UNIVERSIDADE FEDERAL DE MINAS GERAIS
Escola de Engenharia Elétrica
Programa de Pós-Graduação em Engenharia Elétrica

Victor Marcius Magalhães Pinto

Estudo de Aplicação de Técnicas de Aprendizado por Reforço no Problema de Otimização de Portfólio

Belo Horizonte
2022

Victor Marcius Magalhães Pinto

Estudo de aplicação de Técnicas de Aprendizado por Reforço no Problema de Otimização de Portfólio

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Minas Gerais, como requisito parcial à obtenção do título de Mestre em Engenharia Elétrica.

Orientador: Prof. Dr. Cristiano Leite de Castro

Belo Horizonte
2022

Pinto, Victor Marcius Magalhães.
P659e Estudo de aplicação de técnicas de aprendizado por reforço no problema de otimização de portfólio [recurso eletrônico] / Victor Marcius Magalhães Pinto. – 2022.
1 recurso online (174 f. : il., color.) : pdf.

Orientador: Cristiano Leite de Castro.

Dissertação (mestrado) Universidade Federal de Minas Gerais, Escola de Engenharia.

Apêndices: f. 173-174.

Bibliografia: f. 165-169.
Exigências do sistema: Adobe Acrobat Reader.

1. Engenharia elétrica - Teses. 2. Aprendizado do computador - Teses. 3. Otimização - Teses. 4. Mercados financeiros futuros - Teses. I. Castro, Cristiano Leite de. II. Universidade Federal de Minas Gerais. Escola de Engenharia. III. Título.

CDU: 621.3(043)

Ficha catalográfica elaborada pela Bibliotecária Roseli Alves de Oliveira CRB/6 2121
Biblioteca Prof. Mário Werneck, Escola de Engenharia da UFMG

UNIVERSIDADE FEDERAL DE MINAS GERAIS
ESCOLA DE ENGENHARIA
COLEGIADO DO CURSO DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

FOLHA DE APROVAÇÃO

"Estudo de Aplicação de Técnicas de Aprendizado Por Reforço No Problema de Otimização de Portfólio"

VICTOR MARCIUS MAGALHÃES PINTO

Dissertação de Mestrado defendida e aprovada, no dia 28 de abril de 2022, pela Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Minas Gerais constituída pelos seguintes professores:

Prof. Dr. Luiz Carlos Bambirra Torres (DCS (UFOP))

Prof. Dr. Lucas de Souza Batista (DEE (UFMG))

Prof. Dr. Cristiano Leite de Castro (DEE (UFMG)) - Orientador

Belo Horizonte, 28 de abril de 2022.

Documento assinado eletronicamente por **Frederico Gadelha Guimaraes**, **Coordenador(a) de curso de pós-graduação**, em 04/11/2022, às 14:35, conforme horário oficial de Brasília, com fundamento no art. 5º do Decreto nº 10.543, de 13 de novembro de 2020.

A autenticidade deste documento pode ser conferida no site https://sei.ufmg.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **1859801** e o código CRC **03899382**.

Este trabalho é dedicado a todos que, apesar dos obstáculos, nunca perderam a fé na ciência, e se esforçam para que o conhecimento se torne um dos pilares mais importantes da sociedade.

Agradecimentos

Primeiramente, agradeço a meus pais, Regina e Márcio, pela criação e exemplos que contribuíram na construção do meu caráter.

Agradeço à minha noiva, Ludmila, pelo apoio e suporte durante toda essa jornada. Foi meu pilar, e sem ela teria sido muito mais difícil passar por todo esse período, dos momentos fáceis aos difíceis.

À minha família, meus sogros e cunhados, por estarem sempre comigo, também me apoiando, e torcendo por mim.

Ao meu orientador, Cristiano, por toda a construção desse projeto, por ser um orientador excepcional, por todas as discussões e ideias que contribuíram para minha formação.

Aos meus amigos e colegas de trabalho, que participaram comigo de toda esta saga, torcendo e incentivando. Aos meus líderes Victor e Erick, que desde sempre deixaram claro a importância de me preocupar com meu futuro, e que no trabalho eu também encontraria todo o apoio que precisasse para que pudesse completar o mestrado sem preocupações.

Por fim quero agradecer à UFMG, em especial ao Programa de Pós Graduação em Engenharia Elétrica e seu excelente corpo docente, por ser essencial na minha formação acadêmica.

"Nothing in life is to be feared,
it is only to be understood.
Now is the time to understand more,
so that we may fear less."
- Marie S. Curie

Resumo

O problema de gerenciamento de portfólio, foco deste trabalho, consiste em determinar uma alocação ótima para ativos dentro de uma carteira de ações, de forma a maximizar (ou minimizar) um ou mais objetivos. Estes geralmente relacionam-se a medidas de risco e retorno. Na literatura econômica financeira, este problema tem sido resolvido com o uso de modelos de otimização de portfólio, como Markowitz, CAPM e Black Litterman, que são executados a cada instante em que o rebalanceamento da carteira se faz necessário. Este processo de tomada de decisão, incremental e sob incerteza, pode ser visto como um processo de decisão Markoviano, o que torna atraente sua modelagem pelo paradigma de aprendizagem por reforço, sendo esta uma tendência recente discutida na literatura de aprendizagem de máquina. Este trabalho investiga o uso de técnicas de aprendizado por reforço no problema de otimização de portfólio. É realizada uma revisão da literatura existente, com seus principais aprendizados. Em um estudo de caso, um problema de controle de pesos de ativos em uma carteira é modelado como um processo de decisão Markoviano, e são aplicados algoritmos de aprendizado por reforço para atuar na otimização deste portfólio. As implementações são realizadas de maneira incremental, procurando demonstrar a lógica por trás da construção destes algoritmos. Ao final, o desempenho dos modelos é comparado com o de estratégias baseadas em Markowitz, e o resultado mostra que estas abordagens possuem boas performances, e seu uso é promissor para este tipo de problema.

Palavras-chaves: aprendizado de máquina, aprendizado por reforço, otimização de portfólio, mercados financeiros.

Abstract

The portfolio management problem, focus of this work, consists of determining the optimal asset allocation within a wallet, in order to maximize (or minimize) one or more objectives. These objectives are usually related to risk and return metrics. In financial economics literature, this problem has been solved using portfolio optimization models, such as Markowitz, CAPM and Black Litterman, which are executed for each instant when portfolio rebalancing is necessary. This decision process, incremental and under uncertainty, can be seen as a Markovian decision process, which makes modeling under reinforcement learning paradigm attractive, this being a recent trend discussed in machine learning literature. This work aims to investigate the use of reinforcement learning technics in portfolio optimization problem. A literature review is realized, with its main learnings. In a case study, a portfolio asset weight control problem is modeled as a Markovian decision process, and reinforcement learning algorithms are used to optimize it. The implementations are made in an incremental way, aiming to demonstrate the logic behind these algorithms developments. Finally, the model's behavior is compared with Markowitz based strategies, and the result shows that these approaches hold good performances, and have a promising use for this kind of problem.

Key-words: reinforcement learning, machine learning, portfolio optimization, financial markets.

Lista de ilustrações

Lista de tabelas

Lista de Algoritmos

Sumário

1 Introdução

1.1 Motivação e Justificativa

A utilização de algoritmos de otimização e inteligência computacional para a resolução de problemas do mercado financeiro é uma abordagem conhecida e amplamente discutida na literatura (CORNUEJOLS; TUTUNCU, 2007; MANSINI; OGRYCZAK; SPERANZA, 2015). Considerando a alta volatilidade de preços de ativos e a influência de inúmeros fatores externos, que torna complicada a modelagem dos problemas no âmbito financeiro, técnicas de otimização sob incerteza constituem ferramentas essenciais para que um investidor consiga operar no mercado de ações.

Dada uma carteira de ações, que é uma composição de ativos nos quais um investimento é feito, o problema de otimização de portfólios, que é o tema deste trabalho, consiste em encontrar qual a porcentagem deste investimento direcionado a cada um destes ativos, de forma a maximizar o ganho com os mesmos. Definir estes números, também chamados de *pesos* dos ativos, passa por "antecipar"qual será o seu desempenho durante o próximo período de tempo, de forma a concentrar a maior alocação em ativos que se valorizem, e evitar alocar capital em ativos que tenham perdas. Assim, a utilização de técnicas que levem em conta características dos mesmos, e permitam desenvolver uma estratégia que vise otimizar esse retorno é essencial para obter bons ganhos (LUENBERGER et al., 1997; CAPIŃSKI; KOPP, 2014).

Diferentes técnicas de otimização de portfólios não incorporam aprendizado do algoritmo em sua execução, baseando-se em heurísticas ou metodologias matemáticas que, apesar de obterem resultados bons, quando comparadas à uma abordagem de um agente humano, ainda são lentas em perceberam variações de mercado. Portanto, existe um espaço para a utilização de algoritmos de aprendizado de máquina, cuja capacidade adaptativa é promissora em abordar essas lacunas das técnicas de otimização.

Como citado por FISCHER, 2018, grande parte dos trabalhos em aprendizado de máquina orientados para finanças ainda são dedicados a algoritmos de aprendizado supervisionado. Existe, porém, um nicho bem mais interessante dentro desse paradigma que são os algoritmos de aprendizado por reforço. Esta abordagem, que foca em estabelecer procedimentos de tomada de decisão eficientes, e abordam o aprendizado de um agente computacional visando otimizar uma métrica de recompensa, traz vantagens em comparação com modelos de aprendizado supervisionado. O aprendizado por reforço permite por exemplo a consideração de informações difíceis de serem modeladas como features (como custos de transação e liquidez) como parte do processo de aprendizado. Além disso não necessitam de um modelo explícito de como o preço dos ativos se comportam. Mais do que encontrar correlações entre as ações, e predizer movimentação de preços, estes algoritmos concentram-se em modelar um melhor curso de ação a ser tomado *a partir dos preços vistos*, mesmo que não consiga indicar se um determinado valor irá subir ou descer no dia seguinte.

1.2 Objetivos

Este trabalho tem como objetivo investigar trabalhos anteriores que usaram aprendizado por reforço para solucionar o problema de otimização de portfólio. A partir disso, testar algumas dessas técnicas em dados reais, com o intuito de demostrar como um problema de otimização da alocação de ativos pode ser modelado como um problema de aprendizado por reforço e fazer uma análise crítica dos resultados obtidos. Procura também tentar apontar alguns caminhos (como algoritmos, arquitetura do agente, parametrizações, funções de recompensa) na modelagem de problemas de otimização de portfólio a partir de aprendizado por reforço. Ao final do estudo de caso realizado, pudemos observar dos resultados que:

- Os algoritmos de aprendizado por reforço conseguiram um retorno financeiro tão elevado quanto o das abordagens de otimização baseadas em heurísticas clássicas como Markowitz.
- Ainda sim existe espaço para algoritmos mais complexos que busquem um

resultado mais assertivo.

- O estudo de caso mostra que a abordagem de aprendizado por reforço é factível e promissora para este problema.

Conseguimos também concluir alguns pontos com a execução deste trabalho:

- Algoritmos de aprendizado por reforço podem ser utilizados para lidar com problemas complexos como os de mercado financeiro.
- A possibilidade de incorporar aprendizado à rotina, e a capacidade de adequar o comportamento do modelo conforme a resposta do ambiente constituem vantagens dos algoritmos de aprendizado por reforço em comparação a heurísticas de otimização.
- Abordagens que não necessitem de uma representação do ambiente são mais indicadas.
- A utilização de features que considerem séries históricas dos ativos constituem a abordagem mais comum encontrada na literatura.

Os códigos utilizados durante o estudo de caso estão públicos, e podem ser encontrados no repositório do Github através do link <http://github.com/victormmp/rl_mestrado>.

1.3 Organização do Texto

No capítulo 3 é introduzido o problema de otimização de portfólios, e resumida a abordagem clássica em torno do tema. No capítulo 2 é apresentado o tópico de aprendizado por reforço, os principais conceitos, e é desenvolvida a teoria básica em torno do tema. Este capítulo, baseado no livro *Reiforcement Learning: an introduction*, de Richard Sutton e Andrew Barto (SUTTON; BARTO, 2018), serve como uma referência teórica para os assuntos tratados posteriormente. No capítulo 4, é abordado o tema de aplicação de aprendizado por reforço em finanças, com

uma revisão de alguns importantes trabalhos na área. No capítulo 5 é apresentado um estudo de caso, onde um problema de otimização de portfólio é apresentado, para em seguida ser modelado como um problema de aprendizado por reforço. É apresentado o desenvolvimento de uma solução para o mesmo, mostrando o raciocínio de maneira incremental, com o objetivo de mostrar como pode-se desenvolver um projeto no tema, e como aspectos do problema financeiro se relacionam com tópicos da metodologia de aprendizado de máquina.

2 Aprendizado por Reforço

2.1 Introdução

Este capítulo tem por base o livro *Reinforcement Learning - An Introduction*, escrito por Richard Sutton e Andrew Barto (SUTTON; BARTO, 2018), e tem por objetivo trazer um compilado dos principais conceitos de aprendizado por reforço, necessários para o entendimento do tema. O livro é considerado uma das principais referências sobre o assunto, e todos os conceitos aqui abordados podem ser vistos de maneira mais ampla no mesmo.

De acordo com SUTTON; BARTO, 2018, o contexto de aprendizado por reforço, ou *reinforcement learning*, define-se como aprender o que fazer, dada uma situação, de forma a maximizar um retorno numérico. Possuindo ou não um modelo de como o ambiente com o qual se relaciona se comporta, o "indivíduo aprendiz" (também chamado de agente) deve descobrir quais ações tomar de forma a receber a maior recompensa, ao invés de ter um "professor" lhe dizendo qual ação deve ser tomada.

Sendo assim, o aspecto chave de aprendizado por reforço é aprender com a experiência. Um agente tem disponível um conjunto de ações possíveis, e é informado sobre o estado atual do ambiente com o qual se relaciona. A partir dessas condições, ele seleciona uma ação, seguindo alguma lógica para a escolha da mesma, e avalia qual a resposta do ambiente para a sua escolha, após ter realizado esta ação. Cada escolha de ação, a cada estado do ambiente, contribui para o acúmulo de experiência deste agente, e espera-se que o mesmo aprenda quais as melhores ações a serem tomadas com esta experiência, ou que ao menos consiga estimar o ganho que pode obter ao escolher uma ação qualquer, mesmo que esta ação não seja a melhor naquele momento.

O paradigma de aprendizado por reforço difere-se da dinâmica de *aprendizado supervisionado*, mais estudada no campo de aprendizado de máquina (RUSSELL; NORVIG, 2002). Na categoria de aprendizado supervisionado, a tentativa

é de se obter um modelo aproximado da função ou dinâmica que originou uma distribuição de dados. Para tal, uma amostra desta distribuição, com seu respectivo rótulo - quando em tarefas de classificação, ou então valor correspondente, quando em tarefas de regressão - é apresentado à máquina, que tenta adaptar um modelo aproximado do apresentado, a fim de ter um bom desempenho quando for lidar com dados nunca vistos anteriormente. Sendo assim, um conhecimento externo é apresentado à máquina, por um supervisor. Em situações de aprendizado por experiência, nem sempre esse tipo de interação é suficiente, visto que nem sempre é possível construir um conjunto de dados para o treinamento que aborde estados suficientes dos dados desejados, de forma a permitir ao modelo generalizar para situações desconhecidas. Por exemplo, dinâmicas dependentes de um infinito número de fatores, conhecidos ou desconhecidos, como o trânsito, para um carro autônomo, ou fenômenos meteorológicos.

Difere-se também da lógica de *aprendizado não-supervisionado*, bastante estudada no contexto de aprendizado de máquina. Neste tipo de dinâmica, o agente procura identificar padrões associáveis entre as características fornecidas por amostras, sem se preocupar em classificá-las ou fornecer conclusões à respeito das mesmas. Este tipo de algoritmo pode fazer parte do fluxo de aprendizado por reforço, em algum aspecto, como nos trabalhos apresentados por JADERBERG et al., 2016 e GUPTA et al., 2018. Porém, sozinho, não fornece a um agente de aprendizado por reforço meios de aprender com a experiência.

De acordo com SUTTON; BARTO, 2018, diferentemente de outros paradigmas de aprendizado de máquina, em aprendizado por reforço existe o desafio de balancear o que é chamado de *"exploração e exploitação"*[1]. Para obter o melhor resultado a partir de um comportamento, um agente deve explorar as melhores ações conhecidas em cada situação (exploitação). Porém, dado um conjunto de ações possíveis de serem realizadas, um agente também deve explorar outros comportamentos, a fim de verificar se existe uma ação que traga maior resultado do que a melhor ação conhecida até o momento (exploração). E dessa forma, admite-

[1] Oriundos do inglês *"exploration"* e *"exploitation"*, respectivamente. Traduzindo diretamente para o português, ambas as palavras podem ser representadas por "exploração", porém em alguns trabalhos em português, o termo "exploitação"é empregado como tradução de "exploitation", como por exemplo em (FARIA; ROMERO, 2002).

se que o agente falhe nestas situações, uma vez que nem sempre uma nova ação significa um retorno maior.

Uma outra característica chave deste paradigma é que o mesmo considera todo o problema de um agente orientado a resultados interagindo com um ambiente desconhecido e incerto. É diferente, por exemplo, de muitos trabalhos em aprendizado de máquina, que focam em utilizar algoritmo de aprendizado supervisionado para resolver uma parte do problema, sem haver uma definição de como a habilidade aprendida será útil ao final. Por exemplo, considere um modelo classificador que aprenda a identificar fotos de gatos. O objetivo do trabalho é treinar um algoritmo para resolver o subproblema de conseguir realizar a classificação, sem definir o porquê da mesma ser importante, e onde ela se encaixa em algum problema mais amplo, onde classificar imagens de gato seja importante. Agentes em aprendizado por reforço possuem objetivos amplos específicos, interagem com o ambiente e o influenciam através da escolha de suas ações. Ou seja, não se preocupam apenas com os subproblemas, mas já faz parte do planejamento o problema que a máquina sendo treinada deverá resolver. É parte de toda a definição deste tipo de processo considerar que o agente irá atuar em um ambiente cheio de incertezas. Agentes em aprendizado por reforço podem até fazer uso de algoritmos de aprendizado supervisionado, mas o fazem com o objetivo de atuar em subproblemas que possuam objetivos bem definidos dentro do objetivo macro onde o mesmo procura atuar.

De todas as formas de aprendizado de máquina, podemos dizer que aprendizado por reforço seja a que talvez mais se aproxime da forma como humanos e outros animais aprendem.

2.2 Elementos Comuns de Aprendizado por Reforço

Ainda segundo os autores Barto e Sutton (SUTTON; BARTO, 2018), é possível destacar quatro elementos comuns a todos os sistemas de aprendizado por reforço: uma política (*policy*), um sinal de retorno (*reward*), uma função de valor (*value function*) e em alguns casos, um modelo do ambiente (apesar de não ser um ponto essencial para o aprendizado, é possível aprender sem ter conhecimento da dinâmica do ambiente).

A política é como definimos o comportamento de um agente. É o principal conceito deste tipo de aprendizado, e de maneira direta, é o que define quais ações serão escolhidas em cada estado do ambiente. Na prática, é um conjunto de probabilidades associadas a ações em cada estado, que pode ser determinístico, apesar de grande maioria das vezes apresentar um comportamento estocástico. Nas tarefas de *controle*, dentro do contexto de aprendizado por reforço, o objetivo principal é encontrar a política que devolva o maior retorno, considerando os estados visitados.

