Rodrigo Lopes Setti de Arruda

Uma Arquitetura Híbrida Aplicada em Problemas de Aprendizagem por Reforço

Dissertação de Mestrado apresentada à Faculdade de Engenharia Elétrica e de Computação como parte dos requisitos para obtenção do título de Mestre em Engenharia Elétrica. Área de concentração: Engenharia de Computação.

Orientador: Fernando José Von Zuben

Campinas, SP
2012

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

Ar69a Arruda, Rodrigo Lopes Setti de
Uma arquitetura híbrida aplicada em problemas de aprendizagem por reforço / Rodrigo Lopes Setti de Arruda. – Campinas, SP: [s.n.], 2012.

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

1. Inteligência artificial. 2. Aprendizado do computador. 3. Teoria dos automatos. 4. Robótica. 5. Robôs móveis. I. Von Zuben, Fernando José. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Uma arquitetura híbrida aplicada em problemas de aprendizagem por reforço

Título em Inglês:	A hybrid architecture to address reinforcement learning problems
Palavras-chave em Inglês:	Artificial intelligence, Machine learning, Theory of automata, Robotics, Mobile robots
Área de concentração:	Engenharia de Computação
Titulação:	Mestre em Engenharia Elétrica
Banca Examinadora:	Ricardo Ribeiro Gudwin, Roseli Aparecida Francelin Romero
Data da defesa:	07-02-2012
Programa de Pós Graduação:	Engenharia Elétrica

COMISSÃO JULGADORA - TESE DE MESTRADO

Candidato: Rodrigo Lopes Setti de Arruda

Data da Defesa: 7 de fevereiro de 2012

Título da Tese: "Uma Arquitetura Híbrida Aplicada em Problemas de Aprendizagem por Reforço"

Prof. Dr. Fernando José Von Zuben (Presidente): Fernando José Von Zuben

Profa. Dra. Roseli Aparecida Francelin Romero: Roseli Dsp. Francelin Romero

Prof. Dr. Ricardo Ribeiro Gudwin: Ricardo R Gudwin

Resumo

Com o uso de sistemas cognitivos em uma crescente gama de aplicações, criou-se uma grande expectativa e elevada demanda por máquinas cada vez mais autônomas, inteligentes e criativas na solução de problemas reais. Em diversos casos, os desafios demandam capacidade de aprendizado e adaptação. Este trabalho lida com conceitos de aprendizagem por reforço e discorre sobre as principais abordagens de solução e variações de problemas. Em seguida, constrói uma proposta híbrida incorporando outras ideias em aprendizagem de máquina, validando-a com experimentos simulados. Os experimentos permitem apontar as principais vantagens da metodologia proposta, a qual está fundamentada em sua capacidade de lidar com cenários de espaços contínuos e, também, de aprender uma política ótima enquanto segue outra, exploratória. A arquitetura proposta é híbrida, baseada em uma rede neural perceptron multi-camadas acoplada a um aproximador de funções denominado *wire-fitting*. Esta arquitetura é coordenada por um algoritmo adaptativo e dinâmico que une conceitos de programação dinâmica, análise de Monte Carlo, aprendizado por diferença temporal e elegibilidade. O modelo proposto é utilizado para resolver problemas de controle ótimo, por meio de aprendizagem por reforço, em cenários com variáveis contínuas e desenvolvimento não-linear. Duas instâncias diferentes de problemas de controle, reconhecidas na literatura pertinente, são apresentadas e testadas com a mesma arquitetura.

Palavras-chave: Inteligência artificial, Aprendizado de máquina, Aprendizado por reforço, Controle Ótimo, Redes Neurais

Abstract

With the evergrowing use of cognitive systems in various applications, it has been created a high expectation and a large demand for machines more and more autonomous, intelligent and creative in real world problem solving. In several cases, the challenges ask for high adaptive and learning capability. This work deals with the concepts of reinforcement learning, and reasons on the main solution approaches and problem variations. Subsequently, it builds a hybrid proposal incorporating other machine learning ideas, so that the proposal is validated with simulated experiments. The experiments allow to point out the main advantages of the proposed methodology, founded on its capability to handle continuous space environments, and also to learn an optimal policy while following an exploratory policy. The proposed architecture is hybrid in the sense that it is based on a multi-layer perceptron neural network coupled with a function approximator called wire-fitting. The referred architecture is coordinated by a dynamic and adaptive algorithm which merges concepts from dynamic programming, Monte Carlo analysis, temporal difference learning, and eligibility. The proposed model is used to solve optimal control problems, by means of reinforcement learning, in scenarios endowed with continuous variables and nonlinear development. Two different instances of control problems, well discussed in the pertinent literature, are presented and tested with the same architecture.

Keywords: Artificial Intelligence, Machine Learning, Reinforcement Learning, Optimal Control, Neural Networks

Agradecimentos

Ao meu orientador Fernando José Von Zuben, sou grato pela orientação.

Aos demais colegas de pós-graduação, pelas críticas e sugestões.

À minha família, pelo apoio durante esta jornada.

À CAPES, pelo apoio financeiro.

À minha família, pela confiança e apoio

Sumário

Lista de Algoritmos

Lista de Figuras

Capítulo 1

Introdução

1.1 Aprendizado de máquina

Aprendizado de máquina é um dos grandes desafios no campo da inteligência artificial, desde que foi instaurado. De fato, o homem tem sonhado com máquinas inteligentes desde a antiguidade [12].

A questão se é ou não possível produzir consciência em uma máquina, apesar de ser absolutamente relevante e de profundo interesse filosófico, está fora do contexto da maioria dos textos que abordam aprendizado de máquina. Mesmo que a resposta para esta questão seja negativa, ou seja, que a consciência seja um fenômeno exclusivamente humano, no contexto de inteligência artificial, as máquinas devem necessariamente representar conhecimento e também aprender com as suas ações e outras ações observáveis no ambiente. A ciência ainda está distante de sustentar a concepção de uma máquina que exiba graus elevados de autonomia e inteligência, e mesmo animais considerados cognitivamente bastante limitados, como um pernilongo, impõem desafios formidáveis à pesquisa de mecanismos artificiais de controle equivalentes.

Muito embora neste exemplo (do pernilongo) desconfia-se que o agente já tenha *a priori* todos os conhecimentos para a sua operação de controle, não se fazendo necessário "aprender", é fato que muitos animais possuem a faculdade de aprendizado, e tal faculdade é extremamente desejável em sistemas artificiais autônomos, particularmente para resolver problemas não bem definidos em ambientes incertos.

No início do século XX, acreditava-se que as máquinas, com seus algoritmos desempenhando o papel de automatizar sistemas formais, seriam capazes de atingir a idealizada inteligência artificial. Mais tarde, porém, observou-se que, mesmo sendo executados em processadores muito rápidos e dispondo de grande capacidade de memória, esses algoritmos não eram capazes de produzir comportamentos inteligentes e criativos.

Estas descobertas não invalidaram a possibilidade de algoritmos expressarem comportamento inteligente quanto executados. Apenas ficou afastada a possibilidade dos algoritmos seguirem os mesmos paradigmas expressados pela mente humana. De fato, os artefatos de inteligência artificial mais bem sucedidos do nosso tempo são mais suportados pela "força bruta" da máquina que pela reprodução de processos e teorias vinculados à mente humana [4].

1.2 Aprendizagem por Reforço

No mundo em que vivemos, mesmo um sistema muito competente para modelar a realidade seria nada além de um mecanismo reativo se não tivesse objetivos.

Apesar de muitos objetivos serem pré-programados no cérebro, outros são agregados de conhecimentos adquiridos pela experiência. Por exemplo, o homem tem o objetivo de plantar a semente, porque sabe que isso irá gerar uma planta que lhe servirá de alimento. Muito provavelmente a causa/efeito de semear e nascer uma planta não seja pré-programada, mas aprendemos por experiência.

Existem dúvidas a respeito do mecanismo com que as experiências são processadas, mas pode-se realçar que os agentes, pela sua vivência, adquirem conhecimento a respeito da dinâmica na qual o ambiente trabalha e também na consequência imediata e tardia de seus atos. Consequências essas que podem ser positivas ou negativas, e o aprendizado deve levar em conta essa graduação.

A problemática de aprendizagem por reforço (AR) é o desafio de aprender com a experiência, muito embora sem um professor ditando exemplos de ações para cada situação. O único retorno de um ato é uma recompensa imediata. O objetivo final, portanto, é maximizar a soma das recompensas ao longo do tempo [14], Cap. 3.

A história da AR é uma história de convergência de duas linhas de pesquisa.

Uma das linhas é a psicologia comportamental, a qual ganhou notoriedade no início do século XX com expoentes como Edward Thorndike. Em suas palavras, expressando sucintamente o que entendia por "condicionamento operante" (em tradução livre):

> Das muitas respostas produzidas para uma mesma situação, aquelas que forem acompanhadas ou seguidas rapidamente de satisfação para o animal serão, invariavelmente, conectadas mais fortemente à situação, de modo que, quando esta for recorrente, a resposta terá maior probabilidade de ocorrer. Aquelas respostas seguidas de desconforto ao animal terão sua conexão com a situação enfraquecidas, de modo que, quando a situação for recorrente, a resposta terá menor probabilidade de ocorrência. Quanto maior a satisfação ou desconforto, maior o reforço ou enfraquecimento da conexão [15].

Thorndike denominou a essa teoria de "Lei do Efeito".

Uma segunda linha de pesquisa, concorrentemente, se desenvolvia na área de controle automático. O termo "controle ótimo" surgiu no final dos anos de 1950 para descrever o problema de modelar controladores para minimizar uma medida de um comportamento de um sistema dinâmico ao longo do tempo. Uma das abordagens a este problema foi desenvolvida em meados dos anos de 1950 por Richard Bellman e colaboradores ao estender uma teoria do século XIX estabelecida por Hamilton e Jacobi. Esta solução faz uso do conceito de estado de um sistema dinâmico e de uma função valor, ou "função de retorno ótimo", para definir uma equação, hoje conhecida como equação de Bellman [9].

A classe de métodos para a solução de controle ótimo veio a se estabelecer como programação dinâmica (PD). Bellman também introduziu uma versão discreta estocástica do problema de controle ótimo, conhecida como processo de decisão de Markov (PDM) [11].

Foi com a pesquisa de Barto e Sutton [13] nos anos de 1980, inspirados na psicologia comportamental e na disciplina de controle ótimo, que se desenvolveram uma série de métodos e algoritmos capazes de generalizar os problemas de programação dinâmica e progredir no sentido de reproduzir cenários mais comuns no mundo real.

1.3 Representação de Conhecimento

Os agentes artificiais, quando perante problemas desafiadores, devem exibir comportamentos mais avançados do que simples mecanismos reativos. Eles devem efetivamente manter um estado interno que tenha significado, ou seja, um meio de representar internamente um homomorfismo com o mundo real, seja através de processamento simbólico, arquitetura conexionista, híbridos entre estas abordagens, ou outras iniciativas.

É claro que um entendimento preciso destes mecanismos deve ser capaz de encontrar um nível de abstração que faça sentido. Apesar de que, em última instância, circuitos são apenas uma coleção infindável de portas lógicas - assim como o cérebro é uma coleção de neurônios - entender processos complexos em nível de portas lógicas é tão inadequado quanto entender a cognição em nível de ativação de sinapses. É preciso abstrair para camadas "superiores".

Tendo em vista estes pontos, é importante deixar bem claro onde situa-se esta pesquisa. Em primeiro lugar, ainda não está bem estabelecida qual é exatamente a forma mais inteligível de representação de conhecimento do cérebro. Em todo caso, se uma máquina for capaz de apresentar comportamento inteligente, então, para os propósitos que se busca, não é relevante discutir se o mecanismo de representação interna é ou não similar ao natural. Neste projeto, o "conhecimento", por assim dizer, é alocado em pesos sinápticos de uma rede neural (RN).

No tocante ao ponto elucidado no segundo parágrafo desta seção, não é preocupação deste texto definir com precisão quais os símbolos ou significados armazenados em forma de pesos sinápticos da RN. Existem diversas interpretações vinculadas aos vários níveis de redução que se queira. Certamente o que se representa internamente é homomórfico com os elementos do mundo real (ou problema simulado) apenas indiretamente e imprecisamente. O que se busca dar foco, todavia, são justamente a resultados e relacionamentos da máquina com o mundo real, em detrimento dos resultados e relacionamentos da mente artificial da máquina consigo mesma.

1.4 Proposta e Apresentação

A evolução biológica tem produzido seres capazes de adaptação e aprendizado em locais extremamente incertos e dinâmicos. Em diversos ambientes da indústria, pesquisa e outras atividades humanas, encontram-se ambientes como estes, para citar alguns: exploração submarina, exploração espacial, cenários de conflito militar, hospitais e lares com indivíduos debilitados. De fato, a pesquisa em engenharia tem trazido em poucos anos uma variedade de inovações tecnológicas na área de robótica móvel, controle automático, sensores a atuadores, os quais inspiram um futuro em que mentes artificiais sejam capazes de coordenar estes corpos mecânicos com a mesma maestria e propriedade com que os animais fazem há milhões de anos.

O que se busca hoje é oferecer soluções reais de robótica, controladores e outros sistemas inteligentes para os cenários mencionados e inúmeros outros que se vislumbrarem no futuro. Há uma necessidade por mentes artificiais inteligentes, sendo que há manifestações claras deste ideal nas expressões artísticas e na literatura, ao longo dos séculos.

A AR aborda essa questão fazendo uma analogia com a cognição biológica. O reforço, ou seja, a recompensa, vai ser positiva ou negativa, e deve necessariamente ser algo exterior ao próprio sistema cognitivo (neste caso, o cérebro), pois o organismo deve saber indicar o que é bom e o que é ruim

para si, pelo menos em um nível básico (*e.g.* fome, sede, dor, prazer). Apesar disso, o mesmo organismo pode construir modelos mentais para planejar o máximo de recompensa a longo prazo, e, adicionalmente, construir símbolos de recompensa mais abstratos, além das recompensas básicas imediatas, para lidar com esses planejamentos.

O algoritmo proposto neste trabalho, iterativamente constrói de forma direta não um modelo dinâmico de mundo, mas um modelo de "valor". O significado disso é que o agente é um sistema auto-regulador cujo objetivo único é seguir o gradiente de valor (alçar estados com valores cada vez mais altos). O desafio, portanto, se instala principalmente na construção deste senso de valor. Naturalmente, o valor de um estado não deve ser a recompensa imediata, pois é simples encontrar problemas em que estados com boa recompensa imediata não são necessariamente bons candidatos para otimizar a recompensa a longo prazo (*i.e.* para conseguir uma boa recompensa, muitas vezes, é preciso passar por um caminho de más recompensas).

Conforme será elucidado, a PD é capaz de operar com os conceitos de recompensa e valor, e, depois, em modelos mais realistas e probabilísticos, serão propostos algoritmos mais adequados para problemas em que a dinâmica não é conhecida, ajudando o agente a construir sua noção de valor, mesmo sem conhecer as "regras do jogo".

O objetivo deste trabalho, portanto, é tratar especificamente de problemas de AR que podem ser modelados como um PDM. Neste contexto, construir-se-á um algoritmo *on-line* (ou seja, que aprende enquanto interage) que pode ser aplicado em problemas de AR, nos moldes definidos, com pouca dimensionalidade. Este algoritmo é inovador no sentido de que agrega elegibilidade com aprendizado em ambientes contínuos de uma forma inédita, ao combinar as ideias de Sutton & Barto [14], Doya [5] e Gaskett *et. al.* [6].

No capítulo 2 apresenta-se o problema de AR formalmente como um PDM. Em seguida, no capítulo 3, mostram-se alguns métodos clássicos de solução encontrados na literatura, para então, no capítulo 4, introduzir alguns conceitos que serão utilizados para a consolidação do modelo que será proposto, como elegibilidade em espaços contínuos, redes neurais artificiais e o aproximador de funções *Wire-Fitting* (WF). No final deste capítulo, o leitor terá uma descrição completa do algoritmo que representa a principal contribuição deste trabalho.

No capítulo 5, discorre-se sobre a aplicação das técnicas mencionadas junto a dois problemas reais de controle automático para então fechar, no capítulo 6, comentando a respeito dos principais achados, dificuldades encontradas e trabalhos futuros.

Capítulo 2

Problemática

2.1 Processos de Decisão de Markov (PDM)

2.1.1 Elementos de um Processo de Decisão de Markov

Antes de entrar mais formalmente na definição de um PDM, discute-se quais são os elementos componentes e como estes se relacionam [11]:

- **Agente**: O agente é a entidade que atua no ambiente e tem o poder de modificar o estado deste. Além disso, possui um estado interno e consegue ler o estado do ambiente, localmente ou globalmente.
- **Política do Agente**: É o comportamento do agente no sistema. Dado um estado, a política irá definir a ação a ser tomada.
- **Estados**: Um PDM contém um conjunto de estados, os estados podem ser conhecidos totalmente pelo agente (totalmente observáveis) ou em parte (parcialmente observáveis).
- **Ações**: É o conjunto de decisões do agente. Para um dado estado, há uma probabilidade de transição para outro estado, dada uma ação executada pelo agente.
- **Recompensa**: A cada transição de estados há uma recompensa, que é um valor escalar a ser maximizado. A recompensa é o elemento que irá delinear o aprendizado da política.

A figura 2.1 mostra um exemplo de PDM simples, onde cada nó é um estado e cada aresta direcionada representa uma probabilidade de transição entre estados dada a ação descrita.

É interessante notar que tais sistemas devem respeitar a propriedade de Markov: Os estados anteriores visitados pelo agente não influenciam nas probabilidade de transição, ou seja, toda informação a respeito do ambiente (inclusive seu histórico) devem estar em cada estado.

2.1.2 Definição Formal

Um PDM é uma tupla de quatro elementos: $(S, A, P(.,.), R(.,.))$, onde:

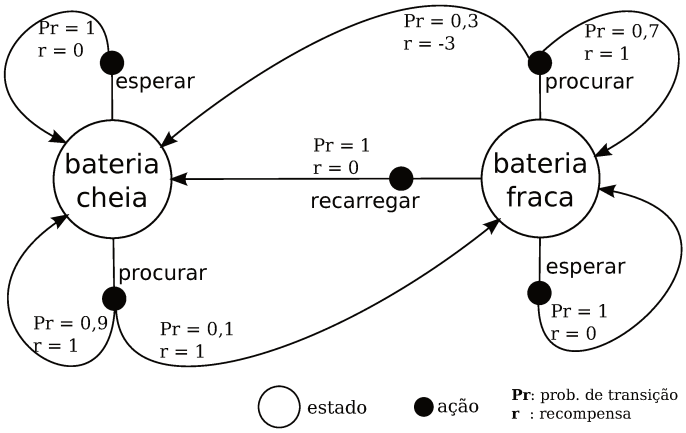

Fig. 2.1: Exemplo de um sistema de Markov de um robô lixeiro. Definem-se dois estados: "bateria cheia" e "bateria fraca"; e três ações: "esperar", "procurar" e "recarregar".

- S é um conjunto de estados.
- A é um conjunto de ações.
- $P_a(s, s')$ é a probabilidade de transição de um estado s ao estado s', sob uma ação a [1].
- $R_a(s, s')$ é a recompensa imediata caso seja consolidada a transição de um estado s ao estado s', sob uma ação a [2].

