a atualização é executada imediatamente após cada passo. Com estas condições, nos métodos de Diferenças Temporais a equação (2.8) converte-se na equação abaixo:

$$V^\pi(s_t) = V^\pi(s_t) + \alpha[r_{t+1} + \gamma V^\pi(s_{t+1}) - V^\pi(s_t)], \quad (2.9)$$

onde a atualização se refere ao valor $r_{t+1} + \gamma V^\pi(s_{t+1}) - V^\pi(s_t)$ que precisamente define a diferença entre os tempos t e $t + 1$. É devido a esta característica que a técnica recebe o nome de método das *Diferenças Temporais*. Como a atualização é feita a partir do estado seguinte, os métodos TDs também são conhecidos como métodos *single-step*.

Algoritmo de predição TD para estimar V^π

INICIALIZAR $\mathbf{V}(\mathbf{s})$ DE FORMA ARBITRÁRIA, E π (POLÍTICA A SER AVALIADA)

REPETE (PARA CADA EPISÓDIO):

 INICIALIZAR s (ESTADO INICIAL)

 REPETE (PARA CADA PASSO DO EPISÓDIO):

$A \leftarrow$ AÇÃO DADA POR π PARA s

 TOMAR A AÇÃO a , OBSERVAR REFORÇO r E PRÓXIMO ESTADO s'

$V(s) \leftarrow V(s) + \alpha[r + \gamma V(s') - V(s)]$

$s \leftarrow s'$

ATÉ s SER O ESTADO FINAL

Vantagens dos Métodos TD

A principal vantagem do método de Diferença Temporal, em relação aos outros métodos tradicionais de Aprendizagem por Reforço, como o próprio método de *Monte*

Carlo visto acima, é que ele pode ser implementado de forma totalmente incremental, sendo que não há necessidade de aguardar até o final de um episódio para obter o retorno verdadeiro do estado atual, bastando, apenas, aguardar o estado seguinte. Existe um algoritmo baseado nas Diferenças Temporais que dispensa a política (*off-policy methods*) e utiliza a função de valor-ação $Q(s_t, a_t)$ para resolver problemas de predição. É o método *Q-learning*. Sua versão mais simples, *One-Step Q-learning*, é definida pela seguinte equação:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)], \quad (2.10)$$

onde a função de valor-ação $Q(s_t, a_t)$ é atualizada a partir do seu valor atual do reforço imediato r_{t+1} e da diferença entre a máxima função valor no estado seguinte (encontrando e selecionando a ação do estado seguinte que a maximize) e o valor da função valor-ação no estado atual. O fato de selecionar a ação que maximize a função valor no estado seguinte faz com que a função valor-ação Q aprendida aproxime-se, diretamente, da função valor-ação ótima Q^* , sem depender da política que está sendo utilizada (SUTTON; BARTO, 1998). Observe que o algoritmo *Q-learning*, da mesma forma que o $TD(\lambda)$, permite ser incremental na busca de soluções para problemas de predição.

O algoritmo de Aprendizagem por Reforço utilizado no sistema *LS-Draughts* é o método das Diferenças Temporais. Ele é utilizado para ajustar os pesos da rede neural do agente jogador de Damas. O processo de reajuste dos pesos e o funcionamento do algoritmo é discutido com mais detalhes nas subseções 4.1.4 e 4.1.5.

2.5 Computação Evolutiva

A Computação Evolutiva (CE) está fortemente embasada em mecanismos evolutivos encontrados na natureza, tais como a auto-organização e o comportamento adaptativo (GOLDBERG; HOLLAND, 1988). Esses mecanismos foram descobertos e formalizados por Darwin em sua Teoria da Evolução Natural, segundo a qual a vida na terra é o resultado de um processo de seleção pelo meio ambiente dos mais aptos e adaptados, tendo estes indivíduos mais oportunidades de reproduzirem-se e de produzir indivíduos cada vez mais aptos. A diversidade da vida, associada ao fato de que todos os seres vivos compartilham uma bagagem genética comum, é um exemplo das possibilidades do mecanismo de evolução natural em diversificar as espécies. Essa diversidade ocorre devido a recombinação gênica e mutação dos indivíduos. A recombinação gênica é responsável pela transmissão das características dos pais para os filhos. A mutação é responsável pelo

surgimento da diversidade nos indivíduos da população, com o surgimento de novas características que, se forem benéficas tornam os indivíduos mais aptos e adaptados facilitando a geração de descendentes com tais características; caso contrário, essas características tendem a ser eliminadas. Esse processo, que é a base da Teoria da Evolução Natural de Darwin, é denominado de seleção natural (DARWIN, 1859).

2.5.1 Algoritmo Genético - AG

Os Algoritmos Genéticos ou AGs foram introduzidos por John Holland na década de 1960 (HOLLAND, 1975), na Universidade de Michigan, com o objetivo de estudar formalmente os conceitos de adaptação que ocorrem na natureza, formalizá-los matematicamente, e desenvolver sistemas artificiais que mimetizam os mecanismos originais encontrados em sistemas naturais.

O AG proposto por Holland é um método que consiste em modificar uma população (conjunto de indivíduos representando as soluções candidatas codificadas na forma de cromossomos) inicial em uma nova população utilizando a seleção natural e os operadores genéticos: recombinação gênica (ou *crossover*) e mutação. Um indivíduo da população é representado por um único cromossomo, que contém a codificação (genótipo) de uma possível solução do problema (fenótipo). Cromossomos são geralmente implementados na forma de listas de atributos, vetores ou *arrays*, onde cada atributo é conhecido como gene, e os possíveis valores que um determinado gene pode assumir são denominados *alelos*. No caso particular do AG proposto por Holland, um cromossomo é geralmente representado por um vetor binário de genes. Dessa forma, o processo de evolução executado por um AG possui características de um procedimento de busca em um espaço de soluções potenciais para o problema.

Apesar dos AGs apresentarem etapas não-determinísticas em seu processo de execução, eles não são métodos de busca puramente aleatórios, pois combinam variações aleatórias com seleção, polarizada pelos valores de adequação da função de adaptabilidade ou *fitness* atribuído a cada indivíduo.

Os AGs possuem um paralelismo implícito decorrente da avaliação independente de cada uma das cadeias de gene que compõem os cromossomos, ou seja, pode-se avaliar a viabilidade de um conjunto de solução para o problema.

O processo de busca é, portanto, multi-direcional, com a manutenção de soluções candidatas que representam a busca em várias partes do domínio e com troca de informações entre essas soluções. A cada geração, soluções relativamente “boas” reproduzem-se mais freqüentemente, enquanto que soluções relativamente “ruins” tendem a ser eliminadas.

Para fazer a distinção entre diferentes soluções é empregada a função de adaptabilidade ou *fitness* que simula o papel da pressão exercida pelo ambiente sobre o indivíduo. O algoritmo a seguir descreve um AG típico.

Algoritmo Genético

FUNÇÃO AG()

```
{  
   $t \leftarrow 0$  (GERAÇÃO ZERO DO AG)  
  GERE OS  $T_P$  INDIVÍDUOS DA POPULAÇÃO INICIAL, ISTO É,  $P(0)$   
  PARA CADA INDIVÍDUO  $i$  DA POPULAÇÃO ATUAL  $P(t)$  FAÇA  
    AVALIE APTIDÃO DO INDIVÍDUO  $i$   
  FIM PARA  
  ENQUANTO CRITÉRIO DE PARADA NÃO FOR SATISFEITO FAÇA {  
     $t = t + 1$  (GERAÇÃO SEGUINTE)  
    SELECIONE A POPULAÇÃO  $P(t)$  A PARTIR DE  $P(t - 1)$   
    APLIQUE OPERADORES DE crossover SOBRE  $P(t)$   
    APLIQUE OPERADORES DE MUTAÇÃO SOBRE  $P(t)$   
    AVALIE APTIDÃO DE  $P(t)$   
    ATUALIZE  $P(t)$  SELECIONANDO OS  $T_P$  MELHORES INDIVÍDUOS DENTRE  $P(t - 1)$   
  E  $P(t)$   
  }  
}
```

No desenvolvimento de um AG para um problema particular devem-se especificar os seguintes componentes:

- representação genética para soluções potenciais (etapa de codificação do cromossomo ou indivíduo);
- procedimento para criar uma população inicial;
- definir o método de seleção dos indivíduos para a próxima geração;
- definir os operadores genéticos com base na codificação utilizada;

- definir a função de avaliação para classificar as soluções em termos de sua adaptação ao ambiente (sua capacidade de resolver o problema);
- definir o critério de parada do AG;
- valores para os diversos parâmetros do AG, como: tamanho da população, probabilidades de aplicação dos operadores genéticos e outros.

2.5.1.1 População e codificação dos indivíduos

Como foi comentado anteriormente, a população de um AG é composta por um conjunto de indivíduos que representam possíveis soluções para um determinado problema. Entretanto, o tamanho da população afeta diretamente o desempenho global e a eficiência dos resultados do AG. Populações muito pequenas tendem a perder a diversidade genética rapidamente e podem não obter uma boa solução, já que a busca realizada pelo AG cobre uma pequena parte do espaço de soluções do problema. Por outro lado, se a população for muito grande o algoritmo tenderá a ser muito caro computacionalmente (lento), principalmente se o cálculo da função de *fitness* for complexo, o que freqüentemente acontece na resolução de problemas difíceis.

No AG clássico proposto por Holland (HOLLAND, 1975), os indivíduos da população são codificados em vetores binários de tamanho fixo. A grande motivação para o emprego da codificação binária está na Teoria dos Esquemas, utilizada por Holland para justificar a eficiência dos AGs: com o passar das gerações, as soluções “boas” tendem a compartilhar partes comuns em seus cromossomos (ou indivíduos). Estas partes são chamadas de padrões. Padrões com maior aptidão do que a média da população tendem a crescer exponencialmente nas próximas gerações, enquanto que padrões com aptidões menores do que a média tendem a diminuir, também exponencialmente, isto é, as soluções convergirão para um ponto de maior aptidão (HOLLAND, 1992). Entretanto, existem outras formas de se codificar um indivíduo a fim de melhor representar as possíveis soluções do problema. O princípio dessa codificação é baseado na biologia: um indivíduo é formado por um cromossomo que contém uma seqüência de genes ou atributos que determinam a representação genótipica de uma possível solução do problema (fenótipo).

A codificação é uma das etapas mais críticas na definição de um AG. A definição inadequada da codificação pode acarretar diversos problemas, entre esses, o problema da convergência prematura do AG. A convergência prematura ocorre quando indivíduos relativamente adaptados, contudo não ótimos, rapidamente dominam a população fazendo com que o AG convirja para um máximo ou mínimo local. Dessa forma, a estrutura de

um cromossomo deve representar uma solução como um todo e deve ser a mais simples possível.

2.5.1.2 Definição da população inicial

O método geralmente utilizado na criação da população inicial é a geração aleatória dos indivíduos. Se algum conhecimento inicial a respeito do problema estiver disponível, este pode ser utilizado na inicialização da população. Por exemplo, no caso da codificação binária, se é sabido que a solução final vai apresentar mais 0's do que 1's, então esta informação pode ser utilizada, mesmo que não se saiba exatamente a proporção. Por outro lado, já em problemas com restrições, deve-se tomar cuidado para não gerar indivíduos inválidos na etapa de inicialização.

2.5.1.3 Método de seleção dos indivíduos para próxima geração

Segundo (BENTLEY, 2002), a seleção é o componente do processo evolutivo responsável por determinar os indivíduos “vencedores” e “perdedores” na luta pela sobrevivência. Caso seja vencedor, o indivíduo terá uma chance maior de ter um descendente. Caso seja perdedor, o indivíduo no mínimo terá uma chance menor de ter um descendente, e, no pior caso, será excluído da população. Portanto, a seleção desempenha papel fundamental na evolução.

Há vários métodos de seleção dos indivíduos que sofrerão as operações genéticas, dentre estes, dois merecem destaque: o método da roleta estocástica e a seleção por torneio estocástico.

O método de seleção por roleta estocástica consiste em uma analogia ao processo de girar uma roleta de um cassino, sendo que o número sorteado pela roleta corresponde ao número do indivíduo selecionado. Roletas de cassino têm a propriedade de que todos os números possuem a mesma probabilidade de serem sorteados. No caso da seleção por roleta em AG, cada indivíduo é representado na roleta por seu valor de *fitness*, ou seja, para indivíduos com maior valor de *fitness* é atribuído um número maior de casas, quando comparado aos indivíduos com baixo valor de *fitness*. A cada vez que a roleta é girada é escolhido um novo indivíduo para a população. A equação (2.11) mostra o cálculo da probabilidade p_i de o i -ésimo indivíduo da população vir a ser selecionado, considerando o valor de seu *fitness* (f_i):

$$p_i = \frac{f_i}{\sum_{j=1}^{T_P} f_j}, \quad (2.11)$$

onde f_i é assumida positiva e T_P é o tamanho da população.

O método do torneio estocástico vem de outra analogia, desta vez com torneios de competição. No método de seleção por torneio estocástico, k indivíduos da população são sorteados através de uma roleta estocástica e copiados para uma sub-população temporária (a variável k também é referenciado na literatura através da nomenclatura *tour*). Este grupo fará parte de um torneio, no qual o indivíduo ganhador é aquele que possuir o maior valor de *fitness*.

Há um fator importante na seleção em Computação Evolutiva, chamado *pressão de escolha* (ou *pressão seletiva*). Em uma seleção com alta pressão de escolha, os indivíduos mais aptos têm maior probabilidade de ser escolhidos, e com baixa pressão de escolha, a competição é menos desigual. Maior pressão seletiva significa mais prospecção e menor pressão seletiva, mais exploração. Na ausência absoluta de pressão, a busca se torna um “passeio ao acaso”. Controlar a pressão seletiva usando a roleta pode ser uma questão trabalhosa, enquanto, no torneio, isso é feito mudando-se o valor de k .

2.5.1.4 Operadores Genéticos

O princípio básico dos operadores genéticos é transformar a população, efetuando modificações em seus indivíduos, permitindo a diversificação e manutenção de características de adaptação adquiridas nas gerações anteriores. Os operadores genéticos mais freqüentemente utilizados em AGs são o *crossover* e a mutação. Esta subseção apresenta os principais aspectos relacionados a esses operadores.

Operador de *Crossover*

O operador de *crossover* ou recombinação permite a criação de dois novos indivíduos combinando-se características de dois indivíduos-pais. Pedacos dos indivíduos-pais são trocados pelo trecho equivalente do outro. A idéia intuitiva por trás do operador de *crossover* é a troca de informação entre diferentes soluções candidatas. No AG clássico é atribuída uma probabilidade fixa de ocorrer *crossover* (P_{rec}) aos indivíduos da população.

O tipo de *crossover* mais comumente empregado é o *crossover* de um ponto. Para a aplicação deste, são selecionados dois indivíduos (pais) e a partir de seus cromossomos são gerados dois novos indivíduos (filhos). Para gerar os filhos, seleciona-se um mesmo ponto de corte aleatoriamente nos cromossomos dos pais e então, os segmentos de cromossomo criados a partir do ponto de corte são trocados. A figura 11 mostra um exemplo de aplicação do operador de *crossover* de ponto único.

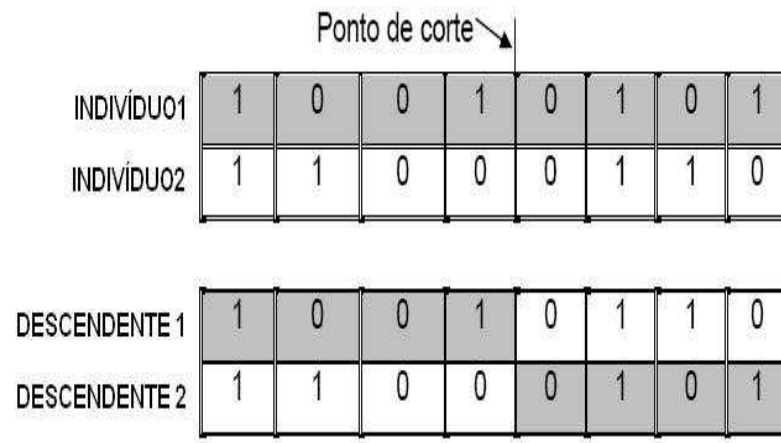


Figura 11: Exemplo do operador de *crossover* de ponto único.

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

Outro tipo de *crossover* muito comum é o *crossover* uniforme (SYWERDA, 1989): para cada *bit* no primeiro filho é decidido (com alguma probabilidade fixa p) qual dos pais vai contribuir com seu valor para aquela posição. Como o *crossover* uniforme troca bits ao invés de segmentos de bits, esse pode combinar características independentemente da sua posição no cromossomo.

Os operadores de *crossover* descritos até aqui também podem ser utilizados em cromossomos com codificação em ponto flutuante. Entretanto, existem operadores de *crossover* especialmente desenvolvidos para serem utilizados em codificação com ponto flutuante e também em outros tipos. Para mais detalhes, vejam as seguintes referências (MICHALEWICZ, 1996; MICHALEWICZ; FOGEL, 2004; HOLLAND, 1992; ESHELMAN; SCHAFER, 1992).

Operador de Mutação

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

Considerando a codificação binária, o operador de mutação padrão simplesmente troca

o valor de um gene em um cromossomo (HOLLAND, 1992). Assim, se um gene selecionado para mutação tem valor 1, o seu valor passará a ser 0 após a aplicação da mutação, e vice-versa. A figura 12 mostra um exemplo de aplicação do operador de mutação.

INDIVÍDUO ORIGINAL	1	1	0	1	0	1	0	1
INDIVÍDUO MUTADO	1	1	0	0	0	1	0	1

Ponto de Mutação

Figura 12: Exemplo do operador de mutação.

A mutação garante, dessa forma, que a probabilidade de pesquisa em qualquer região do espaço de busca nunca seja igual zero e além disso, ajuda a prevenir a perda de material genético durante a seleção.

Existem diversos outros operadores de mutação comuns na literatura. Eles variam de acordo com a codificação do indivíduo e do problema a ser abordado. Para mais detalhes, veja (GOLDBERG; HOLLAND, 1988; MICHALEWICZ, 1996).

2.5.1.5 Função de Avaliação ou *fitness*

Após cada geração de uma nova população, a qualidade de cada indivíduo é avaliada através de uma função de avaliação ou *fitness*. Esta função é considerada um dos elementos críticos do Algoritmo Genético, uma vez que define a forma do espaço de busca (*fitness landscape*). Dessa forma, o *fitness* a ser utilizado em um AG deve ser definido apropriadamente para cada problema específico.

2.5.1.6 Critério de Parada

Alguns dos possíveis critérios de parada adotados por um AG são:

- a) quando o algoritmo atingir um determinado número de gerações;
- b) quando for atingido um determinado valor para a função objetivo, definido *a priori*;
- c) quando não ocorrer melhora significativa no *fitness* do melhor indivíduo por um determinado número de gerações.

Existem outros critérios de parada que podem ser utilizados para melhor adequação ao problema em questão.

2.5.2 Outras Técnicas Evolutivas

Além dos AGs outros algoritmos também foram desenvolvidos seguindo os princípios evolutivos. Esses algoritmos são chamados de Algoritmos Evolutivos (AEs) e são estudados na área de pesquisa de Computação Evolutiva.

No contexto histórico, cinco abordagens para sistemas baseados em evolução foram paralelamente desenvolvidas. As principais diferenças entre elas dizem respeito aos operadores de reprodução empregados, estruturas de dados utilizadas para codificar os indivíduos, métodos para criar a população inicial e métodos para selecionar os indivíduos para a próxima geração. As principais abordagens em Computação Evolutiva são:

- Algoritmo Genético (AG);
- Estratégia Evolutiva (EE);
- Programação Evolutiva (PE);
- Sistemas Classificadores (SC);
- Programação Genética (PG).

O Algoritmo Genético implementado no sistema *LS-Draughts* é utilizado para gerar, automaticamente, um conjunto mínimo de características do domínio do jogo de Damas a fim de mapear os estados do tabuleiro do jogo na entrada da rede neural do agente jogador de Damas. Esse processo é discutido com detalhes na seção 4.2.

2.6 Considerações Finais

Neste capítulo foram introduzidos os primeiros conceitos de agentes inteligentes que são capazes de aprender através da interação com o ambiente. Assim, os agentes inteligentes devem ser capazes de se comportarem como “funções” do ambiente, isto é, se o meio envolvente se alterar, o agente deve ser capaz de se adaptar na mesma medida e, orientado por certos objetivos, deve ser capaz de adaptar os meios aos fins desejados.

É provável que as soluções para a incorporação de inteligência aos agentes de aprendizagem surjam através de uma combinação adequada das várias abordagens vistas neste capítulo, a saber:

- Busca *Minimax*: do planejamento de ações futuras, com base no estado atual, para escolha da melhor ação a ser executada emerge comportamento inteligente (planejamento e classificação de ações);
- Redes Neurais: da interação algorítmicamente controlada entre interligações paralelas de funções de múltiplas variáveis emerge o comportamento inteligente (casamento de padrões);
- Diferenças Temporais: da interação algorítmica entre elementos de sistemas dinâmicos, obtidos por previsões sucessivas e pelos reforços retornados pelo ambiente ao longo do tempo, emerge comportamento inteligente (aprendizagem por previsões sucessivas temporais);
- Algoritmos Genéticos: da interação algorítmica entre elementos de sistemas dinâmicos, estocasticamente mutáveis e sujeitos a pressões ambientais, emerge comportamento inteligente (estratégias aptas a enfrentar as pressões ambientais).

Como este trabalho visa utilizar uma integração entre estas quatro abordagens, para melhor incorporar a inteligência a um agente, uma proposta detalhada será apresentada no capítulo 4 para o domínio de Damas. O objetivo é obter um agente que seja capaz de jogar Damas com alto nível de desempenho.

3 *Estado da Arte*

Devido à vasta aplicabilidade de técnicas de aprendizagem em jogos de computadores, este capítulo apresentará alguns dos aspectos atualmente mais relevantes do estado da arte em programas que aprendem a jogar. O objetivo é clarificar as técnicas mais relevantes para os problemas existentes em diferentes aspectos dos jogos, e também apontar os tópicos mais recompensadores na aplicação dessas técnicas de aprendizagem nos cenários dos jogos.

3.1 Introdução

Para que um jogador automático tenha a eficácia de um perito, isto requer habilidade no tratamento de memória, reconhecimento de padrões e capacidades sofisticadas de planejamento. A maioria dos algoritmos utilizados em jogos normalmente utilizam uma função de avaliação que retorna uma unidade escalar vinculada a uma dada posição do jogo. Em jogos complexos, como o Go, Xadrez e Damas, devem ser utilizadas técnicas de busca rigorosas onde milhões de posições têm de ser avaliadas antes de ser encontrada uma solução viável. A necessidade destas estratégias de busca provém das muitas descontinuidades (ou exceções) na função de avaliação que são causadas pelas diferentes combinações de peças no tabuleiro. Para estes jogos, teríamos de representar todas essas descontinuidades no modelo da função de avaliação, o que é quase impossível, daí a utilização de aproximações que utilizam regras simbólicas ou as Redes Neurais (atualmente, a forma mais popular de aproximador de função) (HAYKIN, 2001). Como exemplo, pode-se citar o *TD-GAMMON* de Tesauro (TESAURO, 1995) e o *NeuroDraughts* de Lynch (LYNCH, 1997). Ambos trabalhos utilizam redes neurais como função de avaliação para treinar seus agentes jogadores.

Por outro lado, o programador do jogo tem de fornecer ao programa um conjunto de funções (rotinas) que calculam importantes propriedades de um estado do tabuleiro (por exemplo, o número de peças de cada oponente no jogo, o tamanho do território ocupado

etc), de tal forma que tais propriedades possam ser combinadas e, a partir daí, extrair importantes informações. São estas informações que caracterizam o domínio e que servem como um meio pelo qual a função de avaliação do agente adquirirá novos conhecimentos.

3.2 Tipo de Treinamento

Para que um agente especialista aprenda a jogar algum tipo de jogo, uma estratégia de treinamento deve ser aplicada à sua função de avaliação de forma que esta possa otimizar as ações do agente e ajudá-lo a se comportar no ambiente que atuará. Neste sentido, pode-se citar alguns tipos de treinamento:

- **Treino Supervisionado:** a função de avaliação é treinada em cima de informações com saídas corretas, isto é, o agente recebe exemplos de posições ou jogadas que são rotuladas pela correta avaliação das mesmas (avaliação esperada) (RUSSELL; NORVIG, 2004). Em jogos, é muito difícil um ser humano fornecer avaliações precisas e consistentes de grande número de posições que seriam necessárias para treinar uma função de avaliação diretamente a partir de exemplos. Entretanto, Johannes Fürnkranz demonstra em (FÜRNKRANZ, 2001) algumas alternativas e propostas de se combinar a aprendizagem supervisionada com outras técnicas de aprendizagem de máquina para jogos. O algoritmo mais famoso de treinamento supervisionado é a retropropagação do erro pela regra delta generalizada (algoritmo *backpropagation*). Um padrão é apresentado às unidades da camada de entrada e, a partir desta camada as unidades da rede calculam sua resposta que é produzida na camada de saída, obtendo um erro em comparação com a saída esperada (saída correta). A partir daí o erro é propagado da camada de saída em direção a camada de entrada, e os pesos das conexões das unidades das camadas internas vão sendo modificadas utilizando a regra delta generalizada (RUMELHART; HINTON; WILLIAMS, 1986);
- **Treino por Métodos Estatísticos:** os agentes manipulam as incertezas do ambiente utilizando teorias probabilísticas de como o domínio funciona a partir de suas experiências. Estes métodos estatísticos de aprendizagem variam desde o cálculo simples de médias até a construção de modelos complexos, como redes bayesianas (RUSSELL; NORVIG, 2004);
- **Treino Evolutivo:** as funções de avaliações, que são responsáveis pela atribuição de um valor escalar as posições de um jogo, são evoluídas utilizando técnicas da

Computação Evolutiva. Andrew Rae em (RAE, 2001) obteve bons resultados com seu agente jogador de Damas, evoluindo-o por meio de um Algoritmo Genético que atua sobre a otimização dos coeficientes e parâmetros de sua função polinomial (mesma função de avaliação utilizada por Samuel em (SAMUEL, 1959));

- **Treino por Reforço:** este é um dos tipos de treinamento mais utilizado em jogos ou até mesmo em outros domínios. Segundo (SUTTON; BARTO, 1998) a Aprendizagem por Reforço significa aprender a jogar de forma a poder, incrementalmente, testar e refinar a função de avaliação. O agente não recebe qualquer informação direta acerca do valor absoluto ou relativo dos exemplos de treino. Em vez disso, recebe um sinal escalar do ambiente que lhe indica a eficiência das jogadas efetuadas. No paradigma da AR é necessário apenas explicitar as regras do jogo, incluindo os estados dominantes, e disponibilizar um módulo de aprendizagem, não sendo necessária a ajuda de peritos. O primeiro a construir um sistema de Aprendizagem por Reforço foi Samuel. Nele, um algoritmo complexo foi utilizado para selecionar ajustamentos nos parâmetros baseando-se na diferença entre as sucessivas avaliações de posições bem sucedidas em um jogo, a fim de aprender a jogar Damas (SAMUEL, 1967). A aprendizagem por Diferença Temporal (TESAURO, 1995; SUTTON, 1988) é um caso especial de Aprendizagem por Reforço que fornece um método eficiente para receber exemplos de treino com uma precisão mais elevada, uma vez que a avaliação de uma dada posição (estado) é ajustada utilizando as diferenças entre a sua avaliação e as avaliações de posições sucessivas. Dessa forma, a previsão do resultado do jogo a partir de uma certa posição está relacionada com as previsões das posições seguintes. Uma das aplicações mais conhecidas em jogos, que utilizam o método das Diferenças Temporais, são: Damas, Xadrez, Go, Gamão e Othello (LYNCH, 1997; SCHAEFFER et al., 2001; SAMUEL, 1959, 1967; SCHRAUDOLPH; DAYAN; SEJNOWSKI, 2001; THRUN, 1995; TESAURO, 1994; LEUSKI, 1995). Nas seções seguintes, serão abordados, com mais detalhe, trabalhos envolvendo jogadores que utilizam Aprendizagem por Reforço como técnica de aprendizagem, uma vez que o jogador utilizado na presente proposta de trabalho utiliza tal técnica.