O sinal de retorno, também chamado de recompensa (*reward* ou *return*), é a resposta do ambiente ao estímulo recebido pelo agente, quando o mesmo executa uma ação, dizendo se a mesma foi boa ou ruim. Este sinal é a resposta imediata do comportamento de um agente, e é a base principal para a modificação de uma política. Em geral, a recompensa é um sinal de comportamento estocástico, em função do estado atual do ambiente e da ação selecionada quando naquele estado.

As funções de valor, por outro lado, indicam qual o *retorno total esperado*, para uma interação longa com o ambiente, se começarmos a contar a partir do estado em questão. Leva em consideração, portanto, não só o estado em questão, mas também todos os possíveis estados que podem se suceder a partir do mesmo, e toda a sucessão de estados a partir deles. E para a definição das sequências possíveis de estados, leva em conta também o quão prováveis eles são de ocorrerem com o comportamento adotado pelo agente (a política do mesmo). Em um jogo de tabuleiro, por exemplo, pode significar o quão provável é ganharmos um jogo, se considerarmos que as peças estão organizadas de uma maneira específica. Avaliamos portanto não só a organização atual das peças (o estado atual do ambiente),

mas também todas as sequências possíveis a partir desta organização, que nos levem ou não à vitória. As funções podem ser relativas aos estados (*state-value functions*), ou às ações em um determinado estado (*action-value functions*). Em tarefas de predição, o objetivo principal é estimar a recompensa total obtida, a cada estado, tendo por base um comportamento específico. A partir daqui, quando nos referirmos a retorno, estaremos falando de todo este valor que um estado ou uma ação pode possuir, considerando todas as recompensas futuras, diferentemente de recompensa, que permanece sendo o sinal imediatamente recebido por uma interação. Nas próximas seções iremos explorar este conceito, e suas diferentes formas de ser calculado.

Apesar de a recompensa indicar a interação imediata do agente com o ambiente, é a função de valor o objetivo principal na tomada de decisões. O interesse do agente é selecionar ações que direcionem a estados de maior valor, e não de maior recompensa, uma vez que uma determinada ação pode trazer uma recompensa baixa em um estado, se considerarmos todas as ações possíveis, mas talvez o maior retorno acumulado quando observado o panorama a longo prazo. Considere, por exemplo, um carro localizado no fundo de um vale, e que deseja subir até uma borda, como na Figura 1, mas que não possui potência o suficiente para ir direto até o objetivo. Para tanto, precisaria se afastar, pela outra borda, para ganhar impulso e conseguir subir. Se a recompensa for maior quanto mais próximo do objetivo se encontra, ir para a borda contrária para pegar impulso apresenta a menor recompensa instantânea para o agente controlando o carro, mas a longo prazo, é a ação que traz maior retorno, uma vez que apenas assim conseguirá chegar ao objetivo.

Além disso, um sistema de aprendizado por reforço pode contar com um modelo do ambiente, apesar de não ser essencial. Neste caso, este modelo fornece informações à respeito do comportamento esperado do ambiente, das probabilidades de transição entre os estados e o que esperar de resposta quando uma determinada ação é tomada. Ter um modelo associado ao sistema de aprendizado permite experimentar eventos simulados, não necessariamente vivenciados pelo agente ou por um indivíduo externo, que colaboram para a melhoria das estimativas de recompensas totais esperadas. Estes métodos são conhecidos como *model-based*, enquanto mé-

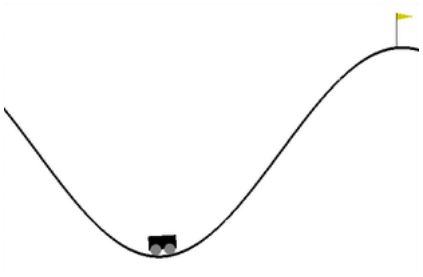


Figura 1 – Carro no fundo de um vale, tentando chegar à borda. Este é um problema clássico de aprendizado por reforço, conhecido como *mountain car* (MOORE, 1990).

todos que aprendem por tentativa e erro são conhecidos como *model-free*. Alguns métodos, entretanto, podem fazer uso de ambas as metodologias.

Para consolidar os conceitos apresentados nesta seção, considere o seguinte exemplo. Um robô (que é o *agente* do nosso problema de aprendizado por reforço) é treinado para conseguir locomover-se através de um labirinto, indo do ponto S ao ponto F, e encontrar o menor caminho para a saída, como na Figura 2. O desempenho do robô em resolver este problema é medido em pontos, que o mesmo recebe ao interagir com o ambiente. A cada quadrado caminhado pelo robô, ele recebe -1 ponto. Caso consiga chegar na saída, ele recebe 100 pontos. Esses pontos recebidos são as *recompensas*. Considere a situação onde o robô encontra-se no ponto A. Para saber o quão "bom"é para o mesmo estar naquele ponto, considerando o objetivo específico que o robô tem de sair do labirinto no menor caminho, o mesmo deve considerar o quanto aquele ponto se aproxima ou se afasta deste menor caminho. Para isso, ele realiza a estimativa de quantos pontos ele pode ganhar (ou perder), considerando todos os caminhos possíveis a partir dali. Essa estimativa é realizada através das *funções de valor*, e o valor calculado por elas é o chamado *retorno*. Ao final, o que o robô vai aprender é a melhor estratégia (ou comportamento) que o possibilite resolver o problema, que no contexto de aprendizado por reforço, é a chamada *política*.

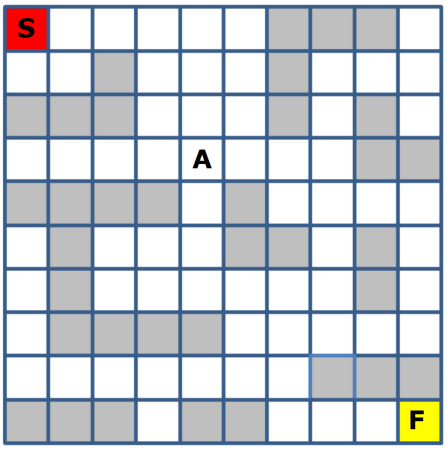


Figura 2 – Exemplo de problema de aprendizado por reforço, onde um robô é treinado para conseguir sair de um labirinto.

2.3 Processo de Decisão Markoviano Finito

2.3.1 Definição

O chamado *Processo de Decisão Markoviano*, (ou MDP[2]) (BELLMAN, 1957), é a forma matemática idealizada do problema de aprendizado por reforço. Tipicamente é representado por um diagrama como o da Figura 3. Este corresponde a uma representação básica da interação entre os diversos componentes de um MDP.

Neste tipo de relação, o indivíduo responsável por aprender e tomar as decisões é chamado de *agente*. Ele interage com o *ambiente* e aprende com as respostas obtidas a partir disso. O agente, a partir do *estado* atual do ambiente, informado ao mesmo, performa uma *ação*, e recebe do ambiente uma resposta sobre o efeito desta ação, a *recompensa*, que já foi mencionada na sessão 2.2. A ação realizada também tem um impacto no ambiente, que altera o seu estado para um *próximo estado*. A partir da informação de qual esse próximo estado, e qual foi a recompensa obtida com a última ação realizada, o agente escolhe e realiza uma próxima ação. Essa interação é contínua, e é realizada indefinidamente, ou até que um estado final seja obtido. A *trajetória* das ações do agente dentro do MDP segue

[2] *Markovian Decision Process.*

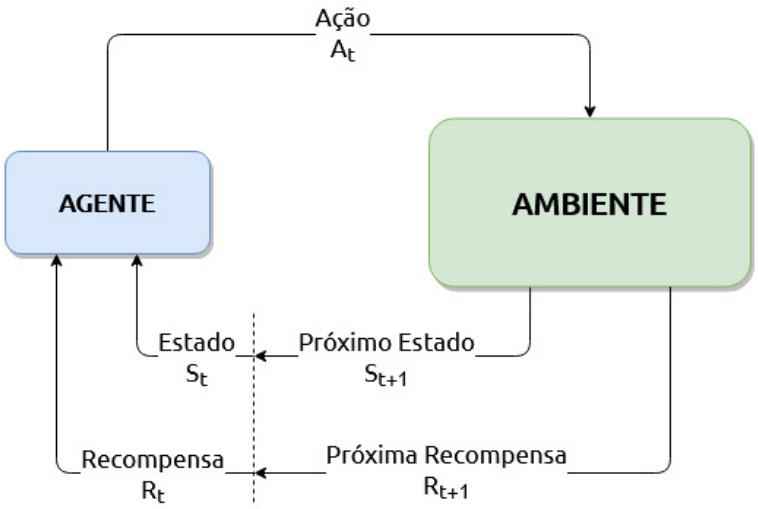


Figura 3 – Representação da interação entre os elementos de um MDP.

uma sequência em intervalos de tempo discreto $t = 0, 1, 2, 3, \ldots$, do tipo

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, \ldots$$

ou seja, a partir de um estado inicial S_0, o agente performa uma ação inicial A_0, recebendo a recompensa do ambiente R_1, que juntamente com a informação do próximo estado S_1 leva o agente a escolher e realizar a ação A_1, e assim por diante. A definição de agente e ambiente é flexível, e depende do escopo sendo trabalhado. Em geral, definimos como parte do ambiente tudo o que não pode ser modificado arbitrariamente pelo agente, ou seja, que é externo a ele.

Em um MDP *finito*, os conjuntos de estados, ações e recompensas têm um número finito de elementos. Neste caso, as variáveis aleatórias R_t e S_t possuem distribuições de probabilidades discretas bem definidas, que dependem apenas do estado anterior e da ação anterior realizada. Ou seja, considerando um estado pertencente ao conjunto de estados possíveis $s' \in \mathcal{S}$ e uma recompensa pertencente ao conjunto de recompensas possíveis, $r \in \mathcal{R}$, existe uma probabilidade de os dois valores ocorrerem ao mesmo tempo em um instante de tempo t, dado que foi observado um estado anterior $s \in \mathcal{S}$, e que uma ação $a \in \mathcal{A}$ foi selecionada,

$$p(s', r|s, a) = Pr\{S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a\}. \tag{2.1}$$

A função p é chamada de dinâmica do MDP, e é a distribuição de probabilidade conjunta de s' e r, condicionada à ocorrência de um estado s e uma ação a, onde

$$\sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r|s, a) = 1, \quad \forall \quad s \in \mathcal{S}, a \in \mathcal{A}. \tag{2.2}$$

Essas probabilidades podem ser compreendidas como um *modelo* do ambiente sendo interagido, uma vez que caracterizam como se dá o comportamento do mesmo em termos dos estados apresentados. Ou seja, é possível prever como o ambiente irá reagir, dado o conhecimento do estado atual, e uma ação escolhida. Sendo assim, o estado atual deve conter toda a informação necessária para que seja estabelecida as condições futuras, incluindo todos os aspectos das interações passadas, quando cabíveis. Esta é uma condição importante, conhecida como *propriedade de Markov* (MARKOV et al., 1960; DURRETT, 2019).

2.3.2 Tarefas Contínuas e Episódicas

O objetivo do agente é maximizar a recompensa total recebida, no longo prazo. Para isso, nem sempre a ação com a maior recompensa é a melhor a ser executada, em um estado, e sim a ação que leve à maior recompensa acumulada, ao longo prazo. Isso por vezes significa que o agente irá escolher uma ação que traga um retorno imediato ruim, mas que leve ao melhor resultado ao final. Este comportamento foi mencionado na seção 2.2, no exemplo do *mountain car* (Figura 1).

Podemos separar a interação entre agente e ambiente em dois tipos, *contínua* ou *episódica*, dependendo da forma como ocorre.

Em tarefas episódicas, toda a interação considerada, entre agente e ambiente, é chamada de *episódio*, e pode ser bem definida em um estado de início e um de fim. Podemos entender um episódio como uma partida de algum jogo, ou uma viagem, desde o momento de saída até o momento de chegada. O estado do último

intervalo de tempo, que encerra um episódio, é chamado de *estado terminal*. No contexto de aprendizado por reforço, interpretamos os episódios como independentes, isto é, a sequência de acontecimentos em um episódio depende apenas da interação entre agente-ambiente, e não da ocorrência de episódios anteriores. No caso de um jogo de xadrez, por exemplo, isso significaria que um episódio terminar em vitória ou derrota do agente depende apenas das ações realizadas pelo mesmo, e não se jogos anteriores terminaram em vitória ou derrota.

Formalmente, definimos a métrica para o retorno total de uma tarefa episódica, que chamaremos de *retorno esperado*, G_t, em um determinado instante de tempo t, a partir da soma de todas as recompensas futuras, como

$$G_t = R_{t+1} + R_{t+2} + \cdots + R_T, \tag{2.3}$$

onde T é o estado terminal. Em geral definimos o conjunto de todos os estados não-terminais como $\mathcal{S}$, e o conjunto de todos os estados possíveis, incluindo o estado terminal, como $\mathcal{S}^+$.

Em tarefas contínuas, a interação entre o agente e o ambiente por vezes não tem uma limitação clara, como um estado de encerramento, mas sim continua indefinidamente. Nestes casos, a definição dada pela equação 2.3 não se aplica, uma vez que $T = \infty$, e o retorno esperado poderia divergir para $+\infty$ ou $-\infty$. Nestes casos, o retorno é definido com base em uma soma ponderada, onde valores de recompensa muito futuros teriam um peso no valor total do retorno inversamente proporcional a quão distantes eles são. Isto é, adicionamos uma *taxa de desconto*, γ, ao cálculo,

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots \\ &= \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}, \end{aligned} \tag{2.4}$$

sendo $0 \leq \gamma \leq 1$. Assim, recompensas imediatas têm um peso maior do que as muito futuras. Uma versão da equação 2.4, importante em implementações de tarefas de aprendizado por reforço, é feita em função da próxima recompensa imediata e o valor do retorno esperado para a próxima iteração, de maneira incremental,

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots \\ &= R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \dots) \\ &= R_{t+1} + \gamma G_{t+1}. \end{aligned} \tag{2.5}$$

Como será visto, o próximo retorno esperado, G_{t+1}, está associado ao próximo estado s do ambiente, no tempo $t+1$, e sua estimativa é aprimorada ao longo do tempo, à medida que o agente interage com o ambiente. Dessa forma, podemos obter o retorno esperado pra um estado no tempo t de maneira incremental, a partir das estimativas que temos para o próximo estado.

De forma geral, podemos utilizar as equações 2.5 e 2.4 para modelar tanto tarefas episódicas, quanto contínuas, da forma

$$G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} R_t, \tag{2.6}$$

onde para tarefas contínuas $T = \infty$.

2.3.3 Políticas

Na seção 2.2 foi introduzido o conceito de *políticas*, e como as mesmas definem o comportamento a ser seguido por um agente. De maneira formal, uma política π é um mapeamento de ações em um estado, às probabilidades dessas ações serem escolhidas. Assim, um agente que esteja seguindo uma política π qualquer, a cada instante de tempo t, possui uma probabilidade $\pi(a|s)$ de escolher uma ação $A_t = a$, estando no estado $S_t = s$. Os métodos de aprendizado por reforço baseiam-se principalmente em estimar o valor que um estado s, ou par de estado-ação (s, a) vai apresentar, se o agente seguir este comportamento determinado por π. Ou então baseiam-se em encontrar a melhor política, $\pi*$, que maximize os valores para cada estado.

Sendo assim, é importante a noção que uma função de valor (também introduzida na seção 2.2) está intrinsecamente associada a uma política. A *função de valor de um estado*, quando o agente segue a política π, denominada v_π, é

$$v_\pi(s) = \mathbb{E}[G_t|S_t = s] = \mathbb{E}\left[\sum_{k=t+1}^{T} \gamma^{k-t-1} R_t \,\middle|\, S_t = s \right], \tag{2.7}$$

ou seja, é o valor esperado da soma de todas as recompensas futuras a esse estado, quando a política π é seguida para tomar decisões. De maneira análoga, a *função de valor de uma ação* (ou *função de valor do par estado-ação*) quando o agente segue a política π, denominada $q_\pi(s, a)$, é

$$q_\pi(s, a) = \mathbb{E}[G_t|S_t = s, A_t = a] = \mathbb{E}\left[\sum_{k=t+1}^{T} \gamma^{k-t-1} R_t \,\middle|\, S_t = s, A_t = a \right]. \tag{2.8}$$

Uma política é denominada *determinística* quando, em cada, estado, apenas uma ação possui uma probabilidade de ser selecionada, e *estocástica* quando múltiplas ações podem ser selecionadas, em um estado, com uma probabilidade diferente de zero.

Segundo Sutton e Barto, estes valores podem ser estimados via experiência. Assim, quanto mais vezes um agente seguindo uma política π encontra um determinado estado s em sua trajetória, e estima o valor de seu retorno, melhor sua estimativa do mesmo, e mais próximo essa estimativa fica de convergir para o valor real. O mesmo vale para estimativas de retorno de uma ação a selecionada em um estado s. Estes métodos de estimativa são chamados de *Métodos de Monte Carlo* (BARDENET, 2013), pois envolvem obter a média para os valores a partir de um conjunto amostral de retornos, obtidos pelas experiências vividas pelo agente. Estes métodos serão melhor explorados na seção 2.4.2.

As relações estabelecidas nas equações 2.7 e 2.8 são expandidas nas chamadas *Equações de Bellman*, para v_π e q_π (BELLMAN, 1966),

$$v_\pi(s) = \sum_a \left(\pi(a|s) \sum_{s',r} \Big(p(s', r|s, a)[r + \gamma v_\pi(s')] \Big) \right), \tag{2.9}$$

$$q_\pi(s, a) = \sum_{s',r} \left(p(s', r|s, a) \left[r + \gamma \sum_{a'} \pi(a'|s') \cdot q_\pi(s', a') \right] \right). \tag{2.10}$$

Analisando a equação 2.9, ela diz que o valor esperado de um estado é igual à soma de todos os estados subsequentes possíveis mais seus retornos correspondentes, ponderados pelas respectivas probabilidades de ocorrerem, de acordo com a definição da equação 2.1, a partir de cada ação a possível. Por sua vez, esses valores obtidos para cada ação são somados, ponderados pela probabilidade $\pi(a|s)$ de cada ação acontecer no estado s. Uma análise similar pode ser feita para a equação 2.10.

Como dito, resolver uma tarefa utilizando aprendizado por reforço significa encontrar uma política que devolva o máximo possível de recompensa a longo prazo. Em MDPs finitos, isso significa definir de maneira precisa uma política ótima, seguindo o seguinte raciocínio, como explicado por Sutton e Barto em seu livro. Podemos utilizar as funções de valor para metrificar uma política em função das outras, isto é, se uma política é melhor ou pior do que outra em comparação. Uma política π é considerada melhor ou igual a uma política π' se seu retorno esperado é maior ou igual ao de π' para todos os estados $s \in \mathcal{S}$. Existe sempre ao menos uma única política que seja melhor ou igual a todas as outras políticas, chamada de *política ótima*. O teorema e a prova de sua existência podem ser vistos na seção A.1 do apêndice. Podem haver mais de uma política ótima, porém todas elas compartilham o mesmo *valor ótimo para a função de valor de um estado*, v_*,

$$v_*(s) = \max_{\pi} v_\pi(s) \quad \forall \quad s \in \mathcal{S}, \tag{2.11}$$

e o mesmo *valor ótimo para a função de valor de uma ação*,

$$q_*(s,a) = \max_{\pi} q_\pi(s,a) \quad \forall \quad s \in \mathcal{S}. \tag{2.12}$$

Definimos as equações ótimas de *Bellman* para v_* e q_* a partir das equações de Bellman correspondentes, das formas à seguir.

$$\begin{aligned} v_*(s) &= \mathbb{E}\left[R_{t+1} + \gamma v_*(S_{t+1}) \,\middle|\, S_t = s, A_t = a\right] \\ &= \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_*(s')]. \end{aligned} \tag{2.13}$$

A partir de $v_*(s)$, podemos determinar $q_*(s)$. Como a função ótima nos permite concluir que a política encontrada também é ótima, não há necessidade de exploração do espaço de soluções à procura de uma política melhor - afinal a política ótima já é a melhor disponível. Podemos dizer que a política ótima orientará para a escolha da melhor ação em cada estado de maneira determinística. Sendo essa escolha ótima, então a ação escolhida também será. Portanto,

$$v_*(s) = \max_a q_*(s, a). \tag{2.14}$$

E assim, a partir de 2.13, podemos chegar em $q_*(s, a)$ como

$$\begin{aligned} q_*(s, a) &= \mathbb{E}\left[R_{t+1} + \gamma v_*(S_{t+1}) \,\middle|\, S_t = s, A_t = a\right] \\ &= \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \,\middle|\, S_t = s, A_t = a\right] \\ &= \sum_{s',r} p(s', r|s, a)\left[r + \gamma \max_{a'} q_*(s', a')\right]. \end{aligned} \tag{2.15}$$

Essas equações expressam o fato de que, sob a política ótima, os valores dos estados são iguais aos valores de retorno da melhor ação sob cada um desses estados. Dessa forma, a política ótima pode ser resumida em encontrar as ações que devolvam os valores ótimos de função de estado, de acordo com a equação 2.13, ou simplesmente selecionar as ações que possuem maior função de valor de ação, de acordo com a equação 2.15.