A teoria de Markov não pressupõe que o conjunto de estados tenha cardinalidade finita, muito embora alguns métodos de solução tenham esse requerimento.

2.1.3 Objetivo

O problema de um PDM é encontrar uma política de agente sendo uma função π que define a ação $\pi(s)$ que o agente irá tomar em um dado estado s. O objetivo é encontrar uma política que maximize alguma função cumulativa das recompensas (FCR), tipicamente a soma descontada das recompensas em um horizonte de tempo infinito:

$$FCR = \sum_{t=0}^{\infty} \gamma^t R_{a_t}(s_t, s_{t+1}) \tag{2.1}$$

onde a escolha de a_t é de acordo com a política π, ou seja, $a_t = \pi(s_t)$.

A constante γ é um fator de desconto que satisfaz $0 \leq \gamma \leq 1$. Esse desconto evita que o somatório divirja, dando pesos menores para recompensas mais tardias e pesos maiores para recompensas mais imediatas.

[1] Também se adota a notação $P^a_{ss'}$.
[2] Também se adota a notação $R^a_{ss'}$.

2.1.4 Variações de PDM

Os PDMs se dividem em quatro grupos, que são basicamente a combinação de duas características - como descreve a tabela 2.1: controlabilidade e visibilidade.

A controlabilidade refere-se à capacidade do agente efetivamente controlar o sistema. Um sistema é dito controlável se a mudança de estados depende apenas das ações do agente. Caso contrário, podem ocorrem mudanças de estado fora do controle do agente, tornando o ambiente parcialmente controlável. Outro atributo de sistemas controláveis é a possibilidade de se atingir qualquer estado do ambiente a partir de qualquer estado inicial, apenas pelas ações do agente.

A visibilidade refere-se à condição do agente de ver o estado completo do ambiente. Se for o caso, o ambiente é observável. Caso contrário, o ambiente é parcialmente observável e o agente terá acesso somente a parte do estado e algumas variáveis podem ser escondidas, resultando no fato de que mais de um estado do sistema pode ser interpretado como o mesmo estado para o agente.

Tab. 2.1: Variações de PDM

	Observável	Parcialmente Observável
Controlável	PDM	PDMPO
Parcialmente Controlável	PDMPC	PDMPOPC

As quatro variações de PDM são:

- **PDM**: PDM observável e controlável.
- **PDMPO**: PDM parcialmente observável.
- **PDMPC**: PDM parcialmente controlável.
- **PDMPOPC**: PDM parcialmente observável e parcialmente controlável.

2.2 Problema de Aprendizagem por Reforço em PDM

O problema de AR surge quando não se conhecem as probabilidades de transição e as recompensas esperadas de um PDM [14], Cap. 3. Desta forma, para construirmos uma política ótima para o agente (que maximize a soma descontada das recompensas) é necessário explorar o sistema e aprender com a experiência, para então determinar as melhores ações para cada estado.

Considere um robô em um ambiente desconhecido coletando objetos. Apesar de conhecer suas ações e ter sensores capazes de ler o estado do ambiente, o robô de início não sabe qual é o resultado de suas ações no ambiente e quais recompensas receberá. Por exemplo, se entrar em uma sala, ele não sabe o que irá encontrar, possivelmente com uma recompensa positiva, se encontrar o objeto que procura, ou negativa, caso encontre uma armadilha.

Desta forma, o robô deverá explorar o ambiente, mas sempre levando em conta suas experiências para maximizar as recompensas.

A AR difere de aprendizado supervisionado por não haver exemplos corretos de estado e ação a serem tomados, impedindo assim que ocorra correção explícita de ações sub-ótimas. Ao invés disso, o agente deve explorar o ambiente e aprender com sua experiência.

Surge então a questão de como gerenciar exploração e explotação (a ser melhor discutida nos capítulos seguintes). Um agente, para aprender, deve explorar. Mas, em contrapartida, para obter bom desempenho, deve tomar ações gulosas. Desta forma, os algoritmos em geral têm estratégias de balanço entre esses dois comportamentos.

No capítulo seguinte, são apresentados três métodos já propostos para a solução de PDMs. A técnica de PD, muito embora garanta a solução ótima, assume o conhecimento das probabilidades de transição e recompensas, de modo que não é aplicável ao problema de AR. Os dois métodos restantes, Método de Monte Carlo e Diferença Temporal, são capazes de resolver o problema de AR, mas de modos distintos, como será descrito.

Capítulo 3

Métodos de Solução de PDMs

Neste capítulo, são introduzidos quatro abordagens consideradas clássicas para tratar do problema de aprendizagem por reforço (AR), quando modelado em um processo de decisão de Markov (PDM). O conteúdo deste capítulo representa uma compilação de conceitos e métodos apresentados em Sutton e Barto [14].

O primeiro dos métodos, programação dinâmica (PD), foi estabelecido por Richard Bellman e colaboradores como uma maneira discreta e estocástica de resolver problemas de controle ótimo, através do uso de sua famosa equação recursiva.

O segundo método, análise de Monte Carlo (MC), apresenta o problema de maneira diferente, não supondo conhecimento completo do sistema. Logo, o agente deverá aprender com exploração e buscar uma convergência para um controle ótimo, porém, não garantindo absolutamente tal resultado. O caráter aleatório e de amostragem dá nome ao método.

O terceiro método, diferença temporal (DT), permite um aprendizado mais rápido ao calcular uma propagação da função valor para os estados vizinhos, permitindo assim que o agente infira o valor de um estado baseado no potencial de se passar para outro estado de valor já conhecido. A DT realiza uma espécie de "difusão" de valor entre os estados.

O quarto e último método, traçado de elegibilidade, é, na verdade, um agregado dos últimos dois, situado na contribuição de Barto e Sutton. Incorpora uma maneira do agente "lembrar" o caminho de estados até um retorno, resultando em um aprendizado ainda mais efetivo, pois ele é capaz de atribuir valor a estados imediatamente anteriores a uma recompensa ou punição, sem a necessidade de passar por estes estados novamente (como na DT), contribuindo para uma difusão de valor mais eficiente.

3.1 Programação Dinâmica

O termo PD refere-se a uma coleção de algoritmos que podem ser utilizados para computar políticas ótimas, dado um modelo completo do ambiente como um PDM.

A PD clássica tem utilidade limitada para problemas de AR, dado que assume um conhecimento total do sistema e também demanda grande quantidade de recursos computacionais. Independente de sua relevância prática no contexto deste trabalho, a PD é de grande validade teórica, pois ela provê a fundamentação para o entendimento dos métodos apresentados no restante desta dissertação.

De fato, todos esses métodos podem ser vistos como tentativas de atingir o mesmo efeito que a

PD, com menos computação e sem considerar onisciência a respeito do ambiente.

A idéia chave da PD é encontrar a função valor, conhecida como equação de Bellman [9]:

$$V^{\pi}(s) = \sum_{s'} \mathcal{P}^{a}_{ss'}[\mathcal{R}^{a}_{ss'} + \gamma V^{\pi}(s')]; a = \pi(s) \tag{3.1}$$

onde $V^{\pi}(s)$ é o valor do estado s, caso seja seguida a política π; $\pi(s)$ é a ação esperada, escolhida pela política π, no estado s; $\mathcal{P}^{a}_{ss'}$ é a probabilidade de transição de um estado s a s', quando efetuada a ação a, e $\mathcal{R}^{a}_{ss'}$ é a recompensa obtida quando transitando de s a s' pela ação a.

A função valor é a soma, descontada por γ, de todas as recompensas obtidas pelo agente a partir do estado s, em um horizonte infinito. O que se tem com essa equação é uma forma de calcular qual o valor de cada estado, dada uma política seguida pelo agente.

O que se deseja, contudo, é encontrar a melhor política. Um primeiro passo, é encontrar a função valor máxima, ou seja, a função valor caso seja seguida a melhor política possível. Isso é possível se a cada "iteração" recursiva da fórmula é escolhida a melhor ação com respeito ao estado em questão, ou seja, escolher a ação cuja fórmula tenha valor máximo. Isso é matematicamente representado com o operador $\max$.

A função valor é máxima (em outras palavras, a soma descontada das recompensas é máxima) se a política é ótima, ou seja, se as ações tomadas pela política levam a uma maximização do valor. Então, considera-se a função de valor máxima na forma:

$$V^{*}(s) = \max_{a} \sum_{s'} \mathcal{P}^{a}_{ss'}[\mathcal{R}^{a}_{ss'} + \gamma V^{*}(s')] \tag{3.2}$$

A transição da equação 3.1 para a equação 3.2 se dá na substituição do termo do somatório das probabilidades de ação de uma política arbitrária π pela escolha da ação a que maximiza o valor recursivamente calculado.

A função valor e a política são interdependentes, de modo que se pode iterativamente escolher as melhores ações para maximizar a função valor e, desta forma, reciprocamente melhorar a função valor para garantir melhores escolhas. Nas subseções a seguir, descrevem-se métodos iterativos para avaliar e melhorar a política e a função valor, sendo apresentados os algoritmos de PD derivados das equações. É importante ter sempre em mente que, como as equações são recursivas, os algoritmos trabaralhão de maneira iterativa e convergente, onde cada iteração é um nível a mais de recursão nas equações.

3.1.1 Avaliação de Política

Nesta subseção, considera-se como computar a função valor V^{π} para uma política arbitrária π.

O algoritmo 1 mostra um método iterativo clássico [14], Cap. 4, para avaliar o valor de V^{π}.

entrada: Política π a avaliar
saída : Função valor $V^{\pi}(s)$ da política
Inicialize $V(s) = 0, \forall s \in \mathcal{S}$
repita
  $\Delta \leftarrow 0$
  para todo $s \in \mathcal{S}$ **faça**
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V(s')]$
    $\Delta \leftarrow \max\{\Delta, |v - V(s)|\}$
  fim
até $\Delta < \epsilon$

Algoritmo 1: Avaliação de Política

A função valor $V^{\pi}(s)$ pode ser compreendida intuitivamente como uma tabela contendo uma lista de estados e seus valores. Esses valores da lista são inicializados com zero, pois não são conhecidos os valores dos estados *a priori*.

No corpo de repetição do algoritmo, é utilizada uma iteração da equação de Bellman (Eq. 3.1) para cada estado conhecido. Nesta versão, a equação não é recursivamente chamada, mas faz referência ao valor aproximado conhecido do próximo estado da política seguida ($V(s')$).

À medida que se calculam as iterações, os valores da função valor dos estados vão convergindo para uma estimativa do valor da política seguida. A condição de parada ocorre quando a maior variação de valor (*i.e.* o maior aprendizado) de todos os estados é menor que um parâmetro de tolerância mínima (ϵ), arbitrariamente pequeno e maior que zero.

3.1.2 Melhoria de Política

A principal razão de se avaliar uma política é determinar se há possibilidade de melhorá-la, escolhendo uma ação a não necessariamente ditada pela política, mas que maximize o valor do próximo estado.

O algoritmo de melhoria da política (Algoritmo 2) altera iterativamente uma política π inicial selecionando a ação que maximiza o valor do próximo estado.

entrada: Política π a melhorar
entrada: Função de valor $V^{\pi}(s)$
saída : Política π gulosa
para cada $s \in \mathcal{S}$ **faça**
  $\pi(s) \leftarrow \arg\max_a \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V(s')]$
fim

Algoritmo 2: Melhoria de Política

Este método de melhoria da política deve ser combinado com a avaliação de política para juntos e iterativamente convergirem para um máximo de função de valor e, consequentemente, de política ótima, como descrito na próxima seção.

3.1.3 Iteração de Política

Dado que uma política π é melhorada a partir da avaliação de política (V^{π}) resultando em π', pode-se então computar a função valor $V^{\pi'}$ e melhorá-la novamente, resultando em π''. Desta forma, constroi-se uma sequência de políticas e funções valores monotonicamente melhores:

$$\pi_0 \xrightarrow{A} V^{\pi_0} \xrightarrow{M} \pi_1 \xrightarrow{A} V^{\pi_1} \xrightarrow{M} \pi_2 \xrightarrow{A} \cdots \xrightarrow{M} \pi_* \xrightarrow{A} V^{\pi_*}$$

onde $\xrightarrow{A}$ denota avaliação de política e $\xrightarrow{M}$ denota melhoria de política. A cada política, é garantida uma melhoria estrita com relação à anterior (exceto a política já ótima). Devido ao fato de um PDM finito conter um número finito de políticas, este processo deverá convergir para uma política e função valor ótimas, em um número finito de passos.

Esse método de encontrar a política ótima, chamado de *Iteração de Política*, é melhor formalizado no Algoritmo 3.

saída : Política π e função valor V^{π} ótimas
Inicialize V, arbitrariamente
Inicialize π arbitrariamente
repita
  repita
    $\Delta \leftarrow 0$
    para todo $s \in \mathcal{S}$ **faça**
      $v \leftarrow V(s)$
      $V(s) \leftarrow \sum_a \pi(s,a) \sum'_s \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V(s')]$
      $\Delta \leftarrow \max\{\Delta, |v - V(s)|\}$
    fim
  até $\Delta < \epsilon$
  política-estável $\leftarrow$ Verdadeiro
  para cada $s \in \mathcal{S}$ **faça**
    $b \leftarrow \pi(s)$
    $\pi(s) \leftarrow \arg\max_a \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V(s')]$
    se $b \neq \pi(s)$ **então**
      política-estável $\leftarrow$ Falso
    fim
  fim
até *política-estável*

Algoritmo 3: Iteração de Política

3.2 Métodos de Monte Carlo (MC)

Nesta seção, considera-se o primeiro método de aprendizagem de um sistema cujo modelo é oculto. No método de MC, não se supõe o conhecimento do ambiente. Este método requer apenas a experiência através de amostras (*i.e.* exemplos) de estados, ações e recompensas de interações com o ambiente [14], Cap. 5.

Isso permite a criação de modelos para aprendizado simulado de forma mais simples, pois este precisa apenas fornecer exemplos e não uma lista completa de probabilidades de transição - o que em alguns casos torna a tarefa muito mais simples.

O método de MC é uma maneira de resolver o problema de aprendizagem por reforço baseado na média das amostras obtidas pela experiência. Desta forma, para garantir um valor bem definido, o aprendizado deve ocorrer em tarefas episódicas - ou seja - considera-se que a experiência é dividida em episódios e, ao final de cada episódio, ocorre o aprendizado.

Episódio, neste texto, pode ser entendido como a sequência (ou cadeia) de estados visitados por um agente, desde o estado inicial até o final. Os estados inicial e final são definidos arbitrariamente pela descrição do experimento. Por exemplo, o primeiro pode ser um estado de "entrada" ou de "custo baixo" para o agente iniciar. O segundo, final, pode ser determinado por um tempo limite (*i.e.* após vários estados visitados, o final é aquele que é o n-ésimo visitado), ou mesmo, um estado considerado de "sucesso" ou "falha irrecuperável".

Sendo assim, a função valor e a política são incrementados episódio a episódio, e não passo a passo (lê-se aqui "passo" como equivalente a "estado"). Apesar do caráter de aleatoriedade (daí o nome Monte Carlo) esse método interage para buscar a otimalidade de forma essencialmente equivalente ao da PD, discutida previamente.

3.2.1 Avaliação de Política por MC

Nesta seção, será tratado o método de avaliação de política por MC, isto é, aprender os valores dos estados de um PDM. Cabe ressaltar que o valor de um estado é o retorno esperado - acumulação descontada das recompensas futuras - iniciando do estado em questão.

Uma maneira óbvia de estimar o valor por experiência é calcular a média dos retornos observados à medida que os estados são visitados. Quanto mais amostras, melhor a convergência para o valor esperado. Esta ideia está por trás de todos os métodos de MC.

Por exemplo, suponha que deseja-se estimar $V^{\pi}(s)$, o valor de um estado s seguindo uma política π. Cada ocorrência do estado s em um episódio é dita "visita" a s. A cada visita, o método de MC estima o valor de s como a média dos valores dos estados seguidos de s em todos os episódios.

Considera-se, nesta seção, a média calculada apenas dos retornos seguidos da primeira visita ao estado s em um dado episódio. Este método é chamado "MC de primeira visita", ilustrado de maneira procedural no Algoritmo 4. A estrutura chave deste algoritmo é, abstratamente, uma "tabela de conjuntos", definida aqui como $R(s)$. Mapeia um estado s para um conjunto de retornos imediatos experimentados pelo agente. A média de $R(s)$ - à medida que novas amostras do retorno forem adicionadas - convergirá para o valor do estado s para a política sendo seguida.

entrada: Política π a avaliar
saída : Aproximação da função valor $V^{\pi}(s)$ da política
Inicialize $V(s)$ arbitrariamente, $\forall s \in \mathcal{S}$
Inicialize $R(s) \leftarrow \emptyset \; \forall s \in \mathcal{S}$
repita
  Gere um episódio utilizando π
  para todo $s \in$ *episódio* **faça**
    $r \leftarrow$ retorno seguido da primeira ocorrência de s
    Adicione r ao conjunto $R(s)$
    $V(s) \leftarrow \overline{R(s)}$
  fim
até *critério de parada*

Algoritmo 4: Avaliação de estados por MC, segundo uma política dada (método de primeira visita)

No Algoritmo 4, $\overline{R(s)}$ denota a média dos elementos do conjunto $R(s)$. O algoritmo converge para $V^{\pi}(s)$ à medida que o número de visitas a s cresce indefinidamente.

3.2.2 Estimativa do Valor das Ações

Quando o modelo do sistema não está disponível, torna-se útil avaliar o valor das *ações* ao invés de *estados*. Com o modelo em mãos, apenas o valor dos estados é suficiente para determinar uma política. O controle observa um passo à frente e toma a decisão de selecionar a ação que leva à melhor combinação de recompensa e próximo estado - como feito nos métodos de PD.

Entretanto, sem um modelo, o valor dos estados sozinho não é suficiente para determinar uma política. É preciso estimar explicitamente o valor de cada ação para que os valores sejam úteis na tomada de decisão de política. Desta forma, o objetivo dos métodos de MC é estimar $Q^{*}(s, a)$ - neste contexto, Q representa o valor do par estado/ação para uma dada política. Para atingir este objetivo, considera-se agora outro problema de avaliação de política: para pares de estado/ação.

O problema de avaliação de política para valores de ação é estimar $Q^{\pi}(s, a)$: o retorno esperado quando iniciando no estado s, tomando a ação a, e depois, seguindo a política π. O algoritmo de avaliação de ações funciona de maneira muito similar ao algoritmo de avaliação de estados (Algoritmo 4).

3.2.3 Controle em MC

Considera-se nesta seção como os métodos de MC podem ser utilizados para controle, incorporando estimativa de valor de estado/ação e melhoria de política no sentido de buscar uma política ótima, de forma similar às ideias de *iteração de política* já apresentadas na seção sobre *programação dinâmica*:

$$\pi_0 \xrightarrow{A} V^{\pi_0} \xrightarrow{M} \pi_1 \xrightarrow{A} V^{\pi_1} \xrightarrow{M} \pi_2 \xrightarrow{A} \cdots \xrightarrow{M} \pi_* \xrightarrow{A} V^{\pi_*}$$

onde $\xrightarrow{A}$ denota avaliação de política e $\xrightarrow{M}$ denota melhoria de política.

Neste caso específico, no qual tratam-se de métodos de MC, a avaliação de política segue o Algoritmo 4 e a melhoria de política é simplesmente tomar a ação gulosa (*i.e.* a ação de maior valor) para cada estado.