3.3 Aprendizagem por Reforço

Na modelagem conceitual de Aprendizagem por Reforço, o agente tem de aprender a se comportar em um ambiente através de percepções e recompensas, selecionando a melhor ação vinculada a cada estado. Contudo, ao contrário da Aprendizagem Supervisionada,

o agente não possui nenhum professor para dizer quais ações devem ser executadas em cada circunstância. Dessa forma, o agente deve explorar as diferentes ações possíveis para um estado e receber do ambiente um sinal escalar de reforço que reflita a qualidade das suas ações. No contexto dos jogos, as ações são, tipicamente, as jogadas legais a partir do estado atual do jogo, e o sinal de reforço indica se o agente ganha ou perde o jogo.

O *MENACE*, aplicação *Matchbox Educable Noughts and Crosses Engine* de Michie, foi uma das primeiras aplicações de técnicas de aprendizagem em jogos. Ele utilizou uma forma de Aprendizagem por Reforço, antes mesmo desta ser considerada uma área científica, para tentar aprender a jogar o “*Tic Tac Toe*” (ou Jogo da Velha). O *MENACE* tinha um peso associado a cada uma das 287 posições (ou estados) diferentes do jogo. Em cada estado, todas as possíveis ações (todos os quadrados ainda não ocupados) também tinham um peso atribuído. A ação seguinte era selecionada de acordo com uma distribuição de probabilidade correspondente aos pesos das diferentes escolhas. O reforço era obtido em função do resultado do jogo, isto é, as jogadas do agente eram penalizadas ou recompensadas através da atualização de seus pesos (MICHIE, 1963).

Por outro lado, o problema principal a ser resolvido por um agente que utiliza a Aprendizagem por Reforço como método de aprendizagem é justamente *o problema da atribuição dos créditos*, isto é, o problema de distribuir a recompensa recebida pela ações responsáveis. Por exemplo, em um jogo perdido, apenas uma jogada pode ter sido decisiva para a derrota. Neste caso, apenas esta jogada deverá receber a recompensa negativa, pois todas as demais jogadas podem ter sido boas. A subseção seguinte abordará esta questão.

3.3.1 O problema da atribuição de crédito

Quando se estudam algoritmos de Aprendizagem por Reforço, é útil considerar a noção da atribuição de crédito (BORGA, 1993). Basicamente, *o problema da atribuição dos créditos* a ser resolvido por um agente é o problema de se atribuir crédito ou culpa por resultados globais a cada uma das decisões internas que tenham sido tomadas por uma máquina de aprendizagem e que tenham contribuído para aqueles resultados.

Em muitos casos, a dependência dos resultados em relação a decisões internas é medida por uma seqüência de ações tomadas pela máquina de aprendizagem. Em outras palavras, as decisões internas afetam a escolha das ações particulares que são tomadas e, com isso, as ações, e não as decisões internas, influenciam diretamente nos resultados globais. Dessa forma, Sutton dividiu o problema de atribuição de crédito em dois subproblemas (SUTTON, 1984):

1. *A atribuição de crédito por resultados a ações*: este é o chamado *problema da atribuição de crédito temporal* que envolve os instantes de tempo quando as ações que merecem crédito foram realmente tomadas.
2. *A atribuição de crédito por ações a decisões internas*: este é o chamado *problema da atribuição de crédito estrutural* que envolve atribuir crédito às estruturas internas das ações geradas pelo sistema.

O *problema da atribuição de crédito estrutural* é relevante no contexto de uma máquina de aprendizagem com múltiplos componentes quando se deve determinar, precisamente, qual componente particular do sistema deve ter seu comportamento alterado e qual medida deve ser tomada a fim de melhorar o desempenho global do sistema. Por outro lado, o *problema da atribuição de crédito temporal* é relevante quando houve muitas ações tomadas por uma máquina de aprendizagem que acarretam certos resultados e que se deve determinar quais dessas ações foram responsáveis pelos resultados. O problema combinado de atribuição de crédito temporal e estrutural é enfrentado por qualquer máquina de Aprendizagem por Reforço que se esforce em melhorar seu desempenho em situações envolvendo comportamento estendido no tempo (WILLIAMS, 1988; HAYKIN, 2001).

Michie (1963) propôs duas técnicas para tentar resolver o *problema da atribuição dos créditos*, em especial a atribuição de crédito temporal. A primeira técnica, simplesmente fornece o mesmo crédito a todas as jogadas de uma partida e, a segunda, assume que as posições que ocorrem mais tarde durante o jogo possuem um impacto maior no resultado final do que as posições que ocorrem no início. Esta técnica simples não impede que as boas jogadas recebam reforço negativo (quando se comete um erro no fim do jogo) e nem que as más jogadas recebam reforço positivo (quando o jogo é ganho porque o oponente não aproveitou o erro cometido). Contudo, a idéia é que, após muitos jogos, as boas jogadas tenham recebido mais reforços positivos do que negativos e vice versa, de forma que a função de avaliação eventualmente convirja para um valor válido.

Mais de três décadas após a criação do *MENACE*, Sutton e Barto confirmariam esta proposição através de teoremas de convergência para a Aprendizagem por Reforço (SUTTON; BARTO, 1998). Entre as diversas técnicas de Aprendizagem por Reforço existentes, o método das Diferenças Temporais (TD) também faz uso desta proposição de resolução do *problema da atribuição dos créditos* para melhor reforçar as ações executadas por um agente durante o seu processo de aprendizagem e interação com o ambiente.

3.4 Método das Diferenças Temporais

Uma pequena revolução no campo da Aprendizagem por Reforço ocorreu quando Gerald Tesauro apresentou os seus primeiros resultados do treino de uma função de avaliação por meio do método das Diferenças Temporais (TESAURO, 1992, 1995). O programa de Tesauro, *TD-GAMMON*, é um jogador de Gamão que, demandando pouco conhecimento sobre o jogo de Gamão, conseguiu atingir resultados ao nível dos maiores jogadores mundiais (TESAURO, 1994). O algoritmo de aprendizagem utilizado no *TD-GAMMON* é uma combinação do algoritmo $TD(\lambda)$ com uma função de aproximação não-linear baseada em uma rede neural multicamada que apresenta duplo papel: atua como previsora de retorno esperado da posição do tabuleiro e como um meio de selecionar jogadas. Em qualquer posição (ou estado), a jogada seguinte é escolhida de forma gananciosa, avaliando todas as posições alcançáveis a partir do estado atual e selecionando então aquela com o melhor retorno. Assim, os pesos da rede neural são atualizados de acordo com o algoritmo $TD(\lambda)$.

Modelar uma função de avaliação como uma rede neural gera algumas dúvidas, como por exemplo: qual é a melhor topologia de rede a ser utilizada? E como deve ser feita a representação e o mapeamento do estado na entrada da rede neural? Tesauro (TESAURO, 1995) adicionou um número de atributos relevantes para o jogo do Gamão à informação que codificava a entrada da rede, de modo a aumentar o conhecimento disponível para a rede neural. Com esta codificação, Tesauro conseguiu aumentar o desempenho do seu programa. Mark Lynch, em seu programa *NeuroDraughts*, também mapeou o tabuleiro em um conjunto de características, com o objetivo de aumentar a habilidade da função de avaliação em generalizar bem e, assim, ter condições de obter maiores informações sobre o jogo (LYNCH, 1997).

Muitos autores, entre os quais (POLLACK; BLAIR, 1998; SCHRAUDOLPH; DAYAN; SEJNOWSKI, 2001), discutiram algumas peculiaridades do Gamão que o tornaram particularmente apto a aprender pelo método das Diferenças Temporais. A próxima subseção analisará estas questões que contribuíram para o sucesso de Tesauro.

3.4.1 O Sucesso de *TD-GAMMON*

Pollack e Blair (POLLACK; BLAIR, 1998) fundamentaram em 1998 a hipótese de que o sucesso do *TD-GAMMON* não está ligado às técnicas de aprendizagem por Diferença Temporal, mas sim a uma predisposição inerente à própria dinâmica do jogo de Gamão, assim como à natureza do próprio processo de treino, no qual a tarefa muda dinamicamente à medida que a aprendizagem ocorre. Os fatores que Pollack e Blair citam como

determinantes para o sucesso do *TD-GAMMON* são:

- **Rapidez do Jogo:** *TD-GAMMON* aprendia a partir de vários jogos contra si mesmo (*self-play*). Nas abordagens que utilizam Aprendizagem por Reforço, o número de jogos de treino é importante para o sucesso da aprendizagem.
- **Suavidade na representação:** a avaliação de um estado do tabuleiro no Gamão é uma função razoavelmente suave de posição, facilitando uma boa aproximação por rede neural;
- **Natureza Estocástica:** devido ao fato de o Gamão ser jogado com lançamento de dados, isto implica na exploração de uma boa parte da quantidade do espaço de estados, forçando o sistema a entrar em regiões desse espaço que ainda não foram vistas pela função de avaliação corrente.

Contudo, e apesar do sucesso do *TD-GAMMON* sobre os seus predecessores, que foram treinados por aprendizagem supervisionada ou treino por comparação, alguns pesquisadores não concordam que a aprendizagem por Diferença Temporal seja a melhor solução para todos os jogos. Como exemplo, pode ser citado o trabalho de Wait-kit que consistiu em investigar a viabilidade da aplicação do algoritmo TD em jogos Go. Através de uma série de experimentos, Wait-kit observou que é praticamente difícil sua aplicabilidade devido à característica determinística do jogo e principalmente à sua complexidade (CHAN, 1996). Uma outra citação é o trabalho de Samuel (SAMUEL, 1967) que em 1967 obteve bons resultados com seu jogador de Damas, treinando-o por comparação a partir de 150.000 jogadas de peritos (aprendizagem supervisionada).

Por outro lado, Sutton demonstrou que a convergência do algoritmo TD está associada a sistemas que tenham como estrutura inerente uma *Cadeia de Markov Absorvente*. Uma *Cadeia de Markov* é um processo estocástico que satisfaz a *Propriedade de Markov* vista na subseção 2.4.3.2, isto é, a probabilidade de evolução para um determinado estado s' só depende do estado atual s e da ação a selecionada em s . Uma *Cadeia de Markov* é *absorvente* se existirem estados finais que uma vez alcançados não mais são abandonados. Em alguns problemas, associam-se aos estados finais um determinado valor ou custo. É este custo que o algoritmo $TD(\lambda)$ pretende estimar em cada estado s . Portanto, a adequada *representação dos estados* torna-se um outro fator importante para garantir a convergência do algoritmo. Além disso, também é necessário para a convergência do algoritmo $TD(\lambda)$ a obtenção de padrões de treino que sejam representativos. Dessa forma, se a amostragem não for representativa da *Cadeia de Markov* o sistema pode não convergir

(SUTTON, 1988). No caso do jogador de Damas de Mark Lynch e Schaeffer (LYNCH; GRIFFITH, 1997; LYNCH, 1997; SCHAEFFER et al., 2001), estes requisitos são satisfeitos, o que torna o domínio de Damas adequado para se utilizar o método das Diferenças Temporais.

3.4.2 Uma proposta de sucesso para outros jogos

Apesar de Pollack, Blair e outros pesquisadores demonstrarem grandes dúvidas sobre a afinação dos pesos de uma função de avaliação por Diferença Temporal ser suficiente para exibir os mais elevados níveis de desempenho, isto é, de que os métodos de aprendizagem por Diferença Temporal sejam eficazes o suficiente para obtenção de programas de jogos com alto nível de desempenho, alguns pesquisadores, entre eles, Jonathan Schaeffer, Mark Lynch e Schraudolph, obtiveram alguns resultados que contra-argumentam estas dúvidas.

O projeto *CHINOOK*, iniciado em 1989 como uma iniciativa para melhor entender as buscas heurísticas, foi campeão mundial de Damas em agosto de 1994 ao empatar 6 jogos com o Dr. Marion Tinsley que até então defendia seu título mundial a mais de 40 anos (SCHAEFFER et al., 2001; SCHAEFFER, 1997). Os pesos da função de avaliação de *CHINOOK*, que consistia numa combinação linear de 84 parâmetros - 21 características baseada em conhecimento para cada uma das 4 fases do jogo, foram afinados manualmente, ao longo de 5 anos, por meio de testes extensivos em jogos contra si mesmo e em centenas de jogos contra os melhores jogadores humanos (incluindo informações perfeitas sobre quais posições de fim de jogo podem levar a vitórias, derrotas ou empates; além de boas estratégias para se começar um jogo).

Recentemente, Jonathan Schaeffer e outros pesquisadores levantaram a seguinte questão: é possível substituir a afinação manual dos pesos da função de avaliação do *CHINOOK* por aprendizagem por Diferença Temporal? Os dados experimentais obtidos em (SCHAEFFER et al., 2001) indicam que a resposta é “sim”. Este trabalho realiza um estudo detalhado de comparação entre uma função de avaliação treinada manualmente por peritos e uma função aprendida por Diferença Temporal. O objetivo disso é então verificar se a aprendizagem TD é capaz de alcançar, por si próprio, um alto nível de desempenho requerido em programas que jogam. Além disso, o autor também apresenta novas pistas sobre a aplicação de aprendizagem por Diferença Temporal em programas que jogam.

A primeira abordagem do agente jogador de Damas em (SCHAEFFER et al., 2001) consistiu em treinar os pesos jogando contra o próprio *CHINOOK* para determinar a eficácia da aprendizagem face ao benefício de jogar contra um oponente de alto desempenho. O segundo conjunto de experiências envolveu o jogo contra si próprio (estratégia de treino

por *self-play*) a fim de verificar se a aprendizagem poderia alcançar um alto nível de desempenho sem ter o privilégio de treinar jogando contra um oponente forte (em ambos os casos, o treino foi realizado utilizando *look-ahead* de 5, 9 e até 13 níveis de profundidade).

Os resultados do treino por *self-play* obtidos por Schaeffer evidenciam que não é necessário um bom professor para que o programa aprenda um conjunto de pesos de uma função de avaliação que alcance um desempenho equiparável ao de um campeão mundial. Note que isto constitui uma ótima notícia para os desenvolvedores de programas que jogam, já que sugere que a afinação manual dos pesos é uma coisa do passado (SCHAEFFER et al., 2001).

Schaeffer também cita que os resultados obtidos com seu jogador de Damas treinado por *self-play* foram bem melhores do que os resultados obtidos por *KnightCap*, um jogador de xadrez que também utiliza TD treinado por *self-play* e que foi desenvolvido por (BAXTER; TRIGDELL; WEAVER, 1998a; BAXTER; TRIDGELL; WEAVER, 1998b). Além disso, o autor aponta que uma das causas do baixo desempenho de *KnightCap*, em relação aos dados de seu jogador, é que, provavelmente, o uso de um número relativamente grande de parâmetros a serem ajustados comprometeu o seu desempenho (de 1500 parâmetros inicialmente utilizados, passou-se a 6000). A conclusão é de que poucos parâmetros na função de avaliação são mais fáceis de serem ajustados.

Apesar de a aprendizagem por Diferença Temporal prometer reduzir o esforço na construção de um programa que jogue com alto nível de desempenho, a escolha das características que melhor representam o conhecimento sobre o domínio a serem adicionadas à função de avaliação ainda é feita, geralmente, por um processo manual (algumas das características da função de avaliação de *CHINOOK* foram o resultado de uma extensa análise humana do jogo do programa para identificar as suas deficiências) (SCHAEFFER et al., 2001). O melhor jogador de Mark Lynch, que chegou a um bom nível de jogo após 2.000 jogos de treino por $TD(\lambda)$, também fez uso de uma escolha manual de características do domínio de Damas para aprender a jogar (LYNCH, 1997).

Schraudolph e outros pesquisadores verificaram que a eficiência da aprendizagem do Go utilizando métodos das Diferenças Temporais com redes neurais, pode ser aumentada, consideravelmente, utilizando não apenas uma arquitetura de rede com estrutura apropriada, mas também através de um sinal de reforço local, mais rico, e de estratégias de treino que incorporam o jogo contra si mesmo, mas sem depender exclusivamente do jogo em questão - a idéia é a de que um sistema inteligente deve aprender pela sua própria experiência, isto é, *self-play*. Assim, além do sinal de reforço fornecido no fim do jogo, foi acrescentado um sinal $r(t)$ de +1 ou -1 de acordo com a captura de prisioneiros durante

o jogo. A experiência mostrou que as vantagens de incorporar sinais de reforço locais compensam largamente a desvantagem de fixar o parâmetro λ , do método $TD(\lambda)$, em 0 (SCHRAUDOLPH; DAYAN; SEJNOWSKI, 2001).

3.5 Complexidade dos Jogos

Em (HERIK; UITERWIJK; RIJSWIJCK, 2002) encontra-se uma análise exaustiva das principais características dos jogos que mais influenciam em sua complexidade. Em particular, são definidas duas medidas de complexidade em jogos: a *complexidade do espaço de estados* e a *complexidade da árvore do jogo*. A complexidade do espaço de estados é definida como o número de posições de jogo legais que podem ser atingidas a partir da posição inicial do jogo. A complexidade da árvore do jogo é definida como o número de folhas na árvore de busca da solução do jogo a partir de uma posição (ou estado) atual. Em outras palavras, a complexidade da árvore do jogo é determinada pelo fator de ramificação do jogo em questão. A principal análise feita em (HERIK; UITERWIJK; RIJSWIJCK, 2002) é a de que uma baixa complexidade do espaço de estados é mais importante do que uma baixa complexidade na árvore do jogo como fator determinante para se resolver os problemas dos jogos.

A figura 13, que foi extraída e compilada por (CAMPOS; LANGLOIS, 2003), compara o fator de ramificação e o espaço de estados de alguns jogos.

Jogo	Ramificação	Estados
Xadrez	30–40	10^{50}
Damas	8–10	10^{17}
Gamão	± 420	10^{20}
Othello	± 5	$< 10^{30}$
Go 19×19	± 360	10^{160}
Abalone	± 80	$< 3^{61}$

Figura 13: Complexidade do espaço de estados e fator de ramificação de alguns jogos

3.6 Representações do Estado

A representação do estado de um sistema de aprendizagem é de fundamental importância para o processo de aprendizagem. Conseqüentemente, esta questão tem sido

um dos segmentos de pesquisa mais investigados e discutidos no mundo dos jogos. A seguir, listam-se dois aspectos considerados mais relevantes no assunto:

1. **Exploração de características espaciais e temporais:** a exploração de características espaciais e temporais do estado de um jogo pode auxiliar e conduzir a uma representação de estado bastante eficiente, permitindo uma maior facilidade na obtenção de boas estratégias de jogo. Schraudolph et al. propuseram uma aproximação baseada em redes neurais que refletisse as características espaciais do jogo do Go 9x9, de tal forma a criar uma função de avaliação mais precisa, tornando o processo de treino mais veloz e estável (SCHRAUDOLPH; DAYAN; SEJNOWSKI, 2001). Como se mostrou na seção anterior, o Go possui um elevado fator de ramificação o que torna a avaliação de posições extremamente difícil. Entretanto, as posições das peças do Go são invariantes no que diz respeito à reflexão x rotação do tabuleiro. Dessa forma, Schraudolph et al. fizeram a rede neural obedecer a esta invariância criando grupos de simetria de oito unidades escondidas, cada uma delas observando a mesma entrada sob uma diferente rotação/reflexão, através de pesos compartilhados. Os resultados mostraram que a inclusão de características espaciais e temporais de um jogo em uma estrutura de rede neural pode auxiliar na obtenção de uma função de avaliação mais precisa, tornando o processo de treino mais veloz e estável;
2. **Representação das relações entre as peças do jogo:** mais importante do que representar as peças de um jogo de tabuleiro, é estabelecer uma boa representação das relações entre uma peça e as demais peças do tabuleiro. Um dos primeiros trabalhos a utilizar uma representação de relações entre as peças de um jogo foi Arthur Samuel (SAMUEL, 1959). Ele implementou 26 características representativas do domínio de Damas baseando-se em análises de peritos sobre as posições e relacionamentos das peças sobre o tabuleiro de Damas. Mark Lynch também utilizou um tipo de representação semelhante ao de Samuel a fim de estabelecer as relações entre as peças de um tabuleiro de Damas. Seu jogador utilizou 12 características para aprender a jogar Damas (LYNCH, 1997). Levinson e Weber (LEVINSON; WEBER, 2002) construíram uma representação interessante para as relações entre as peças de um tabuleiro de Xadrez. Um tabuleiro de Xadrez é representado por 64 vizinhanças: uma para cada quadrado do tabuleiro de Xadrez. Cada vizinhança possui um centro e 16 “satélites” que correspondem às peças que estão imediatamente próximas nas 4 diagonais, 2 ranks, 2 filas e 8 movimentos de cavalo em relação ao quadrado. O jogador de Xadrez de Levinson e Weber consistiu em uma rede neural

de regressão de duas camadas treinada pelo método das Diferenças Temporais e com busca em profundidade de 4 níveis. Para estimar o desempenho do agente desenvolvido, os autores treinaram-no jogando no ICC (*Internet Chess Club*) e também a partir de várias centenas de jogos de Mestres do Xadrez, disponíveis em bases de dados online. O nível de jogo alcançado em apenas alguns dias de treino no ICC fez com que o agente alcançasse uma classificação de 1042, o que constitui uma importante melhora em relação aos sistemas desenvolvidos anteriores como, o MORPH IV (LEVINSON; WEBER, 2000), que necessitou de meses de treino para alcançar o mesmo nível. Este trabalho mostra a importância de se desenvolver um bom modelo de representação das relações entre as peças de um jogo de tabuleiro a fim de acelerar a aprendizagem e diminuir a importância da busca em jogos.

3.7 Estratégias de Treinamento

O conhecimento adquirido por um sistema de aprendizagem é originado na etapa de treinamento. Entretanto, a quantidade de informação de treino a ser fornecida para o agente deve ser, por um lado, suficientemente focada para que se garanta a convergência rápida em uma boa função de avaliação, e, por outro lado, deve oferecer uma diversidade suficiente para permitir a aprendizagem geral de todas as situações que possam surgir durante o jogo.

Em Aprendizagem por Reforço, isto implica em se ter um equilíbrio eficaz entre a exploração de novas situações (estados) e o aproveitamento do conhecimento já adquirido. Este problema assume especial relevância no caso do auto-treino (*self-play*), pois é necessário assegurar que a função de avaliação seja obtida a partir de um conjunto diversificado de treinos a fim de prevenir que o agente fique “preso” em um mínimo local.

Em (LYNCH, 1997), Mark Lynch lista uma série de estratégias de treinamento para que um agente possa aprender a jogar. Entre elas, pode-se destacar quatro estratégias consideradas mais relevantes no assunto:

1. **Jogos Diretos:** este método permite que dois jogadores aprendam a jogar a partir de um conjunto de jogos de treino entre eles. Entretanto, este método apresenta o problema do *benchmark*, isto é, não é possível definir um ponto de referência que determina qual dos dois jogadores está tendo um melhor desempenho nos treinamentos. Um jogador poderia, por exemplo, estar aprendendo a jogar a partir de um oponente fraco ou “estacionar” sua aprendizagem depois de um certo tempo de treinamento. Existem diversos trabalhos em que estes jogos diretos são realizados

entre um jogador, em processo de aprendizagem, e um outro jogador já treinado (uma máquina especialista, por exemplo). Em (SCHAEFFER et al., 2001), Schaeffer obteve bons resultados ao treinar seu jogador de Damas por Diferença Temporal a partir de jogos contra *CHINOOK*, uma máquina especialista e atual campeã de Damas;

2. **Jogos de Especialistas:** os jogos de especialistas são lidos de uma base de dados e o agente aprende a jogar a partir deles. O maior problema com esta estratégia de treinamento é que os jogos de especialistas nunca tendem a terminar a partida até o último momento do jogo. Por exemplo, em Damas, a maioria dos jogos sempre terminam com 6 ou mais peças sobre o tabuleiro. Com isso, o treinamento de final de jogo poderia ser prejudicado, atrapalhando, dessa forma, o processo de aprendizagem do agente. Em (LEVINSON; WEBER, 2002), Levinson e Weber obteve bons resultados com seu jogador de Xadrez, treinando-o a partir de várias centenas de jogos de Mestres do Xadrez, disponíveis em bases de dados online;
3. **Jogos contra humanos:** o agente aprende a jogar a partir de jogos contra oponentes humanos. Esta estratégia de treinamento permite ao agente evoluir sua capacidade de bater oponentes humanos a partir de treinamentos contra jogadores com alto padrão de jogo. Em (BAXTER; TRIGDELL; WEAVER, 1998a), Baxter obteve bons resultados com seu jogador de Xadrez, KNIGHTCAP, treinando-o por TD(λ) a partir de jogos contra jogadores humanos em um servidor de Xadrez FICS (*Free Internet Chess Server*). À medida que o programa aprendia e ia ficando mais forte, eram atraídos jogadores humanos cada vez melhores que orientavam o programa para posições variadas numa ordem crescente de dificuldade. Isto foi determinante para uma boa exploração do espaço de estados;
4. **Self-play com clonagem:** durante o auto-jogo (ou *self-play*) o agente é treinado para um determinado número de jogos contra si próprio e quando o mesmo atinge uma determinada pontuação (indicando uma melhora no seu nível de jogo) um processo de clonagem da sua função de avaliação é realizado. A figura 14 mostra um esboço deste processo de treinamento por *self-play* com clonagem extraído de (LYNCH, 1997). Lynch obteve bons resultados com seu jogador de Damas a partir de treinamentos com o método TD(λ) e utilizando esta estratégia de treinamento. Em (EPSTEIN, 2001), Epstein sugere que uma estratégia de treino na qual as fases de treino são intercaladas entre jogadas de peritos e *self-play* pode produzir resultados melhores do que quando treinado somente pelo *self-play*. No caso do *LS-Draughts*,

é precisamente esta abordagem que se pretende adotar, visto que a idéia é a de que um sistema inteligente deve aprender pela sua própria experiência.

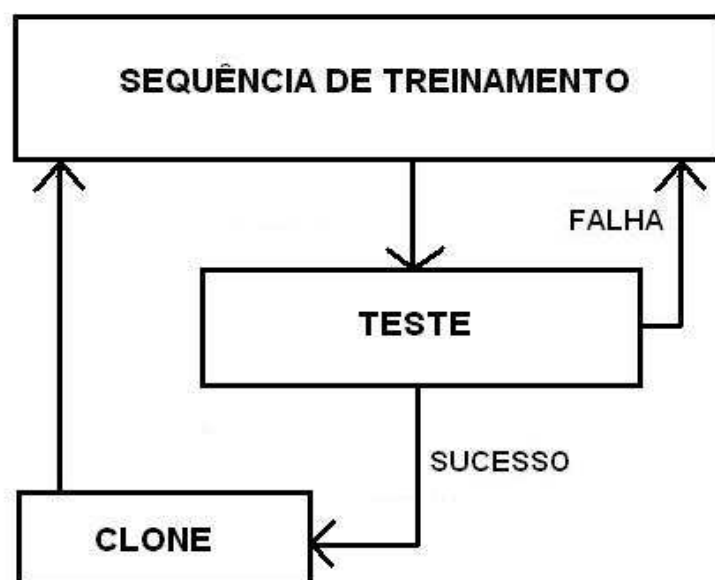


Figura 14: Processo de treinamento por *self-play* com clonagem

3.7.1 Ajuste dos parâmetros de treino

Uma das maiores dificuldades na implementação de sistemas de aprendizagem para jogos consiste no ajustamento favorável dos parâmetros da aprendizagem, o que é crucial na convergência de uma determinada técnica. A maioria dos parâmetros das técnicas utilizadas em sistemas de aprendizagem são ajustados manualmente e definidos após uma série de experimentos. Recentemente, Beal e Smith descreveram um novo sistema que ajusta automaticamente os parâmetros de treino do método TD, em especial a taxa de aprendizagem α e o parâmetro de decaimento de eligibilidades λ (BEAL; SMITH, 1999, 2000). Este sistema não requer conhecimento a priori sobre os valores mais adequados dos parâmetros α e λ para um determinado domínio. O ajustamento é feito de acordo com a própria experiência de aprendizagem, baseando-se no conceito de que a taxa de aprendizagem deve ser mais elevada quando ocorre uma aprendizagem significativa e que, deve ser mais baixa quando as alterações se devem a ruídos nos dados.