Resolver as equações de *Bellman* para encontrar os valores ótimos das funções de valor, ou para encontrar a política ótima depende que estas três afirmativas sejam verdadeiras:

1. Conhecemos as dinâmicas do ambiente, expressos pela componente $p(s', r|s, a)$ das equações de *Bellman*;
2. Possuímos recurso computacional suficiente para realizar os cálculos;
3. A propriedade de *Markov* - cada estado contém toda a informação necessária para estabelecer as condições futuras, sem precisar olhar para estados passados - é cumprida para todos os estados.

Estes pontos raramente são cumpridos na realidade. Portanto, em tarefas de aprendizado por reforço, comumente é feito uso de aproximações das equações de *Bellman* ótimas, utilizando a experiência adquirida pelo agente em diversas interações com o ambiente para obter essas aproximações. Nas seções seguintes, serão abordados diferentes métodos de aproximação das equações de Bellman, baseados tanto em amostragem dos valores oriundas de diversas interações com o ambiente, quando de aproximação das equações a partir de funções parametrizadas.

2.4 Métodos Baseados em Amostragem

Os métodos apresentados nas seções à seguir procuram obter uma aproximação dos resultados das equações de Bellman, a partir de amostras de interações do agente com o ambiente. Isto é, com base nas informações de quais estados foram visitados, quais os retornos obtidos em cada um deles, e quais as ações escolhidas, estes métodos procuram obter uma tabela com aproximações dos valores de cada estado, ou cada ação. Para compreender como realizar estes cálculos, primeiramente será introduzido o conceito de *programação dinâmica* em tarefas de aprendizado por reforço. Segundo Sutton e Barto, esta metodologia de cálculo, apesar de não mais tão utilizada como solução para este tipo de problema, devido aos altos custos computacionais e à necessidade de possuir um modelo do ambiente, possui um raciocínio que ajuda a compreender melhor como outros métodos funcionam.

É importante destacar que esses métodos funcionam melhor quando o espaço de estados, ou de ações, é discreto, e o número de indivíduos em cada estado não é muito grande, de forma a criar limitações computacionais à resolução.

2.4.1 Programação Dinâmica

Algoritmos de programação dinâmica são metodologias de resolução de MDPs que são empregadas quando possuímos um modelo perfeito do ambiente, isto é, das probabilidades de transição de estados, descritas na seção anterior. Juntamente com a necessidade de um grande recurso computacional disponível, seu uso em resolução de problemas de aprendizado por reforço é limitado. Porém, o raciocínio por trás desta forma de resolução de problemas é importante para a compreensão de outros métodos mais robustos, e os conceitos aqui introduzidos permanecem válidos para todos os métodos subsequentes. É plausível considerar que outras metodologias de resolução são tentativas de realizar os processos de programação dinâmica de forma que necessite de menos recursos computacionais, e que seja independente do conhecimento das dinâmicas do ambiente.

A premissa desses métodos é tornar as equações de Bellman em regras de atualização, e, considerando as estimativas dos valores de cada estado, realizar o

ajuste dos mesmos com base nas probabilidades de cada trajetória da interação do agente com o ambiente.

2.4.1.1 Policy Evaluation (Predição)

No âmbito de aprendizado por reforço, são chamados de *policy evaluation*, ou *predição*, algoritmos onde o objetivo é obter os *resultados das funções de valor para cada estado*, seguindo uma política π. Ou seja, são problemas de cálculo do valor v_π para cada estado.

Partindo da equação de Bellman para estado (2.9),

$$v_\pi(s) = \sum_a \left(\pi(a|s) \sum_{s',r} \left(p(s', r|s, a)[r + \gamma v_\pi(s')] \right) \right),$$

teremos uma solução para a mesma se a tarefa for episódica, ou se $\gamma < 1$. Considerando que todas as probabilidades envolvidas sejam conhecidas, então, considerando todos os estados, a equação 2.9 pode ser descrita como um sistema de $|\mathcal{S}|$ equações, com $|\mathcal{S}|$ variáveis desconhecidas, onde $|\mathcal{S}|$ corresponde ao número de estados no conjunto de estados existentes $\mathcal{S}$. A solução deste problema é direta, porém é possível resolver de maneira mais simples computacionalmente se o cálculo for iterativo. Assim, considerando uma sequência de funções valores de estado $v_0(s), v_1(s), \ldots$, que mapeiam cada estado em $\mathcal{S}^+$ para um valor em $\mathbb{R}$, e considerando ainda que a estimativa inicial v_0 é definida arbitrariamente, podemos obter aproximações para v_π utilizando a seguinte regra de atualização,

$$v_{k+1}(s) = \sum_a \left(\pi(a|s) \sum_{s',r} \left(p(s', r|s, a)[r + \gamma v_k(s')] \right) \right), \tag{2.16}$$

para todo estado $s \in \mathcal{S}$. A próxima estimativa da função de valor v_π, referida como v_{k+1} é obtida através da equação de Bellman, considerando a estimativa atual, v_k, para calcular o valor do estado seguinte, presente no termo de recompensa da equação. Esta regra garante que a sequência $v_k \to v_\pi$ quando $k \to \infty$. Algoritmos que fazem uso dessa regra são chamados de *iterative policy evaluation.*

O pseudocódigo para a predição pode ser visto no Algoritmo 1. Para a representação dos valores dos estados, tanto do atual quanto dos novos valores, normalmente são utilizados vetores. Cada iteração realiza a atualização de todos os valores de estados disponíveis, sendo essa passagem por todos os estados chamada de *sweep*.

Algoritmo 1 Iterative Policy Evaluation em Programação Dinâmica

Define a política π a ser seguida e avaliada
Seja θ o threshold da variação mínima necessária para parada do algoritmo
Inicializa o vetor $V(s)$ de valores dos estados em $\mathcal{S}^+$, e faz $V(terminal) = 0$
Inicializa parâmetro de variação dos valores $\Delta = 0$

while $\Delta < \theta$ **do**
  $\Delta \leftarrow 0$
  for $s \in \mathcal{S}$ **do**
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \left(\pi(a|s) \sum_{s',r} \left(p(s', r|s, a)[r + \gamma V(s')] \right) \right)$
    $\Delta \leftarrow max(\Delta, |v - V(s)|)$
  end for
end while

2.4.1.2 Policy Improvement

Calcular os valores de cada estado ajuda a determinar se uma política melhor pode ser utilizada pelo agente. Por exemplo, considerando que um agente siga uma determinada política π, que avaliando um estado s determina que uma ação a_π seja escolhida a seguir. Porém, queremos saber se o agente deve mudar sua política para selecionar uma ação a diferente da que seria escolhida seguindo π. Uma forma de responder a essa pergunta seria escolher, nesse estado, a ação a, diferente do que $\pi(s)$ determina, e em seguida, a partir do novo estado, voltar a seguir $\pi(s)$. Ou seja, para a ação a, queremos determinar o resultado de sua função de valor $q_\pi(s, a)$ seguindo a política π,

$$q_\pi(s, a) = \sum_{s',r} p(s', r|s, a) \left[r + \gamma v_\pi(s') \right]. \tag{2.17}$$

Caso o valor devolvido seja maior do que o que seria obtido se o agente tivesse escolhido a ação a_π, então é plausível concluir que para o estado s, devemos escolher a ação a para obter um melhor resultado de recompensa total. Dessa forma, a política resultante π' se assemelha a π em todos os estados que não o estado s, onde a ação a ser escolhida deverá ser a. Este exemplo é um caso especial do chamado *policy improvement theorem*. Sejam π e π' qualquer par de políticas determinísticas de modo que, para todos os estados $s \in \mathcal{S}$,

$$q_\pi(s, \pi'(s)) \geq v_\pi(s). \tag{2.18}$$

Então a política π' deve ser tão boa ou melhor quanto a política π, ou seja, para todos os estados $s \in \mathcal{S}$,

$$v_{\pi'}(s) \geq v_\pi(s). \tag{2.19}$$

Seguindo essa linha de raciocínio, o agente pode querer determinar uma política que escolha sempre a melhor ação a cada estado, num comportamento chamado de *greedy*.

$$\begin{aligned} \pi'(s) &= \underset{a}{argmax}\ q_\pi(s, a) \\ &= \underset{a}{argmax}\ \mathbb{E}\left[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = a\right] \\ &= \underset{a}{argmax}\ \sum_{s', r} p(s', r|s, a)\left[r + \gamma v_\pi(s')\right], \end{aligned} \tag{2.20}$$

com empates sendo resolvidos aleatoriamente. O processo de construir uma nova política que melhore a anterior é chamado de *policy improvement*. Os raciocínios desenvolvidos aqui para políticas determinísticas podem facilmente ser considerados para políticas estocásticas também, onde a política $\pi(a|s)$ fornece probabilidades para cada ação do estado considerado.

2.4.1.3 Policy Iteration

Utilizando as ideias apresentadas nas seções anteriores, a partir de uma política inicial π, calculamos as funções de valor v_π para cada estado $s \in \mathcal{S}$ (policy

evaluation). Com os valores calculados, podemos procurar melhorar o desempenho do agente, gerando uma política melhorada π'. Com esta nova política melhorada, podemos fazer um novo cálculo das funções de valor, $v_{\pi'}$, e partir para a melhora da política novamente. Podemos seguir neste ciclo indefinidamente, até que idealmente atinjamos tanto a política ótima, quanto à função de valor ótima (SANTOS; RUST, 2004).

$$\pi_0 \xrightarrow{E} v_{\pi+0} \xrightarrow{I} \pi_1 \xrightarrow{E} v_{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \ldots \xrightarrow{I} \pi_* \xrightarrow{E} v_{\pi_*},$$

sendo $\xrightarrow{E}$ um passo de policy evaluation e $\xrightarrow{I}$ um passo de policy improvement. Esta heurística de obtenção da política ótima recebe o nome de *policy iteration*. Um algoritmo completo, retirado do livro base deste capítulo, pode ser visto no algoritmo 2 (SUTTON; BARTO, 2018).

Algoritmo 2 Policy Iteration em Programação Dinâmica

1 - Etapa de Inicialização
Define a política inicial $\pi(s)$ de maneira aleatória
Define a lista de valores para cada estado $V(s)$ de maneira aleatória
Seja θ o threshold da variação mínima necessária para parada do algoritmo

2 - Policy Evaluation
while $\Delta < \theta$ **do**
    $\Delta \leftarrow 0$
    for $s \in \mathcal{S}$ **do**
        $v \leftarrow V(s)$
        $V(s) \leftarrow \sum_a \Big(\pi(a|s) \sum_{s',r} \Big(p(s', r|s, a)[r + \gamma V(s')] \Big)\Big)$
        $\Delta \leftarrow max(\Delta, |v - V(s)|)$
    end for
end while

3 - Policy Improvement
Define parâmetro de parada *policy-stable* $\leftarrow$ *true*
for $s \in \mathcal{S}$ **do**
    old-action $\leftarrow \pi(s)$
    $\pi(s) \leftarrow \underset{a}{argmax} \ \sum_{s',r} p(s', r|s, a) \, [r + \gamma V(s')]$
    if *old-action* $\neq \pi(s)$ **then**
        policy-stable $\leftarrow$ *true*
    end if
end for
if *policy-stable* **then**
    return V e π
else
    Volta para o **passo 2 - Policy Evaluation**
end if

2.4.1.4 Value Iteration

Como citado pelos autores, a proposta de *policy iteration* possui a desvantagem de precisar recalcular os valores dos estados (policy evaluation) a cada iteração, de uma maneira também iterativa que necessita de vários *sweeps* no espaço de estados. A fim de melhorar o desempenho do algoritmo, este passo pode ser truncado de diversas maneiras, sem comprometer a convergência do valor. Uma delas corresponde à chamada *value iteration*, onde os passos de policy evaluation são truncados após um *sweep*, e juntamente com o passo de policy improvement, fornecem um cálculo de atualização da forma

$$v_{k+1}(s) = \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_k(s')] \quad \forall\ s \in \mathcal{S}. \tag{2.21}$$

De outra forma, pode ser entendido como tornar a equação ótima de Bellman uma regra de atualização, da mesma forma feita com a equação de Bellman na equação 2.16. Esta regra de atualização englobando os dois passos do algoritmo 2 faz com que o algoritmo agora precise apenas de um loop para a obtenção da política e das funções de valor, como pode ser visto no algoritmo 3.

O algoritmo para value iteration segue de maneira indefinida, até que a variação obtida na estimativa do valor de $v(s)$ seja menor do que um threshold. Para este valor convergir para $v_*(s)$ seriam necessários infinitos passos, mas estabelecendo um critério de parada dessa forma, podemos considerar que a convergência desse valor acontece assintoticamente.

Algoritmo 3 Value Iteration em Programação Dinâmica

Define a lista de valores para cada estado $V(s)$ de maneira aleatória
Seja θ o threshold da variação mínima necessária para parada do algoritmo

while $\Delta < \theta$ **do**
  $\Delta \leftarrow 0$
  for $s \in \mathcal{S}$ **do**
    $v \leftarrow V(s)$
    $V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$
    $\Delta \leftarrow max(\Delta, |v - V(s)|)$
  end for
end while

Determina a política determinística $\pi \approx \pi_*$ da forma:
$\pi(s) = \underset{a}{argmax} \ \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$

2.4.1.5 Generalized Policy Iteration

Usamos o termo *generalized policy iteration* (GPI) para a ideia geral da interação entre os passos de policy evaluation e policy iteration, independentemente de como ele ocorre. Nas seções anteriores, por exemplo, foram vistas duas formas de interação, policy iteration e value iteration. Este conceito é importante, visto que a ampla maioria dos métodos de aprendizado por reforço podem ser explicados de maneira geral por GPI.

O processo é caracterizado por etapas sucessivas, onde as funções de valor são ajustadas para se adequarem ao que seria obtido pelo agente seguindo uma política específica, e com estes valores uma nova política é determinada. A partir disso, uma nova etapa para adequação dos valores se segue, e esta interação continua indefinidamente, como já visto na seção 2.4.1.3. Um diagrama típico pode ser visto na Figura 4.

Se ambos os processos de evaluation e improvement se estabilizarem, isto é, não mais produzirem melhorias significativas em seus valores, podemos considerar que ambas política e função de valor são ótimos, uma vez que a convergência para a função ótima é garantida (SANTOS; RUST, 2004). Por sua vez, a função de valor estabiliza quando entra em acordo com a política π vigente, e a política π

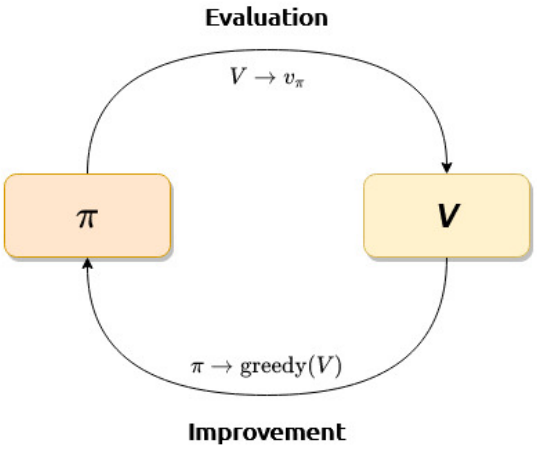


Figura 4 – Generalized Policy Iteration. Adaptado de (SUTTON; BARTO, 2018)

estabiliza quando corresponde a uma política gulosa (*greedy*) com as funções de valor atuais.

Os processos de evaluation e improvement em GPI podem ser compreendidos como competindo e cooperando entre si, ao mesmo tempo. Eles competem no sentido que ambos tentam se sobrepujar, puxando o ciclo em direções opostas. Fazer a política ser gulosa com respeito às funções de valor atuais, em geral às torna incorretas para a política atual, e tornar as funções corretas faz com que a política não seja mais gulosa em relação às funções. Porém, à medida que as interações prosseguem, todo esse processo leva a um mesmo objetivo, que é a convergência para os termos ótimos de política e função de valor, assim, cooperando entre si.

De maneira geral, os autores simplificam visualmente esta interação como visto na Figura 5. Cada processo leva os valores de v e π para o valor consistente com o outro termo atual, e ambos são levados para mais próximos de seus valores ótimos, a cada passo.

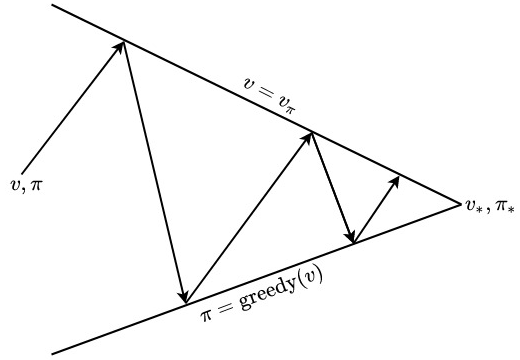


Figura 5 – Diagrama geométrico de interação GPI. Adaptado de SUTTON; BARTO, 2018.

2.4.2 Métodos de Monte Carlo

Sem a necessidade de conhecer as dinâmicas completas do ambiente, fazemos uso então de métodos para estimar as funções de valor. Os métodos de Monte Carlo são o primeiro conjunto deles apresentado neste capítulo, e fazem uso apenas da experiência do agente com o ambiente, a partir de amostras de sequências de estados e recompensas recebidas. Pode fazer uso tanto de experiências reais vivenciadas, quanto de experiências *simuladas*, onde é necessário um modelo dos estados, mas contrário à programação dinâmica, apenas um modelo de amostras das transições, e não da dinâmica completa do ambiente.

Métodos de Monte Carlo procuram resolver o problema de aprendizado por reforço a partir de médias dos valores de recompensas das amostras das interações. Para obtermos uma estimativa mais acurada com a experiência vivida, assumimos que os valores estão disponíveis quando o cálculo for feito. Sendo assim, da mesma forma que os autores destacam no livro, definiremos os métodos apenas para tarefas episódicas. Ao final de cada episódio, as funções de valor são atualizadas com as médias dos retornos observados no mesmo, e com uma série de episódios, refinamos a estimativa.

Métodos de Monte Carlo expandem os conceitos apresentados em programação dinâmica, com a diferença de que, ao invés de calcular as funções de valor, estes métodos procuram aprender as mesmas, seguindo um raciocínio ainda parecido com o GPI apresentado na seção 2.4.1.5.