É importante ressaltar que, neste modelo de controle aparece o problema de exploração/explotação, ou seja, tão logo os valores começam a ser estimados existe o risco da melhoria de política ignorar ações de menor valor estimado, mas com valor real maior, fazendo o controle cair em mínimos locais rapidamente.

É fundamental, portanto, garantir uma forma de exploração das ações durante a geração dos episódios. Não tomar ações sempre gulosas é uma maneira de se certificar de que ações de menor valor também sejam eventualmente exploradas. O Algoritmo 5 mostra a forma procedural do ciclo de avaliação e melhoria que compõe um controlador de MC.

saída : Política $\pi(s)$ aproximadamente ótima
Inicialize $Q(s,a)$ arbitrariamente, $\forall s, a$
Inicialize $R(s,a) \leftarrow \emptyset \; \forall s, a$
repita
  Gere um episódio utilizando π
  para todo *par* $s, a \in$ *episódio* **faça**
    $r \leftarrow$ retorno seguido da primeira ocorrência de s, a
    Adicione r ao conjunto $R(s,a)$
    $Q(s,a) \leftarrow \overline{R(s,a)}$
  fim
  para todo $s \in$ *episódio* **faça**
    $\pi(s) \leftarrow argmax_a Q(s,a)$
  fim
até *critério de parada*

Algoritmo 5: Avaliação de ações por MC

No Algoritmo 5, $\overline{R(s,a)}$ é a média dos valores do conjunto $R(s,a)$.

Observe que o algoritmo é uma composição de um procedimento de avaliação de política e outro de melhoria.

3.3 Diferença Temporal (DT)

O método da DT é considerado por muitos autores a ideia central no tema de aprendizagem por reforço [14], Cap. 6. De certa maneira, é uma combinação das ideias de Monte Carlo e Programação Dinâmica já apresentadas.

Da mesma maneira que MC, o método da DT aprende somente pela experiência, sem um modelo explícito do ambiente, como em PD. No entanto, similarmente à PD, o método da DT atualiza sua aproximação da função de valor baseando-se em estimativas anteriores, de modo que não há necessidade aqui de aguardar pelo final do episódio (aprendizado *on-line*).

3.3.1 Estimativa de Valor

Ambos os métodos de DT e de MC utilizam a experiência para resolver o problema de estimativa de valor. Dada alguma experiência seguindo uma política π, ambos os métodos atualizam sua estimativa V de V^π. Se um estado não terminal s_t é visitado no instante t, então os métodos atualizam sua estimativa $V(s_t)$ baseado no que acontece em consequência desta visita. O método MC aguarda até que o retorno seguido da visita seja conhecido, quando então utiliza este valor como objetivo na atualização de $V(s_t)$, como denotado:

$$V(s_t) \leftarrow V(s_t) + \alpha(R_t - V(s_t)) \tag{3.3}$$

onde R_t é o retorno no tempo t e α é uma constante de tamanho de passo (aprendizado).

Dado que por MC é necessário aguardar até o fim do episódio para determinar o incremento de $V(s_t)$, pois somente então R_t é conhecido, nos métodos de DT é necessário apenas aguardar até o próximo passo. No instante $t+1$, é construído um objetivo imediato e é feita uma atualização útil fazendo uso de r_{t+1} e da estimativa $V(s_{t+1})$. A atualização de DT mais simples, também conhecida como $DT(0)$, como será apresentado nas seções seguintes, é dada por:

$$V(s_t) \leftarrow V(s_t) + \alpha(r_{t+1} + \gamma V(s_{t+1}) - V(s_t)) \tag{3.4}$$

onde α é a taxa de aprendizagem (note que um valor muito alto pode causar oscilação) e γ é o fator de desconto da diferença entre a percepção de valor dos instantes t e $t+1$.

Com efeito, observa-se que o alvo da atualização de MC é R_t, enquanto que na DT o alvo é $r_{t+1} + \gamma V(s_{t+1})$.

Pelo motivo da DT basear sua atualização em uma estimativa anterior, é dito que tal método faz uso de *bootstrapping*, como em PD.

Tendo a regra apresentada na equação 3.4 como fundamento, segue o algoritmo básico para avaliar uma política π.

entrada: Política π a avaliar
saída : Aproximação da função valor $V^\pi(s)$ da política
Inicialize $V(s)$ arbitrariamente, $\forall s \in \mathcal{S}$
para todo *episódio* **faça**
  Inicialize s (estado inicial)
  repita
    $a \leftarrow$ ação dada por π para s
    Tome ação a, observe recompensa r e próximo estado s'
    $V(s_t) \leftarrow V(s_t) + \alpha(r_{t+1} + \gamma V(s_{t+1}) - V(s_t))$
    $s \leftarrow s'$
  até s *ser terminal*
fim

Algoritmo 6: Avaliação de Política por diferença temporal: $DT(0)$

3.3.2 Controle em Diferença Temporal

Com as ideias de DT estabelecidas, agrega-se a estrutura de *iteração de política* à DT como estimador do valor da política para pares de estado/ação, de modo que isso seja útil para construir sucessivamente melhorias na política.

Devido ao fato da atualização da política ocorrer durante o episódio, as decisões tomadas na mudança da política irão efetivamente alterar o fluxo de experiências, o que evidentemente trará mudanças no aprendizado. Portanto, os algoritmos de aprendizado *on-line* são divididos em duas categorias: *On-Policy* e *Off-Policy*.

No primeiro caso, *On-Policy*, que será tratado na próxima sub-seção, deve-se estimar $Q^{\pi}(s, a)$ para a política π corrente e para todos os estados s e ações a. Já nos algoritmos *Off-Policy*, aproxima-se diretamente Q^{*}, independentemente da política seguida no momento.

Controle *On-Policy*

O algoritmo para controle com DT *On-Policy* chama-se *Sarsa* (*state, action, reward, state(next), action(next)*) [13]. A estimativa de $Q^{\pi}(s, a)$ pode ser feita essencialmente através do mesmo método supra-citado para estimar V^{π}. Considera-se, entretanto, agora, transições de pares estado/ação para estado/ação, sendo formalmente idênticos: ambos são cadeias de Markov com recompensas.

A atualização dos pares é dada pela regra:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left(r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\right) \tag{3.5}$$

Esta atualização é realizada para cada transição de um estado não terminal s_t. Caso s_{t+1} seja terminal, então $Q(s_{t+1}, q_{t+1})$ é definido como zero.

O Algoritmo 7 permite visualizar a forma de controle *On-Policy* que faz uso da regra de atualização anterior. Como em todos os métodos *On-Policy*, Q^{π} é continuamente estimado para a política π, ao mesmo tempo em que π é modificado na direção de decisões mais gulosas com respeito a Q^{π}.

Inicialize $Q(s, a)$ arbitrariamente, $\forall s, a \in \mathcal{S}, \mathcal{A}$
para todo *episódio* **faça**
  Inicialize s (estado inicial)
  $a \leftarrow$ ação para s utilizando política derivada de Q
  repita
    Tome ação a, observe recompensa r e próximo estado s'
    $a' \leftarrow$ ação para s' utilizando política derivada de Q
    $Q(s, a) \leftarrow Q(s, a) + \alpha(r + \gamma Q(s', a') - Q(s, a))$
    $s \leftarrow s'$
    $a \leftarrow a'$
  até s *ser terminal*
fim

Algoritmo 7: *Sarsa*: Controle por diferença temporal *On-Policy*

Controle *Off-Policy*

Uma das mais importantes contribuições feitas ao estudo de aprendizado por reforço foi o desenvolvimento de um algoritmo de controle independente da política atual (*Off-Policy*) conhecido como *Q-Learning* [2]. Em sua forma mais elementar, um passo de aprendizado no referido método é definido como:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right] \tag{3.6}$$

Neste caso, a função aprendida de estado/ação Q aproxima diretamente Q^*, a função estado/valor ótima, independente da política atualmente seguida. Isso dramaticamente simplifica a análise do algoritmo. A política, entretanto, ainda tem um efeito no aprendizado no sentido de que ela determina quais pares estado/ação serão visitados e atualizados. Tudo o que é necessário para uma convergência correta é que todos os pares sejam continuamente atualizados.

O método *Q-Learning* é mostrado em sua forma procedural no Algoritmo 8.

Inicialize $Q(s, a)$ arbitrariamente, $\forall s, a \in \mathcal{S}, \mathcal{A}$
para todo *episódio* **faça**
  Inicialize s (estado inicial)
  repita
    $a \leftarrow$ ação para s utilizando política derivada de Q
    $Q(s, a) \leftarrow Q(s, a) + \alpha(r + \gamma \max_{a'} Q(s', a') - Q(s, a))$
    $s \leftarrow s'$
  até s *ser terminal*
fim

Algoritmo 8: *Q-Learning*: Controle por diferença temporal *Off-Policy*

3.4 Traçado de Elegibilidade

O mecanismo de Traçado de Elegibilidade é fundamental em aprendizagem por reforço e, nesta seção, este conceito é abordado em detalhes, apresentando o algoritmo em sua forma final (em conjunto com funções aproximadoras) para a solução dos problemas a serem apresentados no capítulo 5.

Segundo Sutton e Barto [14], Cap. 7, há duas maneiras de visualizar o conceito de elegibilidade: a visão teórica (ou para frente) - mais enfatizada pelos autores - e a visão mecanicista (ou para trás).

Apesar dos diferentes enfoques, ambas as visões descrevem exatamente os mesmos algoritmos. Ao introduzir ambas, consolida-se uma percepção mais sólida do que é computado nos métodos que incorporam elegibilidade. Na primeira subseção, faz-se um paralelo entre MC e DT, ao considerar atualizações com n passos. Em seguida, nas próximas subseções, aprofunda-se nas duas diferentes visões: teórica e mecanicista, para então apresentar variações dos algoritmos *On-Policy* e *Off-Policy* já descritos na seção 3.3, modificados para incluir o novo conceito deste capítulo.

3.4.1 Diferença Temporal com n Passos

Considera-se o espaço de métodos existentes entre MC e DT, cujo objetivo é estimar V^π de episódios gerados a partir de π. O método de MC atualiza cada estado baseado na sequência inteira de recompensas observadas do estado em questão até o fim do episódio. A atualização em DT, por outro lado, é baseada apenas na próxima recompensa e utiliza o valor do estado no próximo passo como aproximação das próximas recompensas.

Um método intermediário, portanto, deveria atualizar V^π baseado em um número intermediário de recompensas: mais do que uma, mas menos que todas até o estado final. Por exemplo, uma atualização com dois passos utilizaria as duas próximas recompensas mais uma estimativa do valor dos estados seguintes. Generalizando, pode-se idealizar um método DT com n passos onde seriam utilizadas as próximas n recompensas mais uma estimativa do valor para estados seguintes depois de n.

Neste sentido, chamar-se-á os métodos de DT descritos na seção 3.3 como DT de um passo, para podermos progredir nossa ideia de generalização do número de passos neste método. Mais formalmente, considera-se o valor do retorno de um estado s_t (estado s do passo t) em MC como:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots + \gamma^{T-t-1} r_T \tag{3.7}$$

onde T é o passo final do episódio. Essa quantidade de fato é o "objetivo" da atualização. No caso de MC, é justamente o retorno completo, enquanto que em DT de um passo é a primeira recompensa mais o valor descontado (γ) do próximo estado:

$$R_t^{(1)} = r_{t+1} + \gamma V_t(s_{t+1}) \tag{3.8}$$

A razão deste retorno é que o componente $\gamma V_t(s_{t+1})$ substitui os termos restantes $\gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots + \gamma^{T-t-1} r_T$. Assim como essa ideia faz sentido para DT de um passo, faz para DT de dois passos:

$$R_t^{(2)} = r_{t+1} + \gamma r_{t+2} + \gamma^2 V_t(s_{t+2}) \tag{3.9}$$

Em geral, tem-se:

$$R_t^{(n)} = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots + \gamma^{n-1} r_{t+n} + \gamma V_t(s_{t+n}) \tag{3.10}$$

Sutton e Barto [14], Cap. 7, chamam esta quantidade de "retorno de n passos truncado e corrigido", porque denota um retorno truncado após n passos, corrigido com a soma de uma estimativa do valor do próximo n-ésimo passo. A partir de agora, utiliza-se essa terminologia referindo-se a $R_t^{(n)}$ como o retorno de n passos no instante t.

É necessário notar que, se o episódio acaba em menos que n passos, então não há truncamento do valor e efetivamente a atualização de DT de n passos torna-se um caso especial de MC (atualizando até o último estado).

É preciso enfatizar que todos os conceitos apresentados nesta seção, com respeito à atualização do valor dos estados (V_t), são perfeitamente aplicáveis para pares de estado/ação no caso da aplicação dos algoritmos em controle.

3.4.2 Visão Teórica ou "Para frente"

O algoritmo DT(λ) - diferença temporal com Elegibilidade - pode ser entendido como uma maneira de atualizar V^{π} baseado em uma média ponderada de todos os retornos de n passos. Cada ponderação ou peso é proporcional à λ^{n-1}, onde $0 \leq \lambda \leq 1$. Um fator de normalização $1-\lambda$ garante que os pesos somam à unidade. A atualização resultante requer a utilização da terminologia da introdução desta seção, formalizada como:

$$R_t^{\lambda} = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} R_t^{(n)} \tag{3.11}$$

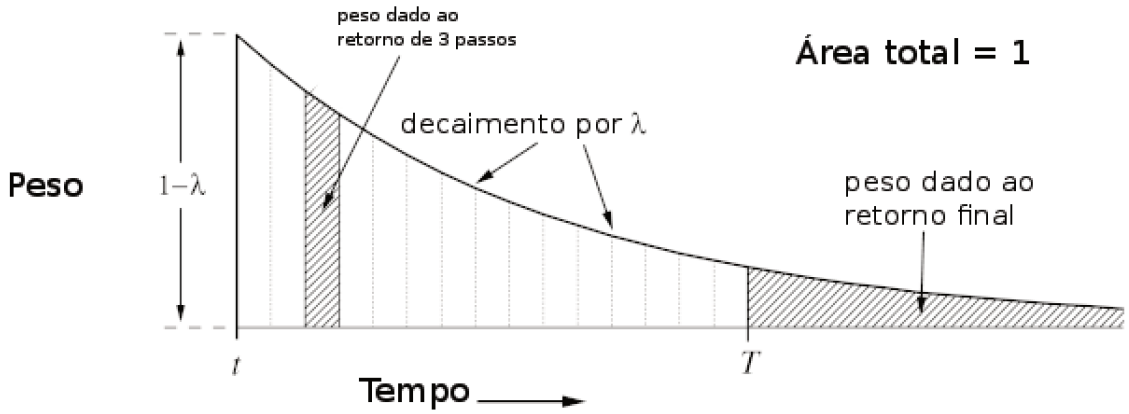

Fig. 3.1: Peso dado a cada retorno de n passos na composição do retorno λ

A Figura 3.1 ilustra essa sequência de pesos. Ao retorno de um passo é dado o maior peso, $1-\lambda$; ao retorno de dois passos é dado o segundo maior peso $(1-\lambda)\lambda$; ao retorno de três passos, $(1-\lambda)\lambda^2$; e assim sucessivamente. O peso é decaído na razão de λ a cada passo adicional. Após o estado terminal, todos os retornos de n passos subsequentes são iguais a R_t. Para enfatizar tal colocação, pode-se separar estes termos da soma principal:

$$R_t^{\lambda} = (1-\lambda)\sum_{n=1}^{T-t-1} \lambda^{n-1} R_t^{(n)} \; + \; \lambda^{T-t-1} R_t \tag{3.12}$$

Esta equação possivelmente deixa mais claras as implicações para $\lambda = 1$. Neste caso, o termo do somatório é zero, e o termo remanescente é reduzido ao retorno convencional, ou seja, R_t. Logo, para $\lambda = 1$, atualizando de acordo com a equação 3.11 ($DT(\lambda)$), resulta o algoritmo de MC. De outra maneira, se $\lambda = 0$, então a mesma equação reduz-se a $R_t^{(1)}$, o retorno de um passo. Logo, para λ nulo, a atualização da equação já supracitada é o mesmo que o método DT de um passo, ou $DT(0)$.

Neste ponto, define-se então o algoritmo de retorno λ cujo incremento do valor do estado s no instante t ($V_t(s_t)$) é dado por:

$$\Delta V_t(s_t) = \alpha \left[R_t^{\lambda} - V_t(s_t) \right] \tag{3.13}$$

onde α é a constante de aprendizado e a atualização dos outros estados diferentes de s_t é nula: $V_t(s) = 0; \forall s \neq s_t$. Esta visão, que está sendo enfatizada nesta subseção, é chamada "para frente" justamente porque é necessário "visualizar" à frente no tempo para calcular o retorno e atualizar o valor.

3.4.3 Visão Mecanicista ou "Para trás"

Na subseção anterior, foi apresentada a visão teórica do algoritmo $DT(\lambda)$ como uma maneira de agregar atualizações que variam parametricamente do método de DT a MC. Nesta subseção, define-se o $DT(\lambda)$ mecanicamente. Entretanto, é equivalente à visão anteriormente apresentada.

A visão mecanicista, ou para trás, do $DT(\lambda)$ é útil porque é conceitualmente e computacionalmente mais simples. Em particular, a visão para frente não é implementável de imediato, porque é não causal, ou seja, utiliza, a cada passo, conhecimento de todos os passos seguintes para atualização. O mecanismo da visão para trás, por outro lado, aproxima a visão para frente de modo causal e incremental, e, na atualização *off-line*, atinge resultado idêntico.

Nesta concepção, há uma variável adicional associada a cada estado, o *traçado de elegibilidade*. O Traçado de Elegibilidade para um dado estado s no instante t é denotado por $e_t(s) \in \mathbb{R}^+$. A cada passo, o traçado de elegibilidade de todos os estados decai em um fator de $\gamma\lambda$, e o traçado de elegibilidade do estado visitado é incrementado em 1:

$$e_t(s) = \begin{cases} \gamma\lambda e_{t-1}(s) & se\ s \neq s_t \\ \gamma\lambda e_{t-1}(s) + 1 & se\ s = s_t \end{cases} \tag{3.14}$$

para todo $s \in S$, onde γ é a taxa de desconto e λ é o parâmetro introduzido na subseção anterior. Refere-se a λ como a *taxa de decaimento da elegibilidade*. Este tipo de traçado de elegibilidade é denominado *acumulativo* [14], Cap. 7, porque é acumulado a cada visita ao estado, e decai gradualmente enquanto o estado não é visitado, como ilustrado exemplificadamente na figura 3.2.