O método, designado *Coerência Temporal*, apresenta a propriedade de a taxa de aprendizagem ser reduzida à medida que os valores dos pesos de uma rede neural se aproximam dos valores ótimos. Permite também que a taxa aumente caso os ajustamentos sejam

seguidos de uma tendência ou inclinação consistente. São mantidas taxas de aprendizagem separadas para cada peso, de forma que os pesos que chegaram perto do ótimo não flutuem, desnecessariamente, adicionando com essa flutuação um ruído que afeta as previsões do método TD.

O uso de uma taxa em separado para cada peso permite que diferentes pesos se tornem estáveis em ocasiões diferentes no decorrer do processo de treino. Por exemplo, se um peso a se torna relativamente estável após 100 atualizações, mas um peso b não, então é preferível que a taxa de aprendizagem do peso b seja mais elevada do que a taxa de aprendizagem do peso a .

O algoritmo foi testado em dois domínios complexos: Xadrez e Shogi (Xadrez Chinês). Os resultados demonstraram três aspectos importantes:

1. eliminação da necessidade de especificar parâmetros;
2. uma aprendizagem mais veloz, e;
3. valores finais mais estáveis.

3.7.2 Combinando Diferença Temporal com Busca Minimax

Como visto na seção 2.2, a busca *minimax* tem por objetivo gerar uma árvore do jogo, a partir de um estado atual, a fim de obter maiores informações sobre o mesmo e, assim, poder selecionar a melhor ação para o jogador max. O uso mais tradicional do método TD(λ) com busca *minimax* é dado da seguinte forma: a árvore do jogo avalia todas as jogadas possíveis para o jogador max como nós filhos da raiz (posição corrente do jogo) e todas as jogadas disponíveis para o oponente (min) como filhas destes nós e assim por diante, até o nível que se desejar. Assim, cada ramificação da árvore representa um movimento que o jogador pode fazer em tal momento do jogo. As folhas da árvore são avaliadas pela função de avaliação do jogador max que atribui um valor para cada folha e estes valores são atribuídos de baixo para cima até chegar na raiz da árvore. Como resultado, a busca retorna a melhor ação a ser executada pelo jogador max naquele estado raiz. Assim, o algoritmo TD(λ) utiliza a diferença temporal entre as previsões de dois estados consecutivos do jogo, obtidos pela execução das ações sugeridas pela busca *minimax*, para atualizar a própria função de avaliação do jogador max. Como exemplo de aplicação desta combinação tradicional do método TD(λ) com a busca *minimax*, pode-se citar os trabalhos de Mark Lynch (LYNCH, 1997) e Thrun (THRUN, 1995) que obtiveram bons resultados com seus agentes jogadores. Esta combinação tradicional entre o método

$TD(\lambda)$ e a busca *minimax* é abordada com detalhes e exemplos nas subseções 4.1.3 e 4.1.4.

Uma abordagem interessante foi criada por Baxter e outros pesquisadores a fim de treinar seu jogador de Xadrez KNIGHTCAP. O método de treino utilizado foi o TD-Leaf(λ), uma variante do algoritmo $TD(\lambda)$ que permite que este seja usado conjuntamente com a busca *minimax* para atualizar a função de avaliação do jogador. Este algoritmo simplesmente usa a posição que surge na folha (daí o seu nome) da árvore de busca *minimax* para atribuir a predição do estado raiz (estado atual do jogo) e assim, poder atualizar a função de avaliação do jogador através da diferença temporal entre esse estado do jogo atual e o próximo estado. Com esta técnica e jogando contra humanos e computadores através da internet, KNIGHTCAP subiu sua classificação ELO (sistema de classificação pontual da Federação Internacional de Xadrez) de 1650 para 2100 em apenas 308 jogos, durante 3 dias (BAXTER; TRIGDELL; WEAVER, 1998a). Os ingredientes que contribuíram crucialmente para o sucesso do KNIGHTCAP foram a disponibilidade de parceiros de treino em grande variedade no servidor de Xadrez e a integração correta da aprendizagem por $TD(\lambda)$ nos procedimentos de busca do programa. Em (SCHAEFFER et al., 2001), Schaeffer também utiliza esta combinação do algoritmo $TD(\lambda)$ com a busca *minimax*, proposto por Baxter, para treinar seu agente jogador de Damas.

O objetivo conjunto de técnicas de Aprendizagem por Reforço, especificamente o do método de Diferença Temporal, com métodos de busca, é justamente tentar obter o máximo de conhecimento sem recorrer a buscas extensas ou à base de dados que assumem o papel de “professor”. A intenção é proporcionar, dessa forma, uma nova conceituação e significado para os termos “ensino” e “treino”, deixando-os cada vez mais próximos da aprendizagem humana e animal.

3.8 Diferenças Temporais x Computação Evolutiva

A aplicação da Computação Evolutiva em jogos têm se mostrado bastante eficiente na obtenção de bons agentes jogadores, tornando-se, assim, um paradigma alternativo ao processo de treinamento convencional. A base da Computação Evolutiva é o *Teorema do Esquema* modelado matematicamente por Holland: com o passar das gerações, as soluções “boas” tendem a compartilhar partes comuns em seus cromossomos. Estas partes são chamadas de padrões. Padrões com maior aptidão do que a média da população tendem a crescer exponencialmente nas próximas gerações, enquanto que padrões com aptidões menores do que a média tendem a diminuir, também exponencialmente, isto é, as soluções convergirão para um ponto de maior aptidão (HOLLAND, 1992).

Em (FOGEL et al., 2004), David Fogel utiliza um algoritmo evolutivo para evoluir os pesos de uma rede neural multicamada a fim de aprender a jogar Xadrez. Seu melhor jogador, BLONDIE25, atingiu um alto nível de jogo chegando a uma classificação ELO (sistema de classificação pontual da Federação Internacional de Xadrez) de 2550 pontos (nível de mestre de Xadrez).

Também em (FOGEL; CHELLAPILLA, 2002), David Fogel obteve sucesso ao utilizar um algoritmo evolutivo para evoluir os pesos de uma rede neural multicamada a fim de aprender a jogar Damas. Seu melhor jogador, ANACONDA, chegou a uma classificação de 2045 pontos ao disputar um torneio internacional de Damas em um website para jogadores de Damas (www.zone.com). Esta classificação representa o nível de um jogador especialista em Damas. Fogel também testou seu jogador contra *CHINOOK*, atual campeão mundial de Damas homem-máquina (SCHAEFFER et al., 1996). Em 10 jogos e sem utilizar base final de jogo, ANACONDA venceu 2 jogos, perdeu 4 e empatou outros 4 jogos. Já considerando a mesma base final de jogo de *CHINOOK*, o resultado passou a ser favorável para ANACONDA: 4 vitórias, 3 derrotas e 3 empates.

Por outro lado, Paul Darwen demonstra em (DARWEN, 2001) a vantagem de se utilizar Diferenças Temporais no treinamento de redes neurais multicamadas devido a rapidez com que a rede aprende um comportamento não linear sobre um determinado problema. Darwen demonstra esta questão ao discutir o porquê da co-evolução conseguir bater, para uma arquitetura de rede linear (perceptron simples), a aprendizagem por Diferença Temporal no jogo do Gamão, mas não conseguir o mesmo feito para uma arquitetura de rede não linear (rede neural com camada oculta). O autor mostra que, se são necessários bilhões de jogos para que uma arquitetura não-linear treinada por um método co-evolutivo consiga bater uma outra arquitetura não-linear treinada pelo método TD(λ), a qual, por sua vez, requer apenas cerca de 100.000 jogos para aprender, então muitos dos bilhões de jogos do método co-evolutivo não estarão, de fato, contribuindo para a aprendizagem.

Este fato demonstrado por Paul Darwen parece também ser aplicado ao domínio Damas, quando se pretende treinar uma rede neural multicamada através de um algoritmo co-evolutivo. Por exemplo, o jogador ANACONDA obtido por David Fogel em (FOGEL; CHELLAPILLA, 2002) foi resultado da evolução de 30 redes neurais multicamadas ao longo de 840 gerações, o que levou 6 meses de execução. Cada geração tinha em torno de 150 jogos de treinamento (5 jogos de treino para cada um dos 30 indivíduos da população). Assim, foram necessários 126.000 jogos de treinamento para que Fogel obtivesse o seu melhor jogador, ANACONDA. Já em (LYNCH; GRIFFITH, 1997), Mark Lynch obteve o seu melhor jogador de Damas depois de apenas 2.000 jogos de treinamento com o

método das Diferenças Temporais e utilizando um conjunto de características selecionadas manualmente para representar o mapeamento do tabuleiro de Damas na entrada da rede neural. Seu melhor jogador, disponível em <http://iamlynch.com/nd.html>, obteve bons resultados ao jogar contra Dave Harte, campeão de Damas U18 na Irlanda, e também contra outros competentes jogadores de damas (LYNCH, 1997). Em (SCHAEFFER et al., 2001), Schaeffer também obteve um bom jogador de Damas treinado-o por Diferenças Temporais e utilizando um conjunto de características selecionadas manualmente. Seu jogador chegou ao nível do atual campeão de Damas, *CHINOOK*, depois de aproximadamente 10.000 jogos de treinamento.

3.9 Diferenças Temporais em outros domínios

A aplicabilidade da técnica de aprendizagem TD em outros domínios se deve ao fato de que nestes domínios o agente de aprendizagem pode ser imaginado como um agente que contém um *elemento de desempenho* que decide que ações executar e um *elemento de aprendizagem* que modifica o elemento de desempenho para que este tome decisões melhores. Como exemplo, pode-se citar algumas aplicações:

- **Planejamento de processos do tipo *Job-Shop*:** Zhang e Dietterich em (ZHANG; DIETTERICH, 1996) foram motivados a aplicar o método das Diferenças Temporais para auxiliar no planejamento da produção em empresas cujos processos são do tipo *Job-Shop* (produção sob encomenda). O problema da programação da produção do tipo *Job-Shop* (JSP) é um problema de alocação de um conjunto de *jobs* para as máquinas, de tal forma que os *jobs* sejam executados em um menor intervalo de tempo. Cada *job* pode consistir de diversas tarefas e cada tarefa deve ser processada numa máquina particular, sendo que esta pode processar no máximo uma operação por vez. Além disso, as tarefas em cada *job* estarão sujeitas às restrições de precedência. Zhang e Dietterich modelaram uma rede neural multicamada como uma função de avaliação e treinou-a com o método TD(λ) a fim de auxiliar na determinação de uma lista ordenada de operações para cada máquina, otimizando o tempo total de execução das tarefas (*jobs*) e minimizando o tempo de ociosidade das máquinas;
- **Controle de descarregadores de navios:** Leonardo Scárdua e outros pesquisadores descreve em (SCARDUA; CRUZ; COSTA, 2003) o uso do algoritmo TD(0) para a obtenção de trajetórias ótimas e controle anti-balanço de um descarregador de

navios. Um problema importante nas operações de descarga de navios é a otimização do movimento entre o navio e a moega, respeitando restrições impostas pelos equipamentos e satisfazendo condições de contorno específicas. O descarregador de navios é basicamente um sistema carro-pêndulo em que o comprimento do pêndulo pode ser variado, independentemente do movimento do carro. Uma caçamba localizada na extremidade do cabo é usada para transportar o material a ser descarregado. Dessa forma, o problema da otimização do movimento da caçamba pode ser encarado como um problema de decisão seqüencial em tempo discreto, no qual um controlador deve decidir, em cada época de decisão, qual a melhor ação a executar, considerando seu objetivo de longo prazo. Os autores utilizaram então uma rede neural multicamada treinada pelo método das Diferenças Temporais para tentar aproximar uma função que pudesse descrever este tipo de comportamento;

- **Planejamento de uma rede móvel de celulares:** Singh e Bertsekas em (SINGH; BERTSEKAS, 1996) foram motivados a utilizarem o método das Diferenças Temporais a fim de tentarem resolver o problema da mobilidade no planejamento de uma rede móvel de celular. Um dos grandes desafios para sistemas de comunicações móveis é conseguir uma utilização eficiente do limitado *spectrum* de frequências disponíveis, provendo serviços com qualidade para seus usuários. Com a evolução das comunicações móveis e o surgimento de novas aplicações, cada vez mais exigentes em termos de largura de banda, estudos sobre os parâmetros de Qualidade de Serviço (QoS), a influência da mobilidade dos usuários e eficiência de diferentes técnicas de alocação de canais são necessários para garantir níveis de QoS compatíveis com cada aplicação. Para resolver este problema, os autores utilizam uma rede neural multicamada treinada pelo método das Diferenças Temporais;
- **Estratégia de seleção de diálogos com humanos:** Cada vez mais, a vida moderna demanda agentes que dialogam com humanos (tais como os atendentes eletrônicos em empresas de prestação de serviços). Como exemplo de sistema que ataca esse tipo de problema, pode ser citado o sistema ELVIS (*Elvis Voice Interactive System*) de Walker (WALKER, 2000). Neste trabalho, o autor cria um agente que aprende a escolher uma ótima estratégia de diálogo por meio de suas experiências e interações com os usuários humanos. A técnica utilizada para treinar seu agente foi o algoritmo *Q-learning*, uma variação do método $TD(\lambda)$.

3.10 Considerações Finais

A criação de programas que jogam com alto nível de desempenho tem sido um dos maiores triunfos em IA. A exemplo disso, têm-se os sucessos obtidos com os jogos Gamão, Xadrez, Damas e Othello, que foram citados neste capítulo. Entre os métodos utilizados para obter estes resultados, a busca e a utilização da capacidade de memória dos computadores tem sido uma das técnicas mais bem sucedidas em IA. Os computadores são ótimos para procurar soluções, considerando milhões de possibilidades por segundo. Já os humanos não possuem esta técnica de efetuarem buscas ótimas rápidas e eficientes, mas em compensação são muito bons para descobrir, generalizar e utilizar o conhecimento. Uma prova disso é que após 50 anos de investigação, ninguém ainda conseguiu representar ou manipular eficientemente o conhecimento nos computadores.

É neste sentido que a Aprendizagem por Reforço e a Computação Evolutiva aparecem como opções em IA para a aquisição autônoma do conhecimento em estratégias de aprendizagem em jogos.

4 LS-Draughts – *Um Sistema de Aprendizagem de Damas*

Neste capítulo, será apresentado o *LS-Draughts* (*Learning System Draughts*): um sistema de aprendizagem de jogos de Damas que tem como objetivo principal construir um agente automático capaz de jogar Damas com alto nível de desempenho. Para isso, esse sistema visa estender o jogador de Damas de Mark Lynch (LYNCH, 1997), *Neuro-Draughts*, gerando, automaticamente, por meio da técnica dos Algoritmos Genéticos, as características mínimas necessárias e essenciais de um domínio de um jogo de Damas, de forma a tentar otimizar o processo de aprendizagem do agente jogador de Lynch.

O sistema *NeuroDraughts* de Lynch implementa o agente jogador de Damas como uma rede neural MLP que utiliza a busca *minimax* para a escolha da melhor jogada em função do estado do tabuleiro do jogo. Além disso, ele utiliza o método de aprendizagem por reforço $TD(\lambda)$, aliado à estratégia de treino por *self-play* com clonagem, como ferramentas para atualizar os pesos da rede. Para tanto, o tabuleiro é representado por um conjunto de funções que descrevem as características do próprio jogo de Damas. A utilização de um conjunto de características para representar o mapeamento do tabuleiro de Damas na entrada da rede neural é definida por Lynch como sendo um mapeamento *NET-FEATUREMAP* (este tipo de mapeamento será discutido com mais detalhes na subseção 4.1.1).

A arquitetura geral do *LS-Draughts* é apresentada na figura 15. O sistema é composto por 4 módulos principais:

- Um primeiro que, por meio da técnica dos AGs, gera, automaticamente, um conjunto mínimo e eficaz de características necessárias e essenciais de um domínio de jogo de Damas. No caso, o AG gerará T_p indivíduos que representam sub-conjuntos de todas as características disponíveis no mapeamento *NET-FEATUREMAP*;
- Um segundo módulo que corresponde à busca *minimax* e a seleção da melhor ação a ser executada pelo agente em função do estado do tabuleiro do jogo;

- Um terceiro módulo que corresponde a implementação do próprio agente jogador de Damas, propriamente dito, representado por uma rede neural MLP. Cada um dos T_p indivíduos gerados no 1º módulo estará associado a uma rede neural MLP deste módulo;
- Um quarto módulo que corresponde a unidade de treinamento do agente através do método de aprendizagem por reforço TD(λ) e a estratégia de treino por *self-play* com clonagem. Este processo de aprendizagem da rede neural é o mesmo do agente jogador do sistema *NeuroDraughts* proposto e desenvolvido por Mark Lynch (LYNCH, 1997). Contudo, o processo de treinamento do *LS-Draughts* é direcionado para o treino dos indivíduos do 1º módulo.

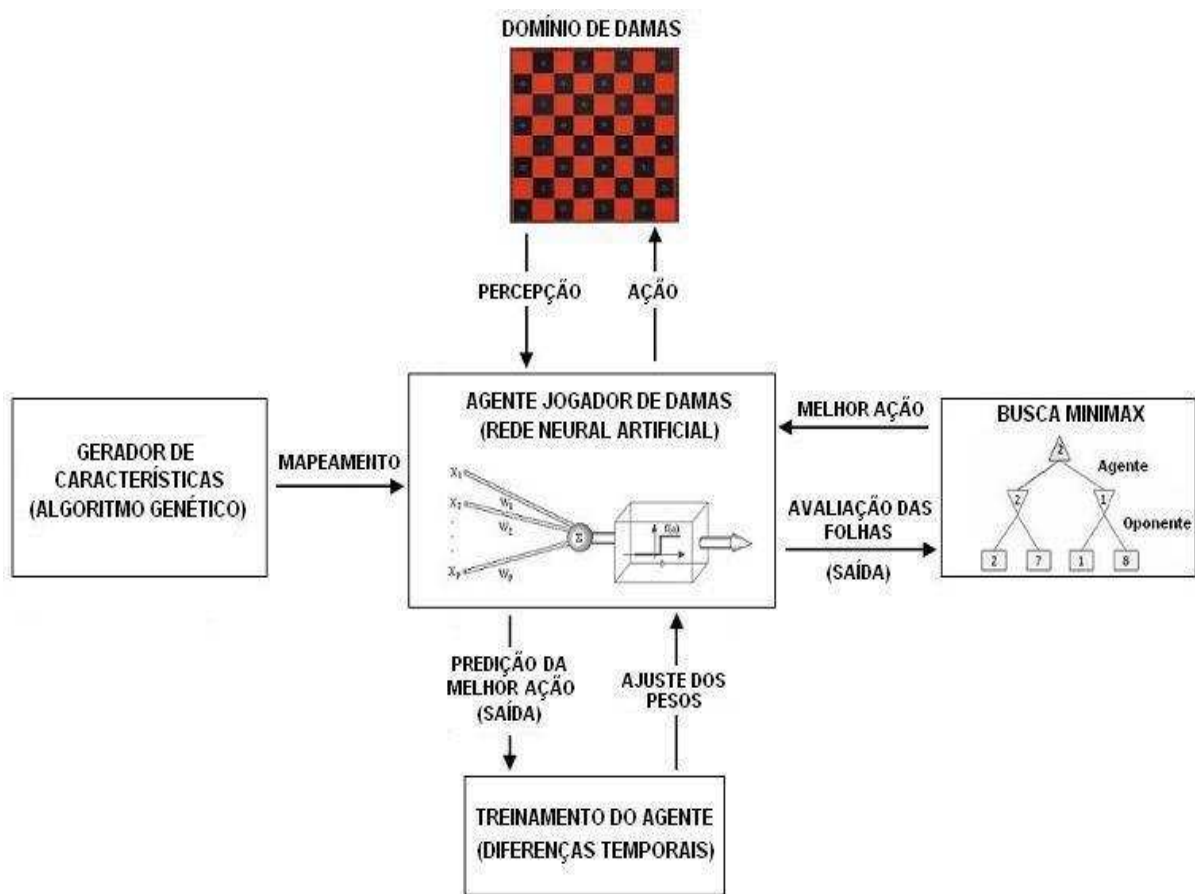


Figura 15: Arquitetura geral do *LS-Draughts*

O 1º módulo do *LS-Draughts* corresponde a uma expansão do *NeuroDraughts*, uma vez que, neste último, o conjunto de características é fixado, meramente, como um dado de entrada, fato que compromete a escolha de um conjunto resumido, otimizado e eficaz

de características. Como será visto neste capítulo, a automatização do processo de escolha de características torna o desempenho do *LS-Draughts* superior ao do *NeuroDraughts*. No *LS-Draughts*, a cada um dos T_p indivíduos gerados no 1º módulo será acoplada uma rede neural que aprende pelo método das Diferenças Temporais. No final, serão obtidas T_p redes treinadas em função de cada indivíduo, ou seja, em função do conjunto de características representativas do tabuleiro do jogo de Damas que caracteriza cada indivíduo. A estrutura dos módulos 2, 3 e 4 do *LS-Draughts* é a mesma utilizada pelo sistema *NeuroDraughts* de Lynch (a interação entre estes três módulos é apresentada com mais detalhes na seção 4.1).

Note que, para o problema de treinar uma rede neural para jogar Damas utilizando o método $TD(\lambda)$ e algum tipo de mapeamento do tabuleiro, a escolha da melhor ação está vinculada a três fatores fundamentais: o primeiro são as percepções que o agente tem disponível sobre cada estado do domínio em que está atuando (no caso do jogador de Lynch, estas percepções sobre os estados do tabuleiro de Damas são fornecidas pelos três tipos de mapeamentos). Assim sendo, a proposta deste trabalho de encontrar um conjunto mínimo de características que melhor represente os estados de tabuleiro em um jogo de damas é de fundamental importância para otimizar o processo de escolha da melhor ação pelo agente jogador, fato que aumentará a eficiência de treinamento por $TD(\lambda)$ da rede neural com arquitetura não-linear. O segundo fator são os pesos da rede treinada, que representam o conhecimento do próprio agente sobre o jogo de Damas (observe que este conhecimento, decorrente do processo de treinamento, também servirá de base para que o agente possa montar sua própria visão *look-ahead* da árvore do jogo e, assim, poder avaliar melhor suas ações futuras). O terceiro fator é a profundidade da busca *minimax*: quanto maior ela for, mais informações sobre as possibilidades de ação do oponente (jogador min) ela trará ao agente (jogador max). Portanto, a função a ser modelada pelo agente para representar a sua política de ações sobre o domínio de Damas deve ser projetada de tal forma a contemplar os três fatores mencionados acima.

Por outro lado, escolher um conjunto de características, referentes ao mapeamento *NET-FEATUREMAP* de Mark Lynch que melhor represente os estados do jogo de Damas (ou configurações de tabuleiro) na entrada da rede neural, envolve um problema de busca e otimização. Dentro da IA, existem diversas áreas que, inspiradas em algumas técnicas da Natureza, buscam desenvolver sistemas inteligentes para tentar resolver este tipo de problema. Os algoritmos genéticos são exemplos destas aplicações. Eles fornecem um mecanismo de busca adaptativa que se baseia no princípio Darwiniano de reprodução e sobrevivência dos indivíduos mais aptos para resolver o problema em questão.

As argumentações apresentadas até aqui motivaram a proposta principal do presente trabalho: aliar os benefícios da utilização da técnica de aprendizagem por reforço $TD(\lambda)$ e dos AGs na construção do sistema *LS-Draughts*, o que representa uma versão expandida do agente jogador de Damas de Mark Lynch. A expansão corresponde à inserção, no sistema *NeuroDraughts* de Lynch, de um módulo que tenta classificar, automaticamente, o conjunto de características que propicie um melhor desempenho do jogador.

Com o AG, o *LS-Draughts* pretende gerar, selecionar e recombinar as melhores características (ou bons “blocos de características”) que se perpetuarão ao longo das gerações a fim de otimizar o treinamento de uma rede neural por Diferenças Temporais. Com o método das Diferenças Temporais, o algoritmo pretende ajustar incrementalmente os pesos sinápticos da rede através de predições sucessivas entre dois estados consecutivos. Estes ajustes são realizados em todos os elos de conexão entre as camadas da rede neural de forma a identificar o quão um determinado conjunto de características é apropriado para representar estados do tabuleiro do jogo, produzindo reforço ou enfraquecimento sináptico a cada vez que o conjunto se mostra mais ou menos apropriado. Ao final do treinamento, as redes neurais que apresentarem baixos pesos sinápticos entre as camadas de entrada e oculta, ou entre esta última e o neurônio de saída, tenderão a obter baixo desempenho na etapa de cálculo do *fitness* do processo evolutivo. Neste caso, os indivíduos correspondentes a tais redes estarão menos propensos a se reproduzirem e se perpetuarem ao longo das gerações, ou seja, cada um desses indivíduos representa um conjunto de características com forte probabilidade de ser inadequado para representar o tabuleiro do jogo de Damas.

As próximas seções abordarão, em detalhe, o desenvolvimento do *LS-Draughts*. Como a compreensão do *NeuroDraughts* é fundamental para o entendimento do presente sistema e para o entendimento da importância de uma boa escolha de características representativas do tabuleiro no desempenho do *LS-Draughts*, inicialmente será apresentado o *NeuroDraughts* de Mark Lynch. Na sequência, serão apresentadas as expansões efetuadas pelo *LS-Draughts* no *NeuroDraughts*, a linguagem de programação utilizada na implementação do sistema e os resultados obtidos com o *LS-Draughts*. Assim, as seções se sucederão de acordo com o disposto a seguir: **i)** jogador de Mark Lynch; **ii)** processo evolutivo do *LS-Draughts*; **iii)** ferramenta utilizada na implementação do *LS-Draughts*; e, **iv)** resultados experimentais.

4.1 Jogador de Mark Lynch e o Processo de Treinamento por TD(λ)

Nesta seção é apresentado o jogador automático de Damas de Mark Lynch, no qual se baseia o *NeuroDraughts*. Conforme já comentado, o jogador de Lynch consiste em uma rede neural MLP treinada pelo método de aprendizagem por reforço TD(λ). Este método utiliza predições (ou estimativas de resultados finais do jogo de Damas) sucessivas para ensinar a rede MLP a jogar Damas. Além disso, o jogador de Lynch também utiliza uma estratégia de treino por *self-play* com clonagem, uma busca minimax com profundidade de 4 níveis e não faz referência a qualquer análise de jogos de especialistas. Com isso, um dos objetivos principais de Lynch foi, justamente, o de tentar obter um bom jogador de Damas que pudesse jogar com um alto nível de desempenho, sem ter que realizar buscas profundas e nem que analisar jogos de especialistas (LYNCH, 1997).

A rede MLP de Lynch utiliza a mesma estrutura dos módulos 2, 3 e 4 da arquitetura do *LS-Draughts* para aprender a jogar Damas e é dividido, basicamente, em quatro etapas fundamentais:

1. **Etapla da escolha da melhor ação:** antes de o agente executar qualquer ação sobre o tabuleiro de Damas, uma busca minimax, com profundidade 4, é realizada com auxílio de uma rede MLP em treinamento (a rede MLP calcula a predição para todos estados folhas da árvore de busca e o minimax propaga estes valores para a raiz da árvore, levando em consideração os níveis de maximização e minimização da mesma). Como resultado, a melhor ação a_{t+1} sugerida pela busca é então retornada para que o agente possa executá-la. Esta etapa utiliza os módulos 2 e 3 da arquitetura do *LS-Draughts*;
2. **Etapla do cálculo da predição do estado resultante da execução da ação a_{t+1} :** após o agente executar a melhor ação a_{t+1} sugerida pela busca minimax, o estado resultante desta ação, S_{t+1} , é apresentado às unidades da camada de entrada da rede MLP e, a partir desta camada, as unidades ocultas mais a unidade de saída calculam a predição P_{t+1} , correspondente a S_{t+1} , que estará disponível na camada de saída. Esta etapa utiliza apenas o módulo 3 da arquitetura do *LS-Draughts*;
3. **Etapla de reajuste dos pesos da rede:** um erro é calculado pela diferença entre a predição P_{t+1} (valor retornado pela rede MLP ao avaliar o estado S_{t+1} resultante da última ação a_{t+1} executada pelo agente) e a predição P_t (valor retornado pela rede MLP ao avaliar o estado S_t resultante da penúltima ação a_t executada pelo

agente). Este erro é então propagado a partir da camada de saída até a camada de entrada, e os pesos das conexões das unidades das camadas internas vão sendo modificadas utilizando o método $TD(\lambda)$. Esta etapa utiliza os módulos 3 e 4 da arquitetura do *LS-Draughts*;

4. Etapa de recálculo da predição do estado resultante da execução da ação

a_{t+1} : após os pesos da rede neural serem ajustados na etapa anterior, o estado S_{t+1} é apresentado novamente às unidades da camada de entrada da rede MLP que então retorna, através da camada de saída, uma nova predição P_{t+1} para aquele estado. A partir daí, o agente assume que $P_t \leftarrow P_{t+1}$, $S_t \leftarrow S_{t+1}$, $a_t \leftarrow a_{t+1}$ e o processo continua na etapa (1) até a finalização de uma partida de treinamento de Damas. Esta etapa utiliza apenas o módulo 3 da arquitetura do *LS-Draughts*. Observe que o treinamento de uma MLP consiste em várias partidas de treinamento de Damas. Como a estratégia de treinamento utilizada é o *self-play* com clonagem, as partidas são disputadas entre uma rede MLP em treinamento (agente) e uma cópia de si própria (oponente). Esta estratégia será abordada com mais detalhe na subseção 4.1.6.