2.4.2.1 Predição com Monte Carlo

Como dito, a premissa principal dos métodos de Monte Carlo é a de que é possível saber qual o valor esperado de um estado obtendo a média dos retornos (média das recompensas de todos os estados futuros, a partir do mesmo) todas as vezes que este estado é visitado pelo agente.

Vamos supor uma situação em que o objetivo seja estimar o valor de um estado sob uma política π, $v_\pi(s)$, observando todas as interações que já ocorreram entre agente e ambiente. Cada ocorrência do estado s nas interações é chamada *visita* ao estado s. Um estado pode ser visitado várias vezes ao longo de um episódio, por isso vamos chamar a primeira vez que o mesmo é observado de *primeira visita a s*. O primeiro algoritmo mostrado aqui, chamado de *Método de Monte Carlo da primeira visita*[3] estima o valor dos estados, $v_\pi(s)$ observando as primeiras ocorrências dos mesmos. O algoritmo 4 mostra a implementação em pseudocódigo do primeiro método.

Algoritmo 4 Predição de Monte Carlo da primeira visita

Define-se a política π a ser seguida.
Inicializa vetor $V(s)$ arbitrariamente, para todo $s \in \mathcal{S}$.
Inicializa uma lista de listas vazias $Retorno(s)$, com $|\mathcal{S}|$ posições.

while {*algum critério de parada não cumprido*} **do**
  Gera ou obtém um episódio seguindo π que siga:
    $S_0, A_0, R_1, S_1, A_1, \ldots, S_{T-1}, A_{T-1}, R_T, S_T$
  Inicializa retorno total do episódio $G \leftarrow 0$
  for cada intervalo de tempo $t = T-1, T-2, \ldots, 0$ **do**
    $G \leftarrow \gamma G + R_{t+1}$
    if estado S_t não estiver em $S_0, S_1, \ldots, S_{t-1}$ **then**
      Apenda G a $Retornos(S_t)$
      Calcula $V(S_t) \leftarrow media(Retornos(S_t))$
    end if
  end for
end while

De acordo com o algoritmo 4, os valores para os estados são calculados de maneira inversa, a partir do último estado não-terminal visitado, e cada estado

[3] Tradução livre do termo utilizado no livro, *first-visit Monte Carlo Method*.

anterior visitado recebe os valores descontados dos estados seguintes, de maneira cumulativa, somados com os retornos correspondentes. É fácil ter a intuição de que, à medida que mais episódios ocorrem, melhor será a estimativa dos valores dos estados, onde pela lei dos grandes números, à medida que o número de episódios tende a infinito, a estimativa v_π converge aos valores reais de cada estado.

Um ponto importante a ser destacado sobre métodos de Monte Carlo é que a estimativa dos valores de cada estado são independentes entre si. Isto é, não fazemos o cálculo do valor com base na estimativa do valor dos próximos estados, dentro de um episódio. Em termos práticos, isto significa que, em cada iteração do loop de episódios, quando calculamos o valor de um estado S_t, usamos o valor final obtido, até este momento, do estado seguinte, S_{t+1}, no cálculo de G_t, e não um valor G_{t+1} *parcial*, ainda a ser aprimorado dentro do loop. Dizemos que os métodos de Monte Carlo não fazem *bootstrap*, como por exemplo é feito no caso de programação dinâmica (veja o algoritmo 3). Ter este conceito em mente é importante, para entender as diferenças destes métodos com os demais, como por exemplo *Temporal-difference learning*. É fácil perceber, também, que para esta categoria de algoritmos, os valores só serão calculados quando os episódios terminarem.

2.4.2.2 Estimativa de Monte Carlo para Valores de Ação

Antes de avançar para tarefas de obtenção de políticas utilizando Monte Carlo, alguns conceitos devem ser abordados. Com um modelo do ambiente disponível, apenas os valores dos estados são suficientes para determinar uma política, uma vez que, estando em um estado $S_t = s$, sabemos quais estados se seguirão para cada ação possível apenas observando as probabilidades $p(s', r|s, a)$. A partir daí, basta escolhermos a ação cujo estado S_{t+1} apresente o maior valor. Sem um modelo, porém, não é possível saber como o ambiente vai se comportar se uma determinada ação for escolhida. Por isso torna-se particularmente útil determinar os valores para as *ações*, do que para os estados. Assim, um dos objetivos principais dos métodos de Monte Carlo torna-se estimar q_*.

Com isto em mente, partimos para o raciocínio dos problemas de *policy evaluation para valores de ação*. Similar ao abordado na seção 2.4.1.1, o objetivo

deste tipo de problema é estimar $q_\pi(s, a)$, ou seja, o valor obtido ao escolhermos a ação a no estado s, e então seguirmos a política π. Os algoritmos seguem de forma similar ao apresentado no algoritmo 4, com a diferença de que agora falamos de primeiras visitas a pares de estado-ação, e não somente a ações.

Existe um problema nessa abordagem, que é o fato de que grande parte das ações existentes nos estados não serão visitadas. Fica claro chegar a essa conclusão se pensarmos que agora não temos só $|\mathcal{S}|$ elementos a serem visitados, mas $|\mathcal{S}| \times |\mathcal{A}|$. Se a política sendo seguida ainda for determinística, isso significa que apenas uma ação de cada estado será visitada, e poderá ser estimada. A solução exige que seja mantida uma exploração do espaço de ações em cada estado, de forma a garantir que os pares de estado-ação sejam visitados ao menos uma vez. Uma das formas de se fazer isso é estabelecer que os episódios se iniciem em um par aleatório de estado-ação, de forma que todos os pares possuam uma mesma probabilidade de ocorrer. Se o número de episódios for infinito, cada par pode ser vistado também um número infinito de vezes. Este tipo de proposição recebe o nome de *exploring starts*.

A proposição de exploring starts é particularmente útil quando falamos de episódios gerados sinteticamente, porém em episódios reais, não é possível contar sempre com esta característica, quando programamos um agente para interagir com o ambiente. Nestes casos, é útil considerarmos apenas políticas estocásticas com probabilidades diferentes de zero de seleção para todas as ações em um estado.

2.4.2.3 Controle com Monte Carlo

No âmbito do aprendizado por reforço, chamamos de *controle* a tarefa de aproximar políticas ótimas. O raciocínio segue o mesmo padrão visto em GPI, com a diferença que, no passo correspondente à *policy evaluation*, o agente procura otimizar o valor das funções de ação.

Considere esta nova versão de *policy iteration*, considerando funções de valor de ação

$$\pi_0 \xrightarrow{E} q_{\pi+0} \xrightarrow{I} \pi_1 \xrightarrow{E} q_{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \ldots \xrightarrow{I} \pi_* \xrightarrow{E} q_{\pi_*},$$

sendo $\xrightarrow{E}$ um passo de policy evaluation e $\xrightarrow{I}$ um passo de policy improvement. Vamos assumir que podemos vivenciar um número infinito de episódios, e que os mesmos são gerados com exploring starts. Sendo assim, nestes passos, os métodos de Monte Carlo serão capazes de calcular q_{π_k} de maneira exata, para a política π_k de cada iteração.

O passo de policy improvement é realizado ao fazermos a política ser gulosa em respeito à função de valor. Como temos disponível a informação dos valores das ações, basta escolhermos a melhor, a cada estado,

$$\pi(s) = \underset{a}{argmax}\ q(s, a) \tag{2.22}$$

Para obter um algoritmo realizável, precisamos lidar com os dois pontos assumidos para a situação, de que temos um número infinito de episódios, e de que conseguimos inicializar cada um com exploring starts. Para o primeiro ponto, podemos utilizar raciocínio similar ao utilizado para o algoritmo 3, e realizar policy evaluation em um passo. O pseudo código para o controle utilizando Monte Carlo, para este caso, pode ser visto no algoritmo 5.

Para lidar com a questão de *exploring starts*, precisamos de uma outra forma de garantir uma contínua exploração do espaço de estados e ações. Neste âmbito, apresentamos dois novos conceitos importantes quando caracterizamos algoritmos de aprendizado por reforço: *on-policy* e *off-policy*. Considere um agente interagindo com o ambiente. Ele toma suas ações seguindo uma política, e ao mesmo tempo procura obter uma política que apresente um resultado melhor, em termos de retorno. Algoritmos *on-policy* são aqueles que avaliam ou procuram aprimorar a mesma política que o agente está seguindo para interagir. Sendo assim, cada passo de melhora da mesma influencia em como o agente vai interagir com os estados nas próximas iterações. Por sua vez, métodos *off-policy* interagem com o ambiente seguindo uma política pré-determinada, enquanto observam os resultados de suas interações para obter uma outra política que seja ótima, mas sem mudar o seu comportamento atual. Por exemplo um agente que procura a melhor política determinística para agir, enquanto que se comporta seguindo uma política estocástica, que o leve a explorar mais todo o espaço de ações possíveis em cada

Algoritmo 5 Controle com Monte Carlo e Exploring Starts

Define uma política inicial $\pi(s) \in \mathcal{A}(s)$ arbitrária (lista de ações a serem tomadas a cada estado)
Define uma matriz de valores de ação por estado $Q(s, a) \in \mathbb{R}$ arbitrária.
Inicializa uma matriz de listas vazias $Retorno(s, a)$, com $|\mathcal{S}| \times |\mathcal{A}|$ posições.

while {*algum critério de parada não cumprido*} **do**
  Exploring Starts: Escolhe um par de Estado e Ação iniciais aleatoriamente, $S_0 \in \mathcal{S}$ e $A_0 \in \mathcal{A}$
  Gera um episódio a partir de S_0 e A_0, seguindo π:
    $S_0, A_0, R_1, S_1, A_1, \ldots, S_{T-1}, A_{T-1}, R_T, S_T$
  Inicializa retorno total do episódio $G \leftarrow 0$
  for cada intervalo de tempo $t = T-1, T-2, \ldots, 0$ **do**
    $G \leftarrow \gamma G + R_{t+1}$
    if par estado-ação S_t, A_t não estiver em $S_0, A_0, S_1, A_1, \ldots, S_{t-1}, A_{t-1}$ **then**
      Apenda G a $Retornos(S_t, A_t)$
      Calcula $Q(S_t, A_t) \leftarrow media(Retornos(S_t, A_t))$
      $\pi(S_t) = \underset{a}{argmax}\ Q(S_t, a)$
    end if
  end for
end while

estado.

Em métodos *on-policy*, a política geralmente segue um padrão do tipo $\pi(a|s) > 0$ para todas as combinações de estados e ações, em $\mathcal{S}$ e $\mathcal{A}$, e é chamada de *soft*. Uma abordagem popular deste tipo de implementação é a chamada *ϵ-greedy*, onde a política, na maior parte do tempo, vai procurar sempre escolher a melhor ação[4] a cada estado (a ação ótima), porém em outros momentos opta por escolher uma ação qualquer aleatória, dentre todo o espaço de ações, com uma probabilidade ϵ. Ou seja,

- Para ações que *não sejam a melhor ação* (ou as melhores ações, que devolvam o maior retorno), a cada estado, a probabilidade de serem escolhidas é de

[4] Ou melhores ações, já que duas ou mais ações podem devolver o mesmo valor de retorno, em um estado.

$\frac{\epsilon}{|\mathcal{A}|}$.

- Para a melhor ação do estado (os as melhores ações), a probabilidade de ser escolhida é de $\frac{\epsilon}{|\mathcal{A}|}$ - quando o agente opta por escolher uma ação aleatória do espaço de ações - adicionado de $\frac{1-\epsilon}{número\ de\ ações\ ótimas}$.

Este algoritmo é um exemplo do chamado grupo ϵ-*soft*, políticas onde $\pi(a|s) > \frac{\epsilon}{|\mathcal{A}|}$ para todos os estados e ações.Outros tipos de implementação para políticas *soft* também podem ser encontradas na literatura, como por exemplo políticas onde a escolha de ações segue uma probabilidade uniforme para todo elemento em $\mathcal{A}$. Uma implementação do método de controle de Monte Carlo, utilizando uma política ϵ-*soft on-policy* pode ser vista no algoritmo 6.

Os métodos *on-policy* procuram aprender não uma política ótima, mas uma que se aproxime, uma vez que ainda precisam ter o compromisso de explorar o espaço de ações, em busca de resultados melhores. Mas para a obtenção de uma política que realmente possa ser considerada ótima para a interação vivida, uma outra abordagem deve ser feita. Nesse contexto, enta a outra metodologia de comportamento dos agentes, os métodos *off-policy*. Nesta abordagem, o agente faz uso de duas políticas, uma que o agente busca aproximar da política ótima, e é a que está sendo de fato aprendida, chamada *política alvo*, e outra que o agente usa para se comportar, chamada de *política comportamental*.

Segundo Sutton e Barto mencionam em seu livro, métodos *on-policy* geralmente são mais simples de serem entendidos e implementados, e apresentam menor variância e maior velocidade de convergência, se comparados com métodos *off-policy*. Porém estes são mais "poderosos"e generalizáveis. Métodos *on-policy* inclusive podem ser considerados como sendo parte de métodos *off-policy*, onde as políticas *alvo* e *comportamental* são a mesma. Métodos *off-policy* apresentam ainda uma variedade de usos úteis, por exemplo, um agente pode aprender uma política ótima enquanto observa as interações de um usuário real, humano. A política alvo é determinada via algoritmo, e a política comportamental é determinada pelo humano.

Para entender melhor o conceito de métodos de controle *off-policy*, vale analisar primeiro como se comporta em tarefas de predição.

Algoritmo 6 Controle *On-policy* com Monte Carlo sem Exploring Starts

Define um parâmetro inicial $\epsilon > 0$ pequeno.
Define uma política ϵ-*soft* inicial $\pi(s) \in \mathcal{A}(s)$ arbitrária
Define uma matriz de valores de ação por estado $Q(s, a) \in \mathbb{R}$ arbitrária.
Inicializa uma matriz de listas vazias $Retorno(s, a)$, com $|\mathcal{S}| \times |\mathcal{A}|$ posições.

while {*algum critério de parada não cumprido*} **do**
  Gera um episódio seguindo π:
    $S_0, A_0, R_1, S_1, A_1, \ldots, S_{T-1}, A_{T-1}, R_T, S_T$
  Inicializa retorno total do episódio $G \leftarrow 0$
  for cada intervalo de tempo $t = T-1, T-2, \ldots, 0$ **do**
    $G \leftarrow \gamma G + R_{t+1}$
    if par estado-ação S_t, A_t não estiver em $S_0, A_0, S_1, A_1, \ldots, S_{t-1}, A_{t-1}$ **then**
      Apenda G a $Retornos(S_t, A_t)$
      Calcula $Q(S_t, A_t) \leftarrow media(Retornos(S_t, A_t))$
      Define a ação ótima do estado, onde empates são resolvidos de maneira arbitrária: $A^* = \underset{a}{argmax}\ Q(S_t, a)$
      for $a \in \mathcal{A}(S_t)$ **do**

$$\pi(a|S_t) \leftarrow \begin{cases} 1 - \epsilon + \frac{\epsilon}{|\mathcal{A}(S_t)|} & \text{se } a = A^* \\ \frac{\epsilon}{|\mathcal{A}(S_t)|} & \text{se } a \neq A^* \end{cases}$$

      end for
    end if
  end for
end while

2.4.2.4 Predição com Métodos *Off-policy*

Relembrando, neste tipo de tarefa, as políticas são pré-fixadas, e o objetivo torna-se ajustar as estimativas dadas pelas funções de valor, seguindo as mesmas. No caso de métodos *off-policy*, teremos o agente tentando estimar os valores que seriam obtidos seguindo a política *alvo* π, enquanto o mesmo comporta-se de acordo com uma política *comportamental* $b \neq \pi$.

Considerando que políticas sejam informações de probabilidade de ações ocorrerem dado um estado, o que desejamos é que, a partir de amostras de ações obtidas a partir da distribuição b, obtenhamos amostras que sigam uma distribuição π. Exitem teoremas estatísticos que nos possibilitam alcançar este resultado, e

neste caso podemos aplicar o que chamamos de *amostragem por importância*. Para que b consiga ser utilizada para estimar π, é necessário que ações em π também apareçam em b, em algum momento. Ou seja, se $\pi(a|s) > 0$, então $b(a|s) > 0$, estamos assumindo *cobertura*. Uma conclusão que este resultado implica é que a política b deve ser estocásticas nos estados onde $b \neq \pi$ (para que as ações de π também possam aparecer), porém π pode ser determinística.

Aplicamos amostragem por importância ponderando os retornos de acordo com a probabilidade relativa de suas trajetórias ocorrendo entre as políticas alvo e comportamental, chamada de *razão de amostragem por importância*. Em outras palavras, dado um estado inicial S_t, e a probabilidade da trajetória subsequente, $A_t, S_{t+1}, A_{t+1}, \ldots, S_T$, que ocorreria sob uma política π, dada por

$$\begin{aligned} &Pr\{A_t, S_{t+1}, A_{t+1}, \ldots, S_T | S_t, A_{t:T-1} \sim \pi\} \\ &\quad = \pi(A_t|S_t)p(S_{t+1}|S_t, A_t)\pi(A_{t+1}|S_{t+1}) \ldots p(S_T|S_{T-1}, A_{T-1}) \\ &\quad = \prod_{k=t}^{T-1} \pi(A_k|S_k)p(S_{k+1}|S_k, A_k), \end{aligned}$$

ou seja, a probabilidade de, dado um estado S_t, a ação escolhida ser A_t, que é dada por π, multiplicada pela probabilidade de o próximo estado ser S_{t+1}, dada por p, e assim por diante, para toda a trajetória. A razão das probabilidades desta trajetória ocorrer, sob π e sobre b é

$$\begin{aligned} \rho_{t:T-1} &= \frac{\prod_{k=t}^{T-1} \pi(A_k|S_k)p(S_{k+1}|S_k, A_k)}{\prod_{k=t}^{T-1} b(A_k|S_k)p(S_{k+1}|S_k, A_k)} \\ &= \prod_{k=t}^{T-1} \frac{\pi(A_k|S_k)}{b(A_k|S_k)}. \end{aligned} \tag{2.23}$$

Como as probabilidades p das transições são as mesmas sob as duas políticas, no final a razão depende apenas das definições das mesmas.

O agente tem apenas os retornos esperados G_t das interações com o ambiente seguindo a política b, consequentemente apenas $\mathbb{E}[G_t|S_t = s] = v_b(s)$. Portanto, para obter o retorno de acordo com π, uma correção deve ser feita, da forma

$$\mathbb{E}[\rho_{t:T-1}G_t|S_t = s] = v_\pi(s). \tag{2.24}$$

Para evoluir o raciocínio de forma a obter $v_\pi(s)$ via Monte Carlo, devemos incluir algumas definições. Primeiramente registraremos todas as visitas a um estado s, considerando todos os episódios ocorridos, em um conjunto $\tau(s)$. Para métodos de primeira-visita, registramos todas os tempos em que um estado foi visitado uma primeira vez, por episódio. No caso de toda-visita, todas as ocorrências de s. Os autores sugerem considerar episódios ocorrendo em sequência, por exemplo, se um episódio foi do tempo 1 ao 100, então o próximo episódio começa no tempo 101, de forma a podermos avaliar as interações sem os limites entre os episódios. Seja $T(t)$ o intervalo de tempo onde ocorre a visita ao estado terminal (no caso do exemplo citado, seria o intervalo 100), e G_t o retorno ao final de um episódio. Sendo assim, $\{G_t\}_{t\in\tau(s)}$ são os retornos do estado s, em todo o processo de interação, e $\{\rho_{t:T(t)-1}\}_{t\in\tau(s)}$ são as razões de amostragem correspondentes a cada retorno. Assim, para obter o valor da estimativa de $v_\pi(s)$, ajustamos os retornos obtidos como 2.24, e obtemos a média considerando o número de registros salvos em $\tau(s)$,

$$V(s) = \frac{\sum_{t\in\tau(s)} \rho_{t:T(t)-1}G_t}{|\tau(s)|}, \tag{2.25}$$

em um cálculo chamado de *amostragem por importância ordinal*. Uma variação é a chamada *amostragem por importância ponderada*, calculada da forma

$$V(s) = \frac{\sum_{t\in\tau(s)} \rho_{t:T(t)-1}G_t}{\sum_{t\in\tau(s)} \rho_{t:T(t)-1}}, \tag{2.26}$$

onde $V(s) = 0$ caso o denominador seja zero. Os dois cálculos podem ser usados, tendo em mente suas características e diferenças. A amostragem ordinal não possui viés, onde a amostragem ponderada apresentaria (como no caso da estimativa quando temos por exemplo um único retorno G_t), mesmo esse viés convergindo assintoticamente para zero. Em compensação, a amostragem ordinal apresenta uma variância não limitada, onde na amostragem ponderada, o maior peso que um retorno pode ter é um.