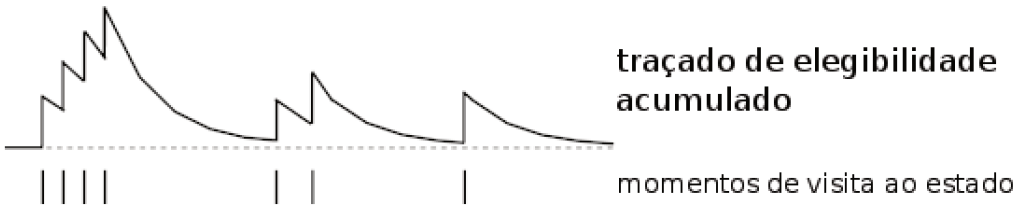

Fig. 3.2: Exemplo de um gráfico do valor de $e(s)$, mostrando acumulação ao longo do tempo

A qualquer momento, o traçado de elegibilidade possui um registro de quais estados foram visitados recentemente, onde "recentemente" é definido em termos de $\gamma\lambda$. O valor de $e(s)$ indica o grau em que cada estado é elegível para mudanças, caso um aprendizado por reforço ocorra. Por exemplo, o erro de DT para predição de valor é dado por:

$$\delta_t = r_{t+1} + \gamma V_t(s_t + 1) - V_t(s_t) \tag{3.15}$$

Nesta argumentação, os valores dos estados são modificados de maneira global, proporcional ao traçado de elegibilidade. Todos os estados com traçado não nulo serão atualizados proporcionalmente ao decaimento, ou quão recentes foram visitados:

$$\Delta V_t(s) = \alpha \delta_t e_t(s), \forall\ s \in S \tag{3.16}$$

Vale observar que estas atualizações podem ser realizadas a cada passo, para formar um algoritmo *on-line*, ou realizadas ao final de cada episódio, compondo um algoritmo *off-line*.

Como última ilustração didática a respeito do funcionamento deste algoritmo, resgatando a mesma linha de raciocínio da última subseção, considere o que acontece para cada um dos vários valores que λ pode assumir: Caso $\lambda = 0$, então pela Eq. 3.14 todos os traçados são nulos no instante t, exceto para aquele correspondente a s_t. Desta forma, a atualização 3.16 é reduzida à atualização mais simplificada $DT(0)$.

Se $\lambda = 1$, então o crédito dado aos estados anteriormente visitados é decaído na razão de γ, provocando um comportamento idêntico ao método de MC, ou $DT(1)$. Nota-se que λ nos permite fazer uma variação paramétrica de DT a MC.

3.4.4 Controle de Traçado de Elegibilidade

Discute-se aqui como a ideia de elegibilidade pode ser usada para controle. Como usual nesta dissertação, estende-se as equações de aprendizado de valor ($V_t(s)$) para pares estado/ação ($Q_t(s,a)$).

Na primeira parte, apresenta-se uma variação do algoritmo *On-Policy Sarsa*, já discutido. Em seguida, apresenta-se o algoritmo *Off-Policy*, cujos conceitos são ligeiramente mais complexos e suas duas formulações mais populares na literatura [2], [10].

Controle *On-Policy*

O traçado de elegibilidade pode ser combinado com o algoritmo *Sarsa* (Alg. 7) de uma maneira direta. O algoritmo, batizado de Sarsa(λ) [13] aplica o método $DT(\lambda)$ para a aproximação dos valores de pares de estado/ação, ao invés de estados. Obviamente, por esse motivo, a variável de elegibilidade aplica-se não somente aos estados, mas também às ações. Ou melhor, aos pares estado/ação.

Denota-se, então, por $e_t(s,a)$ a elegibilidade associada ao par estado/ação, s e a, no instante t, resultando em uma regra de atualização análoga:

$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha \delta_t e_t(s,a),\ \forall\ s,a \tag{3.17}$$

onde:

$$\delta_t = r_{t+1} + \gamma Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t) \tag{3.18}$$

e

$$e_t(s,a) = \begin{cases} \gamma\lambda e_{t-1}(s,a) + 1 & \text{se } s = s_t \text{ e } a = a_t \\ \gamma\lambda e_{t-1}(s,a) & \text{caso contrário} \end{cases} \ \forall\ s,a \tag{3.19}$$

O algoritmo *Sarsa* originalmente proposto (*Sarsa*(0)) e o novo algoritmo *Sarsa*(λ) são *On-Policy*, ou seja, aproximam $Q^\pi(s,a)$ para a política π corrente. Em seguida, melhoram a política baseado nos valores aproximados. O Algoritmo 9 mostra a forma procedural das equações descritas anteriormente.

Inicialize $Q(s, a)$ arbitrariamente e $e(s, a) = 0$, $\forall\ s, a$
para todo *episódio* **faça**
  Inicialize s (estado inicial)
  $a \leftarrow$ ação para s utilizando política derivada de Q
  repita
    Tome ação a, observe recompensa r e próximo estado s'
    $a' \leftarrow$ ação para s' utilizando política derivada de Q
    $\delta \leftarrow r + \gamma Q(s', a') - Q(s, a)$
    $e(s, a) \leftarrow e(s, a) + 1$
    para todo s, a **faça**
      $Q(s, a) \leftarrow Q(s, a) + \alpha\delta e(s, a)$
      $e(s, a) \leftarrow \gamma\lambda e(s, a)$
    fim
    $s \leftarrow s'$
    $a \leftarrow a'$
  até s *ser terminal*
fim

Algoritmo 9: *Sarsa*(λ): Controle por diferença temporal *On-Policy* com uso de elegibilidade

Controle *Off-Policy*

Dois métodos principais e distintos foram propostos para combinar o algoritmo *Q-Learning* e traçado de elegibilidade. São denominados, na literatura, de *Q(λ)-Watkins* e *Q(λ)-Peng*, em homenagem aos seus respectivos autores [2], [10]. O primeiro algoritmo, proposto por Watkins, é, na opinião do autor desta dissertação, conceitualmente mais simples. E, portanto, será abordado com maior ênfase. A segunda proposta, de Peng, será tratada em seguida, apresentando as ideias com menos formalidade.

Q(λ)-Watkins

É importante lembrar, neste ponto, que o *Q-Learning* é um algoritmo *Off-Policy*, significando que a política ótima não é necessariamente a política utilizada para escolher as ações. Em particular, o algoritmo *Q-Learning* aprende a política gulosa enquanto segue tipicamente uma política envolvendo ações exploratórias - escolhas ocasionais de ações sub-ótimas de acordo com Q_t. Por este motivo, um cuidado especial é necessário quando introduz-se o conceito de traçado de elegibilidade.

Suponha que se vá atualizar o par estado/ação (s_t, a_t) no instante t. Suponha que nos dois passos seguintes o agente escolha ações gulosas (ótimas), mas, no terceiro passo, no instante $t + 3$, o agente escolha uma ação sub-ótima exploratória. No aprendizado do valor da política gulosa do par (s_t, a_t), pode-se usar a experiência subsequente somente quando a política gulosa está sendo seguida. Deste modo, pode-se usar os retornos de um passo e de dois passos adiante, mas não, neste caso, o retorno

de três passos adiante. Os retornos de n-passos para $n \leq 3$ não mais necessariamente têm qualquer relação com a política gulosa.

Logo, ao contrário dos métodos $TD(\lambda)$ e Sarsa(λ), o *Q(λ)-Watkins* não considera todos os passos até o fim do episódio em sua atualização. Ele considera apenas até a próxima ação exploratória. Apesar desta diferença, o *Q(λ)-Watkins* é muito similar ao $TD(\lambda)$ e ao Sarsa(λ): suas atualizações somam até o fim do episódio, enquanto que no Q(λ) é somado somente até a próxima ação exploratória ou fim do episódio caso não haja ações exploratórias. Para ser mais preciso, *Q(λ)-Watkins* considera até uma ação depois da ação exploratória, utilizando seu conhecimento a respeito do valor das ações. Por exemplo, suponha que a primeira ação a_{t+1} é exploratória. O método ainda irá realizar a atualização de um passo de $Q_t(s_t, a_t)$ em direção a $r_{t+1} + \gamma max_a Q_t(s_{t+1}, a)$. Em geral, se a_{t+n} é a primeira ação exploratória, então a atualização é na direção de:

$$r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{n-1} r_{t+n} + \gamma^n \max_a Q(s_{t+n}, a)$$

A visão mecanicista ou "para trás" de *Q(λ)-Watkins* é também simples: O traçado de elegibilidade é utilizado da mesma forma que no Sarsa(λ), exceto pelo fato de que o valor é atribuído a zero sempre que uma ação exploratória (não gulosa) é tomada. A atualização do traçado é melhor compreendida como ocorrendo em dois passos. Primeiro, os traçados de todos os pares estado/ação são ou decaídos na razão de $\gamma\lambda$ ou, se uma ação exploratória é tomada, setados em zero. Segundo, o traçado correspondente ao par estado/ação corrente é incrementado em 1. O resultado geral, de maneira mais formal pode ser colocado como:

$$e_t(s,a) = \mathbf{I}_{ss_t} \times \mathbf{I}_{aa_t} + \begin{cases} \gamma\lambda e_{t-1}(s,a) & \text{se } Q_{t-1}(s_t, a_t) = \max_a Q(s_t, a) \\ 0 & \text{caso contrário} \end{cases} \tag{3.20}$$

onde $\mathbf{I}_{xy}$ é uma função indicadora de identidade cujo valor é 1 se $x = y$ e 0 caso contrário. O restante do algoritmo é definido como:

$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha\delta_t e_t(s,a) \tag{3.21}$$

onde

$$\delta_t = r_{t+1} + \gamma \max_{a'} Q(s_t, a') - Q_t(s_t, a_t) \tag{3.22}$$

O Algoritmo 10 mostra, de maneira procedural, a conjunção das ideias apresentadas.

Inicialize $Q(s,a)$ arbitrariamente e $e(s,a) = 0$, $\forall\ s, a$
para todo *episódio* **faça**
  Inicialize s (estado inicial)
  $a \leftarrow$ ação para s utilizando política derivada de Q
  repita
    Tome ação a, observe recompensa r e próximo estado s'
    $a' \leftarrow$ ação para s' utilizando política derivada de Q
    $a^* = argmax_b\ Q(s', b)$
    $\delta \leftarrow r + \gamma Q(s', a^*) - Q(s, a)$
    $e(s,a) \leftarrow e(s,a) + 1$
    para todo s, a **faça**
      $Q(s,a) \leftarrow Q(s,a) + \alpha\delta e(s,a)$
      se $a' = a^*$ **então**
        $e(s,a) \leftarrow \gamma\lambda e(s,a)$
      fim
      senão
        $e(s,a) \leftarrow 0$
      fim
    fim
    $s \leftarrow s'$
    $a \leftarrow a'$
  até s *ser terminal*
fim

Algoritmo 10: $Q(\lambda)$-*Watkins*: Controle por diferença temporal *Off-Policy* com uso de elegibilidade

$Q(\lambda)$-*Peng*

Uma desvantagem recentemente apresentada do algoritmo $Q(\lambda)$ é o desperdício de informação, ou subutilização do traçado de elegibilidade, devido à anulação dos valores quando há uma ação exploratória. Se as ações exploratórias são frequentes, como de costume são no início da aprendizagem, então raramente atualizações com mais de um ou dois passos são feitas, de modo que o processo torna-se apenas ligeiramente mais rápido que o *Q-Learning* de um passo.

Peng e Williams [10] propuseram uma versão alternativa do Q(λ) para tentar remediar essa situação, o algoritmo por eles propostos, denominado aqui $Q(\lambda)$-Peng, pode ser pensado como um híbrido entre o Sarsa(λ) e $Q(\lambda)$-Watkins.

Conceitualmente, diferente do algoritmo anterior, neste não há diferença entre ações exploratórias e ações gulosas. Cada atualização leva em conta vários passos de experiência efetiva, e a todos, exceto o último, é incorporado um componente final de maximização das ações possíveis. Essa atualização, portanto, não é nem *On-Policy* nem *Off-Policy*: as transições iniciais são *On-Policy*, enquanto que a

final (fictícia) é *Off-Policy*, pois faz uso de uma ação gulosa.

Como consequência, para uma política não gulosa fixa, Q_t não converge para Q^π nem para Q^*, mas para um híbrido entre os dois. No entanto, se mais gulosa, então o método ainda assim convergirá para Q^*.

Capítulo 4

Método Proposto

Apresenta-se, neste capítulo, o algoritmo proposto nesta pesquisa. Trata-se de um procedimento de aprendizagem por reforço (AR) *Off-Policy* para ambientes contínuos e com elegibilidade. A arquitetura, a ser elucidada com detalhes nas próximas seções, é um acoplamento de uma rede neural a uma função interpoladora denominada *wire-fitting*. Basicamente, a entrada da rede neural é interpretada como um vetor do estado de um processo de decisão de Markov (PDM). A saída da rede neural é interpretada como os parâmetros da função interpoladora e a saída desta última é interpretada como uma aproximação da melhor ação para um estado. Portanto, o conjunto híbrido descrito é, potencialmente, um aproximador de Q.

As razões para escolha desta arquitetura são defendidas na seção 4.4, porém, fundamenta-se principalmente na baixa complexidade do cálculo da ação máxima para um estado (necessário nos algoritmos de AR *Off-Policy*).

Antes de discutir o algoritmo propriamente, são discorridos os conceitos de PDM em ambientes contínuos e de redes neurais artificiais, e é definida a função interpoladora *wire-fitting*.

4.1 Ambientes Contínuos

Até o presente momento, neste trabalho, considera-se que as estimativas das funções de valores são representadas por tabelas com um registro para cada estado (ou par estado/ação). Esta é uma visão particularmente clara e instrutiva, mas, evidentemente, é limitada a tarefas com poucos estados e ações. A problemática no que tange à implementação abrange não somente a quantidade de memória necessária para representar os estados, mas também o tempo para estimar todos os valores. Em outras palavras, o ponto chave desta questão é a *generalização*: Como pode a experiência com um subconjunto de estados ser generalizada de maneira útil no sentido de produzir uma boa aproximação de um subconjunto muito maior?

Este problema é recorrente em muitas aplicações não limitadas à aprendizagem por reforço. De fato, em diversas instâncias de problemas reais, dificilmente um estado será experimentado novamente de maneira idêntica, sem contar que muitas variáveis são contínuas, efetivamente compreendendo infinitos estados.

A abordagem adotada por Sutton e Barto [14], Cap. 8, é utilizar os algoritmos de aprendizagem por reforço como um *framework* em que serão acopladas as técnicas de aproximação de funções, já

bastante consolidadas na literatura, como instâncias de problemas de aprendizado supervisionado (*e. g.* redes neurais, árvores de decisão, *nearest-neighbor*).

A novidade que é introduzida nesta seção reside no fato de que a função de estimativa de valor V_t, num instante t, não será mais representada por uma tabela, mas por uma função parametrizada com vetor de parâmetros $\vec{\theta_t}$. A significância deste fato é que o valor de V_t depende totalmente de $\vec{\theta_t}$. Tal vetor pode, em teoria, representar a parametrização de qualquer função aproximadora, muito embora estuda-se em particular as redes neurais do tipo perceptron multi-camadas. A escolha deste modelo de aproximação se estabelece na grande quantidade e qualidade de literatura disponível a respeito, e também na gama de aplicações bem sucedidas em classificação, predição e outras generalizações.

O algoritmo já apresentado, Q(λ), será acoplado às redes neurais para tratar de espaços de estado e ação contínuos, segundo a abordagem de Doya [5]. Em seguida, na próxima subseção, será descrito um problema que surge ao tentar encontrar o máximo de $Q(s, a)$ e sua solução com outro método de aproximação via função interpoladora, a saber, *Wire-Fitting*.

4.2 Redes Neurais Artificiais

Uma Rede Neural Artificial, usualmente referenciada apenas como rede neural (RN), é uma abstração matemática ou modelo computacional inspirado na estrutura ou mesmo em aspectos funcionais de redes neurais biológicas. Uma RN consiste de um conjunto interconectado de neurônios artificiais, e é capaz de processar informação através de uma abordagem conexionista. Na maioria dos casos, uma RN é um sistema adaptativo que modifica sua própria estrutura baseada nos fluxos de informação internos ou externos durante o processo de aprendizagem. Redes Neurais, como compreendidas mais recentemente, podem ser interpretadas como ferramentas estatísticas não-lineares, em geral, utilizadas para modelar relacionamentos complexos entre entrada e saída, ou para encontrar padrões em dados [7].

Fundamentação

A inspiração original para o termo "Rede Neural" vem da examinação do sistema nervoso central, composto por neurônios, axônios com seus dendritos e sinapses, os quais constituem os elementos biológicos de processamento de informação, investigados pela neurociência. Em uma RN Artificial, módulos artificiais, comumente chamados de "neurônios" ou "unidades" são interconectados a fim de compor uma rede de módulos mimificando o equivalente natural - daí o termo "Rede Neural Artificial".

Devido ao fato da neurociência hoje estar cheia de perguntas sem resposta, e também ao fato de que existem vários níveis de abstração, que levam a diversas maneiras de se inspirar no cérebro, não há uma definição formal única para modelos matemáticos conexionistas. De forma geral, uma RN é uma rede com elementos de processamento simples, mas onde emerge um comportamento complexo global determinado pelas conexões entre as unidades e os parâmetros relacionados a estas conexões. Apesar de que uma RN não necessita ser adaptativa *per se*, seu uso prático viabiliza-se através de algoritmos arquitetados para alterar os parâmetros (pesos) das conexões na rede, produzindo então o processamento de sinais desejado.

Atualmente, o termo Redes Neurais Artificiais tende a referir-se majoritariamente aos modelos

empregados em estatística, psicologia cognitiva e inteligência artificial. Modelos de RN cujo objetivo é simular efetivamente o sistema nervoso central são objetos de estudo de neurociência teórica e computacional.

Em implementações de software modernos, a abordagem de RN inspirada na biologia tem sido abandonada a favor de uma mais pragmática, baseada em estatística e processamento de sinais. Em muitos destes sistemas - como propriamente é o caso deste trabalho - as redes neurais são de fato partes integrantes de sistemas maiores, os quais combinam elementos adaptativos e não adaptativos.

Perceptron Multi-Camadas

Neste trabalho, faz-se uso de um tipo de RN denominado "Perceptron Multi-Camadas". É uma RN de alimentação à frente (*i.e.* não existem *loops* e nem estados internos) que mapeia um conjunto de entradas em um conjunto de saídas.

Esse tipo de rede consiste em múltiplas camadas de nós em um grafo direcionado, onde cada camada é totalmente conectada com a camada subsequente. Com exceção dos nós de entrada, cada nó é um neurônio (ou unidade de processamento) com uma função de ativação não-linear. A RN utiliza um sistema supervisionado de treinamento que envolve a retropropagação do erro obtido entre a saída produzida pela RN e a saída desejada [7].

Função de Ativação

Se um Perceptron Multi-Camadas contiver funções de ativação lineares em todos os neurônios, então pode ser facilmente provado - com uso de álgebra linear - que qualquer número de camadas pode ser reduzido ao padrão mais simples de uma única camada intermediária. O que traz a capacidade de aproximação universal a um Perceptron Multi-Camadas é o fato de cada neurônio utilizar uma função de ativação não-linear [8].

As funções de ativação mais importantes na literatura são duas, ambas sigmoidais, descritas como:

$$y_i^{(k)} = \tanh(u_i^{(k)}) \tag{4.1}$$

$$y_i^{(k)} = (1 + e^{-u_i^{(k)}})^{-1} \tag{4.2}$$

onde

$$u_i^{(k)} = \sum_{j=1}^{n} w_{ij}^{(k)} x_j^{(k)} + w_{i0}^{(k)} x_0^{(k)} \tag{4.3}$$

é a ativação interna to i-ésimo neurônio da k-ésima camada, $x_j^{(k)}$ ($j = 1, \cdots, n$) são os sinais de entrada desta camada k e $w_{ij}^{(k)}$ são os respectivos pesos sinápticos. O sinal $x_0^{(k)}$ é um sinal de entrada constante, geralmente igual a 1, e $w_{i0}^{(k)}$ é chamado de peso de polarização.