As subseções que se seguem abordam, em detalhe, os principais aspectos dessas quatro etapas de treinamento do jogador de Lynch, apresentando: **i)** mapeamento das características e representação do tabuleiro de Damas na rede MLP, **ii)** cálculo das predições, **iii)** escolha da melhor ação, **iv)** processo de reajuste dos pesos da rede MLP, **v)** cálculo de reajuste dos pesos da rede MLP, e **vi)** estratégia de treino por *self-play* com clonagem.

4.1.1 Mapeamento das características e representação do tabuleiro

A utilização de um conjunto de características para treinar um jogador de Damas foi primeiramente proposta por Samuel (SAMUEL, 1959) com o intuito de prover medidas numéricas para melhor representar as diversas propriedades de posições de peças sobre um tabuleiro. Várias dessas características implementadas por Samuel resultaram de análises feitas sobre o comportamento de especialistas em partidas de Damas. Em termos práticos, estas análises tinham como objetivo tentar descobrir quais características referentes a um estado do tabuleiro, tais como, por exemplo, peças em vantagens, quantidade de rainhas sobre o centro do tabuleiro, quantidade de peças sob ameaça do oponente etc, são freqüentemente analisadas e selecionadas pelos próprios especialistas quando vão escolher seus movimentos de peças (ou ações) durante uma partida de Damas.

Samuel implementou 26 características para treinar seu jogador de Damas, cuja função de avaliação era um polinômio. Os termos deste polinômio representavam subconjuntos das 26 características e, os coeficientes, os pesos (ou a importância) das características para o agente jogador. Para ajustar tais coeficientes, Samuel combinou algumas técnicas heurísticas com aprendizagem de máquina para treinar e melhorar o desempenho do seu jogador (SAMUEL, 1959, 1967). Como resultado, Samuel produziu, não apenas um programa no nível de um mestre de Damas, mas, também, introduziu importantes conceitos na teoria dos jogos e na aprendizagem automática, tornando-se um dos pioneiros das inúmeras técnicas modernas existentes de busca e aprendizagem de máquina. Como exemplo, tem-se o trabalho de Sutton (SUTTON, 1988), que, inspirado nos trabalhos do próprio Samuel, formalizou e provou a convergência do método de aprendizagem por reforço $TD(\lambda)$, ou método das Diferenças Temporais, que, até hoje, continua sendo um dos métodos mais utilizados na área de jogos.

O agente jogador de Mark Lynch é um outro exemplo de aplicação que também utiliza um conjunto de características para tentar ensinar um agente a jogar Damas. Basicamente, o trabalho de Lynch (LYNCH; GRIFFITH, 1997; LYNCH, 1997) consistiu em avaliar o desempenho de um agente ao treiná-lo utilizando três tipos de mapeamento para representar a configuração do tabuleiro de Damas 8x8 na entrada da rede neural MLP (veja figura 16):

1. **NET-BINARYMAP:** cada quadrado do tabuleiro é representado por 3 entradas binárias na rede (000 representa um quadrado vazio, 001 representa um quadrado que possui peça preta, 010 representa um quadrado que possui peça vermelha, 011 representa um quadrado que possui rainha preta e 100 representa um quadrado que possui rainha vermelha). Com essa representação, a entrada na rede neural é formada por 96 neurônios na camada de entrada (32 quadrados do tabuleiro x 3 seqüências binárias);
2. **NET-DIRECTMAP:** cada quadrado do tabuleiro é representado por uma entrada real na rede (0 representa um quadrado vazio, 0.25 representa um quadrado que possui peça preta, 0.5 representa um quadrado que possui peça vermelha, 0.75 representa um quadrado que possui rainha preta e 1 representa um quadrado que possui rainha vermelha). Com essa representação, a entrada na rede neural é formada por 32 neurônios na camada de entrada (32 quadrados do tabuleiro com entrada real);

3. **NET-FEATUREMAP**: o tabuleiro é representado por um determinado número de funções que descrevem as características do próprio jogo de Damas. Lynch implementou 12 características para representar este tipo de mapeamento. Cada característica tem um valor absoluto que é convertido em uma sequência binária. Com essa representação, a entrada na rede neural varia de acordo com o número de características utilizadas e a quantidade de dígitos binários que cada característica utiliza para representar seu valor absoluto. Este tipo de mapeamento será explicado com detalhe mais adiante.

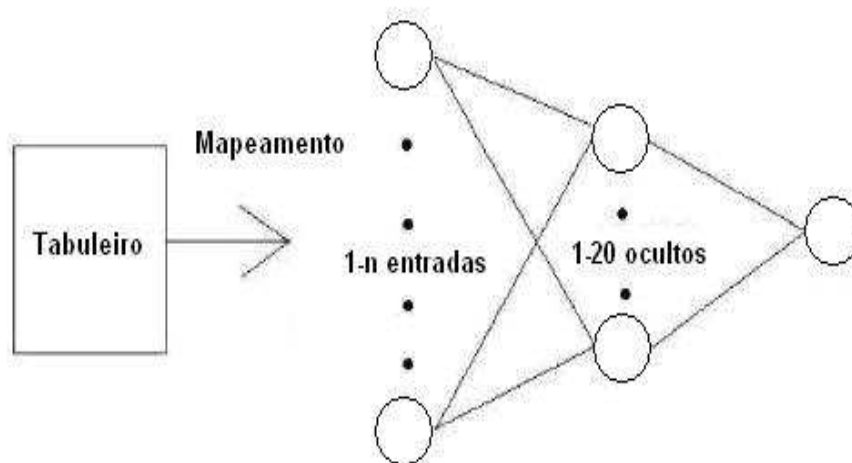


Figura 16: Mapeamento do tabuleiro de Damas utilizado por Mark Lynch em (LYNCH, 1997)

Os melhores resultados obtidos por Lynch foram justamente com os agentes cujas redes MLPs foram treinadas utilizando o mapeamento *NET-FEATUREMAP*. O autor ainda ressalta que este tipo de mapeamento fornece ao agente uma certa especialidade e diferenciação sobre o domínio de Damas. Apesar de Lynch ter obtido um bom jogador de Damas selecionando manualmente o seu conjunto de características, ele ainda coloca como um trabalho a ser explorado a utilização de AGs para tentar otimizar a escolha destas características. Assim, busca-se extrair um mínimo de conhecimento específico sobre o domínio de Damas, de forma a repassá-lo a um agente para que este possa treinar e aprender a jogar Damas com um alto nível de desempenho.

As 12 características implementadas e utilizadas por Lynch no mapeamento *NET-FEATUREMAP* podem ser vistas na tabela 2 abaixo (LYNCH; GRIFFITH, 1997; LYNCH, 1997). Cada característica tem um valor absoluto que representa a sua medida analítica

sobre uma determinada configuração do tabuleiro (ou estado). Este valor é depois convertido em bits binários significativos que, em conjunto com os demais bits das outras características presentes no mapeamento, constituirão a entrada na rede neural. Para exemplificar este valor de medida analítica, suponha que um agente esteja jogando Damas utilizando o mapeamento *NET-FEATUREMAP* e que, para uma determinada configuração de estado do tabuleiro, uma de suas funções do conjunto de características, por exemplo a *PieceThreat*, tem como valor de análise 2. Em termos de conhecimento para a rede neural, este valor quer dizer, que dentre todas as peças do agente que estão no tabuleiro, apenas 2 peças estão sob ameaça do oponente.

CARACTERÍSTICAS	DESCRIÇÃO FUNCIONAL	BITS
<i>PieceAdvantage</i>	Contagem de peças em vantagem para o jogador preto.	4
<i>PieceDisadvantage</i>	Contagem de peças em desvantagem para o jogador preto.	4
<i>PieceThreat</i>	Total de peças pretas que estão sob ameaça.	3
<i>PieceTake</i>	Total de peças vermelhas que estão sob ameaça de peças pretas.	3
<i>Advancement</i>	Total de peças pretas que estão na 5ª e 6ª linha do tabuleiro menos as peças que estão na 3ª e 4ª linha.	3
<i>DoubleDiagonal</i>	Total de peças pretas que estão na diagonal dupla do tabuleiro.	4
<i>Backrowbridge</i>	Se existe peças pretas nos quadrados 1 e 3 e se não existem rainhas vermelhas no tabuleiro.	1
<i>Centrecontrol</i>	Total de peças pretas no centro do tabuleiro.	3
<i>XCentrecontrol</i>	Total de quadrados no centro do tabuleiro onde tem peças vermelhas ou que elas podem mover.	3
<i>TotalMobility</i>	Total de quadrados vazios para onde as peças vermelhas podem mover.	4
<i>Exposure</i>	Total de peças pretas que são rodeadas por quadrados vazios em diagonal.	3
<i>KingCentreControl</i>	Total de rainhas pretas no centro do tabuleiro.	3

Tabela 2: Tabela com as 12 características implementadas por Mark Lynch em (LYNCH, 1997)

A conversão em bits binários do valor de medida analítica retornada por cada característica é dada da seguinte forma:

- Cada característica C_j com valor de análise V_j no mapeamento *NET-FEATUREMAP*

tem uma quantidade pré-fixada de bits significativos que representará o seu valor de análise na entrada da rede neural;

- Cada bit B_i da seqüência que representa V_j é obtido da seguinte forma:

SE $V_i = 1$ **ou** $(V_i > 0 \text{ e } (\text{resto divisão inteira de } V_i \text{ por } 2)=1)$ **ENTÃO**
 RETORNA $B_i = 1$;
SENÃO
 RETORNA $B_i = 0$;

O 1º bit da seqüência binária é “setado” com base na aplicação da condição acima sobre o próprio valor de análise V_j retornado pela função da característica, ou seja, considerando $V_i = V_j$. Cada um dos B_i bits restantes também será “setado” com base na aplicação da condição acima, só que, no caso, V_i assumirá o valor do resultado da divisão inteira do V_i do passo anterior por 2.

Para ilustrar este cálculo de conversão, retornemos ao exemplo acima. Suponha que a função *PieceThreat* tenha 3 bits significativos como quantidade pré-fixada para representar o seu valor de medida analítica na entrada da rede neural MLP (isto é, $V_j = 3$) . Neste caso, a conversão em bits do valor de análise 2 retornado pela função *PieceThreat* seria calculada da seguinte forma:

1. Para $V_i = V_j = 2$ o valor do 1º bit é 0;
2. $V_i = \frac{V_i}{2} = 1$;
3. Para $V_i = 1$ o valor do 2º bit é 1;
4. $V_i = \frac{V_i}{2} = 0$;
5. Para $V_i = 0$ o valor do 3º bit é 0.

Assim, o valor 2 retornado pela função *PieceThreat*, referente a sua análise sobre um determinado estado do tabuleiro, seria representado na entrada da rede neural através da seqüência binária 0 1 0. O mesmo cálculo vale para as demais características que estão representando o mapeamento *NET-FEATUREMAP* de uma determinada rede neural MLP. Se, no exemplo acima, a rede neural tivesse um conjunto de 8 características para representar o seu mapeamento, sendo que cada característica utilizasse 3 bits para representar o seu valor de análise, então haveria 24 neurônios na camada de entrada desta rede.

4.1.2 Cálculo das predições

Calcular a predição P_t para um estado S_t do jogo de Damas, onde S_t é referente a uma configuração do tabuleiro em um instante temporal t , implica em apresentar esta configuração mapeada (utilizando um dos três tipos de mapeamentos citados na subseção anterior) às unidades da camada de entrada da rede neural MLP e, a partir desta camada as unidades calcularem uma resposta, a ser produzida na camada de saída, correspondente à predição P_t .

A figura 17 mostra a rede neural MLP utilizada por Mark Lynch para treinar seu agente jogador de Damas e também calcular a predição dos estados do jogo de Damas. O número de neurônios na camada de entrada varia de acordo com o mapeamento utilizado para representar o tabuleiro de Damas nesta camada. A camada oculta é formada por 20 neurônios e a camada de saída é formada por um único neurônio. Lynch também faz uso de um link de conexão direta entre a camada de entrada e a camada de saída (esse tipo de arquitetura permite aproximar elementos do domínio de Damas que sejam lineares e não-lineares). Além disso, Lynch também utiliza o termo *bias* aplicado a todos os neurônios da camada oculta, representado por $a_0^{(0)} = 1$, e um outro *bias*, aplicado ao neurônio da camada de saída, representado por $a_0^{(1)} = 1$. O efeito do termo *bias* em relação a um neurônio j que está na camada l , para $1 \leq l \leq 2$, é representado por uma sinapse de peso $w_{0j}^{(l-1)}$ conectada a esta entrada fixa $a_0^{(l-1)} = 1$.

Formalmente, o processo de cálculo da predição P_t referente a uma configuração do tabuleiro do jogo de Damas em um instante temporal t , isto é, S_t , pode ser descrito como se segue. Suponha que um vetor de entrada $X(t)$, referente ao mapeamento do estado do tabuleiro S_t , é apresentado à camada de entrada de neurônios sensoriais. Calculam-se os campos locais induzidos e os sinais funcionais da rede prosseguindo para frente através da rede, camada por camada. O campo local induzido $in_j^{(l)}$ para o neurônio j na camada l , para $1 \leq l \leq 2$, é definido por:

$$in_j^{(l)} = \begin{cases} \sum_{i=0}^{m_{(l-1)}} w_{ij}^{(l-1)} \cdot a_i^{(l-1)}, & \text{para neurônio } j \text{ na camada } l=1 \\ \sum_{i=0}^{m_{(l-1)}} w_{ij}^{(l-1)} \cdot a_i^{(l-1)} + \sum_{i=0}^{m_{(l-2)}} w_{ij}^{(l-2)} \cdot a_i^{(l-2)}, & \text{para neurônio } j \text{ na camada } l=2 \end{cases}$$

onde m_l representa o número de neurônios na camada l ; a_i^l é o sinal de saída do neurônio i na camada l ; e w_{ij}^l é o peso sináptico da conexão de um neurônio i da camada l com o neurônio j das camadas posteriores à camada l . Para as camadas ocultas ($l = 1$) e de

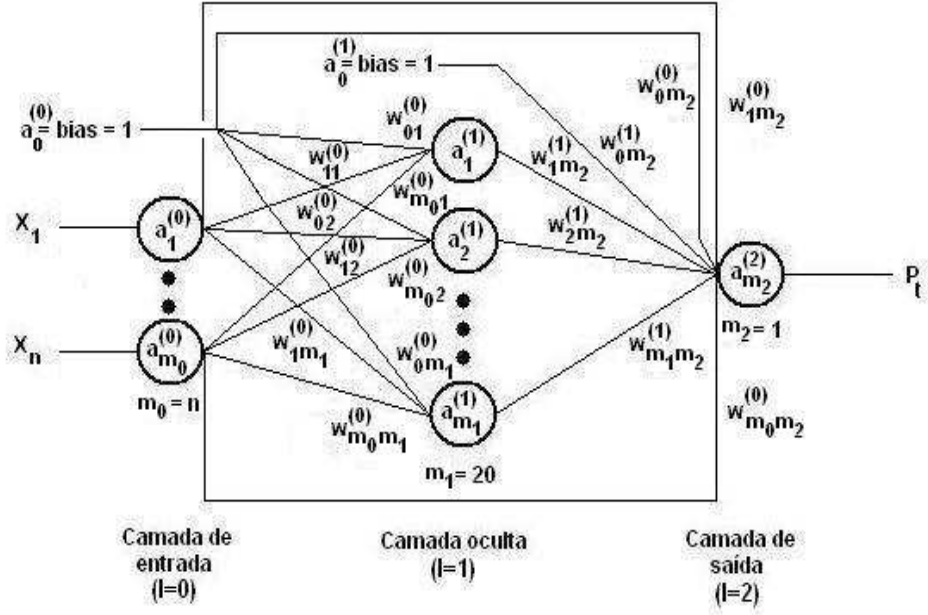


Figura 17: Rede neural MLP utilizada por Mark Lynch em (LYNCH, 1997)

saída ($l = 2$) sendo $i = 0$, tem-se que $a_0^{(l-1)} = +1$ e $w_{0j}^{(l-1)}$ é o peso do bias aplicado ao neurônio j na camada l . Obtido o campo local induzido, o sinal de saída do neurônio j na camada l , para $1 \leq l \leq 2$, é dado por:

$$a_j^{(l)} = g_j(in_j^{(l)}),$$

onde $g_j(x)$ é a função de ativação que descreve a relação funcional de entrada-saída da não-linearidade associada ao neurônio j .

Para o neurônio j que está na camada de entrada, isto é, $l = 0$, faça:

$$a_j^{(0)} = x_j(t)$$

onde $x_j(t)$ é o j -ésimo elemento do vetor de entrada $X(t)$.

Para o neurônio j que está na camada de saída, isto é, $l = 2$, considere que:

$$a_j^{(2)} = a_{m_2}^{(2)} = P_t, \quad (4.1)$$

onde P_t é a predição calculada pela rede MLP para o estado do tabuleiro S_t mapeado na rede através de $X(t)$.

Observe que P_t é uma função dependente do vetor de entrada $\overrightarrow{X(t)}$ e do vetor de pesos $\overrightarrow{W(t)}$ da rede neural no instante temporal t , isto é, $P_t(\overrightarrow{X(t)}, \overrightarrow{W(t)})$. Além disso, o valor

da predição P_t também corresponde a imagem da política π do agente vista na subseção 2.4.2.

Como a função de ativação utilizada por Lynch é a tangente hiperbólica, então a função $g(x)$ é definida por:

$$g(x) = \frac{2}{(1 + e^{(-2x)})} - 1.$$

A função tangente hiperbólica é uma função do tipo $g : R \Rightarrow (-1, +1)$. Esta função tem intervalo de existência entre -1 e +1, sendo assintótica nos dois ramos, isto é, embora ela tenda a -1 e a +1, tais valores não são alcançados nunca. Dessa forma, o objetivo da função de ativação é simplesmente converter uma entrada qualquer em uma saída (bem comportada) entre -1 e +1 (veja figura 18).

Funcionalmente, as predições P_t 's calculadas pela rede neural MLP de Lynch podem ser vistas como uma estimativa do quão o estado S_t se aproxima de uma vitória (representada pelo retorno do valor +1 pelo ambiente), derrota (representada pelo retorno do valor -1 pelo ambiente) ou empate (representado pelo retorno do valor 0, ou próximo de 0, pelo ambiente). Assim, configurações de tabuleiros (ou estados do jogo) que receberem predições próximas de +1 tenderão a ser consideradas como bons estados de tabuleiro, resultantes de boas ações, que poderão convergir para vitória (+1). Da mesma forma, tabuleiros cujas predições estão próximas de -1 tenderão a ser considerados péssimos estados de tabuleiro, resultantes de ações ruins, que poderão convergir para derrota (-1). O mesmo vale para configurações de tabuleiros próximos de 0, que poderão convergir para empate (0 ou valor próximo deste). O valor de retorno do ambiente referente ao resultado de empate de uma partida de Damas é definido em um arquivo de tabuleiro que será discutido, com mais detalhes, na subseção 4.2.3.

No caso do problema de Damas, o ambiente tem a forma de um *Processo de Decisão de Markov* e satisfaz a *Propriedade de Markov* visto na subseção 2.4.3.2:

- a variável de estado representa o tabuleiro de Damas, cujos valores são todas as configurações possíveis do tabuleiro (ou estado). No caso de Damas, o número total de estados possíveis é em torno de 10^{17} (veja figura 13 da seção 3.5);
- os movimentos das peças indicam as transições que alterarão o valor da variável de estado;
- a probabilidade de transição de um estado S_t para um estado S_{t+1} depende apenas do estado S_t e da ação a adotada em S_t . Assim, o estado corrente S_t fornece

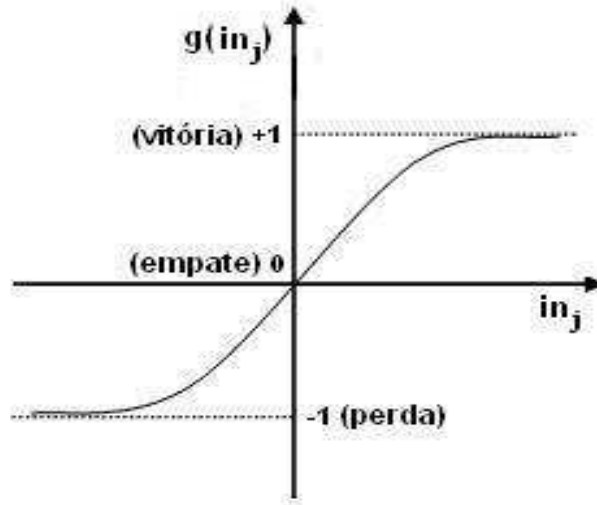


Figura 18: Função tangente hiperbólica e sua funcionalidade no domínio de Damas

informação suficiente para que o agente possa decidir qual ação a ser tomada em S_t (na próxima subseção será visto o processo de escolha da melhor ação a ser tomada em S_t);

- quanto maior for P_t em relação ao estado S_t , mais próximo o agente estará da vitória e maior será a probabilidade de se chegar ao estado de vitória (retorno +1) a partir de S_t ;
- o ambiente evolui probabilisticamente baseado em um conjunto finito e discreto de estados e para cada estado do ambiente existe um conjunto finito de ações possíveis.

4.1.3 Escolha da melhor ação

O problema de treinar uma rede neural para jogar Damas pode ser visto como uma função avaliadora de movimentos ou ações: a partir de posições de tabuleiro corrente, todos os movimentos possíveis legais são avaliados pela função e o movimento com maior valor (ou predição) é, então, selecionado e executado. No caso do jogador de Lynch, é a busca *minimax* que avalia, em conjunto com a rede neural (ou função avaliadora) associada ao jogador max, todos os possíveis movimentos legais detectados para uma determinada posição do jogo corrente. Após esta avaliação, o algoritmo seleciona a ação ou movimento que provê maior predição de vitória para o jogador max.

A busca *minimax* é um método de seleção da melhor ação a ser feita em um jogo, onde dois jogadores se empenham em alcançar objetivos mutuamente exclusivos. Ele se aplica

especialmente na busca em árvores de jogo para determinar qual a melhor jogada para o jogador atual. O algoritmo se baseia no princípio de que, em cada jogada, este jogador irá escolher o melhor movimento possível. No caso do jogador de Lynch, uma árvore de jogo é montada a cada vez que o agente jogador (max) deve escolher uma jogada.

O nó raiz da árvore de jogo representa o estado corrente do tabuleiro. Os nós filhos da raiz correspondem aos estados do tabuleiro que podem ser originados a partir de cada movimento de peça possível para o agente jogador (max) a partir do estado descrito na raiz. Os nós do nível seguinte correspondem a todos os estados do tabuleiro que podem ser originados a partir de cada movimento de peça possível para o jogador oponente (min) a partir dos estados do nível anterior (estados provocados pelas jogadas de max). A mesma estratégia segue em curso até o nível de profundidade que se desejar. Cada ramificação da árvore representa um movimento que o jogador pode fazer em tal momento do jogo. Uma busca mais profunda na árvore fornece mais informações sobre as possíveis vantagens ou armadilhas e, portanto, resulta em uma jogada melhor.

As folhas da árvore de busca são avaliadas pela ótica do jogador max. Logo, maiores valores de predição nas folhas indicam estados mais favoráveis ao agente jogador (max). Os valores dos nós internos são atribuídos de baixo para cima até chegar na raiz da árvore. Os nós do nível minimizar são preenchidos com o menor valor de todos os seus nós filhos, uma vez que o adversário tende a escolher o movimento que levará ao estado menos favorável ao agente. Os nós do nível maximizar são preenchidos com o maior valor de todos os seus nós filhos, uma vez que o agente tende a escolher o movimento que levará ao estado mais favorável a si mesmo. No caso do jogador de Lynch, avaliar as folhas de uma árvore do jogo significa mapear, na entrada da rede neural do jogador max, a configuração do tabuleiro correspondente a cada folha e extrair o valor de predição de vitória retornado pela avaliação da rede neural MLP (ou função avaliadora de movimentos). Note que os estados folha da árvore de busca de Lynch são sempre estados resultantes de ações executadas pelo jogador min (oponente).

A figura 18 mostra uma árvore que simula a progressão de um jogo considerando quatro jogadas a partir do estado corrente (profundidade 4), sendo o jogador preto (max) o jogador a executar o próximo movimento a partir do estado de nível S_0 (raiz) (estado corrente do jogo). Os número reais que ocupam os nós terminais da árvore correspondem às predições de vitória retornadas pela rede. Conforme a figura, a melhor jogada avaliada pela busca *minimax* que maximizará a ação do jogador preto em relação ao estado S_0 do tabuleiro é, portanto, executar a jogada que levará o tabuleiro para o estado S_2 da árvore. Após este movimento, a melhor ação esperada, a ser executada pelo jogador vermelho

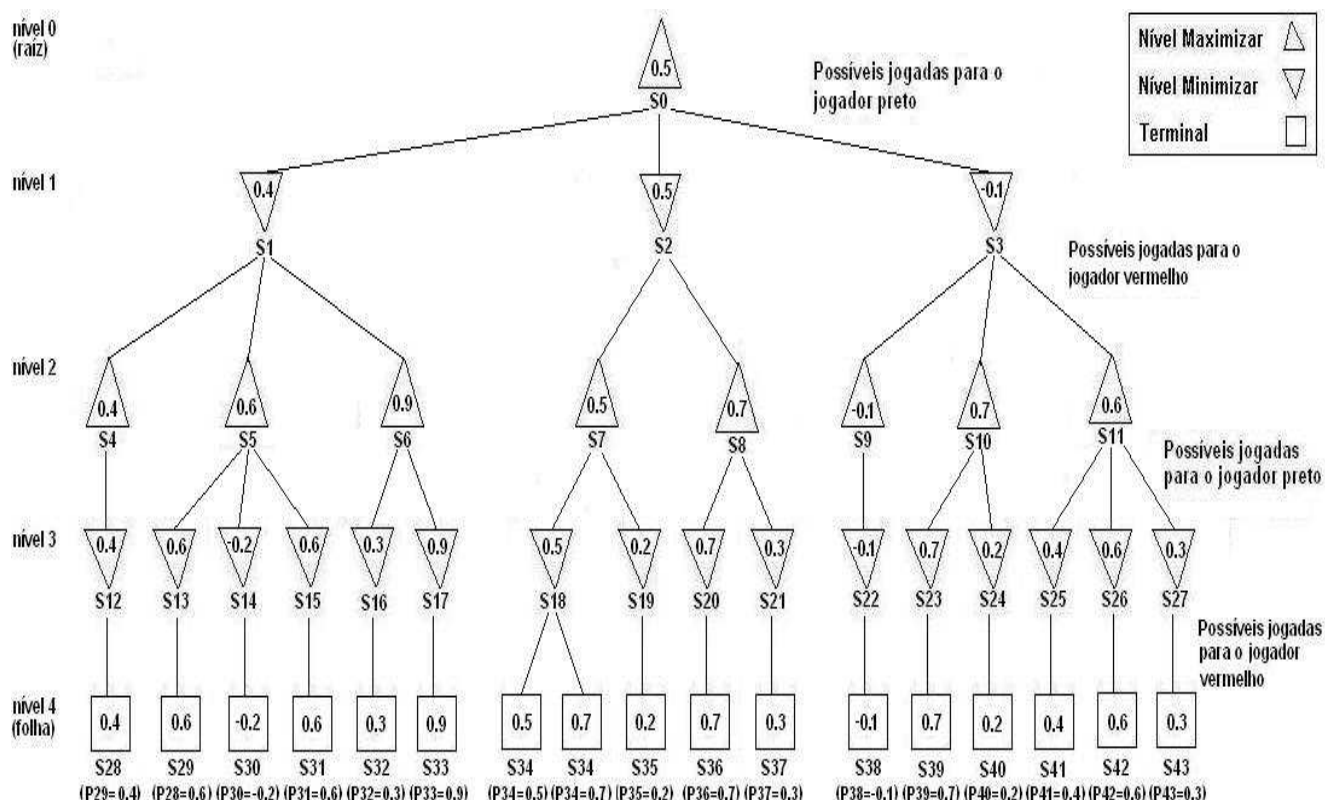


Figura 19: Árvore de busca minimax para o estado raiz S_0 com profundidade 4

(oponente) em resposta a ação executada pelo jogador preto, é a jogada que levará o tabuleiro para o estado S_7 , uma vez que é este o estado que deixa as piores opções para a jogada seguinte do jogador preto (estados S_{18} e S_{19}). Entretanto, se o oponente resolver executar, por exemplo, a jogada para o estado S_8 da árvore, então, de acordo com o princípio do próprio método *minimax*, o jogador vermelho não estará executando o seu melhor movimento e, portanto, o jogador preto passará a ter uma pequena vantagem sobre esta ação executada pelo oponente. Note que este tipo de análise é um exemplo da visão *look-ahead* que o jogador preto, em questão, pode ter a respeito do jogo.