A implementação desta lógica pode ser feita de maneira incremental, episódio a episódio, com particularidades dependendo de qual tipo de amostragem é utilizada. De maneira simplificada, para amostragens ordinais, utilizamos uma regra de atualização para os valores de ação, a cada retorno n de um estado s, aparecendo em sequência, como

$$\begin{aligned} V_{n+1}(s) &= \frac{1}{n}\sum_{k=1}^{n}\rho_k G_k \\ &= \frac{1}{n}\left(\rho_n G_n + \sum_{k=1}^{n-1}\rho_k G_k\right) \\ &= \frac{1}{n}\left(\rho_n G_n + (n-1)\frac{1}{n-1}\sum_{k=1}^{n-1}\rho_k G_k\right) \\ &= \frac{1}{n}\left(\rho_n G_n + (n-1)V_n\right) \\ &= \frac{1}{n}\left(\rho_n G_n + nV_n - v_n\right) \\ &= V_n + \frac{1}{n}\left[\rho_n G_n - V_n\right], \end{aligned} \tag{2.27}$$

onde $n = |\tau(s)|$, que é o número de ocorrências do retorno observadas. No caso de amostragens ponderadas, um raciocínio diferente deve ser utilizado. Suponha uma sequência de retornos $G_1, G_2, \ldots, g_{n-1}$ iniciados a partir de um mesmo estado, onde utilizamos um *peso* W_i correspondente, nesse caso $W_i = \rho_{t:T(t_i)-1}$. A estimativa para um determinado momento da interação do agente com o ambiente é

$$V_n = \frac{\sum_{k=1}^{n-1} W_k G_k}{\sum_{k=1}^{n-1} W_k}, \tag{2.28}$$

que é similar ao que a equação 2.26 define. Considerando que obtemos um novo valor de retorno G_n, e queremos utilizá-lo para atualizar V_n. Adicionalmente, atualizamos a soma dos pesos, $\sum_{k=1}^{n-1} W_k$, com o novo valor de peso correspondente a este retorno, W_n, em uma variável responsável por guardar o valor da soma cumulativa a cada passo, C_n. A regra de atualização, nesse caso, é

$$V_{n+1} = V_n + \frac{W_n}{C_n}\left[G_n - V_n\right] \quad n \geq 1, \tag{2.29}$$

e

$$C_{n+1} = C_n + W_{n+1}, \tag{2.30}$$

com $C_0 = 0$, e V_1 arbitrário. O algoritmo 7 ilustra como seria essa implementação para estimar funções de valor de ação.

Algoritmo 7 Predição *Off-policy* com Monte Carlo

Define uma política inicial $\pi(s) \in \mathcal{A}(s)$ arbitrária
Define uma matriz de valores de ação por estado $Q(s, a) \in \mathbb{R}$ arbitrária.
Inicializa uma matriz $C(s, a) \leftarrow 0$ para todas as posições.

while {*algum critério de parada não cumprido*} **do**
  $b \leftarrow$ uma política comportamental com *cobertura* a π
  Gera um episódio seguindo b:
    $S_0, A_0, R_1, S_1, A_1, \ldots, S_{T-1}, A_{T-1}, R_T, S_T$
  Inicializa retorno total do episódio $G \leftarrow 0$
  Inicializa peso $W \leftarrow 1$
  for cada intervalo de tempo $t = T-1, T-2, \ldots, 0$ **do**
    $G \leftarrow \gamma G + R_{t+1}$
    $C(S_t, A_t) \leftarrow C(S_t, A_t) + W$
    $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_T)}|G - Q(S_t, A_t)|$
    $W \leftarrow W \cdot \frac{\pi(A_t|S_t)}{b(A_t, S_t)}$
  end for
end while

2.4.2.5 Controle com métodos *Off-policy* de Monte Carlo

Com os conceitos abordados, podemos seguir para a definição de métodos de controle *off-policy* utilizando Monte Carlo. Como descrito anteriormente, métodos de controle *off-policy* utiliza uma política chamada de comportamental para relacionar-se com o abmiente, enquanto procuram aprimorar uma outra política, chamada de alvo. Uma vantagem desse tipo de abordagem é que a política alvo pode ser determinística, sem a necessidade de implementar artifícios para incentivar a exploração do espaço de ações, enquanto a política comportamental pode ser estocástica, sem preocupar-se em ser ótima. Um outro benefício é possibilidade de usar eventos ocorridos com agentes reais, ou de interações onde não temos

acesso à política, mas apenas aos acontecimentos em si. Dessa forma, uma tarefa de aprendizado por reforço que procure aprimorar a habilidade de jogar um jogo, por exemplo, pode fazer uso de episódios de jogos vivenciados por humanos, onde os mesmos seguem seu comportamento próprio, não necessariamente preocupados com o agente em si.

No caso de controle *off-policy* utilizando Monte Carlo, existe uma necessidade de que a política comportamental apresente uma probabilidade diferente de zero, para todos os pares de ação-estado, de serem selecionados. Uma vez que procuramos obter a política ótima, existe a necessidade de que possamos explorar todo o espaço possível em sua busca, por isso a política comportamental deve ser também *soft* (diferentemente da predição *off-policy*, onde a política necessitávamos apenas que b oferecesse cobertura à política π). O algoritmo 8, para esta tarefa, pode ser obtido a partir do algoritmo 7 de predição.

Algoritmo 8 Controle *Off-policy* com Monte Carlo

Define uma matriz de valores de ação por estado $Q(s,a) \in \mathbb{R}$ arbitrária.
Inicializa uma matriz $C(s,a) \leftarrow 0$ para todas as posições.
Define uma política inicial $\pi(s) \leftarrow \underset{a}{argmax}\, Q(s,a)\ \forall\ s \in \mathcal{S}$

while {*algum critério de parada não cumprido*} **do**
  $b \leftarrow$ qualquer política *soft*
  Gera um episódio seguindo b:
    $S_0, A_0, R_1, S_1, A_1, \ldots, S_{T-1}, A_{T-1}, R_T, S_T$
  Inicializa retorno total do episódio $G \leftarrow 0$
  Inicializa peso $W \leftarrow 1$
  for cada intervalo de tempo $t = T-1, T-2, \ldots, 0$ **do**
    $G \leftarrow \gamma G + R_{t+1}$
    $C(S_t, A_t) \leftarrow C(S_t, A_t) + W$
    $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_T)}|G - Q(S_t, A_t)|$
    $\pi(S_t) \leftarrow \underset{a}{argmax}\, Q(S_t, a)$
    if $A_t \neq \pi(S_t)$ **then**
      Termina de iterar sobre o episódio atual, e segue para o próximo
    end if
    $W \leftarrow W \cdot \frac{1}{b(A_t, S_t)}$
  end for
end while

Como pode ser visto no algoritmo 8, existe uma verificação no final do loop interno, onde caso a ação escolhida no episódio seja diferente da melhor ação que seria determinada pela política alvo, o algoritmo para de avaliar o episódio atual, e segue para o próximo. Como o objetivo é determinar as funções de valor de ação para a política π, e sendo a mesma determinística, todas as probabilidades para as ações não ótimas, sob π, são zero. Sendo assim, como o algoritmo faz isso ajustando a probabilidade de uma determinado trajeto ocorrer sob π, caso uma ação não ótima seja escolhida, a probabilidade do trajeto como um todo ocorrer, em π, é zero - afinal π escolhe *sempre* a ação ótima a cada estado. Com isso, não há mais necessidade de se avaliar o episódio corrente, e podemos partir para o próximo.

2.4.3 Aprendizado por Diferenças Temporais

Aprendizado por diferenças temporais, ou TD[5] pode ser compreendido como uma "evolução"no raciocínio inicialmente proposto por métodos de Monte Carlo. É quase como uma combinação do mesmo com metodologias vistas em programação dinâmica, e é o ponto de partida para algoritmos mais complexos e mais comumente utilizados na área. Segundo Sutton e Barto descrevem em seu livro, este tipo de aprendizado herda de Monte Carlo a capacidade de aprender através de amostras de experiências vividas pelo agente, sem a necessidade de um modelo da dinâmica do ambiente, enquanto também herda de programação dinâmica a característica de atualizar suas estimativas à medida que as interações ocorrem, sem a necessidade de esperar o fim de um episódio para fazer as atualizações das funções de valor. Justamente por isso, apresentam métodos eficientes para uso tanto em tarefas episódicas quanto em contínuas.

2.4.3.1 Predição por Diferenças Temporais

Como Monte Carlo, métodos que utilizam TD usam a experiência anterior de interações com o ambiente para poder estimar as funções de valor dos estados, ou de pares estado-ação. A diferença principal desta abordagem da anterior é que métodos que fazem uso de Monte Carlo necessitam que um episódio termine para

[5] Temporal-difference.

poderem ajustas suas estimativas, uma vez que o princípio de atualização dos valores parte da obtenção de um valor médio V de todos os retornos observados durante as interações. Uma função de atualização simplificada seria algo como a seguir, onde assim que um novo retorno de um estado é computado, o valor da média atual é atualizado de maneira incremental, semelhante à equação 2.27,

$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)].$$

sendo α um parâmetro de *taxa de aprendizado* (*step size*) da função.

De modo contrário, métodos que fazem uso de diferenças temporais podem atualizar as estimativas V das funções de valor ao longo de um mesmo episódio, baseando-se no valor da estimativa *atual* que ele possui. Esta abordagem, chamada de *bootstrapping*, é semelhante ao que algoritmos de programação dinâmica fazem, utilizando um valor *não-corrigido* para melhorar a estimativa. Ou seja, ao contrário de Monte Carlo, que utilizam o valor da média *final*, obtido dos episódios, como alvo da aproximação, métodos TD utilizam um valor *aproximado* desta média, ainda em processo de ajuste, como alvo da próxima atualização da mesma. Com isso, ao invés de atualizar os valores *a cada episódio*, estes métodos atualizam os valores *a cada intervalo de tempo*. A função de atualização seria algo do tipo

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)], \tag{2.31}$$

ou seja, no próximo intervalo de tempo, para um estado S_t, seu valor é atualizado usando o retorno estimado atual, $R_{t+1} + \gamma V(S_{t+1})$. Esta função é a base dos algoritmos de aprendizado por reforço que serão vistos daqui pra frente. Um algoritmo básico de predição a partir de diferenças temporais pode ser visto no algoritmo 9.

O valor dado entre os colchetes na equação 2.31 pode ser entendido como uma espécie de erro, entre a estimativa atual $V(S_t)$, e uma estimativa melhor $R_{t+1}+\gamma V(S_{t+1})$. Este valor é importante em aprendizado por reforço, é é conhecido como *erro TD*,

$$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t), \tag{2.32}$$

Algoritmo 9 Predição com Diferenças Temporais

Inicializa com uma política π a ser avaliada
Inicializa um $\alpha \in (0, 1]$
Define um vetor $V(s)$ $\forall s \in \mathcal{S}$ arbitrariamente, e $V(terminal) = 0$.

for *episódio* em uma lista de episódios **do**
    Inicializa um estado S como inicial
    while S não é terminal **do**
        $A \leftarrow \pi(S)$
        Realiza a ação A e observa os retornos S' e R
        $V(S) \leftarrow V(S) + \alpha[R + \gamma V(S') - V(S)]$
        $S \leftarrow S'$
    end while
end for

onde o mesmo é relativo ao tempo t, e se altera com o passar das interações, na medida que as próprias estimativas também se alteram. Além disso como utiliza o valor do próximo estado, e usa a próxima recompensa, o erro para o tempo t só estará disponível no tempo $t + 1$. É importante mencionar também que, se os valores das estimativas V não se alterarem ao longo de um episódio, o erro de estimativa que seria obtido com Monte Carlo ($G_t - V(S_t)$) pode ser escrito como um somatório de erros TD,

$$G_t - V(S_t) = \sum_{k=t}^{T-1} \gamma^{k-t} \delta_k. \tag{2.33}$$

Podemos seguir agora para o entendimento de métodos de controle utilizando TD, que formam a base para diversos algoritmos vistos daqui pra frente.

2.4.3.2 Sarsa: Controle TD *on-policy*

Começamos o desenvolvimento dos métodos de controle a partir da definição básica de GPI, alterando a metodologia para utilizar métodos TD para a predição. Assim como visto para métodos de controle com Monte Carlo, também faremos distinção de métodos *on-policy* e *off-policy*. O primeiro método apresentado é de controle *on-policy*, denominado *Sarsa*.

O objetivo é aprender funções de valor de ação, e podemos fazer isso seguindo um raciocínio parecido com o aprendizado de v_π visto na seção anterior. Sendo um episódio composto por uma transição de estados, ações e recompensas,

$$S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}, R_{t+2}, S_{t+2}, A_{t+2}, \ldots,$$

a mesma regra de atualização aplicada para valores de estado, na predição utilizando TD, pode ser aplicada a valores de ação,

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)], \tag{2.34}$$

a cada transição de estado. Para estados terminais, S_T, $Q(S_T, A_T)$ é zero. O algoritmo de controle em questão faz uso da sequência de eventos de um episódio, $S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}$, para calcular a transição de um estado para outro, em 2.34, e esta a razão do nome do método de controle, *Sarsa*.

Um algoritmo de controle baseado neste método pode ser aplicado seguindo uma lógica semelhante ao do algoritmo 9, desta vez observando as funções de valor de ação, e aplicando uma política ϵ-soft exploratória. A convergência para uma política ótima pode ser dada, em geral, com ϵ variando com o tempo, como em $\epsilon = \frac{1}{t}$. Gradualmente a probabilidade de selecionar ações não ótimas diminui, e por fim a política utilizada converge para uma política gulosa.

2.4.3.3 Q-Learning: Controle TD *off-policy*

Uma das principais modificações da metologia Sarsa é o algoritmo conhecido como *Q-learning*, sendo também um dos principais algoritmos em tarefas de controle via aprendizado por reforço. A característica principal deste algoritmo, em comparação com o anterior, é a regra de atualização das funções de valor de ação, agora dada por

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]. \tag{2.35}$$

Algoritmo 10 Algoritmo SARSA

Inicializa um $\alpha \in (0, 1]$ e $\epsilon > 0$, pequeno.
Define uma matriz $Q(s, a)\ \forall s \in \mathcal{S}, a \in \mathcal{A}$ arbitrariamente, e $Q(terminal, \cdot) = 0$.

for *episódio* em uma lista de episódios **do**
  Inicializa um estado S como inicial
  Escolhe uma ação A, para S, usando uma política derivada de Q, como ϵ-greedy, por exemplo
  while S não é terminal **do**
    Realiza a ação A e observa os retornos S' e R
    Escolhe uma próxima ação A', para S', usando uma política derivada de Q, como ϵ-greedy, por exemplo
    $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma Q(S', A') - Q(S, A)]$
    $S \leftarrow S'$
    $A \leftarrow A'$
  end while
end for

Observe que agora procuramos aproximar a função de valor para o estado S diretamente da função ótima, q_*, pois estamos considerando que, no estado atual, iremos selecionar a melhor ação possível. Não mais consideraremos uma política ϵ-soft, mas sim uma política greedy. A lógica de controle pode ser visto no algoritmo 11.

Algoritmo 11 Algoritmo *Q-learning*

Inicializa um $\alpha \in (0, 1]$ e $\epsilon > 0$, pequeno.
Define uma matriz $Q(s, a)\ \forall s \in \mathcal{S}, a \in \mathcal{A}$ arbitrariamente, e $Q(terminal, \cdot) = 0$.

for *episódio* em uma lista de episódios **do**
  Inicializa um estado S como inicial
  while S não é terminal **do**
    Escolhe uma ação A, para S', usando uma política derivada de Q, como ϵ-greedy, por exemplo
    Realiza a ação A e observa os retornos S' e R
    $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$
    $S \leftarrow S'$
  end while
end for

Neste algoritmo, a política alvo pode ser obtida tomando-se

$$\pi(s) = \underset{a}{argmax}\ Q(s,a), \quad \forall\ s \in \mathcal{S}. \tag{2.36}$$

Observe que enquanto a função de valor do estado S é atualizada considerando que a melhor ação será selecionada, a ação tomada realmente foi A, que foi selecionada utilizando-se uma política diferente da política ótima. Portanto, *não necessariamente* A é a melhor ação. Como estamos otimizando uma política assumindo ações ótimas, enquanto escolhemos as ações tomadas a partir de outra política mais exploratória, temos uma política comportamental diferente da alvo. Sendo assim, este é um algoritmo de controle *off-policy*.

Esta metodologia de controle aproxima melhor de uma política ótima. Em contrapartida, como não estamos atuando sobre a política que estamos seguindo, melhorando-a, um agente utilizando Q-learning continuará tomando ações não ótimas, e trajetórias ruins, se comparado a um agente seguindo Sarsa, onde a cada intervalo de tempo a política sendo seguida está sendo aprimorada, e o agente toma menos ações ruins a cada passo. Isto torna o algoritmo Q-learning mais ruidoso. O livro de Sutton e Barto (SUTTON; BARTO, 2018) traz um exemplo dos dois algoritmos sobre um problema conhecido como "Caminhada no Penhasco", que mostra bem o que isto significa na prática.

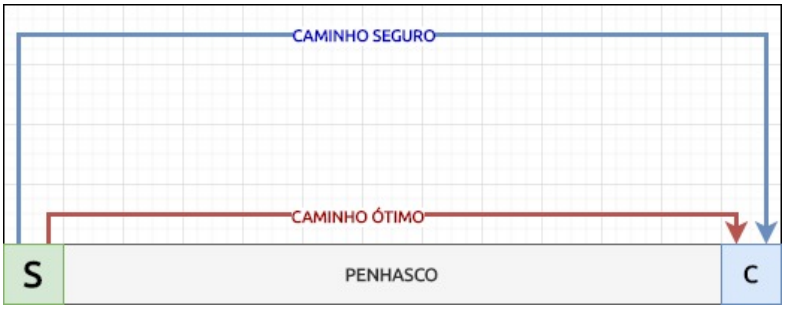


Figura 6 – Esquema do problema Caminhada no Penhasco. Adaptado de (SUTTON; BARTO, 2018).

A Figura 6 mostra o esquemático do problema. Suponha um agente que queira sair do ponto **S** para o ponto **C**. A cada passo dado, o agente recebe uma recompensa de -1, exceto o passo que termina no ponto **C**. Qualquer passo que termine em algum ponto da área do penhasco retorna uma recompensa de -100, e

retorna o agente para o ponto inicial. O problema é episódico, com estado terminal em **C**, e sem desconto γ. Cada estado é um ponto no espaço, e em cada o agente pode escolher quatro ações, que são as direções que o mesmo pode ser mover no plano.