A Eq. (4.1) trata-se de uma tangente hiperbólica cuja intervalo de excursão está entre -1 e 1. A Eq. (4.2) é equivalente em forma, mas o intervalo de excursão está entre 0 e 1.

Algoritmo de treinamento

O Perceptron Multi-Camadas consiste de três ou mais camadas (uma camada de entrada, uma ou mais camadas intermediárias e uma camada de saída) de nós com funções de ativação não-lineares. Para possibilitar que os sinais de saída não fiquem restritos a um intervalo, pode-se adotar neurônios de saída com função de ativação identidade.

A figura 4.1 exemplifica graficamente uma RN com uma camada intermediária. Cada conjunto de pesos entre camadas é denotado por $w_{ij}^{(1)}$ e $w_{ij}^{(2)}$, respectivamente, para as conexões entrada/camada escondida e camada escondida/saída. Em cada camada, o índice i representa o número de neurônios da camada e o índice j representa o número de entradas da camada. Os neurônios são denotados por i, h e o para entrada, camada escondida e saída, respectivamente.

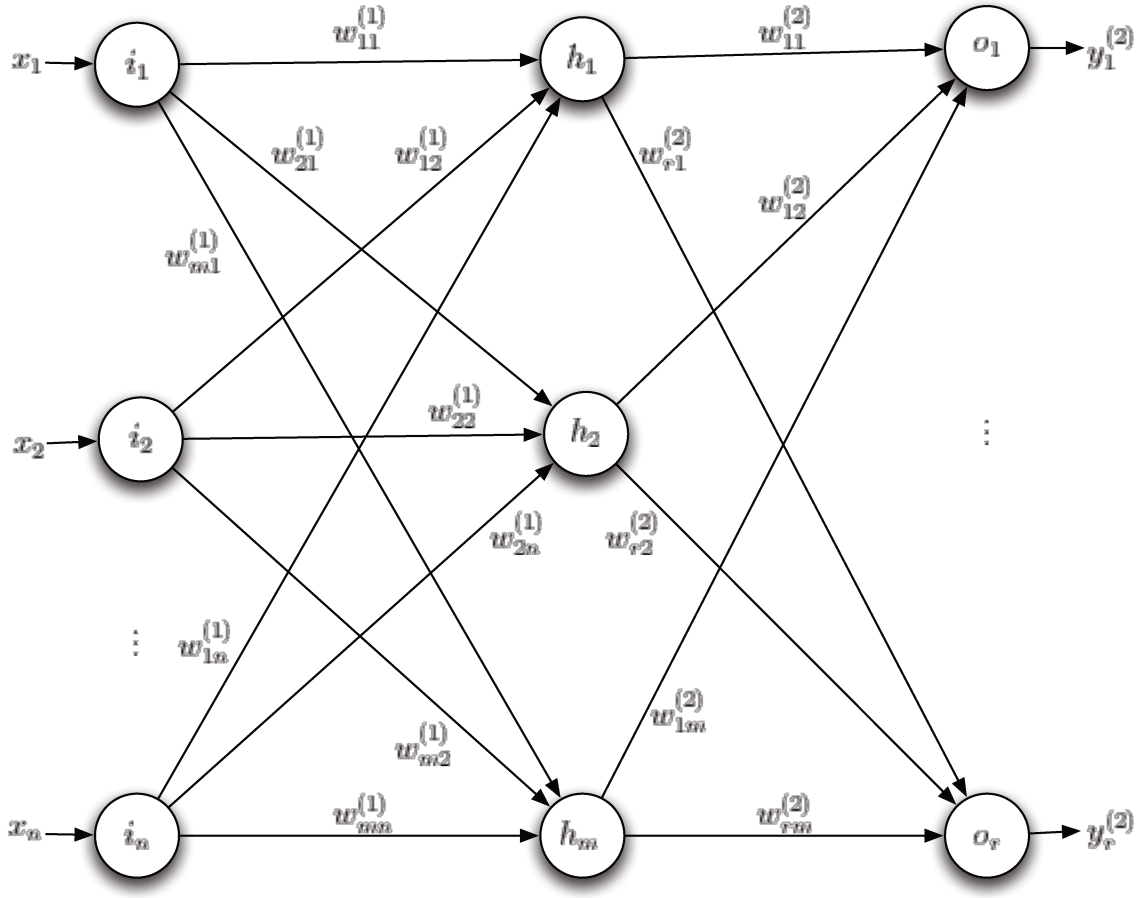

Fig. 4.1: Perceptron Multi-Camadas com três camadas: n entradas, m unidades escondidas e r saídas.

O aprendizado ocorre na rede ao mudar o valor dos pesos depois de cada processamento dos dados, baseado na intensidade do erro da saída comparado ao resultado esperado. Este é um exemplo de aprendizado supervisionado, e ocorre pela aplicação do algoritmo de *backpropagation*, o que permite

a obtenção do vetor gradiente da função de erro em relação aos pesos sinápticos.

Representa-se o erro no neurônio de saída j na n-ésima amostra de dados por:

$$e_j(n) = d_j(n) - y_j^{(2)}(n)$$

onde d_j é o valor esperado (alvo) e $y_j^{(2)}$ é o valor produzido pelo neurônio de saída j. São, então, realizadas correções que minimizam o erro total na saída da rede, dado por:

$$\mathcal{E}(n) = \frac{1}{2}\sum_{j=1}^{r} e_j^2(n).$$

4.3 Função Interpoladora *Wire-Fitting*

Antes de introduzir a arquitetura a ser proposta nesta pesquisa, é importante discorrer brevemente a respeito de uma ideia central a ser incorporada: O interpolador *Wire-Fitting* (WF) [1].

Trata-se de uma função interpoladora, parametrizável por protótipos e modelada para simplificar o cálculo de seu máximo global. Apesar de não ser possível garantir que a função passa por todos os pontos de controle (ou protótipos), é garantido que a função passa pelo ponto de controle máximo (máximo global), e também que a imagem da função tem intervalo aberto entre os pontos de controle mínimo e máximo.

A função WF é calculada como:

$$\hat{w}(a, u\|v) = \frac{\sum_i v_i[\|a - u_i\|^2 + \max_k v_k - v_i]^{-1}}{\sum_i[\|a - u_i\|^2 + \max_k v_k - v_i]^{-1} + \varepsilon} \tag{4.4}$$

onde a constante $\varepsilon \to 0^+$ evita divisões por zero, a é o domínio da função interpoladora resultante e (u_i, v_i) são os pares de pontos de controle. As figuras 4.2 e 4.3 exemplificam graficamente uma plotagem de um segmento de uma função WF contento quatro pontos de controle, ou seja, oito parâmetros. Na figura 4.3, o último ponto é deslocado "para baixo" provocando um máximo em outro ponto.

Como ficará evidente, esse método é importante em um passo específico, porém mandatório, do algoritmo definido e proposto neste trabalho.

Gaskett [6] utiliza WF para criar um algoritmo *Q-Learning* para espaços contínuos. Sua abordagem, porém, não considera elegibilidade. Neste trabalho, propõe-se estender o algoritmo de Gaskett para incorporar elegibilidade em espaços contínuos, resultando em um aprendizado mais rápido.

4.4 Algoritmo Neural-Q(λ)

Como observado na seção 4.2, o desafio no uso de estados contínuos pode ser abordado aproveitando-se do poder generalizador das redes neurais artificiais. Muito embora essas redes sejam adequadas para generalizar o que seriam tabelas infinitas de estados, ainda há dois principais pontos críticos que devem ser estudados para que se obtenha um algoritmo prático de aprendizagem por reforço em ambientes contínuos, a saber:

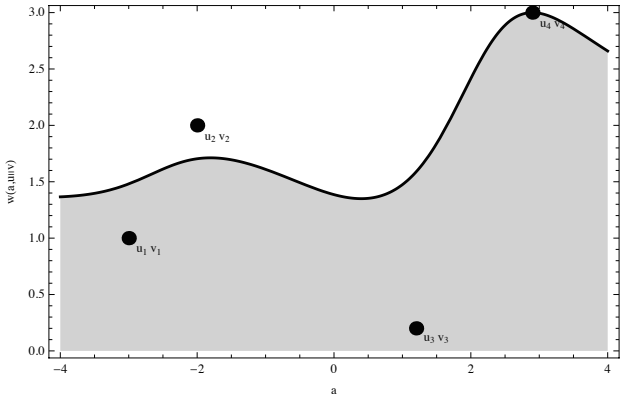

Fig. 4.2: Exemplo de uma função parametrizada por *Wire-Fitting* com quatro protótipos (evidenciados).

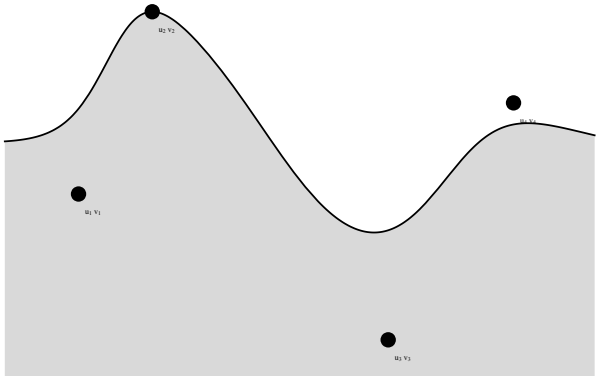

Fig. 4.3: Modificação do exemplo da figura 4.2, com o ponto de controle (u_4, v_4) deslocado para baixo.

1. Elegibilidade: Aplicar este conceito em ambiente contínuo é um desafio, pois, da mesma forma que os estados devem ser generalizados, também é o caso da elegibilidade. Mais ainda, deve haver um mecanismo para "comunicar" os últimos estados alterados. O que é simples a partir de uma tabela de estados, se torna desafiador com uma RN.

2. Aprendizado *Off-Policy*: Como visto anteriormente, é muito desejável que um algoritmo possa aprender uma política enquanto segue outra. Para que isso seja possível em ambientes contínuos, deve haver uma maneira simples de obter a informação da melhor ação dado um estado. Novamente, esta tarefa se torna desafiadora com uma RN. Como será mostrado, é justamente neste ponto que a função WF mostra sua contribuição.

Elegibilidade em ambientes contínuos

Para que a elegibilidade funcione, a estrutura de elegibilidade ($e(s,a)$) deve ser homomórfica à própria estrutura dos estados (RN), ou seja, $e(s,a)$ deve ser - como a rede - um vetor paramétrico.

Em outras palavras, $e(s,a)$ deve assumir a mesma estrutura paramétrica de $Q(s,a)$. $e(s,a)$ deve ser atualizado proporcionalmente à sensibilidade do par (s,a) (estado, ação) à mudança. Mais especificamente, o erro de diferença temporal (δ) é o gradiente dos parâmetros de $Q(s,a)$ para um par (estado, ação) específico.

Estas ideias são eloquentemente delineadas nas equações de Doya [5], apresentadas aqui. Atualizando nossa notação, utiliza-se $\hat{Q}$ para representar a função aproximadora de Q. Como será visto, a estrutura de $\hat{Q}$ é uma composição de uma RN e uma função WF.

A atualização da elegibilidade, como já discutido, é feita em duas partes: o decaimento (primeira equação) e a atualização do estado corrente (segunda equação). Nesta última, a diferença é que e é efetivamente uma média móvel exponencial do gradiente de $\hat{Q}$ nos pares (estado, ação) recentemente visitados:

$$e \leftarrow \gamma\lambda e$$

$$\Delta e = \nabla_\theta \hat{Q}(s_t, a_t)$$

onde θ é o vetor de parâmetros de $\hat{Q}$ (função aproximadora).

O cálculo do erro de diferença temporal pode ser expresso de maneira similar ao que foi apresentado até o momento. Note que utiliza-se Δt para denotar um intervalo de tempo arbitrariamente pequeno (*i.e.* o tempo é discretizado):

$$a^* = \arg\max_a \hat{Q}(s_{t+\Delta t}, a) \tag{4.5}$$

$$\delta = \nabla_\theta [r_{t+\Delta t} + \gamma \hat{Q}(s_{t+\Delta t}, a^*) - \hat{Q}(s_t, a_t)]^2$$

onde a^* é a melhor ação.

A atualização da função valor (*i.e.* vetor paramétrico θ da função aproximadora) é escrita simplesmente em termos da constante de aprendizagem α, o erro de diferença temporal δ e e, o gradiente de elegibilidade:

$$\Delta\theta = \alpha\delta e$$

Off-Policy em ambientes contínuos

Estas equações são precisamente a proposta de Doya, com exceção do cálculo da melhor ação a^*, na equação (4.5). Na proposta original, este valor é baseado em uma política *on-policy*. Enquanto que, neste trabalho, é buscado o desafio de incorporar uma política *off-policy*.

O desafio se situa no fato de que é requerido que a função aproximadora $\hat{Q}$ permita um cálculo de máximo global pouco custoso para uma variável. Redes neurais perceptron multi-camadas infelizmente não têm essa propriedade. Para calcular o máximo de uma variável livre é necessário cálculos de derivada de segunda ordem que se tornariam custosos computacionalmente.

Tipicamente, a ordem de complexidade, neste caso, é quadrática dado o número de parâmetros, e se tornaria pior caso implementada aninhada a um ciclo de aprendizado.

Para resolver esse problema, nossa função aproximadora é de fato uma composição de dois modelos matemáticos: RN e WF. Tal composição foi proposta por Gaskett *et. al.* [6] e é apresentada na equação (4.6):

$$\hat{Q}(s,a) \equiv \hat{w}(a, \hat{g}(s,\theta)) \tag{4.6}$$

onde $\hat{w}$ é a função WF, $\hat{g}$ é uma RN perceptron multi-camadas e θ é o vetor paramétrico da RN (*i.e.* os pesos da RN). A Figura 4.4 mostra um diagrama equivalente à definição acima.

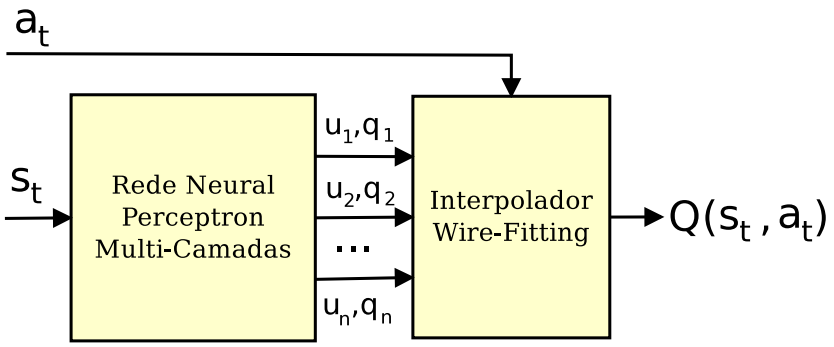

Fig. 4.4: Arquitetura "*Wire-Fitting-Neural-Q*"

O cálculo do valor da ação correspondente ao máximo de $\hat{Q}$ é simplificado pelo uso do componente WF, diagramado na Figura 4.5. O cálculo da ação máxima (ou valor da ação máxima) é realizado em duas etapas:

1. O estado s alimenta a rede neural (esquerda da figura) e tem-se a saída da rede, calculada como os parâmetros da função WF.

2. Dado que os parâmetros da função WF estão definidos, o máximo global desta função é simplesmente o parâmetro com valor máximo. Logo, o valor da ação máxima é o valor correspondente ao parâmetro máximo, e a ação que maximiza a função é justamente o valor do parâmetro.

A atualização dos parâmetros é feita através da propagação de erro, ilustrado na Figura 4.6.

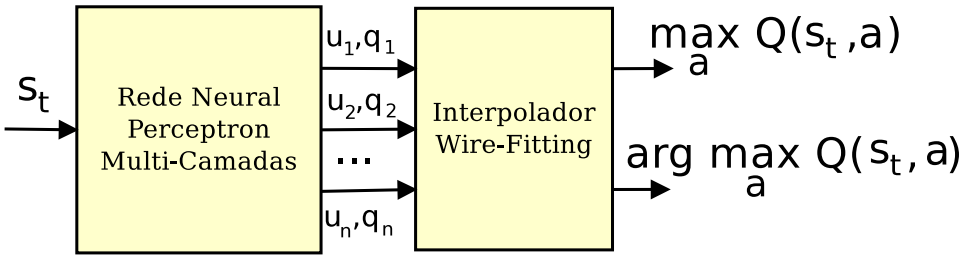

Fig. 4.5: Arquitetura "*Wire-Fitting-Neural-Q*" no cálculo da ação máxima

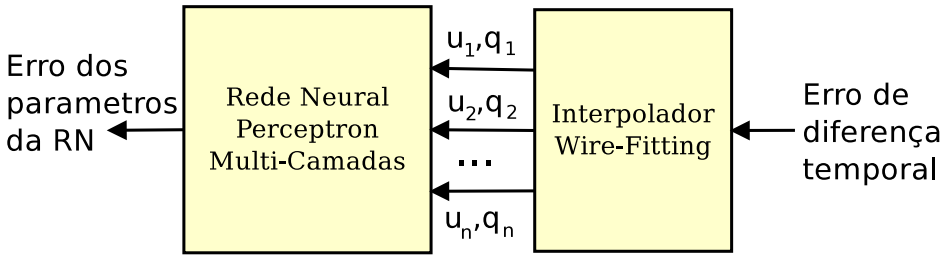

Fig. 4.6: Arquitetura "*Wire-Fitting-Neural-Q*" na propagação de erro

O ponto importante a observar é que a saída da RN é um vetor de parâmetros da função WF, e que a entrada do valor da ação é separada - independente da RN - e alimenta diretamente a função WF. Tal configuração permite o cálculo simples do máximo desta função.

A contribuição desta pesquisa envolve primariamente o algoritmo proposto, que insere a arquitetura apresentada dentro do contexto de aprendizagem por reforço em ambientes contínuos. Ele é denominado "Neural-Q(λ)" e é apresentado a seguir (Alg. 11). Pretende-se empregá-lo, mais adiante, na solução de alguns problemas de aprendizagem por reforço para controle automático.

Inicialize θ arbitrariamente.
para cada *episódio* **faça**
  Inicialize s (estado inicial)
  $a \leftarrow$ ação para s utilizando uma política derivada de $\hat{Q}$
  repita
    Efetue a, observe a recompensa r e próximo estado s'
    $a' \leftarrow$ ação para s' utilizando uma política derivada de $\hat{Q}$
    $a^* \leftarrow \arg\max_a \hat{Q}(s', a)$
    $\delta \leftarrow \nabla_\theta [r + \gamma \hat{Q}(s', a^*) - \hat{Q}(s, a)]^2$
    $\theta \leftarrow \theta + \alpha \delta e$
    $e \leftarrow e + \nabla_\theta \hat{Q}(s, a)$
    $e \leftarrow \exp[-\beta(a^* - a')^2] \gamma \lambda e$
    $s \leftarrow s'$, $a \leftarrow a'$
  até s *ser terminal*
fim

Algoritmo 11: Neural-Q(λ)

A função de base radial $\exp[-\beta(a^* - a')^2]$ é uma versão contínua da ideia de Watkins, no algoritmo de Q-(λ) [2], para anular o traçado de elegibilidade quando a melhor ação não é executada. O seu papel é promover uma redução continuada da participação do termo $\gamma\lambda$ (traçado de elegibilidade) conforme a' se afasta de a^*. A taxa de redução é ditada por β.