De uma forma geral, a busca minimax é um algoritmo recursivo que recebe como parâmetro o estado atual do tabuleiro (raiz da árvore), o jogador que fará o próximo movimento (jogador max), a função de avaliação de movimentos vinculada ao jogador max e a profundidade máxima da busca. Como resultado, a busca retorna para o jogador max qual a melhor ação a ser executada por este. Uma das vantagens da busca *minimax*

é auxiliar o agente na tarefa da investigação de estados já visitados e exploração de novos estados que ainda não foram visitados (como foi visto na subseção 2.4.1).

4.1.4 Processo de reajuste dos pesos da rede MLP

Nesta subseção será abordado o processo de reajuste dos pesos da rede MLP que o agente jogador de Lynch utilizou para aprender a jogar Damas. O processo de reajuste dos pesos da rede é *on-line*, isto é, o agente vai jogando contra o seu oponente e os pesos da rede vão sendo ajustados pelo método TD(λ) de acordo com a escolha das melhores ações e os estados resultantes destas ações. Após o fim de cada partida de treino, um reforço final é fornecido pelo ambiente informando o resultado obtido pelo agente jogador em função da seqüência de ações que executou (+1 ou -1, caso o resultado tenha sido vitória ou derrota, respectivamente). Caso tenha ocorrido empate, o resultado será zero ou valor próximo a zero.

Para ilustrar o processo de reajuste dos pesos, considere o seguinte:

- o estado S_0 da árvore de busca da figura 18 é a configuração do tabuleiro do jogo inicial de uma partida de Damas de treino G_1 a ser disputada entre o agente e o seu oponente;
- a primeira ação a ser executada sobre o tabuleiro inicial da partida G_1 será realizada pelo agente. Em seguida, o oponente faz sua jogada e, assim, as jogadas sucessivas vão se alternando até o fim de G_1 ;
- todos os estados resultantes de ações efetivas executadas pelo agente até o fim de G_1 serão representados pela seqüência $S^* = \{S_1^*, S_2^*, S_3^*, \dots, S_m^*\}$, onde S_m^* é o último estado resultante da última ação executada pelo agente na partida G_1 ;
- todas as predições calculadas pela rede para a seqüência de estados S^* serão representadas pela seqüência $P^* = \{P_1^*, P_2^*, P_3^*, \dots, P_m^*, R\}$, onde P_m^* é a predição calculada pela rede para o último estado S_m^* da seqüência S^* e R é o reforço final (ou retorno) fornecido pelo ambiente com relação ao resultado final da partida G_1 ;
- a atualização dos pesos da rede MLP na camada l , para $0 \leq l \leq 1$, é dada pela equação do método TD(λ) de Sutton (SUTTON, 1988):

$$\begin{aligned} w_{ij}^{(l)}(t) &= w_{ij}^{(l)}(t-1) + \Delta w_{ij}^{(l)}(t) \\ &= w_{ij}^{(l)}(t-1) + \alpha^{(l)} \cdot (P_{t+1} - P_t) \cdot \sum_{k=1}^t \lambda^{t-k} \nabla_w P_k \end{aligned} \quad (4.2)$$

$$= w_{ij}^{(l)}(t-1) + \alpha^{(l)} \cdot (P_{t+1} - P_t) \cdot \text{elig}_{ij}^{(l)}(t),$$

onde $\alpha^{(l)}$ é o parâmetro da taxa de aprendizagem na camada l (Lynch utilizou uma mesma taxa de aprendizagem para todas as conexões sinápticas de uma mesma camada l); $w_{ij}^{(l)}(t)$ representa o peso sináptico da conexão entre a saída do neurônio i da camada l e a entrada do neurônio j da camada $(l+1)$, no instante temporal t . A correção aplicada a este peso no instante temporal t é representada por $\Delta w_{ij}^{(l)}(t)$. O termo $\text{elig}_{ij}^{(l)}(t)$ é único para cada peso sináptico $w_{ij}^{(l)}(t)$ da rede neural e ele representa o traço de elegibilidade das predições calculadas pela rede neural para os estados resultantes de ações executadas pelo agente desde o instante temporal 1 do jogo até o instante temporal t . Como cada predição P_k é uma função dependente do vetor de entrada $\overrightarrow{X(k)}$ e do vetor de pesos da rede neural $\overrightarrow{W(k)}$ no instante temporal k , isto é, $P_k(\overrightarrow{X(k)}, \overrightarrow{W(k)})$, então $\nabla_w P_k$ representa a derivada parcial de P_k em relação aos pesos da rede MLP no instante k (a entrada da rede é considerada uma constante na derivada parcial $\nabla_w P_k$). Visto isso, o termo λ^{t-k} , para $0 \leq \lambda \leq 1$, tem o papel de dar uma “pesagem exponencial” para a taxa de variação das predições calculadas a k passos anteriores de t . Isto implica que, quando maior for λ , mais os reajustes dos pesos da rede realizado em instantes temporais anteriores a t terão maior impacto sobre a atualização dos pesos $w_{ij}^{(l)}(t)$ da equação (4.2).

Na subseção 4.1.5, a equação (4.2) será expandida e melhor detalhada a fim de explicar ao leitor a utilização de cada termo da equação TD(λ) para o treinamento do jogador do sistema *NeuroDraughts* de Mark Lynch;

- o vetor de peso $\overrightarrow{W(0)}$ inicial da rede neural é gerado aleatoriamente;
- as elegibilidades associadas aos pesos sinápticos da rede são todas inicialmente nulas.

Antes de o agente executar qualquer movimento sobre o tabuleiro inicial da partida de Damas G_1 , uma árvore de busca é montada com raiz em S_0 a fim de poder obter, pelo método minimax, qual a melhor ação que o agente deve executar em S_0 (veja figura 18). As predições das folhas $P_{28}, P_{29}, P_{30}, P_{31}, \dots, P_{43}$ são calculadas através da equação (4.1) considerando-se o mesmo vetor de pesos iniciais $\overrightarrow{W(0)}$ e a entrada $\overrightarrow{X(i)}$, para $28 \leq i \leq 43$, variando de modo a representar cada um dos estados $S_{28}, S_{29}, S_{30}, S_{31}, \dots, S_{43}$. Suponha que a melhor ação sugerida pela árvore de busca minimax é a jogada que leva ao estado S_2 . O agente, então, executa seu 1º movimento em G_1 , chegando ao estado S_2 da árvore ou estado S_1^* referente ao instante temporal 1 da sequência S^* . S_1^* é, então, submetido à rede neural, considerando o vetor de pesos iniciais $\overrightarrow{W(0)}$ da rede e a entrada $\overrightarrow{X(1)}$ adequada

a S_1^* . Como resultado, a predição P_1^* para o estado S_1^* (ou S_2) é calculada. Em seguida, calcula-se a elegibilidade $elig_{ij}^{(l)}(1)$ a partir da predição P_1^* . Observe que P_1^* também será utilizada para reajustar os pesos de $\overrightarrow{W(0)}$, obtendo-se um novo vetor de pesos $\overrightarrow{W(1)}$, conforme a equação (4.2). Entretanto, como o agente só executou um movimento efetivo no jogo e a equação (4.2) necessita de dois movimentos efetivos consecutivos do agente, para reajustar os pesos da rede, então será necessário aguardar o próximo movimento do agente para que o reajuste dos pesos ocorra. Observe que somente as predições calculadas a partir de estados resultantes de ações efetivamente executadas pelo agente é que são utilizados na equação (4.2).

Suponha que o oponente escolheu, entre S_7 e S_8 , a ação que leva ao estado S_7 . Uma nova árvore de busca minimax com raiz em S_7 será montada a fim de poder obter, pelo método minimax, qual a melhor ação que o agente deve executar em S_7 (veja figura 19). As predições das folhas P_{48} , P_{49} , P_{50} e P_{51} são calculadas através da equação (4.1) considerando-se o mesmo vetor de pesos iniciais $\overrightarrow{W(0)}$ e a entrada $\overrightarrow{X(i)}$, para $48 \leq i \leq 51$, variando de modo a representar cada um dos estados S_{48} , S_{49} , S_{50} e S_{51} . Suponha que a melhor ação sugerida pela árvore de busca minimax é a jogada que leva ao estado S_{18} . O agente, então, executa seu 2º movimento em G_1 , chegando ao estado S_{18} da árvore ou estado S_2^* da sequência S^* . S_2^* é então submetido à rede neural, considerando o vetor de pesos iniciais $\overrightarrow{W(0)}$ e a entrada $\overrightarrow{X(2)}$ adequada a S_2^* . Como resultado, a predição P_2^* para o estado S_2^* (ou S_{18}) é calculada. Neste caso, o P_{t+1} da equação (4.2) é o P_2^* (predição calculada após a execução da 2ª ação efetiva do agente no jogo) e P_t é o P_1^* (predição calculada após a execução da 1ª ação efetiva do agente no jogo). Neste momento, usando P_1^* , P_2^* e $elig_{ij}^{(l)}(1)$, calcula-se $\Delta w_{ij}^{(l)}(1)$ e em seguida reajustam-se os pesos da rede $w_{ij}^{(l)}(1)$ conforme a equação (4.2), obtendo-se um novo vetor de pesos $\overrightarrow{W(1)}$ (observe que $w_{ij}^{(l)}(0)$ na equação representa o peso inicial da rede gerado aleatoriamente antes do treinamento). Em seguida, S_2^* novamente é submetida à rede neural, só que considerando o novo vetor de pesos ajustados $\overrightarrow{W(1)}$ e a mesma entrada $\overrightarrow{X(2)}$ adequada a S_2^* . Como resultado, uma nova predição P_2^* para o S_2^* (ou S_{18}) é então calculada (esta nova predição sobrepõe a predição anterior). Por fim, a elegibilidade $elig_{ij}^{(l)}(2)$ é então calculada utilizando a predição final P_2^* e parte da elegibilidade de $elig_{ij}^{(l)}(1)$ através do parâmetro λ da equação (4.2).

A partir daí, o oponente executa a única ação que leva ao estado S_{34} . Uma nova árvore de busca minimax com raiz em S_{34} será montada a fim de poder obter, pelo método minimax, qual a melhor ação que o agente deve executar em S_{34} (veja figura 20). As predições das folhas P_{56} , P_{57} , P_{58} e P_{59} são calculadas através da equação (4.1) considerando-se o vetor de pesos ajustados $\overrightarrow{W(1)}$ e a entrada $\overrightarrow{X(i)}$, para $56 \leq i \leq 59$,

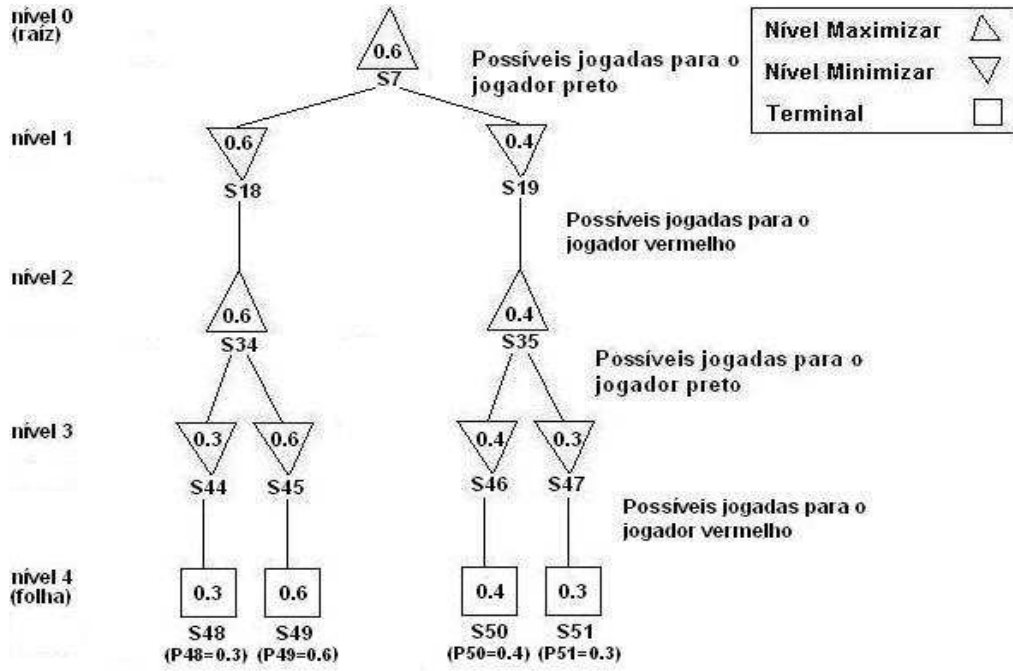


Figura 20: Árvore de busca minimax para o estado raiz S_7 com profundidade 4

variando de modo a representar cada um dos estados S_{56} , S_{57} , S_{58} e S_{59} . Suponha que a melhor ação sugerida pela árvore de busca minimax é a jogada que leva ao estado S_{45} . O agente, então, executa seu 3º movimento em G_1 , chegando ao estado S_{45} da árvore ou estado S_3^* da sequência S^* . Em seguida, S_3^* é submetido à rede neural, considerando o vetor de pesos ajustados $\overline{W}(1)$ e a entrada $\overline{X}(3)$ adequada a S_3^* (ou S_{45}). Como resultado, a predição P_3^* para o estado S_3^* (ou S_{45}) é calculada. Neste caso, o P_{t+1} da equação (4.2) é o P_3^* (predição calculada após a execução da 3ª ação efetiva do agente no jogo) e P_t é o P_2^* (predição final calculada após a execução da 2ª ação efetiva do agente no jogo). Neste momento, usando P_2^* , P_3^* e $elig_{ij}^{(l)}(2)$, calcula-se $\Delta w_{ij}^{(l)}(2)$ e em seguida reajustam-se os pesos da rede $w_{ij}^{(l)}(2)$ conforme a equação (4.2), obtendo-se um novo vetor de pesos $\overline{W}(2)$. Em seguida, S_3^* novamente é submetida à rede neural, só que considerando o novo vetor de pesos ajustados $\overline{W}(2)$ e a mesma entrada $\overline{X}(3)$ adequada a S_3^* . Como resultado, uma nova predição P_3^* para o S_3^* (ou S_{45}) é calculada (esta nova predição sobrepõe a predição anterior). Por fim, a elegibilidade $elig_{ij}^{(l)}(3)$ é então calculada utilizando a predição final P_3^* e parte das elegibilidades de $elig_{ij}^{(l)}(1)$ e $elig_{ij}^{(l)}(2)$ através do parâmetro λ da equação (4.2).

Este processo de reajuste dos pesos da rede MLP prossegue até o final da partida de treino G_1 . Suponha que o agente consiga a vitória após 26 ações efetivas executadas

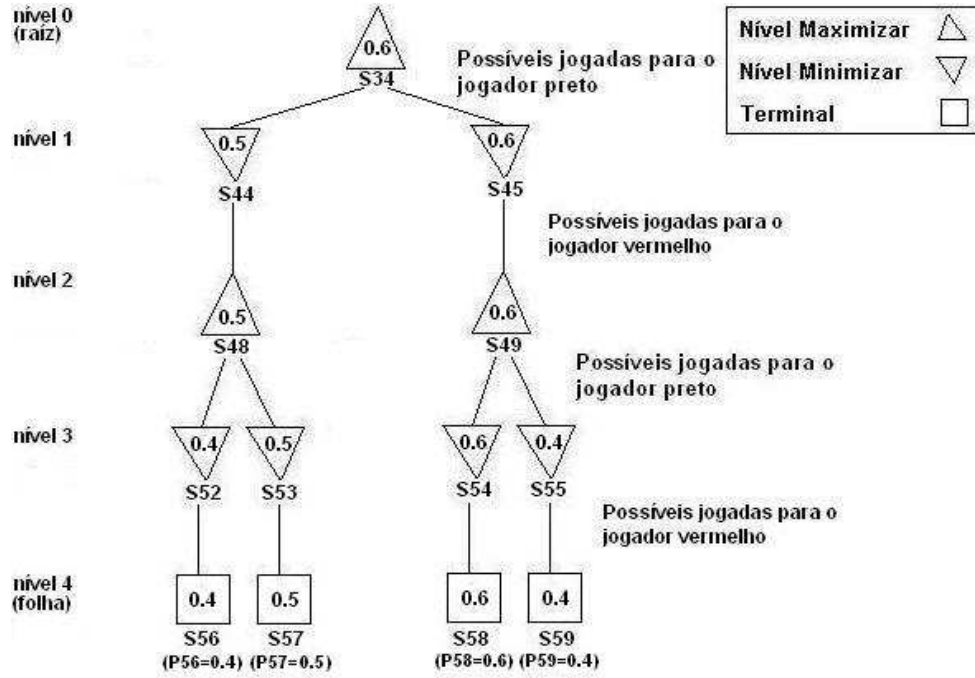


Figura 21: Árvore de busca minimax para o estado raiz S_{34} com profundidade 4

em G_1 , isto é, a sequência de estados $\{S_1^*, S_2^*, S_3^*, \dots, S_{26}^*\}$ foi responsável por levar o agente à vitória. Neste caso, o retorno (ou reforço final) R informado pelo ambiente tem valor 1, isto é, $R = 1$. A partir daí, o P_{t+1} da equação (4.2) é o retorno R (reforço retornado pelo ambiente em virtude do resultado da partida) e P_t é o P_{26}^* (predição final calculada após a execução da 26ª ação efetiva do agente no jogo). Neste momento, usando P_{26}^* , R e $elig_{ij}^{(l)}(26)$, calcula-se $\Delta w_{ij}^{(l)}(26)$ e em seguida reajustam-se os pesos da rede $w_{ij}^{(l)}(26)$ conforme a equação (4.2), obtendo-se um novo vetor de pesos $\overrightarrow{W}(26)$. Observe que este vetor final de pesos ajustados $\overrightarrow{W}(26)$ foi obtido a partir de um vetor de pesos iniciais $\overrightarrow{W}(0)$, gerado aleatoriamente antes do treinamento, que foi sendo ajustado pelas diferenças temporais das predições $\{P_1^*, P_2^*, P_3^*, \dots, P_{26}^*, 1\}$ vinculadas à sequência de ações $\{S_1^*, S_2^*, S_3^*, \dots, S_{26}^*\}$ mais o reforço final resultante da vitória do agente na partida de treino disputada G_1 . O vetor final $\overrightarrow{W}(26)$ servirá como vetor de pesos iniciais $\overrightarrow{W}(0)$ na equação (4.2) quando o agente for disputar a outra partida de treinamento G_2 .

Note que somente no início do jogo de cada partida de treino disputada é que se deve aguardar duas jogadas consecutivas do agente para se atualizar, pela 1ª vez, os pesos da rede neural. Feito o 1º ajuste, a partir daí os reajustes ocorrerão a cada ação executada pelo agente.

4.1.5 Cálculo de reajuste dos pesos da rede MLP

O cálculo de reajuste dos pesos da rede MLP utilizada por Lynch para treinar seu jogador de Damas em *NeuroDraughts* é uma extensão da equação (4.2) vista na subseção anterior. A estrutura da rede neural utilizada por Lynch pode ser vista na figura 17. Formalmente, o cálculo de reajuste dos pesos é definido pelas seguintes etapas:

- Dada duas predições sucessivas calculadas P_t e P_{t+1} referentes a dois estados consecutivos S_t e S_{t+1} resultantes de ações executadas pelo agente durante o jogo, calcula-se o sinal de erro através da equação:

$$e(t) = (\gamma P_{t+1} - P_t)$$

onde o parâmetro γ é uma constante de compensação da predição P_{t+1} em relação a predição P_t ;

- Calculam-se as eligibilidades locais da rede no instante t , isto é, $elig_{ij}^{(l)}(t)$. Cada eligibilidade $elig_{ij}^{(l)}(t)$ está vinculada a um peso sináptico $w_{ij}^{(l)}(t)$ correspondente. A eligibilidade é definida para três casos particulares:

1º) Caso em que o termo $elig_{ij}^{(0)}(t)$ está vinculado ao peso sináptico de conexão entre a saída do neurônio i da camada de entrada ($l = 0$) com a entrada do neurônio j da camada de saída ($l + 2$), isto é, a eligibilidade está vinculada a conexão direta entre a camada de entrada e a camada de saída. Neste caso, a eligibilidade $elig_{ij}^{(l)}(t)$, para $l = 0$, é definida por:

$$elig_{ij}^{(l)}(t) = \lambda \cdot elig_{ij}^{(l)}(t-1) + g'(P_t) \cdot a_i^{(l)};$$

2º) Caso em que o termo $elig_{ij}^{(0)}(t)$ está vinculado ao peso sináptico de conexão entre a saída do neurônio i da camada de entrada ($l = 0$) com a entrada do neurônio j da camada oculta ($l + 1$). Neste caso, a eligibilidade $elig_{ij}^{(l)}(t)$, para $l = 0$, é definida por:

$$elig_{ij}^{(l)}(t) = \lambda \cdot elig_{ij}^{(l)}(t-1) + g'(P_t) \cdot w_{ij}^{(l)}(t) \cdot g'(a_j^{(l+1)}) \cdot a_i^{(l)},$$

onde $a_j^{(l+1)}$ é o sinal de saída do neurônio j da camada oculta ($l + 1$);

3º) Caso em que o termos $elig_{ij}^{(1)}(t)$ está vinculado ao peso sináptico de conexão entre a saída do neurônio i da camada oculta ($l = 1$) com a entrada do neurônio

j da camada de saída ($l + 1$). Neste caso, a elegibilidade $elig_{ij}^{(l)}(t)$, para $l = 1$, é definida por:

$$elig_{ij}^{(l)}(t) = \lambda \cdot elig_{ij}^{(l)}(t-1) + g'(P_t) \cdot a_i^{(l)};$$

Dessa forma, pode-se definir a elegibilidade vinculada aos pesos sinápticos da camada l , para $0 \leq l \leq 1$, como se segue:

$$elig_{ij}^{(l)}(t) = \begin{cases} \lambda \cdot elig_{ij}^{(l)}(t-1) + g'(P_t) \cdot a_i^{(l)}, & \text{para } l=0 \text{ com conexão direta} \\ \lambda \cdot elig_{ij}^{(l)}(t-1) + g'(P_t) \cdot w_{ij}^{(l)}(t) \cdot g'(a_j^{(l+1)}) \cdot a_i^{(l)}, & \text{para } l=0 \\ \lambda \cdot elig_{ij}^{(l)}(t-1) + g'(P_t) \cdot a_i^{(l)}, & \text{para } l=1 \end{cases}$$

Conforme já foi discutido na subseção 4.1.4, a constante λ , para $0 \leq \lambda \leq 1$, tem o papel de dar uma “pesagem exponencial” para a taxa de variação das predições calculadas a k passos anteriores de t . Neste caso, para $\lambda = 0$, o cálculo do termo de elegibilidade $elig_{ij}^{(l)}(t)$ leva em consideração apenas a variação da predição calculada para o instante temporal t (esta variação é obtida pela derivada g'). Para $\lambda > 0$ e cada vez mais próximo de 1, mais o cálculo do termo de elegibilidade $elig_{ij}^{(l)}(t)$ leva em consideração as variações das predições calculadas a k passos anteriores de t , conforme é definido na expressão $\sum_{k=1}^t \lambda^{t-k} \nabla_w P_k$ da equação (4.2).

Para o neurônio j que está na camada de entrada, isto é, $l = 0$, faça:

$$a_j^{(0)} = x_j(t),$$

onde x_j é o j -ésimo elemento do vetor de entrada $X(t)$.

Como a função de ativação utilizada por Lynch é a tangente hiperbólica, então a sua derivada $g'(x)$ é definida por:

$$g'(x) = (1 - x^2)$$

A derivada da função tangente hiperbólica é uma função do tipo $g' : (-1, +1) \Rightarrow (0, +1)$, isto é, esta função tem intervalo de existência aberto entre 0 e +1 e tem como parâmetro de entrada $g(x)$.

Note que o termo $elig_{ij}^{(l)}(t)$ pode ser calculado incrementalmente após o processo de reajuste dos pesos da rede neural ter realizado as seguintes etapas: o agente executa

a ação, a rede reajusta os pesos e a predição final é recalculada para aquele estado resultante da ação executada pelo agente (para mais detalhes sobre o processo de reajuste dos pesos da rede neural, veja a subseção anterior);

- Calculado a elegibilidade local da rede, calcula-se a correção dos pesos $w_{ij}^{(l)}(t)$ da camada l , para $0 \leq l \leq 1$, através da seguinte equação:

$$\Delta w_{ij}^{(l)}(t) = \alpha^{(l)} \cdot e(t) \cdot elig_{ij}^{(l)}(t), \quad (4.3)$$

onde o parâmetro de aprendizagem $\alpha^{(l)}$ é definido por Lynch como:

$$\alpha^{(l)} = \begin{cases} \frac{1}{n}, & \text{para } l=0 \\ \frac{1}{20}, & \text{para } l=1 \end{cases}$$

onde n representa o número de neurônios na camada de entrada da rede neural.

Observe que Lynch utiliza um mesmo parâmetro de aprendizagem α para o ajuste de todos os pesos sinápticos de uma mesma camada l da rede. O parâmetro da taxa de aprendizagem é responsável por determinar a velocidade com que as correções dos pesos sinápticos da rede são efetuadas conforme a equação (4.3). Por exemplo, quando menor for o parâmetro da taxa de aprendizagem α , menor serão as variações dos pesos sinápticos da rede, de uma iteração para a outra, e mais suave será a trajetória no espaço de pesos.

Existe um problema típico associado ao uso de redes MLPs, que é o fato de a convergência estar assegurada para um mínimo local do erro, e não necessariamente para o mínimo global do erro. Quando a superfície de erro é boa, como na figura (22b), isto não representa um problema, mas quando a superfície é semelhante à figura (22a), com muitos mínimos locais, a convergência não é assegurada para o melhor valor. Nestes casos, geralmente se utiliza o termo momento μ para tentar solucionar este tipo de problema. A adição do termo momento no método TD(λ) determina o efeito das mudanças anteriores dos pesos na direção atual do movimento no espaço de pesos. Em outras palavras, o termo momento evita que o equilíbrio da função de avaliação se estabeleça em regiões cujo erro mínimo seja sub-ótimo (FAUSETT, 1994). Para resolver este problema Lynch empregou uma checagem de direção na equação (4.2) quando aplicado o termo momento. Neste caso, a checagem de direção tem por objetivo aplicar o termo momento μ somente quando a correção do peso atual $\Delta w_{ij}^{(l)}(t)$ e a correção anterior $\Delta w_{ij}^{(l)}(t-1)$ estiverem na mesma direção.

Note que, com isso, a parcela do momento jamais será aplicada no primeiro instante em que ocorre uma inversão no sentido do reajuste de peso, fato que evita um reforço prematuro do novo sentido.