Leve em consideração que em ambos os algoritmos uma política exploratória ϵ-soft é seguida como comportamental, com ϵ fixo, e que o objetivo é descobrir um caminho que devolva o menor prejuízo para o agente. Um agente utilizando Sarsa tende a selecionar, a cada episódio, um caminho que diminua a probabilidade de obter um retorno muito baixo, isto é, que diminua a probabilidade de cair no penhasco, considerando que o mesmo pode selecionar ações aleatórias a cada estado, com uma probabilidade ϵ. No final, irá aprender que, caso utilize uma política ϵ-greedy para escolher as ações, o melhor caminho a ser tomado é o *caminho seguro*.

Em contrapartida, caso utilize Q-learning, o agente irá passar por vários episódios com quedas no penhasco, mas irá tentar aprender um caminho considerando que sempre a melhor ação será tomada, de maneira determinística. Com isso, ao final aprenderá que o melhor caminho a ser seguido, caso sempre escolha a melhor ação, sem escolhas aleatórias, é o *caminho ótimo*, que apresenta a menor distância entre os estados inicial e final. Caso utilizamos um ϵ variável com o tempo para o algoritmo Sarsa, veremos sua política alvo convergir para a política ótima também.

2.4.4 Planejando e Aprendendo com Métodos Tabulares

Relembrando o conceito de modelo, chamamos de modelo do ambiente qualquer coisa que o agente possa usar para predizer como o ambiente vai responder às ações realizadas. Alguns modelos são compostos pela descrição de todas as possibilidades de estados e ações, e suas probabilidades, sendo chamados de *modelos distributivos*. Outros apresentam apenas uma das possibilidades, que foi amostrada conforme as probabilidades de amostragem, sendo chamados de *modelos amostrais*. Por exemplo, considerando um modelo de somatório dos valores obtidos com lançamentos de um dado. Um modelo distributivo apresentaria todas as somas possíveis, e as probabilidades de ocorrência. Já um modelo amostral produziria uma obser-

vação obtida de acordo com sua distribuição de probabilidade. Podemos aplicar essa classificação para os diferentes tipos de modelos vistos neste capítulo. O modelo assumido em programação dinâmica, caracterizado pelas probabilidades de transição dos estados, $p(s', r|s, a)$, é um exemplo de modelo distributivo. Já modelos observados em métodos de Monte Carlo, que caracterizam-se de ocorrências possíveis de estados e ações são exemplos de modelos amostrais.

Modelos distributivos são mais robustos do que modelos amostrais, na medida que podem ser utilizados para gerar amostras. Porém, em diversos casos é muito mais simples obter um modelo amostral do que um distributivo. Seja por complexidade da modelagem da transição de estados, seja pela quantidade de estados possíveis. No caso do dado, por exemplo, é muito mais simples escrever um programa que simulasse diversos lançamentos, do que modelar uma função que descreve todas as somas possíveis, e suas probabilidades.

Modelos podem ser usados para simular experiência real. A partir de um estado e uma ação, um modelo amostral fornece uma uma transição possível, enquanto um modelo distributivo produz todas as possíveis transições, e suas probabilidades de ocorrência. Dado um estado inicial e uma política, um modelo amostral fornece um episódio completo, enquanto um modelo distributivo fornece todos os episódios possíveis, e suas probabilidades de ocorrência. Em todos esses casos, utilizamos o modelo para *simular* o ambiente, e produzir uma *experiência simulada*.

O conceito de *planejamento* baseia-se no princípio da experiência simulada, e é um conceito importante em tarefas de aprendizado por reforço. Parafraseando aqui os autores, "*chamamos de planejamento qualquer processo computacional que toma um modelo como entrada e produz ou melhora uma política ao interagir com o ambiente modelado* (SUTTON; BARTO, 2018)". O planejamento ocorrendo pelo espaço de estados possui uma estrutura básica, com duas ideias principais:

1. São calculadas as funções de valor como passo intermediário para o aprimoramento da política;
2. As funções de valor são calculadas por atualização ou operações de backup aplicadas em experiência simulada.

Em suma, métodos que se utilizam de planejamento usam um modelo do ambiente para simular interações do agente com o mesmo, e assim poder aprimorar a política, diferente de métodos de aprendizagem, que utilizamo experiência real neste mesmo processo. Sozinho, este tipo de estratégia possibilita a um agente obter uma política para uma determinada tarefa, quando interações reais não são possíveis, ou são mais difíceis de serem obtidas. O algoritmo 12 mostra como um método de planejamento pode ser utilizado com um algoritmo Q-learning de maneira simples.

Algoritmo 12 Algoritmo *Q-planning*

while *algum critério de parada não cumprido* **do**
Seleciona um estado, $S \in \mathcal{S}$ e uma ação $A \in \mathcal{A}$ aleatoriamente
Envia S e A para o modelo, e obtém uma amostra de recompensa R, e um próximo estado S'.
Aplica a regra de atualização Q-learning
$Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',a) - Q(S,A)]$
end while

Em conjunto com interações reais, este tipo de estratégia é importante para acelerar o tempo de convergência de uma política sendo aprendida, inserindo mais interação do agente com o meio do que apenas a interação real, para assim o mesmo poder realizar as atualizações necessárias, tanto nas funções de valor, quanto na política, mais rápido. Veremos algoritmos baseados nesta lógica adiante.

2.4.4.1 Dyna: Integrando Planejamento, Ação e Aprendizado

Este algoritmo, também chamado de *Dyna-Q*, parte do princípio da utilização de planejamento no aprendizado *on-line*, ou seja, enquanto o agente está interagindo com o ambiente em uma experiência real.

Em um agente que realiza planejamento, experiências reais podem ser utilizadas de duas formas: tanto para aprimorar o modelo de ambiente utilizado (*aprendizado de modelo*), quanto para aprimorar diretamente as funções valor e política, utilizando métodos de aprendizado por reforço que já foram abordados aqui (*aprendizado por reforço direto* ou *RL direto*). Um diagrama visual das interações possíveis neste tipo de abordagem está representadona Figura 7. O loop

mais interno (1) corresponde ao fluxo de aprendizado por reforço visto até então. Inserir planejamento expande as interações possíveis, para um fluxo como em (2). Experiências reais podem agora tanto alterar as funções de valor e a política diretamente, quando através do modelo, no fluxo chamado de *aprendizado por reforço indireto*, ou *RL indireto*.

O algoritmo Dyna-Q inclui todo esse fluxo de aprendizado, onde:

- O planejamento é realizado utilizando Q-planning, como no algoritmo 12;
- O passo de RL direto é realizado via Q-learning, como no algoritmo 11;
- O método de aprendizado do modelo também é baseado em representação por tabela (uma matriz de númeto de estados por número de ações, como nas representações de $Q(S, A)$ vistas nos algoritmos de controle até então), e assume um ambiente *determinístico*, ou seja, se escolhermos uma ação A no estado S, então *sempre* teremos um próximo estado S', com uma recompensa R.

A cada transição de estados o modelo salva na tabela, na posição (S_t, A_t) a predição de qual recompensa R_{t+1} e qual próximo estado S_{t+1} vai se seguir. Assim, a interação entre entrada e saída de um modelo do ambiente se dá inserindo qual o estado atual, e qual ação neste estado foi tomada, e acessando na tabela as entradas (S, A) para obter o par de (R, S') correspondente. Na etapa de planejamento, o algoritmo amostra de maneira aleatória *apenas de pares estado-ação que já tenham sido visitados*, e portanto são conhecidos do modelo. Por isso maiores interações com o ambiente são importantes não só para aprimorar o conhecimento sobre esses pares, mas também para expandir o conhecimento que seu modelo tem do ambiente que representa.

Tipicamente, o mesmo mecanismo de RL utilizado para aprender com experiências reais também é utilizado para aprender com a experiência simulada, isto é, o mesmo tipo de algoritmo de controle é aplicado aos dois fluxos. Assim, planejamento e aprendizado passam a estar intimamente integrados, de forma que quase toda a mecânica aplicada pelo agente é a mesma, diferenciadas apenas pela

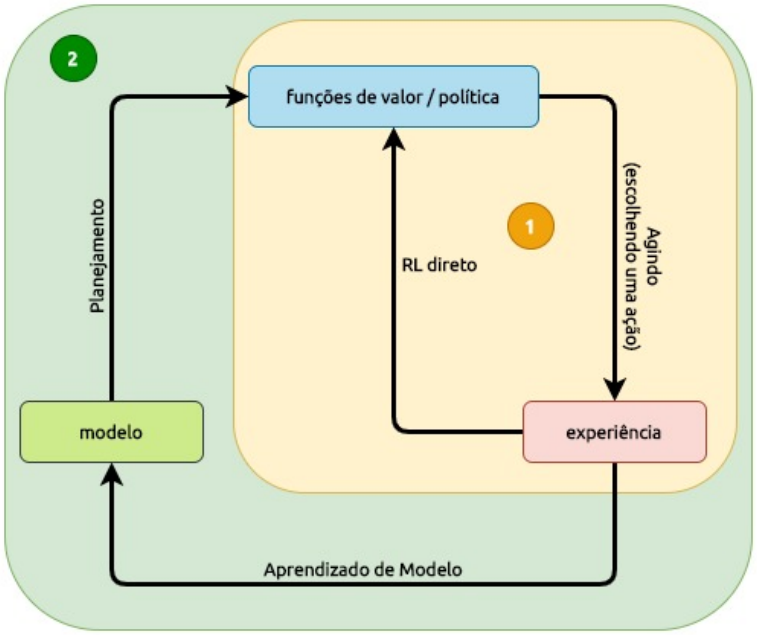


Figura 7 – Diagrama do fluxo relacional Dyna. Adaptado de (SUTTON; BARTO, 2018).

origem do dado sendo aprendido. O algoritmo 13 mostra a implementação do Método Dyna-Q.

Algoritmo 13 Algoritmo *Dyna-Q Tabular, com ambiente determinístico*

Inicializa matrizes $Q(s,a)$ e $Modelo(s,a)$ $\forall s \in \mathcal{S}, a \in \mathcal{A}$ arbitrariamente, e $Q(terminal,\cdot) = 0$.
while *algum critério de parada não cumprido* **do**
  Seleciona um estado, $S \in \mathcal{S}$ e uma ação $A \in \mathcal{A}$ aleatoriamente
  Realiza a ação A e observa R e S'
  $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',a) - Q(S,A)]$
  $Modelo(S,A) \leftarrow (R,S')$
  for *n repetições* **do**
    Escolhe um par estado-ação (S,A) já visitado
    $S, S' \leftarrow Modelo(S,A)$
    $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',a) - Q(S,A)]$
  end for
end while

Como assumimos um ambiente que seja determinístico, caso o mesmo apre-

sente algum comportamento estocástico, ou se altere com o tempo, o algoritmo Dyna-Q não se aplica. Somado a isso, em casos onde as interações do agente com o ambiente real são baixas, o modelo pode não ter sido ajustado o suficiente para que a sua representação leve a bons ajustes das funções de valor e/ou política. Nestes casos, um aprimoramento do algoritmo Dyna-Q é utilizado, chamado *Dyna-Q+*.

Um agente utilizando este algoritmo ajustado mantém um acompanhamento do tempo desde a última visita aos pares de estado-ação, na interação real com o ambiente. Quanto maior este tempo, maior a chance da dinâmica relativa a este par ter se alterado, e portanto, estar incorreta. Assim, um "bônus"é acrescido na recompensa a este par, nas experiências simuladas. Supondo que o par não tenha sido visitado em τ intervalos de tempo, a recompensa obtida durante a simulação, r, passa a ser $r + \kappa\sqrt{\tau}$, com um κ pequeno. Assim, o agente é encorajado a visitar este par quando selecionar uma ação com a interação real, atualizando o modelo com a dinâmica mais atual do ambiente. Existe um custo envolvido nesta tática, mas dependendo da situação, ainda sim o ganho de incentivar a contínua exploração do espaço de estados pode ser compensatória.

Este algoritmo também pode ser utilizado para incentivar a visita do agente a pares de estado-ação ainda não explorados. Nestes casos, estes pares possuem uma dinâmica inicial no modelo onde selecioná-los leva o agente ao mesmo par, com uma recompensa mais elevada.

2.5 Aprendizado por Aproximação de Funções

Até o momento, os métodos citados levavam em consideração que o espaço de estados, ou de pares estado-ação, era representado por vetores ou tabelas, ou seja, *tabulares*. Esta representação funciona bem para problemas onde o espaço de estados é computacionalmente pequeno[6], porém à medida que um número grande de estados são existentes, ou então que o conjunto de ações possível aumenta, este tipo de representação torna-se ineficiente. Representação ainda apresenta outros problemas, oriundos da limitação da complexidade de representação conseguida através dela. Problemas onde o espaço de estados, ou as ações disponíveis, são contínuos, não são possíveis de ser abordados dessa forma. Outra limitação apresentada é que representações tabulares desconsideram a influência de um estado sobre o outro. Nos problemas apresentados até o momento, atualizar os valores de um estado altera diretamente apenas o valor *para aquele estado* (o mesmo raciocínio aplica-se para pares estado-ação), e os valores para outros estados continuam inalterados, a não ser que os mesmos sejam visitados, e ativamente alterados. Assim, considera-se que os estados sejam independentes, o que não é sempre o caso. Existem situações onde alterar o valor para um estado implica em alteração para o valor de outros, de alguma forma.

Assim, faz-se necessário o aprimoramento das técnicas de representação do espaço de estados, de forma que seja possível obter uma representação que seja *generalizável*, visto que muitas vezes a maior parte dos estados vistos serão desconhecidos à princípio, e que consiga lidar com problemas complexos. Esse tipo de generalização é conhecido, e pode ser obtido através de *aproximação de funções*, ou seja, tentar obter uma relação matemática que represente o espaço de estados de maneira satisfatória, através de amostras deste espaço. É, portanto, uma instância do *aprendizado supervisionado*, e diversos métodos ou algoritmos utilizados neste tipo de problema podem ser aplicados neste ponto do aprendizado por reforço.

[6] Nesta colocação, significa que os recursos computacionais disponíveis são capazes de processar o espaço de estados existente.

2.5.1 Predição On-policy com Aproximação de Funções

A utilização de funções para a representação de estados altera o raciocínio utilizado até o momento na caracterização dos problemas. Ao pensarmos em estados, não mais os caracterizaremos como uma lista de elementos possíveis, ou uma tabela de estados por ações, mas sim como uma *função parametrizada* por um vetor de pesos $\mathbf{w} \in R^d$. A partir de agora, ao nos referirmos às funções de valor, utilizaremos a notação $\hat{v}(s, \mathbf{w})$, que é o mesmo que nos referir à função de valor do estado s, $\hat{v}(s)$, dado o vetor de pesos $\mathbf{w}$. Aqui, podemos imaginar, por exemplo, $\hat{v}$ como sendo uma função computada por uma rede neural, com $\mathbf{w}$ sendo os valores de peso das conexões entre as camadas. Ou ainda, $\hat{v}$ pode ser uma árvore de decisão, e $\mathbf{w}$ um vetor contendo informações de divisão dos nós, e de valores para as folhas. Em todos esses casos, alterar os parâmetros para ajustar a função de valor para um estado acaba alterando os valores para todos os outros estados, *generalizando* sua representação para um espaço de estados bem maior do que seria possível até então. Porém, é valido citar que nem todos os métodos de aproximação de funções podem ser utilizados para problemas de aprendizado por reforço. Métodos que assumem conjuntos de dados estáticos, e que baseiam-se em trinos contínuos sobre os mesmos não apresentam bom rendimento com os problemas propostos, uma vez que é importante que o aprendizado possa ocorrer *online*, e com isso as representações comportem-se bem utilizando conjuntos de dados incrementais, além de conseguirem lidar com ambiências dinâmicos. Por exemplo, métodos utilizando GPI, que procuram aprender q_π enquanto π se altera.

2.5.1.1 O Objetivo de Predição ($\overline{VE}$)

Uma aproximação mais exata de um espaço de estados leva em conta a influência de um estado sobre os demais. Assumindo que temos mais estados do que pesos utilizados em sua representação, ao tornarmos o valor de um estado mais preciso, estamos invariavelmente tornando os valores de outros estados menos precisos. Dessa forma, torna-se preciso definir, de alguma forma, os estados cujas representações damos mais importância em estarem corretos. Assim, precisamos definir uma distribuição de estados $\mu(s) \geq 0$, $\sum_s \mu(s) = 1$, indicando o quanto nos importamos com o valor do erro em cada um dos estados s. Erro esta computado

como sendo o quadrado da diferença entre o valor aproximado do estado, $\hat{v}(s, \mathbf{w})$, e o seu valor real $v_{pi}(s)$. Ponderando isso sobre o o espaço de estados, por μ, temos um valor para a função objetivo, o *Erro Médio Quadrático* (MSE), aqui denotado como $\overline{VE}$,

$$\overline{VE}(\mathbf{w}) = \sum_{s' \in \mathcal{S}} \mu(s) \left[v_\pi(s) - \hat{v}(s, \mathbf{w}) \right]^2 . \tag{2.37}$$

Uma possibilidade de definir $\mu(s)$ é utilizando a fração do tempo gasto pelo agente em cada estado.

2.5.1.2 Métodos de Gradiente Estocástico e Semi-gradiente

Partimos para a definição de uma nova classe de métodos de aprendizado, que utiliza gradiente descendente estocástico (SGD) em suas computações. Nestes métodos, o vetor de pesos possui um número fixo de componentes reais, $\mathbf{w} = (w_1, w_2, \ldots, w_d)^T$, e a função de valor aproximada, $\hat{v}(s, \mathbf{w})$ é uma função diferenciável de $\mathbf{w}$, para todo $s \in \mathcal{S}$. Em cada passo do algoritmo iremos atualizar o valor deste vetor. A princípio, assumiremos que, a cada intervalo de tempo, t, iremos obter uma observação do espaço de estados, $S_t \mapsto v_\pi(S_t)$, sendo $v_\pi(S_t)$ o valor real deste estado sob a política π. Vamos assumir também, a título de simplificação, que os estados aparecem, nas interações do agente com o ambiente, com uma mesma distribuição μ. A estratégia utilizada é tentar minimizar $\overline{VE}$, ajustando o vetor de pesos após cada observação por um valor pequeno, na direção que mais reduziria o erro na mesma:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{1}{2}\alpha \nabla [v_\pi(S_t) - \hat{v}(S_t, \mathbf{w}_t)]^2 \tag{2.38}$$

$$= \mathbf{w}_t + \alpha [v_\pi(S_t) - \hat{v}(S_t, \mathbf{w}_t)] \nabla \hat{v}(S_t, \mathbf{w}_t) \tag{2.39}$$

onde α é um parâmetro de passo, e $\nabla f(\mathbf{w})$ é a função gradiente de uma função $f(\mathbf{w})$ qualquer. Pela equação 2.38, vemos o cálculo incremental do vetor de pesos, a partir do gradiente do erro. Como desejamos seguir na direção que *diminui* o erro, e o gradiente nos fornece a direção do *aumento* da função, o sinal da componente somada ao valor anterior do vetor de pesos é negativo.

Podemos repensar a equação 2.38 de outra forma, observando diretamente o valor da estimativa obtida, ao invés de calcularmos o gradiente do erro. Considere o valor da estimativa para a função de valor, $\hat{v}(S_t, \mathbf{w})$. Tomar o gradiente dessa função nos fornece a direção do seu aumento, o que nos dá o termo $\nabla\hat{v}(S_t, \mathbf{w})$ da equação 2.39. Mas como saber qual o sentido deste gradiente seguir? Podemos obter essa resposta observando o erro $v_\pi(S_t) - \hat{v}(S_t, \mathbf{w})$. Caso o valor da estimativa seja superior ao valor real da função, o erro será negativo, levando a um sinal negativo no segundo termo da equação, e consequentemente o sentido contrário do gradiente, diminuindo o valor da estimativa. O mesmo raciocínio se aplica ao caso contrário.