Nos algoritmos *off-policy*, o traçado de elegibilidade é simplesmente anulado quando a ação efetiva (a^*) é diferente da indicada (a'). Aqui, a elegibilidade é gradualmente anulada pela gaussiana da expressão, quanto mais a ação efetiva se afasta da indicada. O formato da função de base radial pode ser visto na Fig. 4.7.

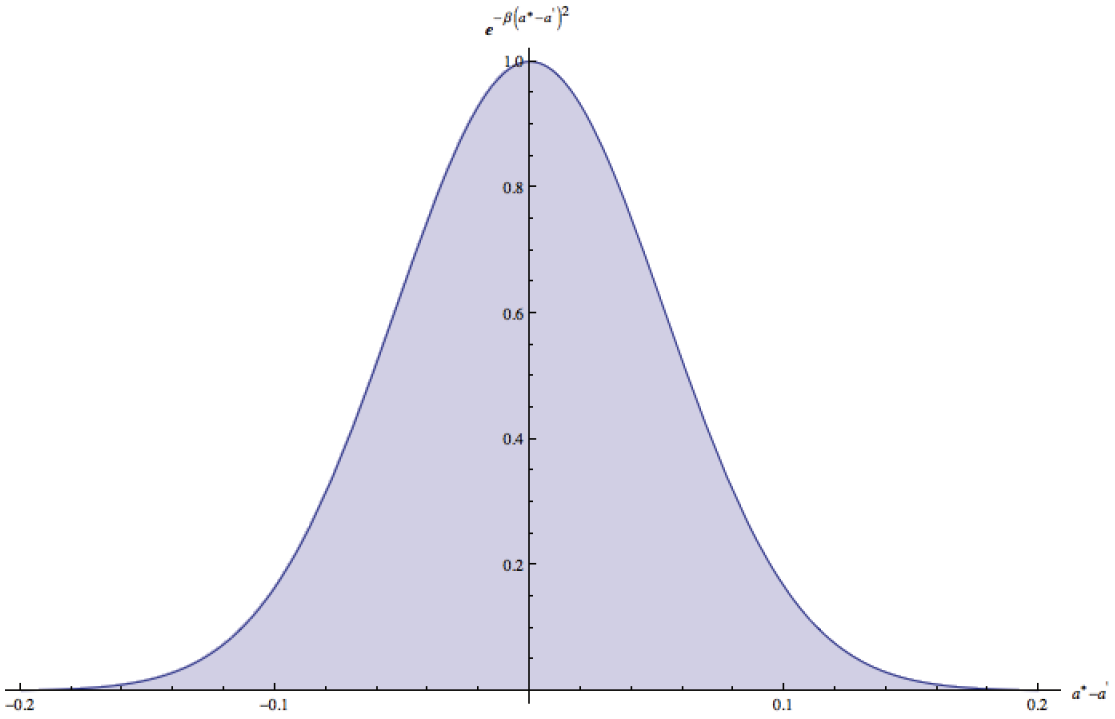

Fig. 4.7: Forma do ajuste da elegibilidade em função da diferença entra a ação ótima e a ação efetuada, para $\beta = 180$.

Capítulo 5

Experimentos

5.1 Problema Proposto

Nas próximas seções, aplica-se o algoritmo proposto no capítulo 4, o qual é referenciado no texto pelo nome Neural-Q(λ). Antes de apresentar propriamente os resultados dos experimentos, serão introduzidos os sistemas e suas correspondentes formulações matemáticas dentro do paradigma de aprendizagem por reforço.

São dois problemas clássicos em teoria de controle: Pêndulo Invertido e Carro na Montanha. Ambos são problemas de visibilidade total, com duas variáveis de estado contínuas e uma variável de ação, também contínua.

5.1.1 Pêndulo Invertido

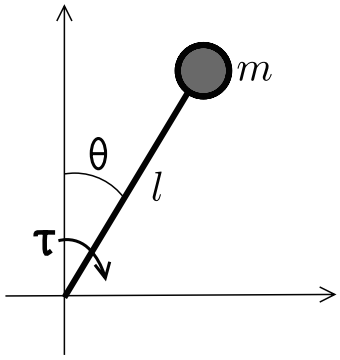

Fig. 5.1: Representação diagramática do problema do Pêndulo Invertido

O Pêndulo Invertido é um problema clássico em teoria de controle e dinâmica e é comumente usado na literatura como *benchmark* de teste para algoritmos de controle. Variações deste problema existem, e podem incluir múltiplas hastes e também uma base móvel. Entretanto, neste trabalho faz-se uso de sua instância mais simples, a qual contém uma base fixa com um motor de torque (capaz de girar o pêndulo) e a massa concentrada na esfera (vide figura 5.1).

O objetivo é construir um controlador que lê posição angular e velocidade angular e aplica um torque a fim de manter o pêndulo em equilíbrio com a haste na vertical.

Para o problema se tornar desafiador, o torque máximo aplicável pelo motor não pode ser superior ao torque produzido pelo peso da esfera, quando a haste está na posição horizontal. Desta forma, o controle deverá construir *momentum* através de um movimento similar ao de uma criança quando acumula *momentum* em um balanço de parque, para finalmente ser capaz de atingir (e manter) a posição de equilíbrio vertical (razão do nome "Pêndulo Invertido").

As equações que regulam a dinâmica do sistema são:

$$ml^2\dot{\omega} = -\mu\omega + mglsin\theta + \tau$$

$$\omega = \dot{\theta}$$

onde θ e ω são o ângulo e a velocidade angular, respectivamente. Estas duas variáveis compõem o vetor de estado. A ação é um escalar, τ, cujo valor é o torque aplicado ao pêndulo, e é limitado no intervalo aberto $-4,9 \leq \tau \leq 4,9$. Qualquer resultado fora deste intervalo é truncado.

$\dot{\theta}$ e $\dot{\omega}$ são as derivadas temporais do ângulo e velocidade angular, respectivamente. A dinâmica é implementada em uma simulação computacional através do uso do método de Euler para a solução de equações diferenciais ordinárias (ODE) cujos valores iniciais são ($\omega = 0$, $\theta = \pi$), e usando $\Delta t = 0,1$ s. como tamanho do passo de tempo discretizado. As condições iniciais indicam que o pêndulo parte do repouso a 180° da posição desejada.

Por tratar-se de um problema de aprendizagem por reforço, modela-se também a função de recompensa imediata, dada pela seguinte função degrau:

$$r = \begin{cases} 1 & |\theta| < \frac{\pi}{12} \\ -1 & |\theta| \geq \frac{\pi}{12} \end{cases} \tag{5.1}$$

em oposição a algumas abordagens que adotam $cos(\theta)$. Com a função de recompensa da equação 5.1, dificulta-se o aprendizado ao não permitir que seja embutida implicitamente a informação de valor real do objetivo.

5.1.2 Carro na Montanha

O problema do Carro na Montanha tem suas similaridades com o já apresentado Pêndulo Invertido. Como o anterior, também tem duas variáveis reais de estado e uma de ação, o estado do sistema é completamente observável e, como o anterior, é considerado um clássico *benchmark* para algoritmos de controle em geral.

Neste problema, o objetivo é levar um carro, partindo do fundo de um vale até o topo de um dos montes (no caso do diagrama da Fig. 5.2, o direito). O carro tem um motor que pode acelerar para frente ou para trás (aceleração negativa, ou "ré"). Porém, para tornar o problema desafiador e interessante, o motor, mesmo a 100%, não é capaz de vencer a ação da gravidade no trecho mais íngreme. Logo, recorrendo-se apenas à força do motor e supondo energia cinética baixa ou nula, não é possível chegar ao topo (solucionando o problema) apenas acelerando para frente (veja a figura 5.2).

A solução deve, analogamente ao pêndulo, produzir um efeito de balanço para vencer a subida íngreme, finalmente alcançando o topo. As equações de dinâmica do sistema são dadas por:

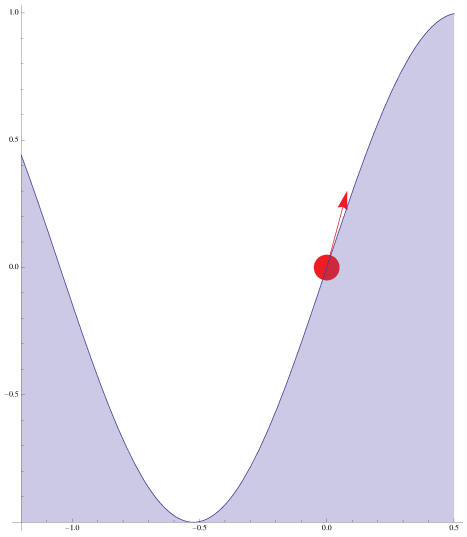

Fig. 5.2: Representação diagramática do problema do Carro na Montanha

$$\dot{x} = v$$
$$\dot{v} = \frac{1}{1000}a - \frac{1}{400}cos(3x)$$

onde x e v são a posição horizontal e velocidade, respetivamente. $\dot{x}$ e $\dot{v}$ são a variação da posição e a variação da velocidade, respectivamente. E a é a aceleração exercida pelo motor do veículo.

Existem algumas restrições do sistema, a saber: a posição é limitada ao intervalo $-1,2 \leq x \leq 0,5$, a velocidade é limitada ao intervalo $-0,07 \leq v \leq 0,07$, e a aceleração do carro obedece $-1 \leq a \leq 1$. Valores fora do intervalo são truncados. A limitação do valor de velocidade no intervalo é uma aproximação ao efeito de atrito.

A equação que define a recompensa imediata r é dada por:

$$r = \begin{cases} -1 & x \leq -1,2 \\ 1 & x \geq 0,5 \\ 0 & \text{caso contrário} \end{cases}$$

Um ponto distinto desta proposta reside no final do episódio: enquanto que no pêndulo cada episódio é limitado por um tempo máximo, no caso do Carro da Montanha, além do limite de tempo, o episódio termina quando uma recompensa diferente de zero é atingida (*i.e.* $x = -1,2$ ou $x = 0,5$).

5.2 Resultados

5.2.1 Resultados do Problema do Pêndulo Invertido

Para este problema, o algoritmo foi ajustado com algumas constantes fixas.

A primeira delas, Δt, é a discretização do tempo para a simulação física pelo método de Euler. Valores maiores podem causar inexatidões na simulação. Por outro lado, valores muito pequenos podem consumir desnecessariamente recursos computacionais. O valor determinado aqui é, para os propósitos deste experimento, suficientemente pequeno.

A segunda constante a ser considerada é γ: o desconto a ser atribuído no cálculo da diferença temporal e da elegibilidade. O efeito intuitivo deste valor é o "inverso da taxa de esquecimento", ou "importância" dos eventos de recompensa imediata. Um valor mediano é bastante apropriado, muito embora, em teoria, qualquer valor no intervalo aberto entre zero e um é operável, embora o tempo de aprendizagem pode se tornar impraticável.

Finalmente, o terceiro valor a ser considerado, ao qual ainda não foi dado o devido foco, é o parâmetro β do algoritmo, utilizado no cálculo de uma curva radial (normal) cujo propósito é delimitar a similaridade entre uma ação e outra.

Em certo momento do algoritmo, o traçado de elegibilidade deve ser zerado caso a ação tomada não seja a ótima. Isto porque, se o agente toma uma ação exploratória (ao invés da ótima calculada), a elegibilidade a partir daquele momento não faz mais sentido, devido ao fato da ação tomada não ser consequência direta da noção de valor do agente, e sim de uma decisão aleatória.

Esta lógica é simples para ambientes com ações discretas. Porém, neste caso, como a ação é contínua, também há uma gradação (pela curva normal) do quanto o traçado de elegibilidade é eliminado. Intuitivamente, quanto mais distante uma ação é tomada da ação considerada ótima, mais distante do centro da normal, portanto, mais próximo de zero é a participação do traçado de elegibilidade. O valor a ser determinado, neste caso, deve ser granular o suficiente para fazer sentido dentro do espaço de ações cabíveis. Em teoria, também, qualquer valor é aceitável, embora algumas escolhas podem levar a dificuldades práticas.

- $\Delta t = 0,1$ s., o intervalo de tempo discretizado.
- $\gamma = 0,5$, o fator de desconto.
- $\beta = 180$, o parâmetro de tolerância da função radial (veja figura 4.7) para comparação entre ações.

A rede neural (RN) utilizada possui três camadas, com dois neurônios na entrada (um para cada dimensão da entrada), três neurônios intermediários e seis neurônios na saída, correspondendo a três pares de parâmetros regulando os pontos de controle da função *Wire-Fitting* (WF).

Para efeitos de comparação, e também para reforçar a tese deste trabalho, foram realizados experimentos com cinco valores distintos para a constante de elegibilidade λ, a saber: 0; 0,5; 0,75; 0,9 e 1,0. O experimento foi executado 50 vezes para cada um dos valores de λ. Logo, o resultado apresentado adiante é a média de 50 experimentos. Cada episódio de aprendizado foi configurado para levar 20 segundos em tempo de simulação - onde o controle tenta conduzir o pêndulo e simultaneamente aprender - de fato, com o valor de Δt ajustado em 0,1; haverá 200 passos de aprendizagem por episódio.

A política derivada de Q é uma versão simples e contínua do correspondente discreto ϵ-guloso. Nesta versão, as ações são escolhidas de uma distribuição normal com média na ação de maior valor tomada anteriormente. A variância foi fixada arbitrariamente em 0,15.

O algoritmo, posteriormente à realização dos experimentos descritos, foi ligeiramente modificado para uma versão *on-policy*. Ou seja, a diferença temporal é calculada baseando-se na ação tomada (não na ação ótima). Em substituição à ação de valor máximo, a próxima ação selecionada pela política é tomada, de modo que o aprendizado dependa da política sendo seguida. Outra mudança é a inexistência do fator $\exp[-\beta(a^* - a')^2]$, pois a atualização é baseada sempre na mesma ação. Este procedimento diferenciado é apresentado no Algoritmo 12.

Inicialize θ arbitrariamente.
para cada *episódio* **faça**
  Inicialize s (estado inicial)
  $a \leftarrow$ ação para s usando uma política derivada de $\hat{Q}$
  repita
    Tome a ação a, observe a recompensa r e o próximo estado s'
    $a' \leftarrow$ para s' usando uma política derivada de $\hat{Q}$
    $\delta \leftarrow \nabla_\theta [r + \gamma \hat{Q}(s', a') - \hat{Q}(s, a)]^2$
    $\theta \leftarrow \theta + \alpha \delta e$
    $e \leftarrow e + \nabla_\theta \hat{Q}(s, a)$
    $e \leftarrow \gamma \lambda e$
    $s \leftarrow s'$, $a \leftarrow a'$
  até s *é terminal*
fim

Algoritmo 12: Neural-Q(λ), versão *on-policy*

É importante enfatizar que esta versão *on-policy* do algoritmo é particularmente um desperdício de recursos. Note que o objetivo da arquitetura proposta (rede neural + *Wire-Fitting*) é tornar pouco custoso o cálculo do máximo global. Quando não é necessário realizar esse cálculo, toda a estrutura torna-se obsoleta. Porém, para propósitos de pura comparação de resultados, as modificações no algoritmo foram apenas as mínimas necessárias para adaptá-lo ao paradigma *on-policy*.

A Tabela 5.1 mostra os resultados compilados para cada versão do algoritmo *Neural-Q(λ)* (*off-policy* e *on-policy*). Os números são uma média de 50 experimentos, cada um com episódios de 20 segundos em tempo de simulação. Eles representam o número de episódios de treinamento até que o controlador seja capaz de manter o pêndulo equilibrado durante 10 segundos.

Tab. 5.1: Resultados para a média de episódios antes de atingir o objetivo de 10 segundos consecutivos com o Pêndulo Invertido equilibrado

versão	$\lambda = 0$	$\lambda = 0.5$	$\lambda = 0.75$	$\lambda = 0.9$	$\lambda = 1.0$
off-policy	338,6	237,2	198,7	190,2	227,8
on-policy	517,1	454,8	403,6	374,3	399,4

Naturalmente, percebe-se pela Tabela 5.1 que, para este conjunto de testes e constantes, o algo-

ritmo *off-policy* apresentou um desempenho melhor em geral (menos é melhor). Estes resultados, portanto, representam uma forte motivação para a investigação mais profunda do *framework off-policy*. A respeito da elegibilidade, é possível concluir que os melhores resultados localizam-se próximos de 0,9, o que indica claramente que o uso de um valor considerável de elegibilidade realmente melhora a qualidade do treinamento, quando comparado a resultados em que a elegibilidade não é usada ($\lambda = 0$ (diferença temporal (DT)) e $\lambda = 1$ (Monte Carlo (MC))).

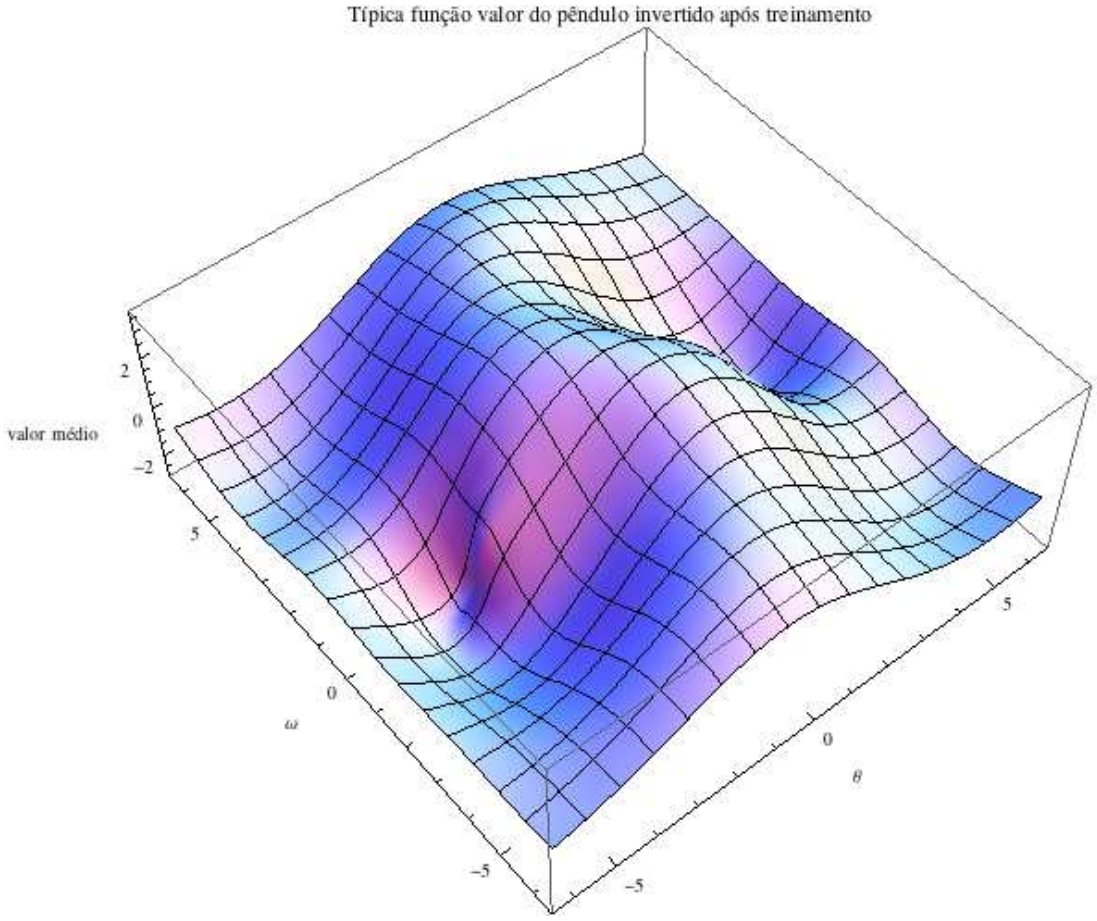

Fig. 5.3: Representação gráfica de uma função valor Q típica após intenso treinamento para o problema do Pêndulo Invertido.