Portanto, a equação final TD(λ) utilizada por Lynch para calcular o reajuste dos pesos da rede MLP na camada l , para $0 \leq l \leq 1$, é definida por:

$$w_{ij}^{(l)}(t) = w_{ij}^{(l)}(t-1) + \Delta w_{ij}^{(l)}(t), \quad (4.4)$$

onde $\Delta w_{ij}^{(l)}(t)$ é obtido nas seguintes etapas:

1º) Calcule $\Delta w_{ij}^{(l)}(t)$ através da equação (4.3);

2º) Se $(\Delta w_{ij}^{(l)}(t) > 0 \text{ e } \Delta w_{ij}^{(l)}(t-1) > 0)$ ou $(\Delta w_{ij}^{(l)}(t) < 0 \text{ e } \Delta w_{ij}^{(l)}(t-1) < 0)$ então faça:

$$\Delta w_{ij}^{(l)}(t) = \Delta w_{ij}^{(l)}(t) + \mu \cdot \Delta w_{ij}^{(l)}(t-1).$$

Observe que o termo momento μ é utilizado para reforçar tendências de estabilização nas direções dos reajustes dos pesos já manifestadas em tempos anteriores e mantidas no instante presente analisado. Caso não haja tal tendência, a parcela do termo momento não é aplicada (o que faz “freiar” o processo de reajuste dos pesos), conforme exposto na subseção 2.3.5.1.

4.1.6 Estratégia de treino por *self-play* com clonagem

Diversas estratégias de treino foram realizadas em (LYNCH, 1997) a fim de verificar qual estratégia de treinamento era a mais adequada para treinar o jogador de Damas através do sistema *NeuroDraughts*. Entre as estratégias avaliadas, podem ser citados os jogos diretos, jogos de especialistas, jogos contra oponentes humanos e o próprio treinamento por *self-play* que, segundo Lynch, quando aliado à técnica de clonagem e com pelo menos uma pequena quantidade de *look-ahead* em seus movimentos, é provavelmente a melhor estratégia de treinamento, porque é completamente automática e requer um aumento do nível de desempenho do agente a cada clonagem realizada.

A base do treinamento por *self-play* com clonagem é a idéia de se treinar um jogador através de vários jogos contra uma cópia de si próprio. À medida que o jogador for melhorando seu nível de desempenho de forma a conseguir bater esta cópia, uma nova clonagem é realizada e o jogador passa a treinar contra este novo clone. O processo se repete para um determinado número de jogos de treinamento.

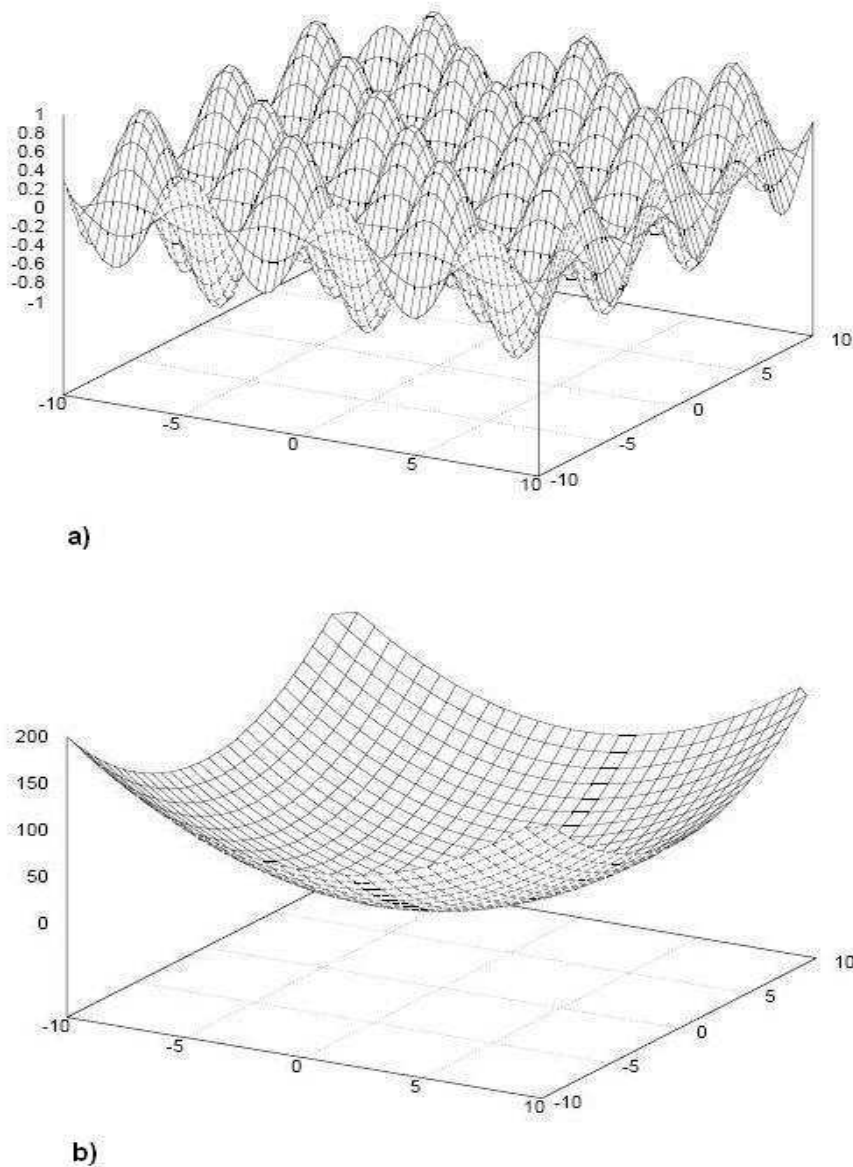


Figura 22: Exemplos de superfície de erro. (a) Uma má superfície de erro, com muitos mínimos locais. (b) Uma boa superfície de erro, cujo mínimo ótimo pode ser facilmente obtido, por exemplo, por regressão linear.

A estratégia de treinamento por *self-play* com clonagem utilizado por Lynch para treinar seu jogador de Damas pode ser dividida nas seguintes etapas:

1. Primeiramente, os pesos da rede neural MLP net_1 do agente são gerados aleatoriamente;
2. Antes de iniciar qualquer treinamento, é feita uma cópia da rede net_1 do agente, obtendo-se o seu 1º clone $cnet_1$;

3. O agente então inicia o treinamento da sua rede net_1 jogando contra a rede $cnet_1$ (oponente). O treinamento consiste em n jogos de treinamentos. Somente os pesos da rede net_1 do agente é que serão reajustados, conforme o que foi visto nas seções 4.4 e 4.5, durante estes n jogos de treinamento. Conseqüentemente, os pesos da rede $cnet_1$ do oponente permanecem inalterados durante o processo de treinamento do agente. Ambas as redes utilizam o mesmo tipo de mapeamento do tabuleiro, a mesma estrutura de rede e a mesma técnica de busca em profundidade para escolherem suas melhores ações;
4. Ao fim dos n jogos de treinamento, dois jogos-teste são realizados entre as redes net_1 e $cnet_1$ a fim de saber se o nível de desempenho da rede net_1 melhorou o suficiente para bater seu clone $cnet_1$. Caso consiga, uma cópia dos pesos da rede net_1 para a rede $cnet_1$ é realizada. Caso contrário, ambas as redes permanecem com seus mesmos pesos e nenhuma cópia será realizada;
5. Se o número de sessões s de treinamento não foi atingido, vá para etapa 3 e execute uma nova sessão de n jogos de treinamentos entre a rede net_1 e o seu último clone $cnet_1$.

Observe que esta estratégia de treinamento utilizado por Lynch é eficiente, uma vez que o agente deve sempre procurar melhorar o seu nível de desempenho a cada sessão de n jogos de treinamentos, de forma a poder bater o seu clone nos dois jogos-teste. No primeiro jogo “teste” a rede net_1 do agente joga com as peças pretas do tabuleiro de Damas e a rede $cnet_1$ joga com as peças vermelhas. Já no segundo jogo, as posições de ambas as redes sobre o tabuleiro de Damas são invertidas. O objetivo dessa troca de posições dos jogadores sobre o tabuleiro de Damas é permitir uma melhor avaliação do desempenho das redes net_1 e $cnet_1$ ao jogarem entre si, em ambos os lados do tabuleiro, uma vez que as características se referem a restrições sobre peças pretas e/ou vermelhas. A pontuação mínima que a rede net_1 precisa alcançar nos dois jogos-teste, para que seus pesos possam ser copiados para a rede $cnet_1$, é definida em um arquivo de tabuleiro (este arquivo é um parâmetro de entrada no programa *NeuroDraughts* de Lynch).

4.2 O Processo Evolutivo do *LS-Draughts*

Nesta seção, são apresentados os aspectos particulares do *LS-Draughts*. Conforme já dito, o aprendizado das redes é guiado pelo método das Diferenças Temporais, tal

como no *NeuroDraughts* descrito na seção 4.1. Contudo, o *LS-Draughts* expande o *NeuroDraughts* por incluir um módulo de geração automática de características (módulo 1 da arquitetura geral do *LS-Draughts* visto na figura 15) que lhe confere aspectos peculiares no processo de treinamento, conforme será apresentado a seguir. Assim, as subseções se sucederão de acordo com o disposto a seguir: **i)** população e codificação dos indivíduos; **ii)** seleção dos indivíduos e aplicação dos operadores genéticos; **iii)** treinamento da rede MLP acoplada aos indivíduos; e, **iv)** função de avaliação e atualização da população para próxima geração.

4.2.1 População e codificação dos indivíduos no *LS-Draughts*

Cada indivíduo da população é formado por um cromossomo fixo de 15 genes cuja representação binária indica se, em cada gene G_i , onde $i \in \{1, 2, 3, \dots, 15\}$, há a presença, ou não, de uma determinada característica F_i referente ao mapeamento *NET-FEATUREMAP* do jogador de Mark Lynch (veja figura 23).

F1	F2	F3	F4	F5	F6	F7		F14	F15
1	1	0	0	1	1	1	...	1	0
G1	G2	G3	G4	G5	G6	G7		G14	G15

Figura 23: Representação dos 15 genes de um cromossomo vinculado a um determinado indivíduo da população

Neste caso, as 15 características utilizadas para representar os 15 genes da figura 23 estão indicadas na tabela 3.

Uma vez definida e gerada a estrutura cromossômica de cada indivíduo (a geração do indivíduo é discutida mais adiante), uma rede neural MLP é acoplada ao mesmo. Esta rede utiliza as características ativas ($G_i = 1$) presentes na estrutura do cromossomo para, então, representar o conjunto de características do mapeamento *NET-FEATUREMAP* que será utilizado no treinamento por Diferenças Temporais da mesma (a criação dessa rede e o seu treinamento será discutido com mais detalhes na subseção 4.2.3).

A população do AG deste trabalho é composta por um conjunto de 50 indivíduos, isto é, $T_P = 50$. Portanto, a população será formada por 50 estruturas cromossômicas (ou indivíduos) que estarão associadas à 50 redes neurais MLPs (cada rede para cada indivíduo). São esses 50 indivíduos que estarão evoluindo dentro do AG ao longo de 30

gerações.

O processo de geração dos indivíduos em uma determinada geração do *LS-Draughts* ocorre de duas formas:

1. Na população inicial (geração 0) cada um dos 50 indivíduos são gerados como se segue: há uma escolha aleatória de ativação (1 ou 0) da característica F_i correspondente ao gene G_i , para cada $i \in \{1, 2, 3, \dots, 15\}$. A partir daí, uma rede é acoplada a cada um dos 50 novos indivíduos gerados e o treinamento dessas redes é realizado. Em seguida, estes 50 indivíduos gerados e treinados recebem uma avaliação (ou *fitness*) e seguem para a próxima geração (geração 1). Tal avaliação será vista em detalhes na subseção 4.2.4;
2. Em cada uma das demais 29 gerações GE_i , onde $1 \leq i \leq 29$, repete-se o mesmo processo descrito a seguir para produzir os 50 novos indivíduos que serão repassados à geração GE_{i+1} : 50 novos indivíduos são gerados a partir da população repassada à geração GE_i pela geração GE_{i-1} , através do método de seleção e da aplicação dos operadores genéticos de *crossover* e mutação aos indivíduos da referida população. Note que, no contexto de uma geração GE_i , a população que ele recebe de GE_{i-1} é chamada de “população de pais”. Assim sendo, a população de GE_i é representada pelos 50 indivíduos de sua população de pais acrescido dos 50 novos indivíduos gerados. Nesta etapa, uma rede também é acoplada aos 50 novos indivíduos gerados e o treinamento dessas redes é realizado. Em seguida, os 50 novos indivíduos gerados e treinados, mais os 50 indivíduos da população de pais recebem uma avaliação (ou *fitness*) que determinará, dentre os 100 indivíduos, quais são os 50 melhores que seguirão para a geração GE_{i+1} . Este processo se repete para cada geração, até o fim da geração 29.

4.2.2 Seleção dos indivíduos e aplicação dos operadores genéticos

O método de seleção utilizado pelo *LS-Draughts* para selecionar os pais, a fim de aplicar os operadores genéticos que darão origem aos novos indivíduos, é o torneio estocástico com *tour* de 3: dentre uma população de 50 pais, 3 indivíduos são selecionados pelo método da roleta e submetidos a um torneio, no qual o indivíduo ganhador é aquele que possuir o maior valor de *fitness*. Para cada 2 pais escolhidos pelo torneio estocástico, 2 novos filhos são gerados. O operador de *crossover* utilizado é o cruzamento simples de genes (*crossover* de um único ponto de corte) e a mutação é aplicada a todos os indivíduos

da população. A taxa de probabilidade de mutação, P_{mut} , utilizada é de 0.3 por gene, isto é, todos os indivíduos sofrem mutação em 5 genes escolhidos aleatoriamente.

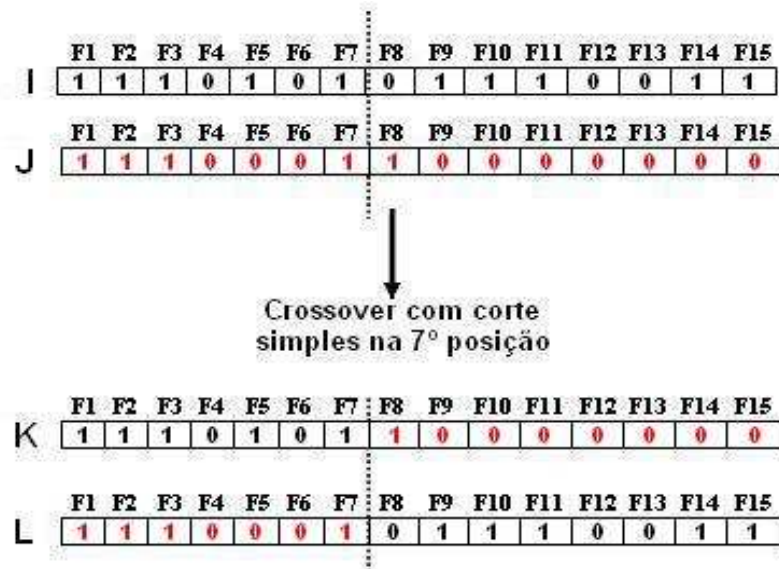


Figura 24: Operação de *crossover*, com um único ponto de corte, aplicado a um par de indivíduos pais para gerar dois novos indivíduos.

A figura 24 mostra um exemplo de aplicação do operador genético de cruzamento simples na geração de dois novos indivíduos (filhos *K* e *L*) a partir de um par de indivíduos pais (*I* e *J*). Na figura 25, exemplifica-se a aplicação do operador genético de mutação ao filho *K*, produzindo o indivíduo *M*.



Figura 25: Operação de mutação de gene com uma taxa de 0.3 sobre o indivíduo *K*.

4.2.3 Treinamento da rede MLP acoplada aos indivíduos

Após a geração de um novo indivíduo I_i , por qualquer uma das duas formas descritas na subseção 4.2.1, onde $i \in \{1, 2, 3, \dots, 50\}$, um filtro é aplicado à sua sequência de genes selecionando apenas as N_A características que estão presentes (genes ativos). A partir daí, uma rede neural MLP com N_A neurônios na camada de entrada, 20 neurônios na camada oculta e um único neurônio na camada de saída é gerada e acoplada ao novo indivíduo I_i . Veja um exemplo na figura 26: a rede acoplada ao indivíduo M utilizará apenas as características F_1 , F_2 e F_{14} (características ativas no indivíduo) para representar o tabuleiro de Damas na entrada da rede neural acoplada. Os pesos iniciais da rede neural MLP vinculada ao indivíduo I_i são gerados aleatoriamente entre -0.2 e +0.2 e o termo *bias* é fixado em 1. Este processo se repete para todos indivíduos I_i , onde $1 \leq i \leq 50$.

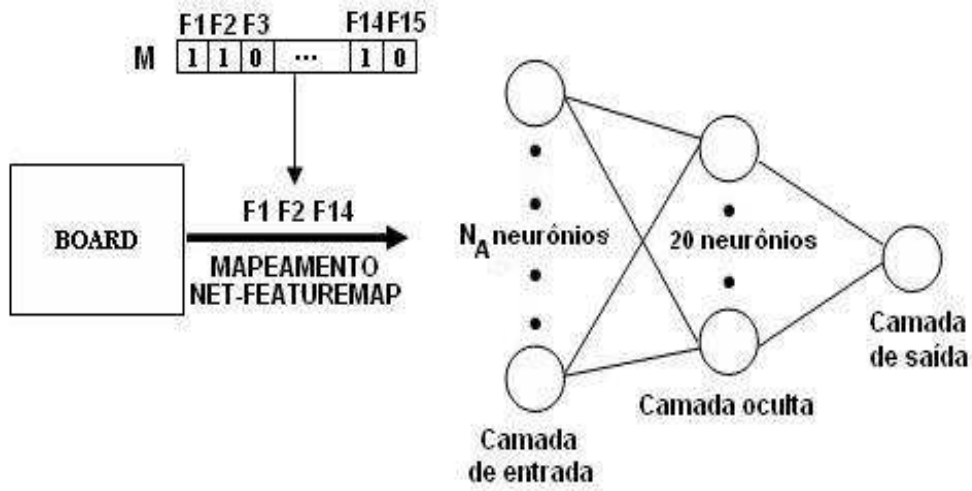


Figura 26: Seleção das características ativas do indivíduo M para definição do mapeamento *NET-FEATUREMAP* que a rede acoplada estará utilizando no treinamento.

Após a criação da rede neural para cada I_i , o *LS-Draughts* inicia a etapa de treinamento das 50 redes neurais MLPs acopladas aos 50 indivíduos gerados, isto é, a aprendizagem de cada agente jogador de Damas através do método de aprendizagem por reforço $TD(\lambda)$ aliada a técnica de *self-play* com clonagem e com busca *minimax* de profundidade 4. Esta etapa utiliza o mapeamento de tabuleiro *NET-FEATUREMAP* de Mark Lynch e os módulos 2, 3 e 4 da arquitetura do *LS-Draughts* que foram apresentados na seção 4.1 desta dissertação.

O treinamento de cada rede neural consiste em um conjunto de 4 sessões de 400 jogos de treinamento. Dos 400 jogos de treinamento realizado em cada sessão, a rede neural

joga metade deles como jogador preto (i.e., joga com as peças pretas do tabuleiro) e a outra metade como jogador vermelho (neste caso, a rede joga com as peças vermelhas do tabuleiro). Tal estratégia tem como objetivo treinar o agente para situações ainda mais diversificadas, uma vez que as características se referem a restrições sobre peças pretas e/ou vermelhas.

Antes de iniciar as 4 sessões de treinamento por *self-play*, é feita uma cópia da rede neural net_i vinculada ao indivíduo I_i , obtendo-se a rede clone $cnet_i$. Feito isso, a 1ª sessão de 400 jogos de treinamento é realizada entre a rede net_i (agente) e o seu clone $cnet_i$ (oponente), sendo que apenas os pesos da rede net_i é que são ajustados pelo algoritmo TD(λ) durante os 400 jogos da sessão. Ao fim desta, dois jogos-teste são realizados para verificar se o nível de jogo da rede net_i melhorou o suficiente para bater o seu clone $cnet_i$. Caso consiga, os pesos da rede net_i são, então, copiados para a rede clone $cnet_i$. Caso contrário, ambas as redes permanecem com os mesmos pesos que tinham durante os jogos-teste para a próxima sessão de treinamento. Este processo se repete até o fim da 4ª sessão de treinamento. Observe que, neste processo de treinamento, ambas as redes utilizam a mesma técnica de busca em profundidade, sendo que a única diferença entre as duas redes é que, diferentemente de $cnet_i$, net_i sofre reajuste de pesos durante o processo.

O objetivo dos dois jogos-teste utilizado na técnica de *self-play* com clonagem é o de avaliar o desempenho das duas redes neurais ao jogarem em ambos os lados do tabuleiro de Damas, isto é, no primeiro jogo, a rede net_i joga como jogador preto enquanto que a rede $cnet_i$ joga como jogador vermelho; no segundo jogo, as posições de ambas as redes no tabuleiro são invertidas. Se net_i superar $cnet_i$ nestes dois jogos-teste, obtendo uma quantidade de pontos suficientes para garantir sua clonagem, então seus pesos são copiados para a rede clone $cnet_i$ e, assim, net_i passa a treinar contra a sua melhor cópia. Caso contrário, net_i continua treinando até conseguir bater seu último clone $cnet_i$. Com a técnica de *self-play* com clonagem, a rede neural net_i procurará sempre aprender a jogar Damas por si própria, evoluindo seu nível de desempenho a cada clonagem e batendo seus clones.

Por outro lado, nem sempre a rede net_i que tenha finalizado as 4 sessões de treinamentos é melhor do que todos os seus clones gerados. Isso ocorre porque a rede final net_i pode se especializar, durante o treinamento, em bater apenas seu último clone. Portanto, para garantir que a melhor rede neural esteja realmente vinculada ao indivíduo I_i , um pequeno torneio é realizado entre a rede final net_i e todos os seus clones gerados nas 4 sessões de treinamento. Note que, neste torneio, haverá de 1 a 5 jogos, dependendo do número de êxitos do indivíduo net_i nos jogos-teste das 4 sessões de treinamento (con-

siderando os extremos, será 1, no caso de ele não ter tido êxito em nenhum deles, ou será 5, no caso de ele ter tido êxito em todos). Cada confronto deste torneio consiste em duas partidas-teste, tal como é feito nos dois jogos-teste da própria técnica de *self-play* com clonagem. Vence o torneio a rede que obtiver maior pontuação. Se a melhor rede deste torneio for um dos clones, então os seus pesos serão copiados para a rede net_i vinculada ao indivíduo I_i . Caso contrário, nada será feito, uma vez que o indivíduo I_i já estará vinculado com a sua melhor rede, isto é, net_i .

Os parâmetros de entrada utilizados no treinamento por TD(λ) de cada rede neural net_i vinculada ao indivíduo I_i , são: um arquivo de tabuleiro (a ser detalhado a seguir), a taxa de aprendizagem para camada de entrada $\alpha^{(0)} = \frac{1}{N_A}$, a taxa de aprendizagem para camada oculta $\alpha^{(1)} = \frac{1}{20}$, o momento = 0.9, gamma = 0.98 e lambda = 0.1.

A figura 27, mostra o arquivo de tabuleiro utilizado no processo de treinamento por TD(λ) de uma rede net_i vinculada ao indivíduo I_i no *LS-Draughts*. Neste arquivo, têm-se as 32 posições (ou seqüência dos 32 quadrados) de um tabuleiro inicial 8x8 de Damas, a partir do qual os jogadores iniciarão as 400 partidas de treinamento de cada uma das 4 sessões de treino. Os números 0, 1 e 2 desta seqüência representam, respectivamente, quadrados vazios, quadrados ocupados por peças simples pretas e quadrados ocupados por peças simples vermelhas. Os valores -0.1 e +0.1 representam os valores de reforços a serem repassados ao agente (ou rede net_i) caso o mesmo empate a partida iniciada pela seqüência do tabuleiro relacionado. O valor -0.1 está vinculado ao reforço de empate no caso em que a rede net_i empata a partida jogando com as peças pretas do tabuleiro e o valor +0.1 está vinculado ao reforço de empate no caso em que a rede net_i empata a partida jogando com as peças vermelhas do tabuleiro. Isto permite treinar a rede net_i em situações onde o empate é realmente bom, por exemplo, quando se tem peças em desvantagens já no início da partida de treinamento.

O valor 4 indicado na figura 27 representa a quantidade de pontos mínimos que a rede net_i necessita obter nos dois jogos-teste para que os seus pesos possam ser copiados para a rede clone $cnet_i$. Os pontos em cada jogo “teste” são distribuídos da seguinte forma: 2 pontos por vitória, 1 ponto por empate e nenhum ponto para derrota. No caso em questão, a pontuação 4 indica que a rede net_i deverá vencer os dois jogos-teste para que os seus pesos possam ser copiados para a rede clone $cnet_i$.

Note que o treinamento utilizado aqui é realizado para um determinado conjunto de características selecionadas pelo AG e que representam as percepções do agente sobre o domínio de Damas. Na figura 26, tem-se que o agente acoplado ao indivíduo M estará em treinamento utilizando apenas as características ativas $F1$, $F2$ e $F14$ como funções

1-Quantidade de tabuleiros iniciais

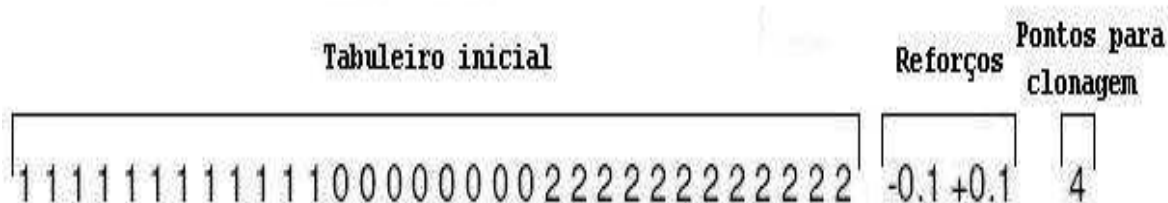


Figura 27: Arquivo de tabuleiro para treinamento.

de mapeamento dos estados do tabuleiro de Damas na entrada da rede neural MLP. Suponha que $F1$, $F2$ e $F14$ sejam respectivamente as características “TotalMobility”, “PieceAdvantage” e “Threat” indicadas na tabela 3. Neste caso, a rede acoplada ao indivíduo M terá apenas 11 neurônios na camada de entrada para mapear os estados do tabuleiro de Damas (os 11 neurônios representam o número total de bits alocados pelas 3 funções ativas).

4.2.4 Função de avaliação e atualização da população para próxima geração

Em uma dada geração GE_j , onde $1 \leq j \leq 29$, após o término de treinamento de todas as 50 redes neurais MLPs acopladas aos 50 novos indivíduos gerados a partir da população de pais, um torneio é, então, realizado entre todos os 100 indivíduos de sua população. O objetivo deste torneio é obter um *fitness* para cada um dos 100 indivíduos que estão no torneio, de forma a poder atualizar a população de 50 indivíduos que irão para a próxima geração. O torneio consiste, basicamente, em calcular os pontos (ou *fitness*) obtidos por cada jogador I_i , onde $1 \leq i \leq 100$, após este disputar 10 jogos contra cada um dos indivíduos do torneio, menos contra ele próprio. Em outras palavras, I_i joga 10 partidas de Damas contra cada um dos 99 jogadores do torneio. Assim, o *fitness* calculado para I_i será obtido em função dos resultados das 990 partidas disputadas. Em cada geração diferente de 0 (i.e., da geração 1 até geração 29), a etapa do cálculo do *fitness* dos 100 indivíduos do torneio terá sempre 99.000 partidas no total (100 jogadores disputando 990 partidas dentro do torneio). No caso especial da geração GE_0 (geração 0), o torneio é

realizado entre 50 indivíduos e tem um total de 24.500 jogos de *fitness* (50 jogadores disputando 490 partidas dentro do torneio).

A pontuação por jogo disputado é dada da seguinte forma: 2 pontos por vitória, 1 ponto por empate e 0 por derrota. Suponha, por exemplo, que o desempenho do indivíduo I_i em 990 jogos de torneio disputado seja: 590 vitórias, 320 empates e 80 derrotas. Assim, o *fitness* calculado para o jogador I_i será de 1500 pontos.