Existe ainda o caso onde o valor alvo da função, U_t, para uma observação S_t, não é o valor exato $v_\pi(S_t)$, mas uma aproximação do valor. Por exemplo de amostras de valor que são coletadas, do ambiente, com algum ruído, alguma variância intrínseca. Um caso conhecido seria de valores oriundos de *bootstrapping*. Nesses casos, a atualização dos pesos não é exata, mas sim uma aproximação,

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha[U_t - \hat{v}(S_t, \mathbf{w}_t)]\nabla\hat{v}(S_t, \mathbf{w}_t). \tag{2.40}$$

Se U_t for um estimador não enviesado do valor real, ou seja, $\mathbb{E}[U_t|S_t = s] = v_\pi(S_t)$, então $\mathbf{w}_t$ é garantido para convergir para um ótimo local, para um α decrescente. Desta forma, podemos por exemplo utilizar métodos de amostragem, como Monte Carlo, para obter uma função alvo na atualização dos pesos. O algoritmo 14 exemplifica uma predição com aproximação de funções neste caso. Para Monte Carlo, como o retorno do episódio é obtido através da média dos retornos observados, é portanto um estimador não enviesado do valor real, e pode ser utilizado.

No caso de utilizarmos *booststrap* para obter U_t, ao invés de Monte Carlo, como nos algoritmos de diferenças temporais, o valor alvo para o estado depende da estimativa do valor *atual* do mesmo, e não de uma média de todos os valores observados. Assim, a função alvo dependerá do valor atual de $\mathbf{w}_t$, irá apresentar um viés, e um método de gradiente descendente não convergirá em um ótimo local. Portanto, utilizar métodos de *bootstrapping* leva em consideração o efeito da alteração do vetor de pesos na estimativa da função de valor, mas ignora o

Algoritmo 14 Algoritmo Gradiente de Predição usando Monte Carlo

Inicializa política π a ser avaliada
Inicializa uma função de valor diferenciável $\hat{v} : \mathcal{S} \times \mathbb{R}^d \to \mathbb{R}$
Insere parâmetro de passo $\alpha > 0$
Inicializa vetor de pesos $\mathbf{w} \in \mathbb{R}^d$ arbitrariamente
for *cada episódio* **do**
  Obtém um episódio $S_0, A_0, R_1, S_1, A_1, \ldots, R_T, S_T$ seguindo π
  for *cada passo* $t = 0, 1, \ldots, T-1$ **do**
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha[G_t - \hat{v}(S_t, \mathbf{w})]\nabla\hat{v}(S_t, \mathbf{w})$
  end for
end for

efeito dos mesmos sobre o alvo, ou seja, consideram apenas parte do cálculo de um gradiente, e por isso métodos que o utilizem são chamados de *semi-gradientes*.

Apesar de não apresentarem uma convergência robusta, como em métodos de gradiente, métodos *semi-gradiente*, porém, apresentam uma série de vantagens que os fazem ser mais utilizados do que sua contrapartes. Em geral apresentam maior velocidade de convergência, além de poderem ser utilizados em tarefas contínuas, e aprendizado online, uma vez que não dependem do final de um episódio. Um exemplo de método *semi-gradiente* pode ser visto no algoritmo 15.

Algoritmo 15 Algoritmo Semi-Gradiente de Predição usando Diferenças Temporais

Inicializa política π a ser avaliada
Inicializa uma função de valor diferenciável $\hat{v} : \mathcal{S} \times \mathbb{R}^d \to \mathbb{R}$
Insere parâmetro de passo $\alpha > 0$
Inicializa vetor de pesos $\mathbf{w} \in \mathbb{R}^d$ arbitrariamente
for *cada episódio* **do**
  Obtém um estado inicial S
  for *cada passo* $t = 0, 1, \ldots, T-1$ **do**
    Escolhe uma ação $A \sim \pi(S)$
    Executa a ação A e observa R e S'
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha[R + \gamma\hat{v}(S', \mathbf{w}) - \hat{v}(S, \mathbf{w})]\nabla\hat{v}(S, \mathbf{w})$
    $S \leftarrow S'$
  end for
end for

Aliado a predições utilizando gradiente e semi-gradiente, uma metodologia

aplicada é a chamada *agregação de estado*, onde os estados são agrupados em uma representação com um mesmo valor estimado para todos, e uma componente do vetor de pesos associado para cada grupo. Esta forma de representação aumenta a generalização do modelo de representação. Uma visualização simples seria imaginar um espaço de estados contínuos, com estados podendo assumir qualquer valor entre 0 e 1000. Uma agregação possível seria definir intervalos de tamanho 100, que delimitariam 10 grupos. Assim, generalizamos a representação, uma vez que estados muito próximos possuirão um mesmo valor dentro de um grupo, e simplificamos o modelo de representação. Na Figura 8 podemos ver o efeito da agregação na função de valor se comparado com o real.

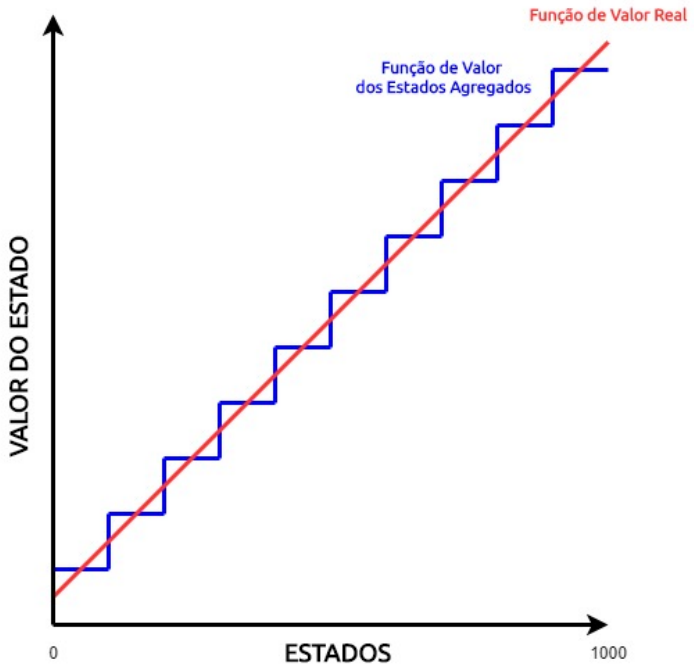


Figura 8 – Agregação de estados e seu efeito na função de valor.

No livro utilizado como referência para este capítulo (SUTTON; BARTO, 2018) podem ser encontrados diversos exemplos de representação de estados, para funções de valor aproximadas por funções lineares, que utilizam diferentes tipos de agregação.

2.5.2 Controle On-policy com Aproximação de Funções

A partir do entendimento de aproximação de funções em problemas de predição, o próximo passo é expandir a lógica para problemas de controle. Como nos métodos vistos até o momento, o foco dos algoritmos torna-se aproximar funções de valor de ação, desta vez considerando a utilização de funções paramétricas, com $\hat{q}(s, a, \mathbf{w}) \approx q_\pi(s, a)$.

2.5.2.1 Controle Semi-Gradiente Episódico

A adaptação dos métodos de predição da seção anterior, para tarefas episódicas, é direta, alterando o objetivo de aproximação, de valores de estado, para valores de ação. As observações consideradas agora são do tipo $S_t, A_t \mapsto U_t$. As funções de atualização do valor de $\mathbf{w}$ tornam-se

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha[U_t - \hat{q}(S_t, A_t, \mathbf{w}_t)]\nabla\hat{q}(S_t, A_t, \mathbf{w}_t). \tag{2.41}$$

Um exemplo de algoritmo de controle pode ser visto no algoritmo 16, que utiliza a lógica Sarsa, para o controle por aproximação de funções. Um passo subsequente de *policy improvement* é realizado, seguindo os moldes de GPI, para obter a política, a partir de $A_t^* = \underset{a}{argmax}\ \hat{q}(S_t, a, \mathbf{w})$.

2.5.2.2 Recompensa Média

Relembrando, em tarefas episódicas o valor de um estado sob uma determinada política π é dado avaliando a média das recompensas obtidas ao longo de toda a trajetória do episódio. Assim, precisamos esperar até o final do mesmo para poder obter a estimativa das funções de valor. Em tarefas contínuas, não é possível esperar até o final de uma trajetória, uma vez que ele pode não existir. Sendo assim, as funções de valor são estimadas tendo por base o retorno descontado até o momento, consistindo da recompensa atual somada às recompensas anteriores descontadas por um fator γ. Em tarefas contínuas que utilizam aproximação de funções, porém, o retorno descontado apresenta um problema em sua aplicação, e por isso um novo conceito se faz necessário para substituí-lo.

Algoritmo 16 Algoritmo Episódico Semi-Gradiente de Controle Sarsa

Inicializa uma função de valor diferenciável $\hat{v} : \mathcal{S} \times \mathbb{R}^d \to \mathbb{R}$
Insere parâmetro de passo $\alpha > 0$
Insere parâmetro $\epsilon > 0$
Inicializa vetor de pesos $\mathbf{w} \in \mathbb{R}^d$ arbitrariamente
for *cada episódio* **do**
  Obtém um par estado-ação inicial, S, A (por exemplo usando ϵ-greedy)
  for *cada passo* $t = 0, 1, \ldots, T-1$ **do**
    Executa a ação A e observa R e S'
    if S' é estado terminal **then**
      $\mathbf{w} \leftarrow \mathbf{w} + \alpha[R - \hat{q}(S, A, \mathbf{w})]\nabla\hat{q}(S, A, \mathbf{w})$
      Vai para próximo episódio
    end if
    Escolhe próxima ação A' em função de $\hat{q}(S', \cdot, \mathbf{w})$ (por exemplo usando ϵ-greedy)
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha[R + \gamma\hat{q}(S', A', \mathbf{w}) - \hat{q}(S, A, \mathbf{w})]\nabla\hat{v}(S, A, \mathbf{w})$
    $S \leftarrow S'$
    $A \leftarrow A'$
  end for
end for

Introduzimos então o conceito de *recompensa média*. Esta nova configuração procura substituir o retorno descontado mas de modo diferente ao outro, não possui desconto nas recompensas antigas.

Neste tipo de configuração, avaliamos a esperança das recompensas obtidas seguindo uma política π, denominada como sendo $r(\pi)$,

$$\begin{aligned} r(\pi) &= \lim_{h\to\infty} \frac{1}{h} \sum_{t=1}^{h} \mathbb{E}[R_t | S_o, A_{o:t-1} \sim \pi] \\ &= \lim_{t\to\infty} \mathbb{E}[R_t | S_o, A_{o:t-1} \sim \pi] \\ &= \sum_s \mu_\pi \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) r, \end{aligned} \tag{2.42}$$

ou seja, significa que estamos somando todos os retornos que podem ser obtidos em um estado a, ponderados pela probabilidade de a nos levar ao estado s' com recompensa r. Fazemos isso para todas as ações de um estado, e então somamos, ponderando pela probabilidade de selecionar cada ação no estado s. Após

isso, somamos novamente este cálculo para todos os estados, ponderando pela probabilidade μ_π de permanência (ou visita) em cada um, se seguirmos a política π. Fazemos isso condicionado a um estado inicial S_0, e cujas ações subsequentes $A_0, A_1, A_2, \ldots A_{t-1}$ são obtidas de acordo com a política π.

Nesta configuração, os retornos são dados em termos das diferenças entre os retornos obtidos a cada passo, e a média calculada até o momento, ou seja,

$$\begin{aligned} G_t &= (R_{t+1} - r(\pi)) + (R_{t+2} - r(\pi)) + (R_{t+3} - r(\pi)) + \ldots \\ &= R_{t_1} - r(\pi) + G_{t+1}, \end{aligned} \tag{2.43}$$

o que é conhecido como *retorno diferencial*, cujas funções de valor correspondentes recebem o nome de *funções de valor diferencial*, que são definidas da mesma forma sendo feita até então, com

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[G_t, S_t = s] \\ q_\pi(s,a) &= \mathbb{E}_\pi[G_t, S_t = s, A_t = a] \end{aligned}$$

Existe ainda uma forma diferencial do erro TD, visto na equação 2.32,

$$\delta_t = R_{t+1} - \bar{R}_t + \hat{v}(S_{t+1}, \mathbf{w}_t) - \hat{v}(S_t, \mathbf{w}_t) \tag{2.44}$$

$$\delta_t = R_{t+1} - \bar{R}_t + \hat{q}(S_{t+1}, A_{t+1}, \mathbf{w}_t) - \hat{q}(S_t, A_t, \mathbf{w}_t), \tag{2.45}$$

para valores de estado e ação, respectivamente, e $\bar{R}_t$ é a estimativa da recompensa média $r(\pi)$ no instante de tempo t. Dessa forma, é possível ajustar os algoritmo estudados anteriormente em tarefas contínuas para casos com aproximação de funções, sem que sejam necessárias grandes alterações. Por exemplo, o algoritmo 16 pode ser ajustado para a utilização do erro TD, fazendo

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha\delta_t \nabla \hat{q}(S_t, A_t, \mathbf{w}_t), \tag{2.46}$$

cujo pseudocódigo pode ser visto no algoritmo 17.

Algoritmo 17 Algoritmo Diferencial Semi-Gradiente de Controle Sarsa

Inicializa uma função de valor diferenciável $\hat{v} : \mathcal{S} \times \mathbb{R}^d \to \mathbb{R}$
Insere parâmetros de passo $\alpha > 0$ e $\beta > 0$
Insere parâmetro $\epsilon > 0$
Inicializa vetor de pesos $\mathbf{w} \in \mathbb{R}^d$ arbitrariamente
Inicializa a estimativa para a recompensa média $R \in \mathbb{R}$ arbitrariamente

Obtém um par estado-ação inicial, S, A (por exemplo usando ϵ-greedy)
for *cada passo* $t = 0, 1, \ldots$ **do**
    Executa a ação A e observa R e S'
    Escolhe próxima ação A' em função de $\hat{q}(S', \cdot, \mathbf{w})$ (por exemplo ϵ-greedy)
    $\delta \leftarrow R - \bar{R} + \hat{q}(S', A', \mathbf{w}) - \hat{q}(S, A, \mathbf{w})$
    $\bar{R} \leftarrow \bar{R} + \beta\delta$
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha\delta\nabla\hat{v}(S, A, \mathbf{w})$
    $S \leftarrow S'$
    $A \leftarrow A'$
end for

2.5.3 Métodos de Gradiente de Política

Esta seção aborda uma metodologia nova para a abordagem de métodos de aprendizado por reforço. Até o momento, todos os métodos considerados lidavam com valores de ação e estado, procurando aprender as funções modeladoras dos mesmos, e a partir disso selecionar ações. As políticas aprendidas tinham por base esses valores, e dependiam dos mesmos para existir. Agora, apresentamos um novo conceito, de *políticas parametrizadas*, onde a seleção de ações pode ser realizada sem a necessidade de consultar qualquer função de valor, apesar de ainda serrem necessárias para *aprender* os parâmetros das políticas. Utilizaremos a notação $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ para representar o vetor de parâmetros da política. Assim, a mesma passa a ser definida como

$$\pi(a|s, \boldsymbol{\theta}) = Pr\{A_t = a|S_t = s, \boldsymbol{\theta}_t = \boldsymbol{\theta}\}, \tag{2.47}$$

ou a probabilidade de selecionarmos a ação a no tempo t, dado que estamos no estado s, e os parâmetros da política sendo utilizada é $\boldsymbol{\theta}$. Estes métodos ainda podem coexistir com aprendizado de funções de valor, portanto as definições vistas

até agora ainda são válidas.

Para seguir com o processo de aprendizado de políticas, definimos uma métrica de performance alvo, a ser otimizada, $J(\boldsymbol{\theta})$, definida em função dos parâmetros da política, e definimos uma regra de atualização dos mesmos baseado em *gradiente ascendente*,

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha\widehat{\nabla J(\boldsymbol{\theta}_t)}, \tag{2.48}$$

sendo $\widehat{\nabla J(\boldsymbol{\theta})} \in \mathbb{R}^{d'}$ uma estimativa estocástica cuja esperança se aproxima do gradiente da métrica de performance. Métodos que fazem uso desta regra de atualização de $\boldsymbol{\theta}$ são chamados de *métodos de gradiente de política*, mesmo que também aprendam funções de valor. Nos casos onde isso acontece, os métodos também são chamados de *métodos ator-crítico*, onde *ator* é uma referência à política aprendida, e *crítico*, à função de valor aprendida.

2.5.3.1 Aproximação de Política

Em métodos que utilizem gradiente de política, a mesma pode ser uma função parametrizada qualquer, desde que seja diferenciável em respeito a seus parâmetros, ou seja, desde que $\nabla\pi(a|s,\boldsymbol{\theta})$ existe e seja finito para todo $s \in \mathcal{S}, a \in \mathcal{A}$, e $\boldsymbol{\theta} \in \mathbb{R}^{d'}$. Na pática, para incentivar a exploração do espaço de ações, um requerimento é que a política nunca se torne determinística.

Para casos onde o espaço de ações é discreto e não seja muito grande, então um tipo comum de parametrização aplicado é formar preferências numéricas $h(s, a\boldsymbol{\theta}) \in \mathbb{R}$ para cada par de estado-ação. As ações com maiores preferências em cada estado recebem as maiores probabilidades de serem selecionadas, por exemplo, através de um mapeamento realizado via função *soft-max*,

$$\pi(a|s,\boldsymbol{\theta}) = \frac{e^{h(s,a,\boldsymbol{\theta})}}{\sum_b e^{h(s,b,\boldsymbol{\theta})}}. \tag{2.49}$$

As preferências de ação h podem ser calculadas de maneira arbitrária. Poderiam ser a saída de uma rede neural, onde $\boldsymbol{\theta}$ seria o vetor de pesos da mesma, ou então de um regressor linear.

Existem algumas vantagens em utilizar soft-max em preferências de ação. Aplicar a função diretamente sobre os valores de ação, sozinho, não garantiria a convergência para uma política determinística ótima, mesmo que a seleção de ações seguisse uma lógica ϵ-greedy. Preferências de ação tendem a se aproximar da política ótima estocástica, e mesmo que a política ótima seja determinística, a utilização do soft-max levaria a probabilidades maiores para ações correspondentes à esta política, mas ainda mantendo uma probabilidade diferente de zero para as outras ações, possibilitando ainda uma estocasticidade da política obtida. Outra vantagem deste método é que quando estudamos métodos que fazem uso de $Q(S, a)$, a política era determinada como $\pi(s, a) = \underset{a}{argmax}\ Q(s, a)$, portanto determinística, e não estocástica. Além disso, uma vantagem fundamental desta metodologia é que a política pode ser uma função simples de ser aproximada, e sua seleção possibilita injetar um conhecimento à priori sobre a forma desejada da política em um sistema de aprendizado por reforço.

2.5.3.2 Teorema do Gradiente de Política

Uma outra vantagem de utilizar parametrização de políticas é que as probabilidades das ações variam de maneira gradual, ao contrário de uma mudança drástica das mesmas como em métodos ϵ-greedy.

A definição da métrica de performance $J(\boldsymbol{\theta})$ é definida de maneiras diferentes para casos episódicos e contínuos. No primeiro, assumindo que um episódio inicie em um estado s_0, a performance pode ser definida como sendo

$$J(\boldsymbol{\theta}) = v_{\pi_{\boldsymbol{\theta}}}(s_0), \tag{2.50}$$

onde $v_{\pi_{\boldsymbol{\theta}}}$ é a função de valor verdadeira para uma política parametrizada $\pi_{\boldsymbol{\theta}}$ com parâmetros $\boldsymbol{\theta}$. Daqui pra frente, consideraremos episódios sem desconto, com $\gamma = 1$, apesar de o mesmo ser incluído nos algoritmos e equações definidas.