A Figura 5.3 representa o gráfico da função Q no intervalo do escopo do problema, após intenso treinamento. Pode-se perceber que os gradientes indicam as decisões do controlador quando em determinados estados. A Figura 5.4 mostra um exemplo de caminho de subida de gradiente durante um episódio típico, posteriormente a intenso treinamento.

A Figura 5.5 mostra a evolução de um experimento de treinamento durante quinhentos episódios. A evolução é denotada em tempo (em segundos simulados) que o controlador levou para conseguir equilibrar o pêndulo na posição vertical desejada. Cada episódio tem 20 segundos. Observa-se na Figura 5.6 que, mesmo bastante treinado, o controle precisa de aproximadamente 9,5 segundos, depois de treinado, em média, para conseguir levar o pêndulo para a posição invertida.

Com base no seu conhecimento de valor (veja Fig. 5.3) é que o controlador toma as decisões

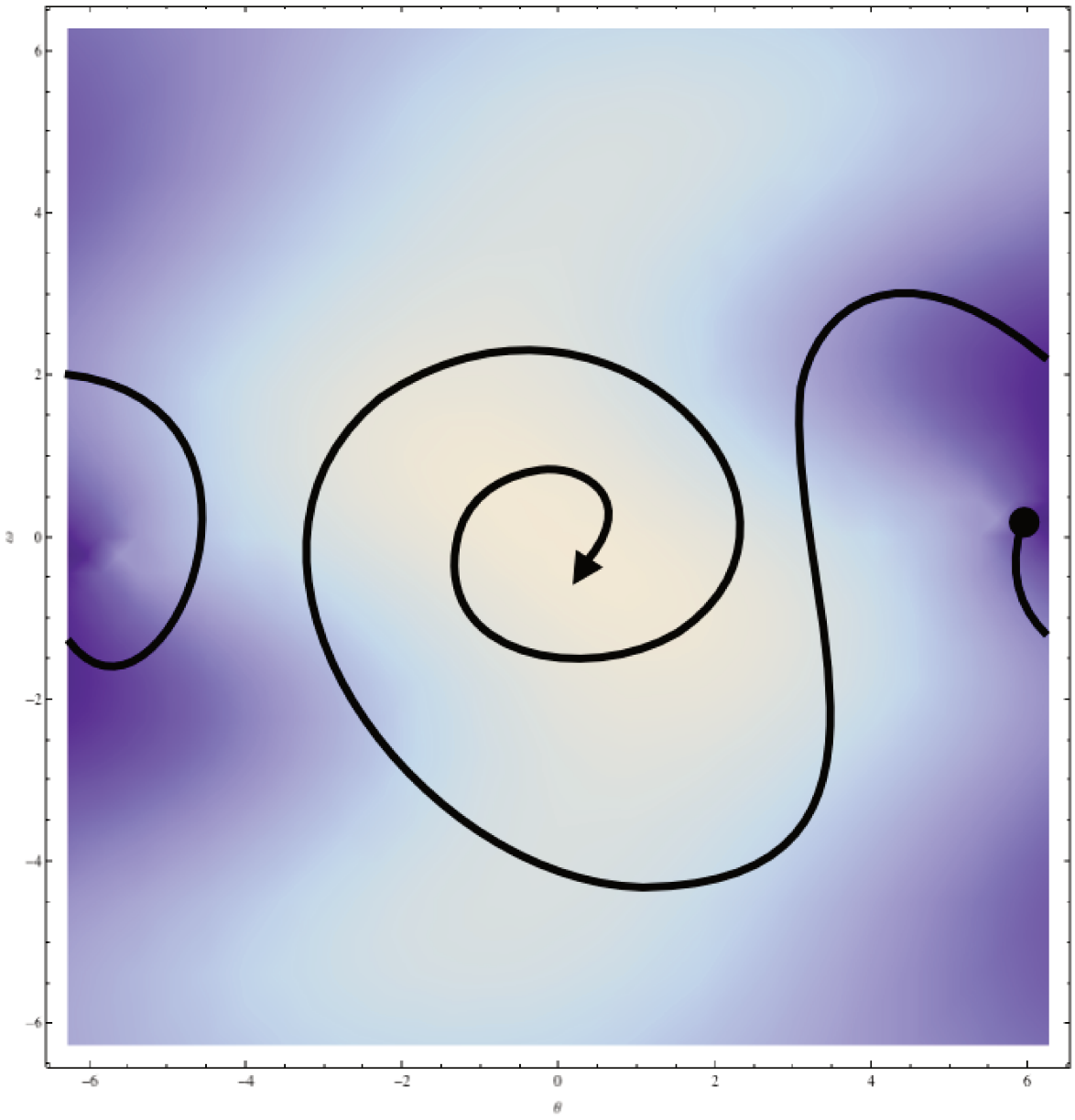

Fig. 5.4: Exemplo de caminho de subida no gradiente, sob uma função valor Q, durante um episódio típico posterior a intenso treinamento.

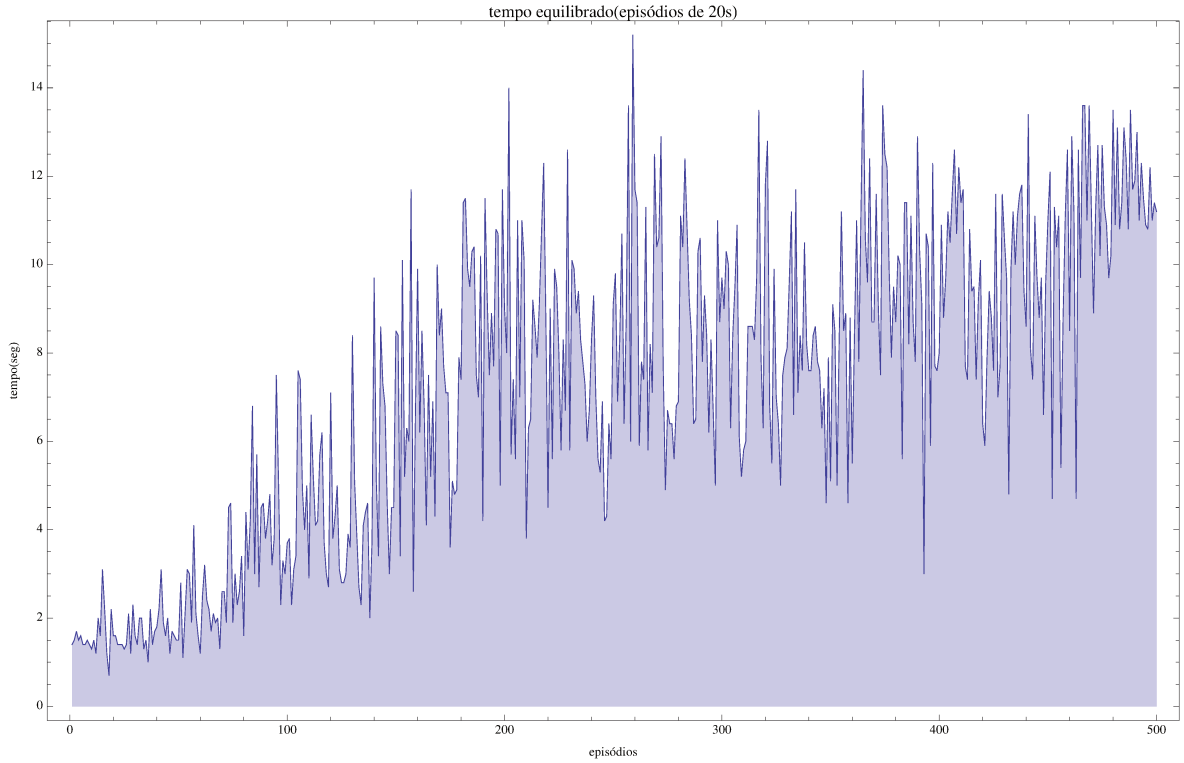

Fig. 5.5: Gráfico do tempo equilibrado (em segundos simulados) para cada episódio de treinamento do Pêndulo Invertido, durante 500 episódios, com episódios de 20 segundos.

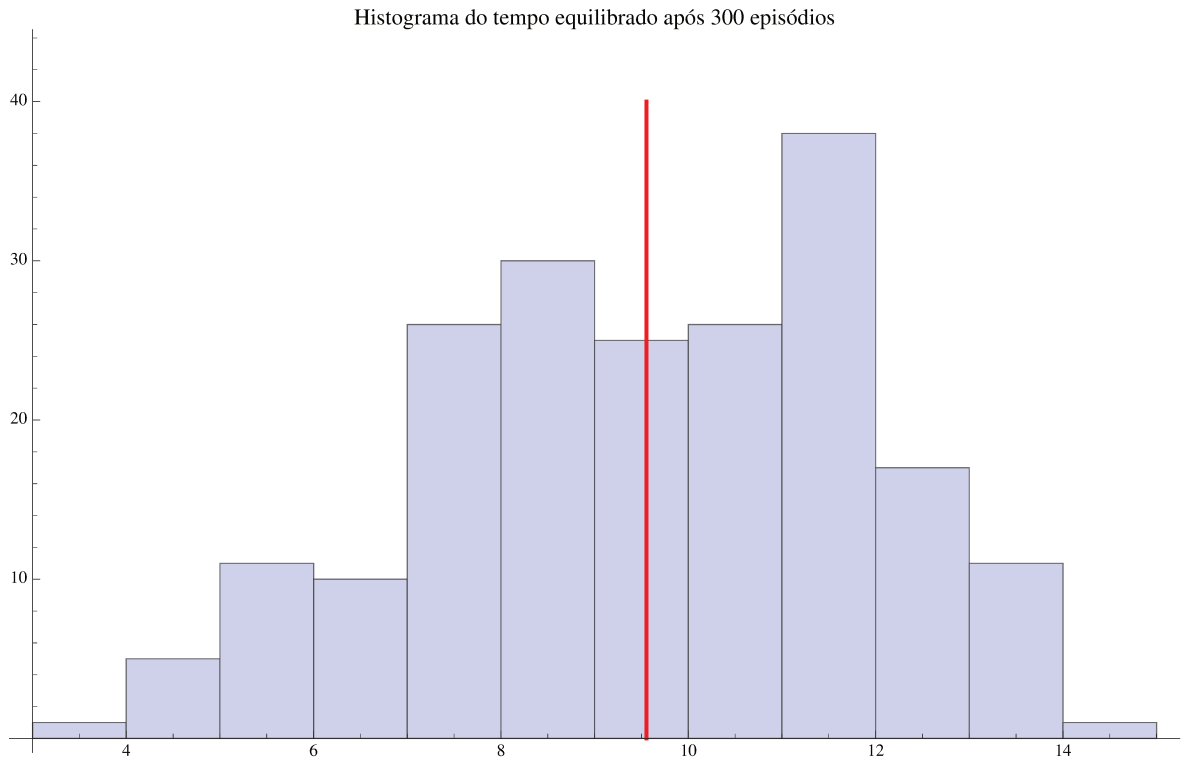

Fig. 5.6: Histograma e média (linha vermelha) do tempo equilibrado do pêndulo após 300 episódios de treinamento, onde é possível discernir uma convergência.

de controle (torque). O gráfico de controle típico depois de um aprendizado do pêndulo invertido é apresentado na Figura 5.7.

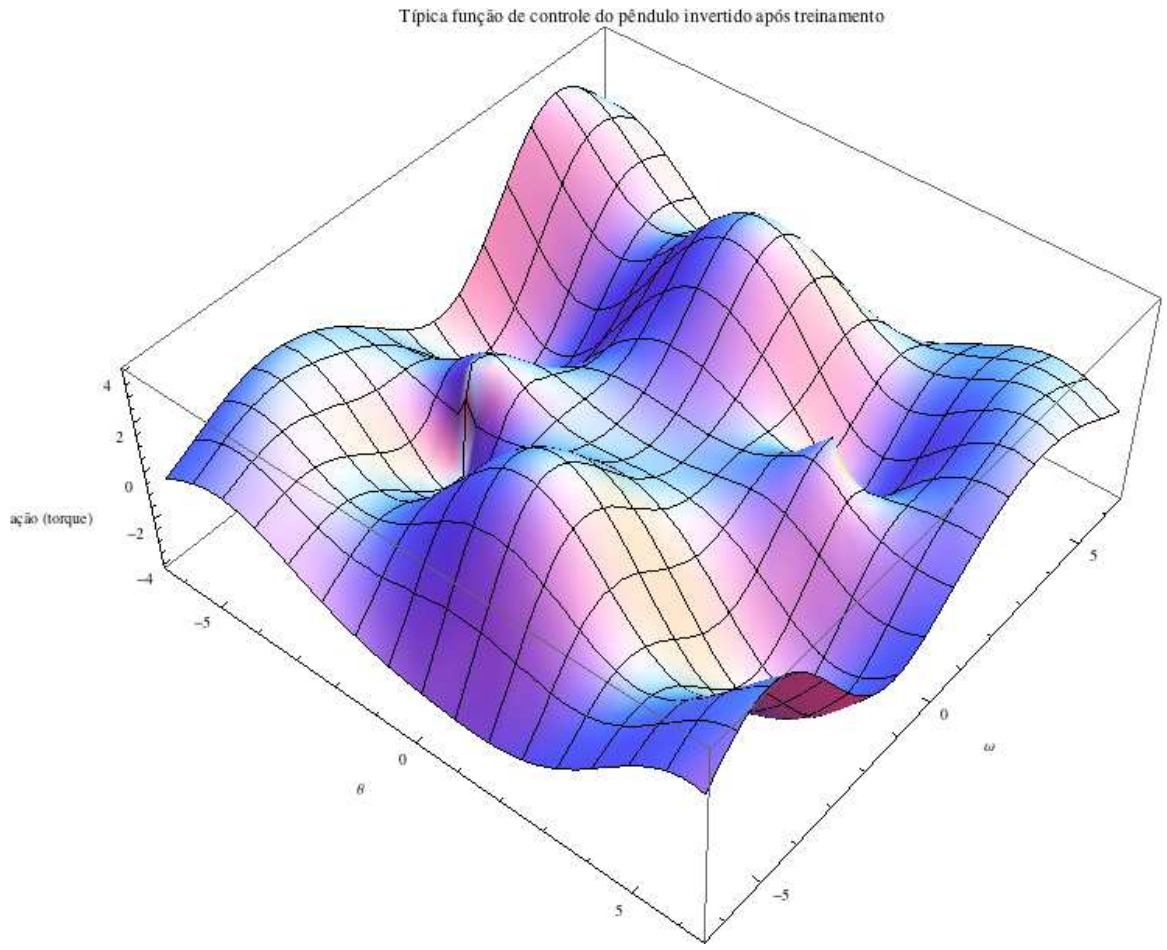

Fig. 5.7: Representação gráfica de uma função de ação típica após intenso treinamento para o problema do Pêndulo Invertido.

5.2.2 Resultados para o problema do Carro na Montanha

O problema do carro na montanha, em certos aspectos, é bem similar ao problema anterior do pêndulo invertido. Entre as similaridades, pode-se citar o fato de que ambos correspondem a um problema com duas variáveis reais, cujo valor é completamente conhecido e sem ruído.

O tempo máximo do episódio é de 20 segundos. Porém, ele pode terminar caso o carro receba uma recompensa diferente de zero (positiva ou negativa).

As dinâmicas envolvidas, obviamente, são distintas, muito embora utiliza-se neste experimento exatamente a mesma configuração para as constantes já anteriormente descritas:

- $\Delta t = 0,1$ s., o intervalo de tempo discretizado.
- $\gamma = 0,5$, o fator de desconto.

- $\beta = 180$, o parâmetro de tolerância da função radial para comparação entre ações.

A razão disso é defender a tese de que o algoritmo proposto tem poderes de generalizar e ser aplicado a problemas diferentes. O resultado para esta instância, apesar de bem sucedido, apresentou uma performance inferior à instância do pêndulo.

A topologia da RN utilizada foi a mesma para o Pêndulo: três camadas com dois neurônios na camada de entrada, três na intermediária e seis na saída.

A Tabela 5.2 mostra os resultados compilados para cada versão do algoritmo *Neural-Q(λ)* (*off-policy* e *on-policy*). Os números são uma média de 50 experimentos, cada um com, no máximo, episódios de 20 segundos em tempo de simulação. Eles representam o número de episódios de treinamento até que o controlador fosse capaz de chegar em até 10 segundos à posição de sucesso (extrema direita). Note que o critério aqui é diferente do apresentado no problema do pêndulo.

Tab. 5.2: Resultados para a média de episódios antes de atingir o objetivo de se alcançar a posição de sucesso em até 10 segundos

versão	$\lambda = 0$	$\lambda = 0.5$	$\lambda = 0.75$	$\lambda = 0.9$	$\lambda = 1.0$
off-policy	362.3	260.9	222.4	213.9	251.5
on-policy	557.4	495.1	443.9	414.6	439.7

Um resultado surpreendente que logo se observa é, novamente, o valor de λ em torno de 0,9 ter tido o melhor desempenho em ambos os tipos de algoritmos. Além disso, novamente, o algoritmo *off-policy* apresentou um melhor desempenho. Estes resultados evidenciam o potencial da abordagem *off-policy*, além de um valor experimental satisfatório para λ.

As Figuras 5.8 e 5.9 apresentam, para um treinamento típico de vários episódios, a função valor Q e a função de ação (derivada da função valor Q), respectivamente.

A Figura 5.10 mostra a evolução de um experimento de treinamento durante quinhentos episódios. A evolução é denotada em tempo (em segundos simulados) necessários para que o controlador consiga levar o carro até a posição desejada. Notadamente, há uma evolução. Entretanto, aproxima-se de um limite onde é sempre necessário um tempo mínimo para se acumular *momentum* a fim de atingir o objetivo. Cada episódio tem no máximo 20 segundos. Portanto, os pontos que atingem esse limite indicam que, naquele episódio, o controlador não conseguiu levar o carro até o objetivo.