O objetivo dos 10 jogos disputados em cada confronto é avaliar o desempenho do jogador I_i ao jogar 5 jogos como jogador preto, cada um deles partindo de um tabuleiro inicial B_i distinto dos demais, e, os 5 outros, como jogador vermelho, partindo dos mesmos 5 tabuleiros B_i 's utilizados anteriormente. Note que cada um desses tabuleiros iniciais B_i não corresponde ao tabuleiro inicial padrão B_P da figura 27, mas, sim, ao tabuleiro resultante da aplicação de uma das ações iniciais possíveis a B_P (que correspondem a 7 ações, das quais foram escolhidas 5).

Finalizado o cálculo do *fitness* para todos os I_i indivíduos do torneio, onde $1 \leq i \leq 100$, um *rank* de seleção dos 50 melhores, dentre os 100 indivíduos avaliados no torneio, é realizado a fim de atualizar a população de 50 indivíduos que irão para a próxima geração. Assim, dentre 100 indivíduos que disputaram o torneio de *fitness*, apenas metade deles é que serão mantidos para próxima geração, isto é, somente os indivíduos mais aptos e com valores de *fitness* maiores é que serão mantidos para a próxima geração. Observe que este *rank* dos 50 melhores indivíduos é aplicado apenas nas gerações 1 à 29. No caso especial da geração 0, todos os 50 indivíduos da população inicial que foram avaliados no torneio de *fitness* seguem para a próxima geração.

4.3 Ferramenta utilizada na implementação do *LS-Draughts*

A escolha de uma linguagem de programação para a implementação dos dois módulos do sistema *LS-Draughts* foi definida em função da linguagem adotada por Lynch na implementação do seu sistema *NeuroDraughts*. Em <http://iamlynch.com/nd.html>, Lynch deixa disponível, para *download*, o código fonte da sua implementação em C++ referente ao seu jogador de Damas. Como o 2º módulo do *LS-Draughts* é praticamente este jogador de Lynch, então o objetivo foi ganhar tempo com boa parte da implementação deste módulo já garantida, em função do reaproveitamento do código em C++ do jogador de Lynch, e, assim, poder propor uma extensão a este jogador, o que deu origem ao 1º módulo do *LS-Draughts*. Convém lembrar que o próprio Lynch, como trabalho futuro, havia pro-

posto tal extensão como uma estratégia de aprimoramento de seu jogador. Os resultados experimentais obtidos pelo *LS-Draughts*, descritos a seguir, confirmam o aumento do desempenho do jogador original de Lynch com a inserção do módulo de geração automática de características do *LS-Draughts*.

Foi graças a disponibilidade de acesso ao código fonte da implementação do jogador de Mark Lynch que foi possível o desenvolvimento deste trabalho, uma vez que a complexidade do método das Diferenças Temporais exigia a necessidade de uma observação prática a fim de melhor entender o seu funcionamento teórico e funcional.

Por fim, a ferramenta escolhida para a implementação destas duas etapas do sistema *LS-Draughts* foi o compilador Borland C++ Builder 5.0.

4.4 Resultados Experimentais

O sistema *LS-Draughts* foi executado para 30 gerações e gastou quase 4 meses de execução (média de 2 gerações por semana). Após cada 5 gerações, começando da geração 0 até a geração 29, o *fitness* do melhor indivíduo foi comparado com a média do *fitness* dos demais indivíduos da população, conforme é mostrado na figura 28. O *fitness* dos indivíduos da geração 0 estão baixos devido ao fato de o torneio de *fitness* envolver apenas 50 indivíduos (neste caso, o *fitness* máximo que um indivíduo pode alcançar na geração 0 é de 980 pontos = 2 pontos por vitória * 49 confrontos de 10 partidas cada). Já nas gerações subseqüentes, o *fitness* máximo que um indivíduo pode alcançar é 1980 pontos (2 pontos por vitória * 99 confrontos de 10 partidas cada).

Fazendo uma análise geral da evolução dos indivíduos da população do *LS-Draughts* durante as 30 gerações, foi observado que: até a 7ª geração do *LS-Draughts*, o melhor indivíduo de cada geração não conseguiu obter bons resultados em relação ao jogador de Mark Lynch. Em muitos casos, ao se avaliarem os indivíduos de uma mesma geração, aqueles que possuíam *fitness* abaixo do melhor indivíduo da população obtiveram resultados melhores do que este ao jogarem contra o jogador de Lynch. Somente na 8ª geração é que o melhor indivíduo da população do *LS-Draughts* conseguiu bater o melhor jogador do *NeuroDraughts*: em 7 partidas disputadas, o melhor jogador da 8ª geração do *LS-Draughts* venceu 1 partida contra o melhor jogador do *NeuroDraughts* e empatou as outras 6 partidas restantes. Este melhor jogador da 8ª geração do *LS-Draughts* utilizou 12 das 15 características disponíveis no mapeamento *NET-FEATUREMAP* para aprender a jogar Damas e conseguir bater o jogador de Mark Lynch (veja as características deste jogador na figura 29). Apesar de este resultado bem sucedido ter sido atingido já na 8ª

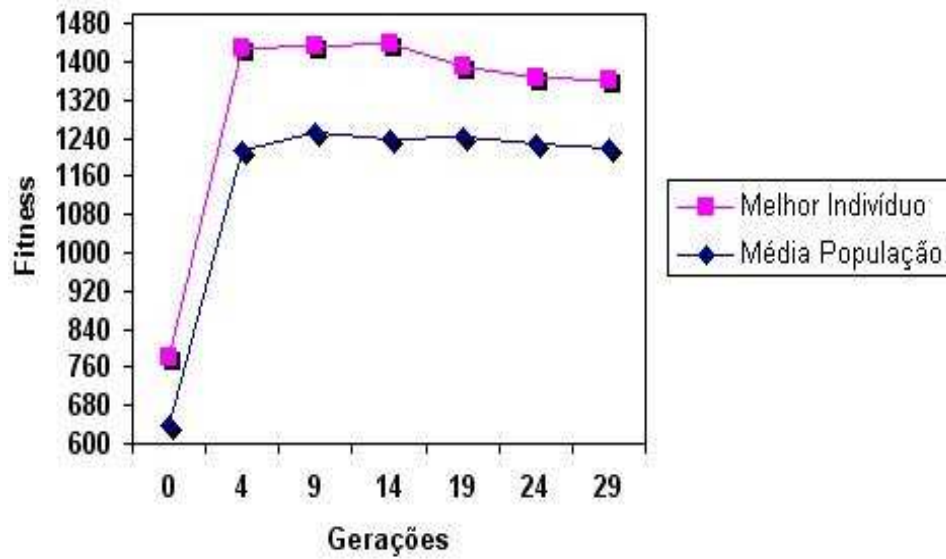


Figura 28: Gráfico de evolução do melhor indivíduo em relação a média da população nas gerações 0, 4, 9, 14, 19, 24 e 29.

geração, optou-se pela continuidade da execução até a 29ª geração com o objetivo de se tentar obter um jogador que batesse o jogador de Mark Lynch utilizando um conjunto de características menor do que aquele de 12 características utilizado pelo melhor jogador da 8ª geração.

CARACTERÍSTICAS	BITS
<i>PieceAdvantage</i>	4
<i>PieceDisadvantage</i>	4
<i>PieceThreat</i>	3
<i>PieceTake</i>	3
<i>Advancement</i>	3
<i>DoubleDiagonal</i>	4
<i>Backrowbridge</i>	1
<i>XCentrecontrol</i>	3
<i>TotalMobility</i>	4
<i>Exposure</i>	3
<i>KingCentreControl</i>	3
<i>DiagonalMoment</i>	3

Figura 29: Conjunto de características do melhor jogador da 8ª geração do *LS-Draughts*.

O melhor jogador da 8ª geração continuou como melhor indivíduo do *LS-Draughts* até a 14ª geração. Da geração 15 até a geração 23, o desempenho do melhor indivíduo de cada geração, em relação ao jogador de Lynch, voltou a ser insatisfatório: o melhor indivíduo não conseguia obter mais vitórias do que derrotas, chegando, no máximo, a empatar todas as partidas disputadas contra o melhor jogador de Mark Lynch. Somente na 24ª geração é que o melhor indivíduo da população do *LS-Draughts* conseguiu bater o melhor jogador do *NeuroDraughts*: em 7 partidas disputadas, o melhor jogador da 24ª geração do *LS-Draughts* venceu 2 partidas contra o melhor jogador do *NeuroDraughts* e empatou as outras 5 partidas restantes utilizando apenas 7 características do mapeamento *NET-FEATUREMAP*. Este melhor jogador da 24ª geração continuou como melhor indivíduo do *LS-Draughts* até o fim da 29ª geração, quando a execução do sistema *LS-Draughts* foi finalizada para levantamento dos resultados.

Os resultados experimentais obtidos com o melhor indivíduo de cada geração do *LS-Draughts* foram avaliados em relação a 2 propósitos:

1. Encontrar um conjunto mínimo de características que possa otimizar o treino de um agente jogador de Damas, isto é, otimizar o treino por Diferenças Temporais de uma rede neural MLP que, acoplada ao indivíduo representado por este conjunto mínimo de características, após treinada, jogue com alto nível de desempenho. Este conjunto mínimo de características deve representar o conhecimento mínimo de que o agente necessita sobre o domínio de Damas para poder jogar com eficiência. Observe que este propósito não necessariamente deve ser atendido pelo melhor jogador de cada geração do *LS-Draughts*, uma vez que a função de avaliação (ou *fitness*) dos jogadores não leva em consideração o número de características utilizadas no mapeamento *NET-FEATUREMAP*. Como foi apresentado anteriormente, o melhor indivíduo da 8ª geração do *LS-Draughts* obteve um bom desempenho contra o jogador de Mark Lynch utilizando 12 características para representar o tabuleiro de Damas na entrada da rede neural (mais adiante, este indivíduo será comparado com o melhor indivíduo da 24ª geração através de um torneio que avaliará a eficiência das características utilizadas por ambos, uma vez que os dois jogadores conseguiram bater o jogador de Lynch e apresentarem um bom nível de jogo);
2. Jogar contra o melhor jogador do sistema *NeuroDraughts* de Lynch disponível, a fim de verificar se o conjunto mínimo de características geradas pelo melhor indivíduo de cada geração do sistema *LS-Draughts* foi o suficiente para otimizar o treino da rede neural MLP acoplada a este indivíduo e, com isso, garantir um alto nível

de desempenho no jogo de Damas. Como o melhor jogador do *NeuroDraughts* é considerado um bom jogador de Damas (LYNCH, 1997), então o propósito desta etapa é prover uma medida de eficiência para o conjunto de características utilizadas pelo melhor jogador do *LS-Draughts* no seu treinamento. O melhor jogador de Lynch está disponível para *download* no website <http://iamlynch.com/nd.html>. Este jogador somente deixa como opção para seus adversários jogar com as peças pretas do tabuleiro, uma vez que a interface do tabuleiro do jogo que está amarrado ao mesmo é fixa.

Avaliando o 1º propósito, o conjunto mínimo de características geradas pelo melhor indivíduo da 24ª geração do *LS-Draughts* está indicado na figura 30. De um total de 15 características disponíveis no mapeamento *NET-FEATUREMAP*, o melhor indivíduo da 24ª geração do *LS-Draughts* utilizou apenas 7 características para representar o tabuleiro de Damas e assim, otimizar o processo de aprendizagem de uma rede neural através do método das Diferenças Temporais. Esse resultado mostra que o melhor indivíduo da 24ª geração do *LS-Draughts* foi capaz de manter o bom nível de jogo obtido, primeiramente, pelo melhor indivíduo da 8ª geração (batendo, como este último, o melhor jogador de Mark Lynch), utilizando, porém, um número menor de características que ele.

CARACTERÍSTICAS	BITS
<i>PieceAdvantage</i>	4
<i>PieceDisadvantage</i>	4
<i>PieceThreat</i>	3
<i>XCentrecontrol</i>	3
<i>TotalMobility</i>	4
<i>Exposure</i>	3
<i>Threat</i>	3

Figura 30: Conjunto de características do melhor jogador da 24ª geração do *LS-Draughts*.

Com relação ao 2º propósito e, conforme já foi apresentado anteriormente, a fim de se testar a eficiência do melhor indivíduo da 24ª geração do *LS-Draughts*, ele disputou um torneio de 7 partidas (total de aberturas iniciais possíveis, quando se joga com as peças pretas do tabuleiro) contra o melhor jogador do sistema *NeuroDraughts* de Lynch.

O resultado foi: 2 vitórias para o jogador do sistema *LS-Draughts* e 5 empates. Porém, dos 5 jogos que terminaram empatados, 2 jogos só não resultaram em vitória por parte do jogador do *LS-Draughts* devido ao problema do *loop* de final de jogo. Nestes 2 jogos, o melhor jogador do *NeuroDraughts* terminou a partida com apenas 1 rainha sobre o tabuleiro final, enquanto que o seu oponente (melhor jogador da 24ª geração do *LS-Draughts*) terminou a partida com 3 rainhas e 1 peça simples sobre o tabuleiro. Apesar de o jogador do *LS-Draughts* ter brigado bastante durante a partida, cercando e atacando as peças do jogador do *NeuroDraughts*, o ritmo diminuiu bastante nos instantes finais do jogo, o que acabou resultando no empate. A figura 31 e 32 mostram o *loop* de final de jogo resultante destas duas partidas, que apresentaram o mesmo *loop* final: a partir do tabuleiro da figura 31, resultante do 43º movimento do jogador do *NeuroDraughts*, o jogo chegou ao tabuleiro da figura 32, resultante da 44ª jogada de ambos os jogadores. Em seguida, na 45ª jogada, o jogo retornou para o tabuleiro da figura 31, iniciando, dessa forma, um *loop* infinito alternado entre estas 2 configurações de tabuleiro. Nas outras 3 partidas que terminaram empatadas, houve uma pequena vantagem de peças por parte do jogador do *NeuroDraughts* sobre o tabuleiro final.

O torneio de 7 jogos disputado entre o melhor indivíduo da 8ª geração do *LS-Draughts* e o melhor jogador do sistema *NeuroDraughts* também apresentou o problema do *loop* de final de jogo: em 6 partidas que terminaram empatadas, o jogador do *LS-Draughts* levou vantagem sobre 3 partidas, chegando a terminar, em uma delas, com 2 rainhas e 3 peças simples sobre o tabuleiro final do jogo, enquanto que o jogador do *NeuroDraughts* terminou com apenas 1 rainha e 1 peça simples sobre o tabuleiro. Mesmo com esta vantagem, o jogador do *LS-Draughts* não foi capaz de pressionar o jogador do *NeuroDraughts* nos instantes finais da partida para tentar finalizá-la com vitória.

O anexo A dessa dissertação contém uma das partidas na qual o melhor jogador da 24ª geração do *LS-Draughts* bate o melhor jogador do sistema *NeuroDraughts* de Lynch. Conforme pode ser visto, nesta partida, o jogador do *LS-Draughts* joga muito bem contra o jogador do *NeuroDraughts*, mantendo um alto nível de padrão de jogo, terminando a partida, inclusive, com 3 rainhas e 1 peça simples sobre o tabuleiro final do jogo. A outra partida em que o jogador do *LS-Draughts* vence o jogador do *NeuroDraughts*, o primeiro termina a partida com 1 rainha e 2 peças simples sobre o tabuleiro final do jogo.

É importante ressaltar que a rede neural MLP do melhor jogador do *NeuroDraughts* e também do melhor jogador de cada geração do *LS-Draughts* utiliza o mesmo número de neurônios na camada oculta e na camada de saída: 20 neurônios na camada oculta e um único neurônio na camada de saída, além do link de conexão direta entre a camada



Figura 31: Posição do tabuleiro de Damas em *loop* depois do 43º movimento do jogador vermelho.

de entrada e a camada de saída.

Por fim, um torneio foi realizado entre o melhor jogador da 8ª geração do *LS-Draughts* e o melhor jogador da 24ª geração do *LS-Draughts* a fim de verificar qual deles teriam o melhor desempenho, uma vez que ambos conseguiram bater o melhor jogador do *Neuro-Draughts* e apresentarem um bom nível de jogo. De um total de 14 partidas disputadas (7 partidas como jogador preto e as outras 7 como jogador vermelho), o resultado do torneio foi: 2 vitórias para cada lado e 10 empates. Das 10 partidas que terminaram empatadas, o melhor jogador da 8ª geração do *LS-Draughts* levou vantagem sobre 6 partidas, chegando a terminar, em uma delas, com 2 rainhas e 2 peças simples sobre o tabuleiro final, enquanto que o seu oponente terminou com apenas 1 rainha sobre o tabuleiro. Estes resultados mostram, novamente, o problema do *loop* de final de jogo enfrentado pelos agentes jo-

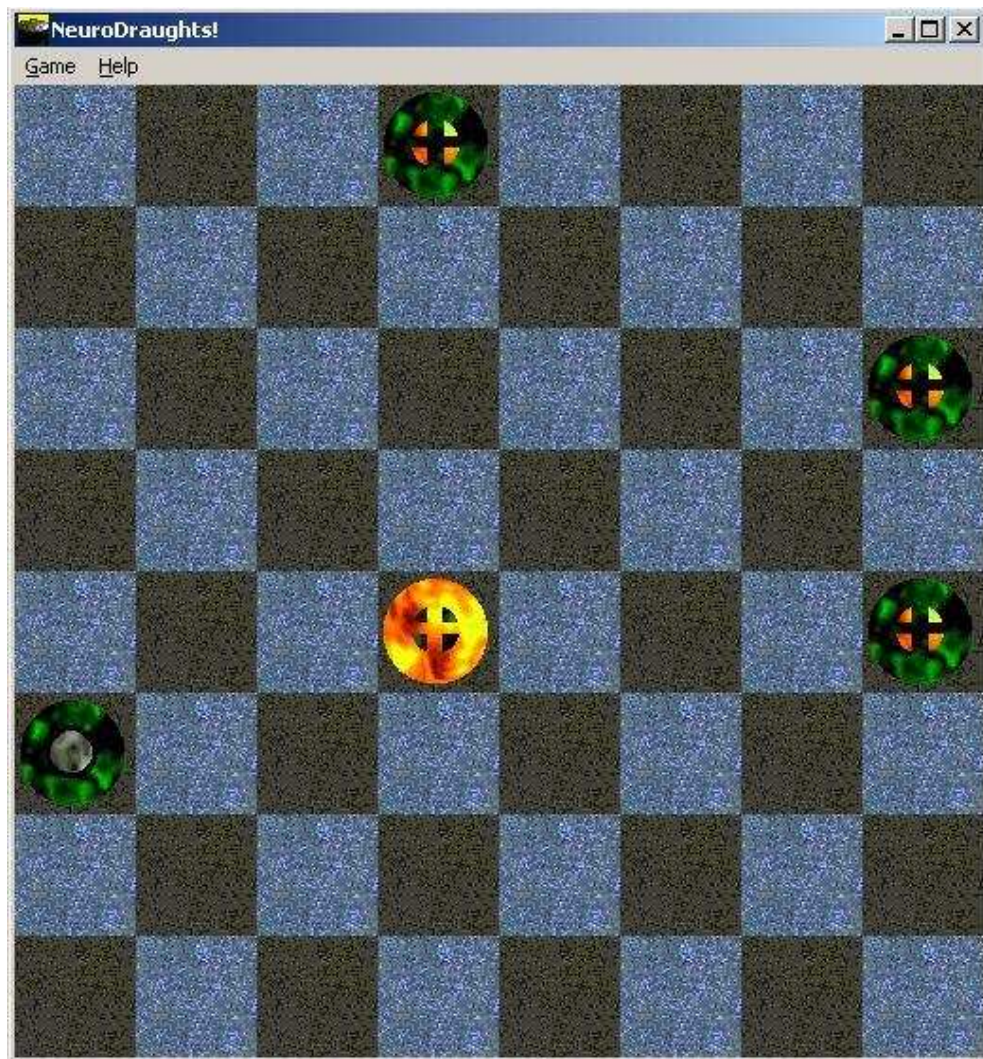


Figura 32: Posição do tabuleiro de Damas em *loop* depois do 44º movimento do jogador vermelho.

gadores do sistema *LS-Draughts* e levanta a possibilidade de se utilizar o melhor jogador da 8ª geração do *LS-Draughts* como uma boa solução para o AG aqui implementado, uma vez que o tempo gasto para se obter o melhor jogador da 24ª geração do *LS-Draughts* foi muito extenso (foram necessários quase 3 meses de execução). Apesar de este último apresentar um desempenho melhor contra o jogador do *NeuroDraughts* (2 vitórias e 5 empates) e utilizar apenas 7 características no mapeamento *NET-FEATUREMAP*, enquanto que o primeiro utiliza 12 características e apresenta um desempenho um pouco menor contra o jogador do *NeuroDraughts* (1 vitória e 6 empates), a solução do problema aqui abordado (encontrar um conjunto mínimo de características que possa otimizar o treino de um agente jogador de Damas) deve ser atribuída em função do custo/benefício pretendido em termos do critério “tempo”, ou seja, se for importante otimizar o critério do

tempo de obtenção de um bom jogador que utilize um conjunto de características menor que o proposto por Mark Lynch, o melhor jogador da 8ª geração poderia ser apresentado como solução. Caso contrário, o melhor jogador da 24ª geração seria o escolhido.

4.5 Considerações Finais

Este capítulo mostrou, primeiramente, o processo de aprendizado das redes neurais através dos módulos 2, 3 e 4 do sistema *LS-Draughts* (estes módulos são baseados no *NeuroDraughts* de Mark Lynch). Tal processo utiliza uma rede MLP com conexão direta entre a camada de entrada e a camada de saída, mapeamento *NET-FEATUREMAP* de tabuleiro, estratégia de treino por *self-play* com clonagem, busca *minimax* com profundidade 4, método de Diferenças Temporais $TD(\lambda)$ e sem qualquer análise de jogos de especialistas. Além disso, o capítulo mostrou, também, todas as etapas da implementação do 1º módulo do sistema *LS-Draughts* (geração de características para o mapeamento *NET-FEATUREMAP*), bem como a sua relação com os módulos 2, 3 e 4 (processo de aprendizado) e os resultados obtidos com tal sistema proposto.

No capítulo a seguir serão apresentadas as conclusões do trabalho realizado, uma perspectiva de desenvolvimento e aplicações futuras.

CARACTERÍSTICAS	DESCRIÇÃO FUNCIONAL	BITS
F1: <i>PieceAdvantage</i>	Contagem de peças em vantagem para o jogador preto.	4
F2: <i>PieceDisadvantage</i>	Contagem de peças em desvantagem para o jogador preto.	4
F3: <i>PieceThreat</i>	Total de peças pretas que estão sob ameaça.	3
F4: <i>PieceTake</i>	Total de peças vermelhas que estão sob ameaça de peças pretas.	3
F5: <i>Advancement</i>	Total de peças pretas que estão na 5ª e 6ª linha do tabuleiro menos as peças que estão na 3ª e 4ª linha.	3
F6: <i>DoubleDiagonal</i>	Total de peças pretas que estão na diagonal dupla do tabuleiro.	4
F7: <i>Backrowbridge</i>	Se existe peças pretas nos quadrados 1 e 3 e se não existem rainhas vermelhas no tabuleiro.	1
F8: <i>Centrecontrol</i>	Total de peças pretas no centro do tabuleiro.	3
F9: <i>XCentrecontrol</i>	Total de quadrados no centro do tabuleiro onde tem peças vermelhas ou que elas podem mover.	3
F10: <i>TotalMobility</i>	Total de quadrados vazios para onde as peças vermelhas podem mover.	4
F11: <i>Exposure</i>	Total de peças pretas que são rodeadas por quadrados vazios em diagonal.	3
F12: <i>KingCentreControl</i>	Total de rainhas pretas no centro do tabuleiro.	3
F13: <i>DiagonalMoment</i>	Total de peças pretas em diagonal, distribuídas por pontuação específica.	3
F14: <i>Threat</i>	Total de quadrados vazios para o qual peças vermelhas podem mover e em fazendo isto, ameaçam a captura de peças pretas.	3
F15: <i>Taken</i>	Total de quadrados vazios para o qual peças pretas podem mover e em fazendo isto, ameaçam a captura de peças vermelhas.	3

Tabela 3: Relação das 15 características utilizadas na representação do indivíduo em *LS-Draughts*.

5 Conclusões

Foi apresentado o *LS-Draughts*, um sistema gerador de agentes jogadores de Damas em que os jogadores correspondem a redes neurais MLPs treinadas pelo método de aprendizagem por reforço $TD(\lambda)$ aliado com a busca *minimax*, com o mapeamento de tabuleiro *NET-FEATUREMAP* e com a técnica de treinamento por *self-play* com clonagem. As entradas das redes neurais correspondem aos estados do tabuleiro de Damas representados por conjuntos de características que os definem (mapeamento *NET-FEATUREMAP*). Tais conjuntos são indivíduos gerados automaticamente por um AG. A escolha desses conjuntos é fundamental para a obtenção de bons jogadores durante a etapa de treinamento das redes MLPs.

Conforme apresentado, a estratégia de aprendizagem das redes neurais é análoga à utilizada por Mark Lynch no sistema *NeuroDraughts*. Contudo, neste último, as características são fixas e selecionadas manualmente. O próprio Lynch deixou como sugestão de aprimoramento de seu jogador, a geração automática de características do jogo de Damas.

Esta foi, então, a motivação da proposta do *LS-Draughts*. Os torneios efetuados entre este último e o *NeuroDraughts* de Mark Lynch confirmaram o melhor desempenho do *LS-Draughts* e, conseqüentemente, ratificam a contribuição da inserção do módulo de geração automática de características no jogador original de Mark Lynch.

Assim sendo, a proposta de trabalho aqui apresentado e os resultados alcançados prometem abrir um novo caminho na busca pela geração e combinação de características de qualquer domínio que necessite de uma boa representação para os seus estados, otimizando, dessa forma, o processo de aprendizagem do agente. Com o AG é possível escolher um mínimo de atributos que melhor caracterizam o domínio e que sirva como um meio pelo qual a função de avaliação adquirirá novos conhecimentos, o que é uma questão fundamental para acelerar a aprendizagem e obter novos agentes com alto nível de desempenho.

Os resultados obtidos com o *LS-Draughts* mostram que é possível jogar Damas com alto nível de desempenho sem ter que utilizar grande quantidade de características no

mapeamento *NET-FEATUREMAP*. Dessa forma, o AG aparece como uma poderosa ferramenta para tentar resolver este tipo de problema, uma vez que auxilia na busca pela melhor combinação de características que possa otimizar o treino por Diferenças Temporais de um agente jogador de Damas que utiliza este conjunto de características para aprender a jogar.

5.1 Perspectiva de Trabalhos Futuros

Apesar do bom desempenho geral do *LS-Draughts*, foram constatadas algumas situações em que sua atuação não é satisfatória. Tais situações se referem ao caso de *loop* de final de jogo. Um exemplo desse *loop* foi apresentado na seção 4.6: a partir do tabuleiro da figura 29, resultante do 43º movimento do jogador do *NeuroDraughts*, o jogo chegou ao tabuleiro da figura 30, resultante da 44ª jogada de ambos os jogadores. Em seguida, na 45ª jogada, o jogo retornou para o tabuleiro da figura 29, iniciando, dessa forma, um *loop* infinito alternado entre estas 2 configurações de tabuleiro. Neste *loop*, o jogador do *LS-Draughts* tinha 3 rainhas e 1 peça simples sobre o tabuleiro enquanto o jogador do *NeuroDraughts* tinha apenas 1 rainha. Mesmo assim, o jogador do *LS-Draughts* não conseguiu pressionar o jogador do *NeuroDraughts* para vencer a partida.

Em virtude disso, propõe-se como perspectiva de trabalhos futuros, a realização de um estudo e análise para resolução do problema de *loop* de final de jogo de Damas sem ter que utilizar uma base supervisionada de jogadas finais, uma vez que o atual campeão de Damas *CHINOOK* (SCHAEFFER et al., 1996), para atacar tal problema, já utiliza uma base supervisionada de jogadas de fim de jogo para tabuleiros finais com até 8 peças sobre o mesmo. Esta base supervisionada está disponível no website <http://www.cs.ualberta.ca/~chinook/>.

Apesar de o jogador *CHINOOK* ser o atual campeão de Damas, ele não foi utilizado como ponto de partida do presente trabalho porque seu código não é aberto, diferentemente de Mark Lynch, o que inviabilizaria o cumprimento do propósito do presente projeto em tempo hábil, pois as técnicas envolvidas são extremamente complexas e variadas. Assim sendo, a proposta há pouco apresentada de trabalho futuro tem como objetivo tornar o *LS-Draughts* competitivo com relação ao *CHINOOK*.

Outras propostas de trabalhos futuros são:

- Implementar uma versão multi-agente para o jogo de Damas: cada agente seria responsável por especializar em fases diferentes do jogo de Damas, por exemplo, jogadas iniciais, jogadas de meio de jogo e jogadas finais. Para tanto, poderia

utilizar o arquivo de tabuleiro, uma vez que o mesmo permite treinar um agente partindo de várias configurações do tabuleiro de Damas;

- Estudo de um modelo matemático que permite criar novas características de Damas a partir de características já existentes e também utilizar combinações de posições de quadrados do tabuleiro de Damas para tal propósito. Esta proposta pode ser implementada utilizando Programação Genética, uma vez que cada característica representa uma função do domínio de Damas.

Referências

- BAXTER, A. T. J.; WEAVER, L. Learning to play chess using temporal differences. *Machine Learning*, v. 40, n. 3, p. 243–263, 2000.
- BAXTER, J.; TRIDGELL, A.; WEAVER, L. Experiments in parameter learning using temporal differences. *International Computer Chess Association Journal*, v. 21, n. 2, p. 84–99, 1998b.
- BAXTER, J.; TRIDGELL, A.; WEAVER, L. Knightcap: a chess program that learns by combining TD(λ) with game-tree search. In: *Proc. 15th International Conf. on Machine Learning*. [S.l.]: Morgan Kaufmann, San Francisco, CA, 1998a. p. 28–36.
- BEAL, D. F.; SMITH, M. C. Temporal coherence and prediction decay in td learning. In: *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*. [S.l.]: Morgan Kaufmann Publishers Inc., 1999. v. 1, p. 564–569.
- BEAL, D. F.; SMITH, M. C. Temporal difference learning for heuristic search and game playing. *Information Sciences*, v. 122, n. 1, p. 3–21, 2000.
- BELLMAN, R. *Dynamic Programming*. Princeton, New Jersey: Princeton University Press, 1957.
- BENTLEY, P. J. *Digital Biology: How Nature Is Transforming Our Technology and Our Lives*. New York: Simon & Schuster Inc, 2002.
- BORGA, M. Hierarchical reinforcement learning. In: *Proceedings of the International Conference on Artificial Neural Networks, Amsterdam, The Netherlands*. Springer Verlag, 1993. p. 13–16. Disponível em: <http://www.cvl.isy.liu.se/ScOut/Publications/Papers/ICANN93b.pdf>.
- CAMPOS, P.; LANGLOIS, T. Abalearn: A risk-sensitive approach to self-play learning in abalone. In: *Proceedings of the European Conference on Machine Learning*. [S.l.: s.n.], 2003. p. 35–46.
- CHAN, H. W. kit. *Application of Temporal Difference Learning and Supervised Learning in the Game of Go*. Tese (Doutorado) — Universidade de Hong Kong, China, 1996.
- DAHL, F. A. Honte, a go-playing program using neural nets. In: *Proceedings of the 16th International Conference on Machine Learning*. [S.l.: s.n.], 1999.
- DARWEN, P. J. Why co-evolution beats temporal difference learning at backgammon for a linear architecture, but not a non-linear architecture. In: *Proceedings of the 2001 Congress on Evolutionary Computation CEC2001*. COEX, World Trade Center, 159 Samseong-dong, Gangnam-gu, Seoul, Korea: IEEE Press, 2001. p. 1003–010.

- DARWIN, C. *On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life*. [s.n.], 1859. Disponível em: <<http://www.zoo.uib.no/classics/origin.html>>.
- DAYAN, P.; SEJNOWSKI, T. J. $Td(\lambda)$ converges with probability 1. *Machine Learning*, v. 14, n. 3, p. 295–301, 1994.
- EPSTEIN, S. *Learning to Play Expertly: A tutorial on Hoyle*. Huntington, NY: Machines That Learn to Play Games. Nova Science Publishers, 2001.
- ESHELMAN, L. J.; SCHAFFER, J. D. Real-coded genetic algorithms and interval-schemata. In: *Foundations of Genetic Algorithms, FOGA*. [S.l.: s.n.], 1992. p. 187–202.
- FAUSETT, L. V. *Fundamentals of Neural Networks: Architectures, Algorithms & Applications*. Englewood Cliffs, New Jersey: Prentice Hall, 1994.
- FOGEL, D. B.; CHELLAPILLA, K. Verifying anaconda's expert rating by competing against chinook: experiments in co-evolving a neural checkers player. *Neurocomputing*, v. 42, n. 1-4, p. 69–86, 2002.
- FOGEL, D. B. et al. A self-learning evolutionary chess program. *Proceedings of the IEEE*, v. 92, n. 12, p. 1947–1954, 2004.
- FÜRNKRANZ, J. Machine learning in games: A survey. In: FÜRNKRANZ, J.; KUBAT, M. (Ed.). *Machines that Learn to Play Games*. Huntington, NY: Nova Science Publishers, 2001. p. 11–59.
- GOLDBERG, D. E.; HOLLAND, J. H. Genetic algorithms and machine learning. *Machine Learning*, v. 3, p. 95–99, 1988.
- HAYKIN, S. *Redes Neurais: Princípios e Prática (2ª edição)*. Porto Alegre, RS: Bookman Editora, 2001.
- HERIK, H. J. van den; UITERWIJK, J. W. H. M.; RIJSWIJCK, J. van. Games solved: now and in the future. *Artificial Intelligence*, Elsevier Science Publishers Ltda, Essex, UK, v. 134, n. 1-2, p. 277–311, 2002.
- HOLLAND, J. H. *Adaptation in natural and artificial systems*. [S.l.]: University of Michigan Press, 1975.
- HOLLAND, J. H. *Adaptation in natural and artificial systems (2nd edition)*. Cambridge, MA, USA: MIT Press, 1992.
- HOPFIELD, J. J. Neural networks and physical systems with emergent collective computational abilities. *PNAS*, v. 79, n. 8, p. 2554–2558, April 1982.
- KOVACS, Z. L. *Redes Neurais Artificiais: Fundamentos e Aplicações (2ª edição)*. São Paulo: Collegium Cognitio, 1996.
- LEUSKI, A. *Learning of position evaluation in the game of othello*. [S.l.], January 1995. Disponível em: <<http://people.ict.usc.edu/~leuski/publications/index.html>>.

- LEVINSON, R.; WEBER, R. Pattern-level temporal difference learning, data fusion, and chess. In: *SPIE's 14th Annual Conference on Aerospace/Defense Sensing and Controls: Sensor Fusion: Architectures, Algorithms and Applications IV*. [S.l.: s.n.], 2000.
- LEVINSON, R.; WEBER, R. Chess neighborhoods, function combination, and reinforcement learning. In: *Revised Papers from the Second International Conference on Computers and Games*. London, UK: Springer-Verlag, 2002.
- LYNCH, M. *NeuroDraughts: An Application of Temporal Difference Learning to Draughts*. [S.l.], May 1997. Disponível em: <<http://iamlynch.com/nd.html>>.
- LYNCH, M.; GRIFFITH, N. Neurodraughts: the role of representation, search, training regime and architecture in a td draughts player. In: *Eighth Ireland Conference on Artificial Intelligence*. [s.n.], 1997. p. 64–72. Disponível em: <<http://iamlynch.com/nd.html>>.
- MCCULLOCH, W.; PITTS, W. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, v. 5, p. 115–133, 1943.
- MICHALEWICZ, Z. *Genetic algorithms + data structures = evolution programs (3rd edition)*. London, UK: Springer-Verlag, 1996.
- MICHALEWICZ, Z.; FOGEL, D. B. *How to solve it: modern heuristics*. New York, NY, USA: Springer-Verlag New York, Inc., 2004.
- MICHIE, D. Experiments on the mechanization of game-learning - part i. characterization of the model and its parameters. *The Computer Journal*, v. 6, n. 3, p. 232–236, 1963.
- NEUMANN, J. V.; MORGENSTERN, O. *Theory of games and economic behavior*. Princeton University Press, 1944. Disponível em: <http://en.wikipedia.org/wiki/Theory_of_Games_and_Economic_Behavior>.
- POLLACK, J. B.; BLAIR, A. D. Co-evolution in the successful learning of backgammon strategy. *Machine Learning*, v. 32, n. 1, p. 225–240, 1998.
- RAE, A. *Evolutionary Programming and the Game of Checkers*. [S.l.], 2001. Disponível em: <<http://www.cs.cornell.edu/boom/2001sp/rae/index-1.htm>>.
- RIBEIRO, C. H. C.; MONTEIRO, S. T. Aprendizagem da navegação em robôs móveis a partir de mapas obtidos autonomamente. In: *IV Encontro Nacional de Inteligência Artificial (ENIA)*. [s.n.], 2003. Disponível em: <<http://www.lbd.dcc.ufmg.br/bdbcomp/servlet/Trabalho?id=2414>>.
- RICH, E.; KNIGHT, K. *Inteligência Artificial (2a edição)*. [S.l.]: Makron Books, 1992.
- RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Learning internal representations by error propagation. MIT Press, Cambridge, MA, USA, p. 318–362, 1986.
- RUSSELL, S.; NORVIG, P. *Inteligência Artificial - Uma Abordagem Moderna (2a edição)*. [S.l.]: Editora Campus, 2004.
- SAMUEL, A. L. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, v. 3, n. 3, p. 211–229, 1959.

- SAMUEL, A. L. Some studies in machine learning using the game of checkers ii - recent progress. *IBM Journal of Research and Development*, v. 11, n. 6, p. 601–617, 1967.
- SCARDUA, L. A.; CRUZ, J. J.; COSTA, A. H. Controle Ótimo de descarregadores de navios utilizando aprendizado por reforço. *Sba Controle & Automação*, v. 14, n. 4, p. 368–376, 2003.
- SCHAEFFER, J. *One Jump Ahead: Challenging Human Supremacy in Checkers*. 175 Fifth Avenue New York, NY, USA: Springer-Verlag New York Inc., 1997.
- SCHAEFFER, J. et al. Temporal difference learning applied to a high performance game-playing program. In: *In Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. [S.l.: s.n.], 2001. p. 529–534.
- SCHAEFFER, J. et al. Chinook: The world man-machine checkers champion. *AI Magazine*, v. 17, n. 1, p. 21–29, 1996.
- SCHRAUDOLPH, N. N.; DAYAN, P.; SEJNOWSKI, T. J. Learning to evaluate go positions via temporal difference methods. In: BABA, I.; JAIN (Ed.). *Computational Intelligence in Games Studies in Fuzziness and Soft Computing*. Springer Verlag, 2001. v. 62. Disponível em: <<http://users.rsise.anu.edu.au/nici/bib2html/index.html>>.
- SINGH, S.; BERTSEKAS, D. Reinforcement learning for dynamic channel allocation in cellular telephone systems. *NIPS 9*, MIT Press, 1996.
- SUTTON, R. S. *Temporal credit assignment in Reinforcement Learning*. Tese (Doutorado) — University of Massachusetts, Amherst, 1984.
- SUTTON, R. S. Learning to predict by the methods of temporal differences. *Machine Learning*, v. 3, n. 1, p. 9–44, 1988.
- SUTTON, R. S.; BARTO, A. G. *Reinforcement Learning: An Introduction*. Cambridge: MIT Press, 1998.
- SYWERDA, G. Uniform crossover in genetic algorithms. In: *Proceedings of the third international conference on Genetic algorithms*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1989. p. 2–9.
- TESAURO, G. Practical issues in temporal difference learning. In: MOODY, J. E.; HANSON, S. J.; LIPPMANN, R. P. (Ed.). *Advances in Neural Information Processing Systems*. [S.l.]: Morgan Kaufmann Publishers, Inc., 1992. v. 4, p. 259–266.
- TESAURO, G. Td-gammon, a self-teaching backgammon program, achieves master-level play. *Neural Computation*, v. 6, n. 2, p. 215–219, 1994.
- TESAURO, G. Temporal difference learning and td-gammon. *Communications of the ACM*, v. 38, n. 3, p. 19–23, 1995.
- THRUN, S. Learning to play the game of chess. In: *Advances in Neural Information Processing Systems 7*. [S.l.]: The MIT Press, 1995. p. 1069–1076.
- WALKER, M. A. An application of reinforcement learning to dialogue strategy selection in a spoken dialogue system for email. In: *Journal of Artificial Intelligence Research* 12. [S.l.: s.n.], 2000. p. 387–416.

- WIDROW, B.; HOFF, M. E. Adaptive switching circuits. *IRE WESCON Convention Record*, p. 96–104, 1960.
- WIERING, M. Multi-agent reinforcement learning for traffic light control. In: *Proc. 17th International Conf. on Machine Learning*. [S.l.]: Morgan Kaufmann, San Francisco, CA, 2000. p. 1151–1158.
- WILLIAMS, R. J. *Toward a Theory of Reinforcement-Learning Connectionist Systems*. [S.l.], 1988.
- XING, L.; PHAM, D. T. *Neural Networks for Identification, Prediction, and Control*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 1995.
- ZHANG, W.; DIETTERICH, T. G. High-performance job-shop scheduling with A time-delay TD(λ) network. In: TOURETZKY, D. S.; MOZER, M. C.; HASSELMO, M. E. (Ed.). *Advances in Neural Information Processing Systems*. [S.l.]: The MIT Press, 1996. v. 8, p. 1024–1030.

Anexo A

Partida de Damas entre o melhor jogador do *LS-Draughts* e o melhor jogador do *NeuroDraughts*

Este anexo contém uma sequência completa de movimentos de uma partida de Damas disputada entre o melhor indivíduo da 24ª geração do *LS-Draughts* e o melhor jogador de Damas de Mark Lynch referente ao seu sistema *NeuroDraughts*. A notação de cada movimento é dado no seguinte formato: “*a-b*”, onde *a* é a posição de origem da peça no tabuleiro de Damas antes de executar o movimento e *b* é a posição de destino para onde esta peça será movimentada (por exemplo, o movimento “9-14” representa que uma peça que estava na posição “9” no tabuleiro de Damas foi movimentada para a posição “14”).

Jogador Preto (<i>LS-Draughts</i>)	Jogador Vermelho (<i>NeuroDraughts</i>)
1. 9-13	1. 24-20
2. 6-9	2. 23-18
3. 1-6	3. 22-17
4. 13-22	4. 26-17
5. 11-16	5. 20-11
6. 8-15-22	6. 25-18
7. 9-14	7. 18-9
8. 6-13-22	8. 27-24

Após 5 movimentos iniciais, o jogador preto começa a atacar o jogador vermelho, tomando 2 peças em diagonal no 6º movimento. A figura 33 mostra o estado do tabuleiro da partida após a execução do 6º movimento do jogador vermelho.

No 8º movimento o jogador preto toma 2 peças em diagonal do jogador vermelho e aumenta sua vantagem sobre o mesmo (veja figura 34).

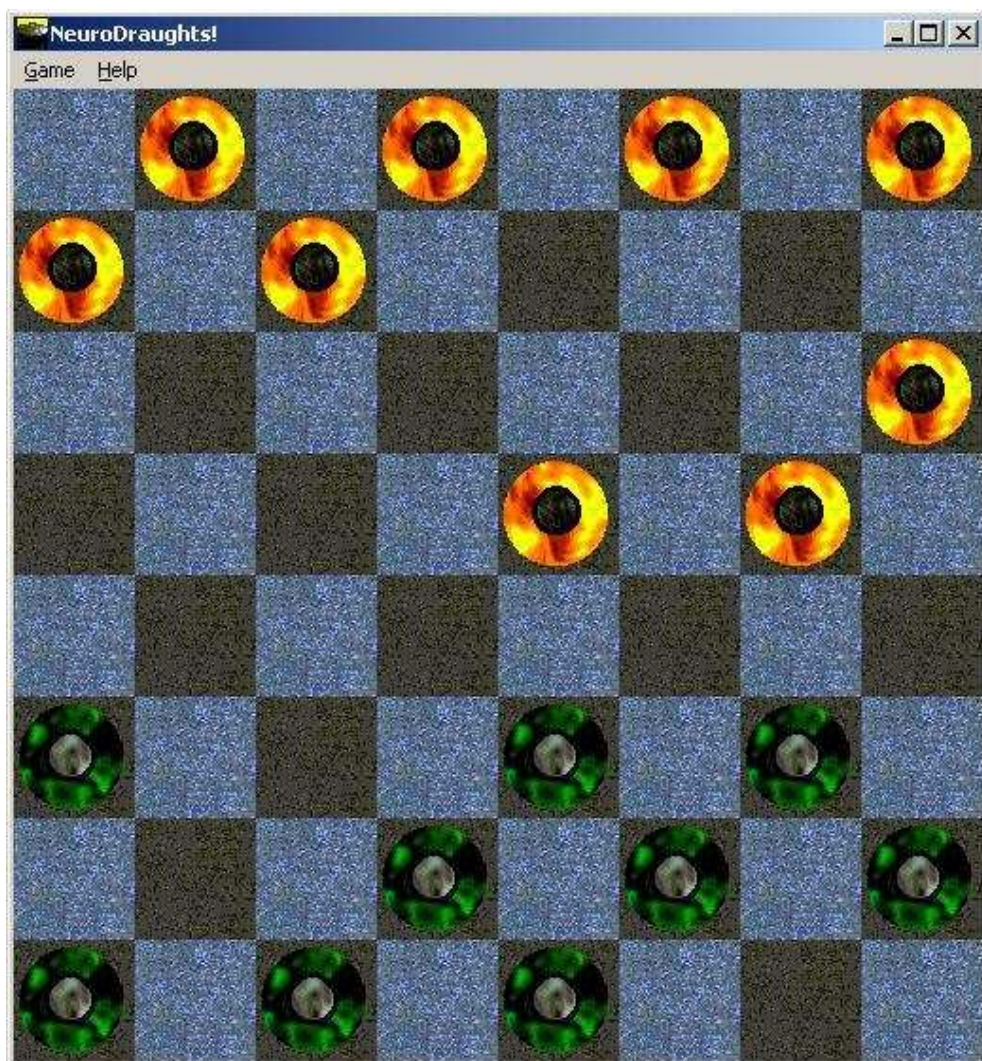


Figura 33: Posição do tabuleiro depois do 6º movimento do jogador vermelho.

9. 10-14
10. 5-9
11. 14-18
12. 9-14
13. 7-14

9. 24-19
10. 31-27
11. 21-17
12. 17-10
13. 19-15

Após o 13º movimento do jogador preto, o mesmo começa a fortalecer o centro do tabuleiro a fim de cercar o jogador vermelho mais adiante (veja figura 35).

14. 2-6
15. 3-8

14. 28-24
15. 30-25

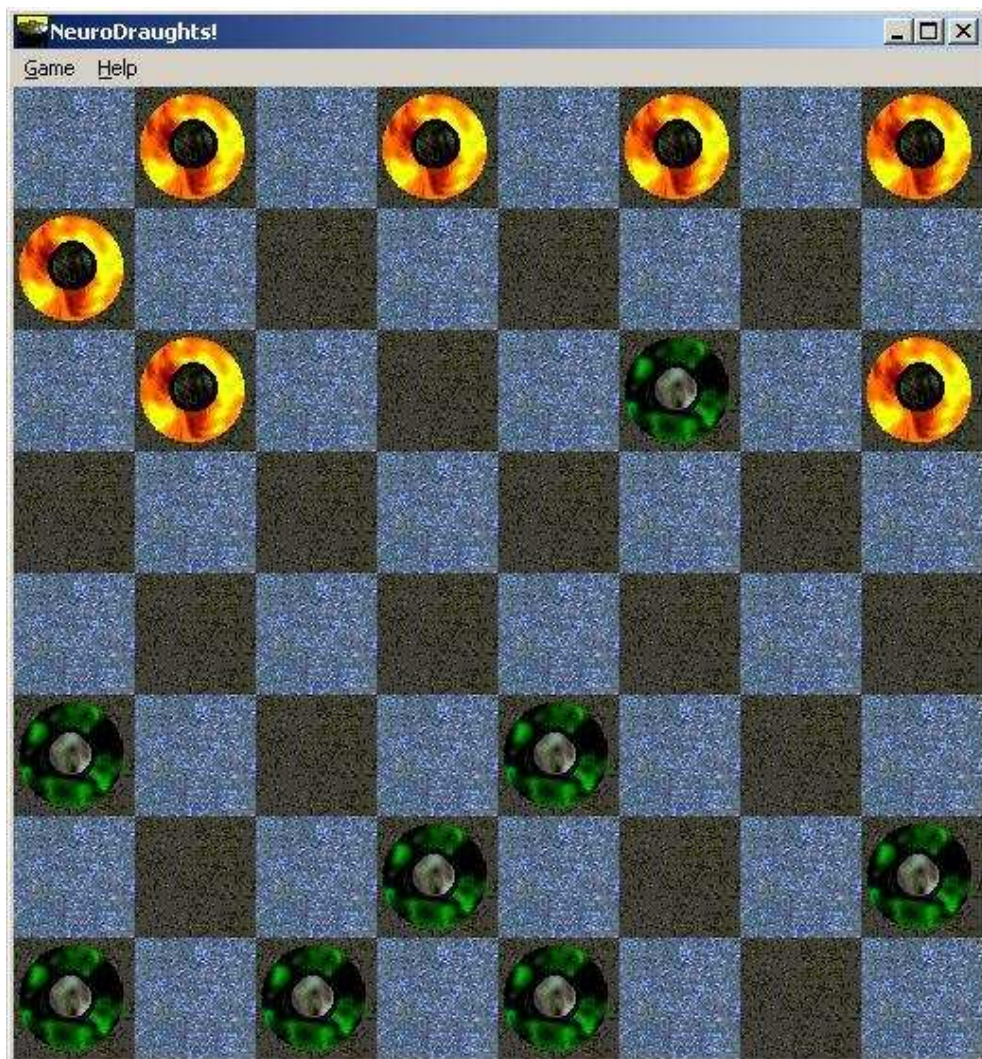


Figura 34: Posição do tabuleiro de Damas depois do 8º movimento do jogador vermelho.

16. 22-26
17. 26-30
18. 18-25
19. 30-26
20. 26-17
21. 12-19

16. 32-28
17. 25-22
18. 29-22
19. 24-19
20. 19-16
21. 27-24

No 17º movimento o jogador preto forma sua 1ª rainha e começa a cercar o jogador vermelho pelo centro do tabuleiro. A figura 36 mostra a configuração do tabuleiro após o 21º movimento do jogador vermelho. Veja que a vantagem de peças do jogador preto é bem maior que a do jogador vermelho.

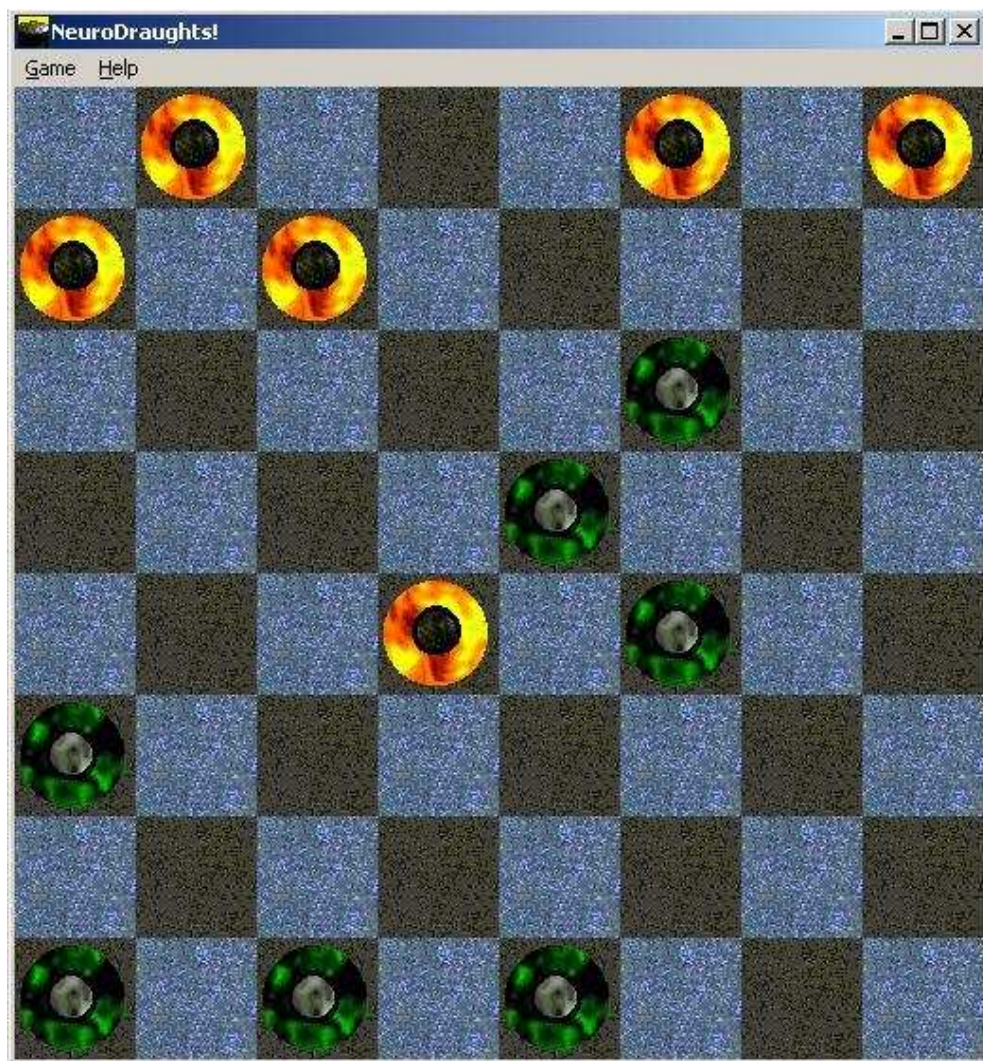


Figura 35: Posição do tabuleiro de Damas depois do 13º movimento do jogador vermelho.

22. 17-22
 23. 8-15
 24. 19-23
 25. 15-18
 26. 6-10
 27. 10-15
 28. 14-17
 29. 15-19
 30. 4-8

22. 15-11
 23. 24-20
 24. 20-16
 25. 16-11
 26. 11-7
 27. 7-2
 28. 2-6
 29. 6-9
 30. 9-6



Figura 36: Posição do tabuleiro de Damas depois do 21º movimento do jogador vermelho.

Após o 30º movimento do jogador preto, o mesmo se fecha no meio do tabuleiro a fim de preparar um ataque final sobre o jogador vermelho (veja figura 37).

31. 17-21
32. 23-26
33. 19-23
34. 21-25
35. 25-30
36. 8-12
37. 26-31

31. 6-10
32. 10-14
33. 28-24
34. 14-10
35. 10-7
36. 7-11
37. 11-7



Figura 37: Posição do tabuleiro de Damas depois do 30º movimento do jogador vermelho.

A figura 38 mostra a vantagem do jogador preto sobre o jogador vermelho. Note que com 3 rainhas, o jogador preto prepara para vencer o jogo.

38. 31-27

39. 23-26

40. 22-17

41. 12-19

42. 17-22

38. 24-20

39. 7-10

40. 20-16

41. 10-15

42. 15-24-31

Após o 42º movimento do jogador vermelho, que tomou 2 peças pretas em diagonal, o jogador preto finalmente cerca o jogador vermelho para vencê-lo com apenas 2



Figura 38: Posição do tabuleiro de Damas depois do 37º movimento do jogador vermelho.

movimentos seguintes (veja figura 39).

43. 18-23

43. 31-27

44. 23-32

44. -

Após o 44º movimento do jogador preto, a partida é finalizada com vitória do jogador preto. Note que o jogador preto vence a partida com uma boa vantagem numérica de peças: 3 rainhas e uma peça simples (veja figura 40).



Figura 39: Posição do tabuleiro de Damas depois do 42º movimento do jogador vermelho.



Figura 40: Posição do tabuleiro de Damas depois do 44º movimento do jogador preto (vitória do jogador do *LS-Draughts*).