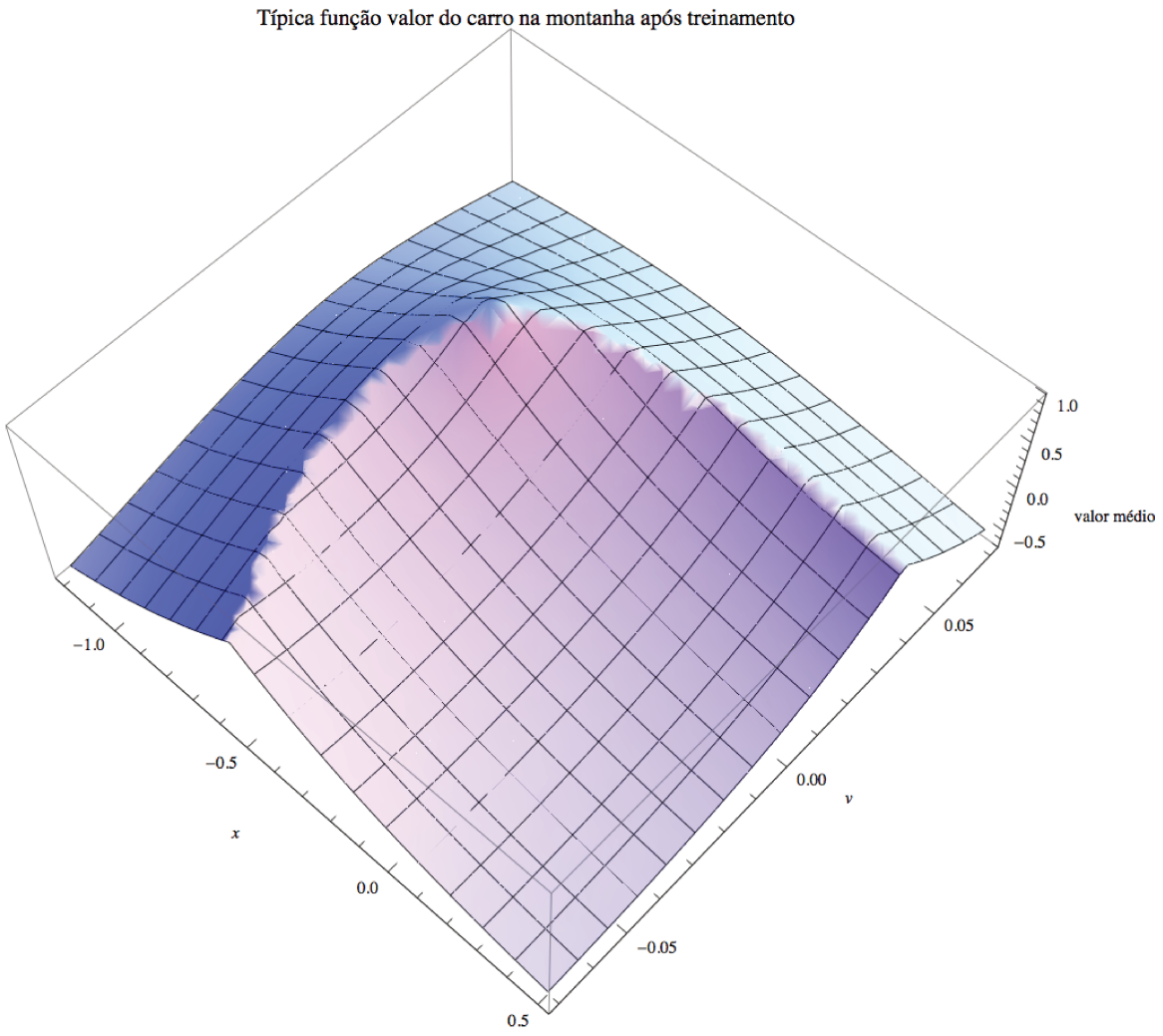

Fig. 5.8: Representação gráfica de uma função valor Q típica após intenso treinamento para o problema do Carro na Montanha.

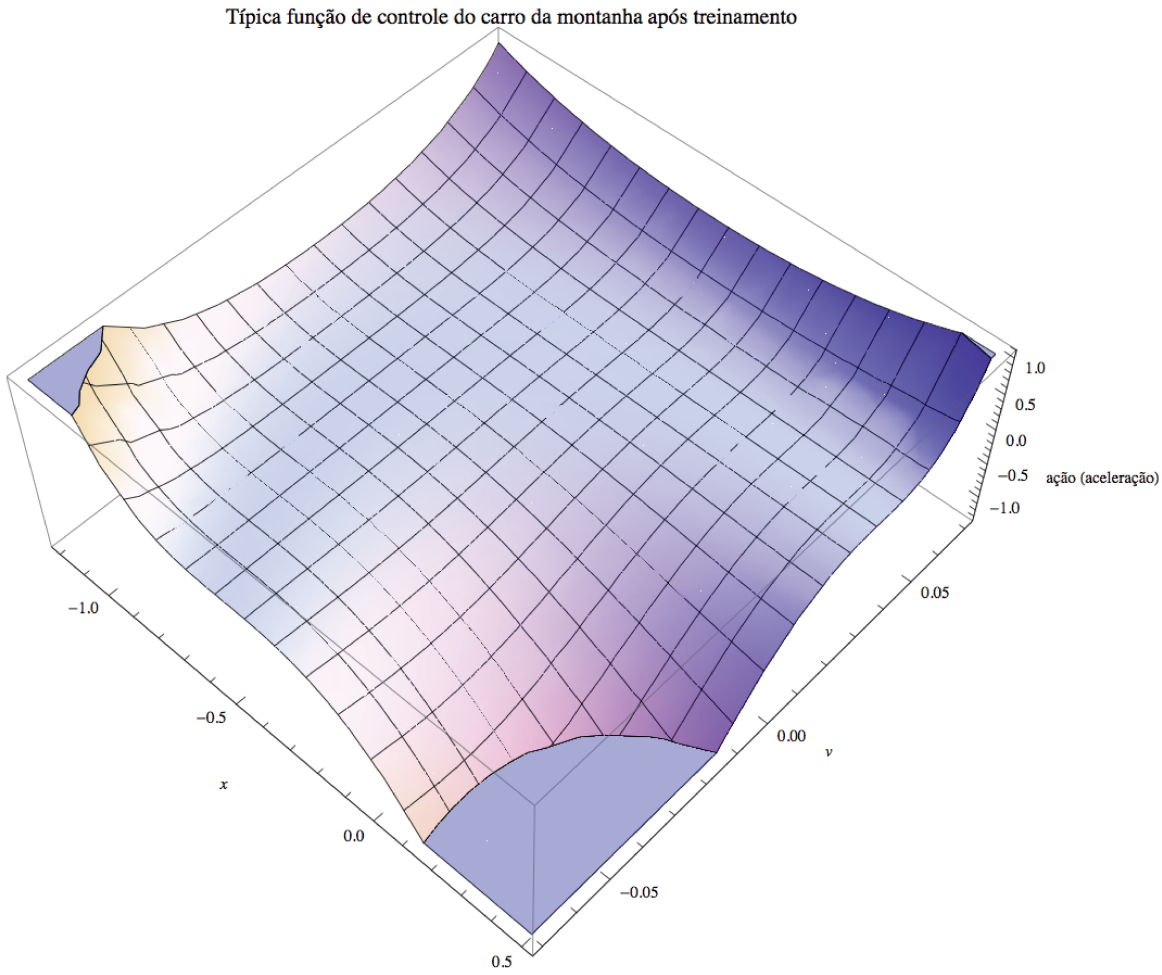

Fig. 5.9: Representação gráfica de uma função de ação típica após intenso treinamento para o problema do Carro na Montanha.

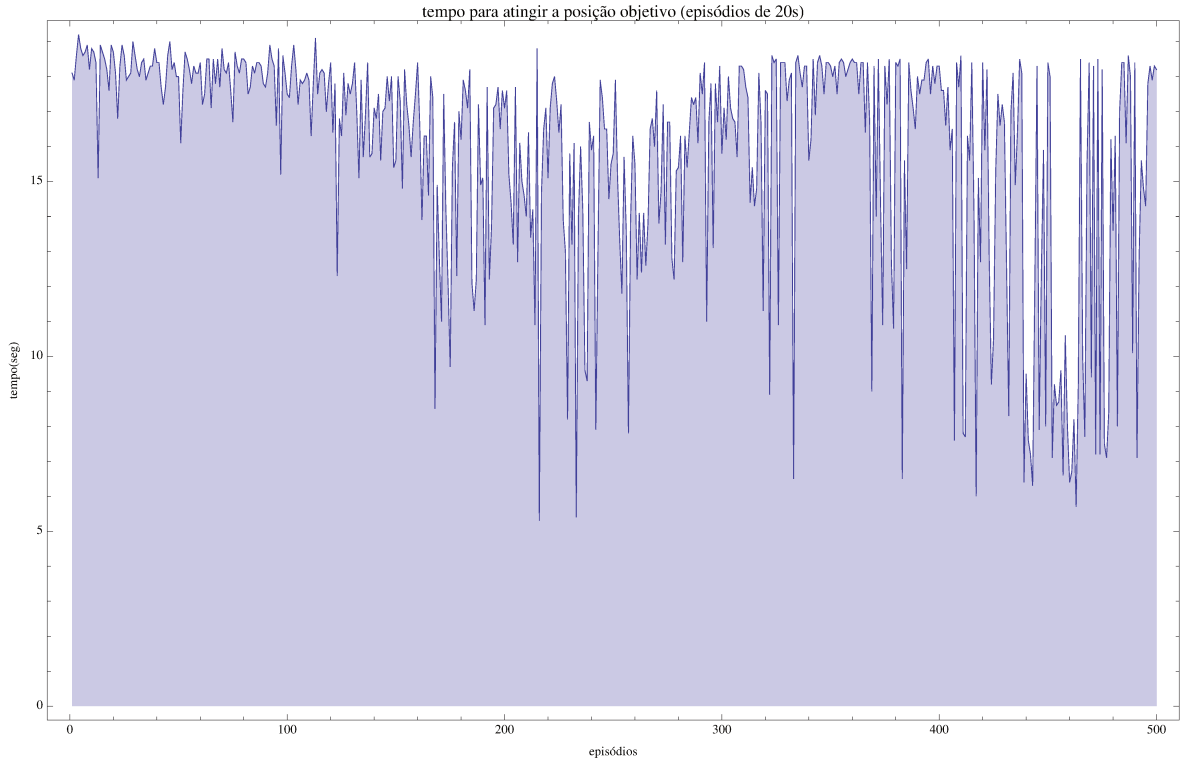

Fig. 5.10: Gráfico do tempo necessário para atingir a posição desejada (em segundos simulados) para cada episódio de treinamento do Carro na Montanha, durante 500 episódios, com episódios de no máximo 20 segundos.

Capítulo 6

Conclusão

6.1 Principais Contribuições

Este trabalho apresentou uma maneira promissora de integração entre algoritmos de aprendizagem por reforço (AR) *off-policy* e elegibilidade. É fato que existem propostas na literatura de se fazer o mesmo, porém é importante ressaltar que é crucial para a comunidade científica pesquisar abordagens distintas para o mesmo problema, cujo propósito enriquece a coleção de ferramentas que servirão de fundamento para projetos futuros nesta área.

Intuitivamente, como já comentado na seção 3.4, a elegibilidade pode ser compreendida como a capacidade da máquina "lembrar" o caminho até uma recompensa ou punição, a fim de aprender a associação deste caminho ao retorno obtido. Também intuitivamente, a aprendizagem *off-policy* pode ser entendida como uma máquina que segue uma política e aprende sobre as "regras" e relações de valor entre os estados, generalizando para construir uma noção interna do que seria uma política ótima, cujo objetivo será justamente maximizar o valor segundo tais conceitos aprendidos *a priori*.

O ambiente com variáveis contínuas apresenta alguns desafios à integração destas duas ideias, por fatores discorridos no capítulo 4. A forma original desenhada por Gaskett *et al.* [6] foi fazer uso de uma estrutura matemática denominada *Wire-Fitting* (WF). A contribuição da pesquisa, foi incorporar a elegibilidade em ambientes contínuos - engenhosamente articulada por Doya [5] - ao *framework* de Gaskett. O ganho disso, evidenciado pelos resultados, é uma aprendizagem mais veloz.

É notável, em diversos algoritmos de aprendizado de máquina, incluindo o deste trabalho, a constatação de que o mecanismo algorítmico em si não define ou referencia quase que absolutamente nada relacionado à instância de um problema específico. É esse caráter que lhe permite ser genérico e lhe concede a capacidade de solucionar problemas, a princípio, tão diferentes. É na faculdade de aprendizado que reside a "mágica" da solução, pois, sendo capaz de aprender, a máquina ganha um potencial poderoso de generalização e "inteligência".

O Neural-Q(λ) é dotado da capacidade de aprender e de julgar valor, o que permitiu tratar com sucesso duas instâncias de problemas de controle ótimo diferentes em alguns aspectos. Em outros aspectos, entretanto, estas duas instâncias são bem similares, particularmente na dimensionalidade de entrada e saída.

6.2 Dificuldades Encontradas e Propostas Futuras

Esta pesquisa foi guiada por objetivos ambiciosos, porém, também encontrou sérias dificuldades. As quais são comentadas nesta seção, para, em seguida, apontar caminhos motivando possíveis frentes de pesquisas futuras.

A principal dificuldade de aplicação do algoritmo relaciona-se ao conhecido termo já cunhado pela comunidade de otimização: "maldição da dimensionalidade" [3]. Ao tentar aplicar o algoritmo Neural-Q(λ) a problemas com dimensões maiores, este apresentou problemas de escalabilidade. Será comentado a respeito de dois tipos de estresse de dimensão efetuados, e os resultados obtidos.

6.2.1 Alta dimensionalidade na Entrada

Quando o problema apresenta muitas dimensões de entrada, é necessário aumentar o número de entradas na RN (a qual é acoplada, na saída, à função WF). Este tipo de estresse, apesar de diminuir a performance, não foi tão crítico quanto o segundo tipo de estresse de dimensionalidade (na saída), que é descrito adiante. Foi possível observar que, naturalmente como é sabido em problemas de RNs perceptron multi-camadas, o aprendizado fica cada vez menos efetivo em dimensões maiores.

O módulo de RN é perfeitamente substituível por qualquer abordagem de generalização de funções com aprendizagem supervisionada. De fato, o módulo escolhido é bastante canônico na literatura, e sabidamente existem outras abordagens mais apropriadas para certos tipos de problema, as quais evidentemente podem ser experimentadas.

6.2.2 Alta Dimensionalidade na Saída

Este segundo tipo de estresse é mais dramático. Mesmo incrementando a dimensão de saída em uma unidade, observou-se uma performance bastante inadequada do algoritmo em termos de tempo de convergência.

O fato é que o acoplamento do módulo de RN para a função WF se dá pela tradução da saída da primeira para os parâmetros de ajuste da segunda. Este acoplamento fica bastante delicado quando é aumentada a complexidade da função WF de uma para duas dimensões na saída. Apesar de matematicamente ser plausível a generalização da função WF para múltiplas dimensões, tal abordagem trouxe problemas de convergência.

Uma alternativa seria o uso de uma função WF para cada dimensão de saída, ou seja, o módulo de RN teria sua saída dividida em dois ou mais conjuntos, um para cada dimensão de saída do problema, acoplando cada um a uma função WF, traduzidos como parâmetros de ajuste desta. A preocupação desta alternativa é o aumento da dimensão de saída da RN. Porém, neste módulo tem-se muito mais espaço para experimentação, não estando limitado à necessidade de cálculo do máximo global simplificado do WF.

Para atacar este problema, desenha-se então dois caminhos: substituir ou melhorar a função WF para lidar com alta dimensionalidade na saída. Ou então substituir ou melhorar o módulo de RN para o mesmo fim. Porém, neste caso, a dimensionalidade da saída é multiplicada pelo tamanho de entrada de cada WF, ou seja, o tamanho do conjunto de parâmetros de ajuste desta.

6.2.3 Necessidade de Ajuste dos Parâmetros *a priori*

Este problema é comum a diversos algoritmos sofisticados que empregam vários parâmetros (também chamados constantes) "mágicos" em sua operação.

Naturalmente, para se diminuir o número necessário de constantes pré-definidas sem modificar o algoritmo original, é preciso definir um meta-algoritmo que englobe o primeiro, e cuja tarefa seja - com menos constantes pré-definidas - descobrir programaticamente constantes para o algoritmo encapsulado que façam sentido.

Uma abordagem bastante interessante é o uso de algoritmos genéticos, aplicados à "evolução", por assim dizer, dos parâmetros do algoritmo, buscando melhores ajustes no espaço vetorial de configurações paramétricas. Evidentemente, além de um problema de busca por algoritmos genéticos, este é, em primeiro lugar, um problema de busca e otimização. A opção por algoritmos genéricos (ou outras meta-heurísticas generalistas) torna-se justificável pela natureza bastante complexa da forma da função-objetivo.

6.2.4 Aplicação em Robótica Móvel

A disciplina de robótica móvel tem se tornado ubíqua em inúmeras atividades humanas. Em caráter motivacional, são enumeradas algumas áreas que demandam soluções ótimas em robótica móvel autônoma:

- Produção industrial: fábricas de veículos e bens eletrônicos.
- Exploração submarina: robôs capazes de navegar e reagir adequadamente a imprevistos.
- Defesa: Robôs de desarmamento de bombas e robôs militares.
- Exploração espacial.
- Robôs domésticos e de cuidado para pessoas necessitadas: úteis tanto em hospitais quanto lares.

Estes projetos exigem alguns atributos junto aos quais a abordagem de aprendizado de máquina desta pesquisa foi capaz de demonstrar, mesmo que timidamente:

- Generalização de ação, estado e resultado esperado.
- Aprendizado de sistemas dinâmicos e não-lineares, os quais, podem ser dinâmicos em suas próprias regras.
- Aprendizado bem sucedido de controle ótimo.
- Aprendizado bem sucedido em ambientes com variáveis contínuas.

Unindo a demanda da indústria e pesquisa com os resultados do algoritmo, surge a motivação de se continuar a pesquisa, buscando melhorias e incorporando novas ideias para prover soluções reais. Os desafios são inúmeros, porém, pode-se contar a curto prazo com cada vez mais recursos computacionais e mais ferramentas de aprendizado de máquina, nutrindo ainda mais o sonho de se criar máquinas inteligentes.

Bibliografia

[1] L. Baird and A.H. Klopf. Reinforcement learning with high-dimensional continuous actions. *US Air Force Technical Report WL-TR-93-1147, Wright Laboratory, Wright-Patterson Air Force Base, OH*, 1993.

[2] A.G. Barto, R.S. Sutton, and C.J.C.H. Watkins. Learning and sequential decision making. *M. Gabriel and J. Moore (Eds.), Learning and Computational Neuroscience: Foundations of Adaptive Networks, chapter 13*, pages 539–602, Bradford Books/MIT Press, 1990.

[3] R. Bellman. *Dynamic Programming*. Rand Corporation research study. Princeton University Press, 1957.

[4] N. Cristianini. Are we there yet? *Neural Networks*, 23(4):466–470, 2010.

[5] K. Doya. Reinforcement learning in continuous time and space. *Neural Computation*, 12(1):219–245, 2000.

[6] C. Gaskett, D. Wettergreen, and A. Zelinsky. Q-Learning in Continuous State and Action Spaces. *Advanced Topics in Artificial Intelligence*, pages 417–428, 1999.

[7] S. Haykin. *Neural Networks and Learning Machines*. Prentice Hall, 3 edition, 2008.

[8] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[9] R.A. Howard. Dynamic programming and Markov process. *MIT Press*, 1960.

[10] J. Peng and R.J. Williams. Incremental multi-step Q-learning. *Machine Learning*, 22(1):283–290, 1996.

[11] M.L. Puterman. *Markov decision processes: Discrete stochastic dynamic programming*. John Wiley & Sons, 1994.

[12] S. Russell and P. Norvig. *Artificial intelligence: A Modern Approach*. Prentice Hall, 3rd edition, 2009.

[13] R.S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44, 1988.

[14] R.S. Sutton and A.G. Barto. *Reinforcement Learning*. MIT Press, 1998.

[15] E.L. Thorndike. *Animal Intelligence*. Columbia University Press, 1898.