A definição da métrica em 2.50 traz a questão de como estimá-la em função dos parâmetros da política, quando o gradiente da mesma depende do efeito desconhecido das alterações da própria política sobre a distribuição de estados. Procurando resolver este problema, apresentamos a definição do *teorema do gradi-*

ente de política, que fornece uma expressão analítica desta estimativa independente da derivada da distribuição dos estados. Portanto, em casos episódicos, o gradiente da métrica de performance é definido sendo proporcional a

$$\nabla J(\boldsymbol{\theta}) \propto \sum_{s} \mu(s) \sum_{a} q_\pi(s,a) \nabla \pi(a|s,\boldsymbol{\theta}), \tag{2.51}$$

onde os gradientes são vetores coluna das derivadas parciais em respeito aos componentes de $\boldsymbol{\theta}$, e π corresponde à política sendo definida pelo vetor de parâmetros. A constante de proporcionalidade da expressão acima é a média de tamanho de um episódio.

2.5.3.3 REINFORCE: Gradiente de Política com Monte Carlo

Retomando da equação 2.50 a necessidade de obter uma estimativa do gradiente $\widehat{\nabla J(\boldsymbol{\theta})}$ que se aproxime de seu valor real. Da equação 2.51, obtermos uma expressão que nos possibilita aproximar desse valor. Considerando que esta expressão indica um somatório dos valores dos estados, ponderados por suas ocorrências no espaço amostral, podemos expandir a equação 2.51 da forma

$$\begin{aligned} \nabla J(\boldsymbol{\theta}) &\propto \sum_{s} \mu(s) \sum_{a} q_\pi(s,a) \nabla \pi(a|s,\boldsymbol{\theta}) \\ &= \mathbb{E}_\pi \left[\sum_{a} q_\pi(S_t,a) \nabla \pi(a|S_t,\boldsymbol{\theta}) \right]. \end{aligned} \tag{2.52}$$

A expressão em 2.52 indica a esperança sobre uma soma sobre todo o espaço de ações, para um determinado estado visitado no tempo t, mas podemos expandir o raciocínio para incluir o conhecimento das ações a serem escolhidas, dada a política π sendo seguida. Assim, dada uma amostra de ação $A_t \sim \pi$ observada, ajustamos a equação 2.52 da forma

$$\begin{aligned}\nabla J(\boldsymbol{\theta}) &= \mathbb{E}_\pi \left[\sum_a q_\pi(S_t, a) \nabla \pi(a|S_t, \boldsymbol{\theta}) \right] \\ &= \mathbb{E}_\pi \left[\sum_a \pi(a|S_t, \boldsymbol{\theta}) q_\pi(S_t, a) \frac{\nabla \pi(a|S_t, \boldsymbol{\theta})}{\pi(a|S_t, \boldsymbol{\theta})} \right] \\ &= \mathbb{E}_\pi \left[q_\pi(S_t, A_t) \frac{\nabla \pi(A_t|S_t, \boldsymbol{\theta})}{\pi(A_t|S_t, \boldsymbol{\theta})} \right] \\ &= \mathbb{E}_\pi \left[G_t \frac{\nabla \pi(A_t|S_t, \boldsymbol{\theta})}{\pi(A_t|S_t, \boldsymbol{\theta})} \right],\end{aligned} \tag{2.53}$$

afinal $\mathbb{E}_\pi[G_t|S_t, A_t] = q_\pi(S_t, A_t)$. A expressão final obtida, portanto, é uma quantidade que pode ser amostrada, e cuja expectativa é igual ao gradiente. Assim, utilizando a conclusão da expressão em 2.53 na equação 2.50, chegamos na regra de atualização que é base para o primeiro algoritmo de gradiente de política apresentado, *REINFORCE*,

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha G_t \frac{\nabla \pi(A_t|S_t, \boldsymbol{\theta}_t)}{\pi(A_t|S_t, \boldsymbol{\theta}_t)}. \tag{2.54}$$

Esta expressão indica que cada incremento nos parâmetros é proporcional ao produto do retorno obtido, G_t, e um vetor, que é o gradiente da probabilidade de selecionar a ação recém selecionada neste estado, dividido pela probabilidade de selecionar esta ação. O vetor indica a direção no espaço dos parâmetros que mais incremente a probabilidade de selecionar novamente a ação A_t em visitas futuras ao estado S_t. Esta atualização aumenta o vetor de parâmetros nesta direção de maneira proporcional ao retorno observado, e inversamente proporcional à probabilidade desta ação ser selecionada. A proporcionalidade a G_t faz sentido ao querermos que o parâmetro se mova mais em direções das ações que forneçam o maior retorno. E ser inversamente proporcional a π se faz necessário na medida em que queremos evitar que ações selecionadas com maior frequência possuam uma vantagem que não seja relacionada ao seu retorno, e acabe beneficiando ações que devolvam os maiores retornos para um estado.

Como o algoritmo utiliza o retorno completo do episódio, a partir de um tempo t, pode ser caracterizado como um Método de Monte Carlo, e aplicável apenas a tarefas episódicas. O pseudocódigo pode ser visto no algoritmo 18.

Algoritmo 18 REINFORCE: Controle Monte Carlo com Gradiente de Política

Inicializa uma política parametrizável diferenciável $\pi(a|s, \boldsymbol{\theta})$
Insere parâmetros de passo $\alpha > 0$
Inicializa vetor de parâmetros $\boldsymbol{\theta} \in \mathbb{R}^{d'}$

for *cada espisódio* **do**
  Gera uma trajetória para o episódio $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$ seguindo π
  for *cada passo* $t = 0, 1, \ldots, T-1$ **do**
    $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \gamma^t G \nabla \ln \pi(A_t|S_t, \boldsymbol{\theta})$
  end for
end for

No algoritmo, a função de atualização dos parâmetros utiliza uma versão compacta da expressão em 2.54, obtida a partir da identidade $\nabla \ln x = \frac{\nabla x}{x}$. Chamamos esse vetor de *vetor de elegibilidade*. Na função de atualização ainda incluímos o termo γ. Como mencionado anteriormente, estamos tratando o caso de episódios sem desconto ($\gamma = 1$), mas nos algoritmos mostramos o caso geral descontado. Fazemos isso pois episódios com desconto apresentam alguma complexidades adicionais que distraem da ideia geral da seção.

2.5.3.4 REINFORCE com baseline

Podemos alterar a equação 2.51 para generalizá-la, adicionando uma comparação do valor da ação com um *baseline* $b(s)$,

$$\nabla J(\boldsymbol{\theta}) \propto \sum_s \mu(s) \sum_a \Big(q_\pi(s, a) - b(s) \Big) \nabla \pi(a|s, \boldsymbol{\theta}). \tag{2.55}$$

Este baseline pode ser qualquer função , contanto que seja independente do valor de a. Adicionar o baseline à equação 2.51 não causa alteração à mesma, uma vez que a subtração adicionada é zero, e as duas equações são equivalentes.

$$
\begin{aligned}
\nabla J(\boldsymbol{\theta}) &\propto \sum_s \mu(s) \sum_a \Big(q_\pi(s,a) - b(s) \Big) \nabla \pi(a|s,\boldsymbol{\theta}) \\
&\propto \sum_s \mu(s) \left[\sum_a q_\pi(s,a) \nabla \pi(a|s,\boldsymbol{\theta}) - \sum_a b(s) \nabla \pi(a|s,\boldsymbol{\theta}) \right] \\
&\propto \sum_s \mu(s) \left[\sum_a q_\pi(s,a) \nabla \pi(a|s,\boldsymbol{\theta}) - b(s) \nabla \sum_a \pi(a|s,\boldsymbol{\theta}) \right] \\
&\propto \sum_s \mu(s) \left[\sum_a q_\pi(s,a) \nabla \pi(a|s,\boldsymbol{\theta}) - b(s) \nabla 1 \right] \\
&\propto \sum_s \mu(s) \left[\sum_a q_\pi(s,a) \nabla \pi(a|s,\boldsymbol{\theta}) - 0 \right].
\end{aligned}
$$

E uma regra de atualização dos parâmetros pode ser obtida a partir da equação 2.55, similarmente à equação 2.54,

$$
\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \Big(G_t - b(S_t) \Big) \frac{\nabla \pi(A_t|S_t,\boldsymbol{\theta}_t)}{\pi(A_t|S_t,\boldsymbol{\theta}_t)}. \tag{2.56}
$$

Adicionar o termo $b(s)$ não apresenta efeito algum no valor esperado da atualização, porém apresenta um grande efeito em sua *variância*, e consequentemente na sua velocidade de convergência. Como ele precisa variar conforme o estado, uma escolha natural para o mesmo é utilizar a estimativa do valor do estado, $\hat{v}(S_t, \mathbf{w})$, sendo $\mathbf{w} \in \mathbb{R}^d$ o vetor e pesos da função de valor, conforme visto anteriormente. Um pseudocódigo completo pode ser visto no algoritmo 19.

Um comparativo do desempenho do algoritmo REINFORCE com e sem baseline pode ser visto na Figura 9, extraída e adaptada de (SUTTON; BARTO, 2018), para o problema do corredor curto, visto na Figura 10. Este problema consiste em um corredor com 4 casas, sendo uma delas uma casa "final". Começando em S, o agente deve seguir para G, podendo apenas tomar a ação de mover-se para esquerda ou para a direita. Para a segunda casa, porém, a direção da ação é invertida, i.e., caso escolha ir para a direita, o agente irá mover-se para a casa à esquerda, e vice-versa. Este é um problema difícil, pois todos os estados parecem semelhantes utilizando apenas aproximação de funções.

Algoritmo 19 REINFORCE utilizando baseline

Inicializa uma política parametrizável diferenciável $\pi(a|s, \boldsymbol{\theta})$
Insere uma função de valor de estado diferenciável $\hat{v}(s, \mathbf{w})$
Insere parâmetros de passo $\alpha_{\boldsymbol{\theta}} > 0$ e $\alpha_{\mathbf{w}} > 0$
Inicializa vetor de parâmetros $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ e o vetor de pesos $\mathbf{w} \in \mathbb{R}^{d}$

for *cada espisódio* **do**
Gera uma trajetória para o episódio $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$ seguindo π
for *cada passo* $t = 0, 1, \ldots, T-1$ **do**
$G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$
$\delta \leftarrow G - \hat{v}(S_t, \mathbf{w})$
$\mathbf{w} \leftarrow \mathbf{w} + \alpha_{\mathbf{w}} \delta \nabla \hat{v}(S_t, \mathbf{w})$
$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha_{\boldsymbol{\theta}} \gamma \delta \nabla \ln \pi(A_t|S_t, \boldsymbol{\theta})$
end for
end for

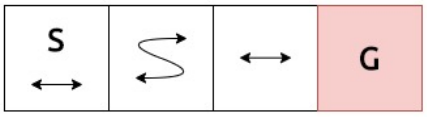


Figura 9 – Esquema do problema do corredor curto. Extraído e adaptado de (SUTTON; BARTO, 2018).

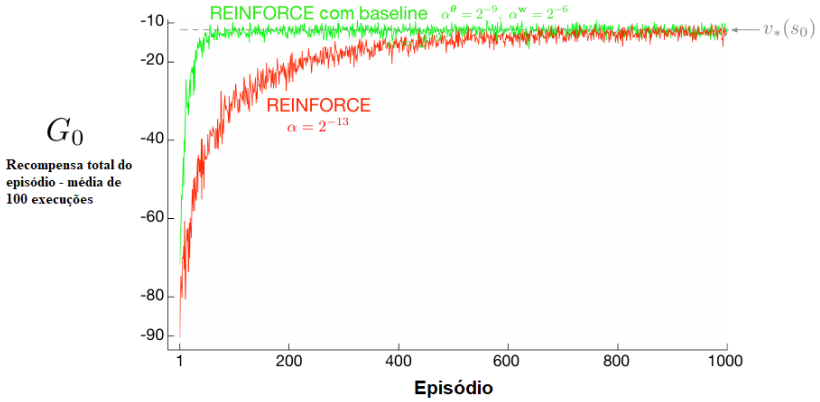


Figura 10 – Resultado para o problema do corredor curto. Extraído e adaptado de (SUTTON; BARTO, 2018).

2.5.3.5 Métodos Ator-Crítico

Métodos ator-crítico são métodos que englobam tanto a parte de aprendizado dos parâmetros da política, quanto fo aprendizado das funções de valor. Diferente do que ocorre do método REINFORCE com baseline, onde a função de valor de estado é utilizada somente como baseline, calculada a partir do seu valor real G, os métodos agora abordados realizam bootstrapping para incrementar o valor da estimativa atual com base na última estimativa conhecida das funções.

Como métodos de Monte Carlo, como o último algoritmo visto, tipicamente possuem uma baixa velocidade de aprendizado, e são inconvenientes para a utilização em aprendizado online contínuo, podemos fazer uso de métodos de gradiente de política que usam métodos ator-crítico com algoritmos bootstrap como críticos.

Considere inicialmente métodos ator-crítico análogos aos algoritmos implementados para diferenças temporais baseados em amostras (Sarsa, Q-learning). Estes métodos possibilitam uma implementação online e incremental. Os algoritmos agora considerados substituem a etapa de retorno dos métodos REINFORCE com uma atualização incremental,

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \left(G_{t:t+1} - \hat{v}(S_t, \mathbf{w}) \right) \frac{\nabla \pi(A_t|S_t, \boldsymbol{\theta}_t)}{\pi(A_t|S_t, \boldsymbol{\theta}_t)} \tag{2.57}$$

$$= \boldsymbol{\theta}_t + \alpha \left(R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w}) - \hat{v}(S_t, \mathbf{w}) \right) \frac{\nabla \pi(A_t|S_t, \boldsymbol{\theta}_t)}{\pi(A_t|S_t, \boldsymbol{\theta}_t)} \tag{2.58}$$

$$= \boldsymbol{\theta}_t + \alpha \delta_t \frac{\nabla \pi(A_t|S_t, \boldsymbol{\theta}_t)}{\pi(A_t|S_t, \boldsymbol{\theta}_t)}. \tag{2.59}$$

O pseudocódigo para esta implementação pode ser visto no algoritmo 20.

2.5.3.6 Gradiente de Política para Tarefas Contínuas

Sem os limites de um episódio, precisamos adaptar a métrica de performance para utilizar o conceito de recompensa média,

Algoritmo 20 Método Ator-Crítico Episódico

Inicializa uma política parametrizável diferenciável $\pi(a|s, \boldsymbol{\theta})$
Insere uma função de valor de estado diferenciável $\hat{v}(s, \mathbf{w})$
Insere parâmetros de passo $\alpha_{\boldsymbol{\theta}} > 0$ e $\alpha_{\mathbf{w}} > 0$
Inicializa vetor de parâmetros $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ e o vetor de pesos $\mathbf{w} \in \mathbb{R}^{d}$

for *cada espisódio* **do**
  Inicializa um estado inicial S
  $I \leftarrow 1$
  while S *não é terminal* **do**
    $A \sim \pi(\cdot|S, \boldsymbol{\theta})$
    Toma ação A e observa S' e R
    $\delta \leftarrow R + \gamma\hat{v}(S', \mathbf{w}) - \hat{v}(S, \mathbf{w})$
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha_{\mathbf{w}}\delta\nabla\hat{v}(S, \mathbf{w})$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha_{\boldsymbol{\theta}} I\gamma\delta\nabla \ln \pi(A_t|S_t, \boldsymbol{\theta})$
    $I \leftarrow \gamma I$
    $S \leftarrow S'$
  end while
end for

$$
\begin{aligned}
J(\boldsymbol{\theta}) = r(\pi) &= \lim_{h\to\infty} \frac{1}{h} \sum_{t=1}^{h} \mathbb{E}[R_t|S_0, A_{0:t-1} \sim \pi] \\
&= \lim_{t\to\infty} \mathbb{E}[R_t|S_0, A_{0:t-1} \sim \pi] \\
&= \sum_s \mu(s) \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)r,
\end{aligned}
\tag{2.60}
$$

sendo $\mu(s) = \lim_{t\to\infty} Pr\{S_t = s|A_{0:t-1} \sim \pi\}$ a distribuição em regime permanente dos estados segundo π. O pseudocódigo para este tipo de implementação pode ser visto no algoritmo 21.

2.5.3.7 Parametrização de Política para Ações Contínuas

Métodos baseados em política ainda oferecem alternativas práticas para lidar com largos espaços de ações, até mesmo espaços contínuos. Ao invés de calcular as probabilidades específicas para cada ação, agora aprendemos as estatísticas da *distribuição de probabilidade* das mesmas. Por exemplo, o espaço de ações pode

Algoritmo 21 Método Ator-Crítico Contínuo

Inicializa uma política parametrizável diferenciável $\pi(a|s, \boldsymbol{\theta})$
Insere uma função de valor de estado diferenciável $\hat{v}(s, \mathbf{w})$
Insere parâmetros de passo $\alpha_{\boldsymbol{\theta}} > 0$, $\alpha_{\mathbf{w}} > 0$ e $\alpha_{\bar{R}} > 0$
Inicializa vetor de parâmetros $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ e o vetor de pesos $\mathbf{w} \in \mathbb{R}^d$
Inicializa $\bar{R} \in \mathbb{R}$
Inicializa $S \in \mathcal{S}$

while *critério de parada não atingido* **do**
  $A \sim \pi(\cdot|S, \boldsymbol{\theta})$
  Toma ação A e observa S' e R
  $\delta \leftarrow R + \gamma\hat{v}(S', \mathbf{w}) - \hat{v}(S, \mathbf{w})$
  $\bar{R} \leftarrow \bar{R} + \alpha_{\bar{R}}\delta$
  $\mathbf{w} \leftarrow \mathbf{w} + \alpha_{\mathbf{w}}\delta\nabla\hat{v}(S, \mathbf{w})$
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha_{\boldsymbol{\theta}} I\gamma\delta\nabla \ln \pi(A_t|S_t, \boldsymbol{\theta})$
  $S \leftarrow S'$
end while

ser descrito por uma variável aleatória amostrada de uma distribuição gaussiana. Sendo assim, considerando a PDF de uma distribuição normal como sendo

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right), \tag{2.61}$$

uma política para o espaço de ações pode ser definida como sendo a PDF gaussiana de um valor de ação escalar, com média e desvio padrão dadas por uma função aproximada parametrizada, dependente do estado,

$$\pi(a|s, \boldsymbol{\theta}) = \frac{1}{\sigma(s, \boldsymbol{\theta})\sqrt{2\pi}} \exp\left(-\frac{(a-\mu(s, \boldsymbol{\theta}))^2}{2\sigma(s, \boldsymbol{\theta})^2}\right). \tag{2.62}$$

As funções aproximadas para a média ($\mu(s, \boldsymbol{\theta})$) e para o desvio padrão ($\sigma(s, \boldsymbol{\theta})$) podem ser calculadas ainda da seguinte forma: considere o vetor de parâmetros como sendo composto por

$$\boldsymbol{\theta} = [\boldsymbol{\theta}_\mu, \boldsymbol{\theta}_\sigma]^T, \tag{2.63}$$

ou seja, uma componente correspondente à parametrização da função de média, e outra para a função de desvio padrão. Assim, sendo $\mathbf{x}_\mu(s)$ e $\mathbf{x}_\sigma(s)$ as features de estado correspondentes à média e ao desvio padrão, as respectivas funções podem ser calculadas como

$$\mu(s, \boldsymbol{\theta}) = \boldsymbol{\theta}_\mu^T \mathbf{x}_\mu(s) \tag{2.64}$$

$$\sigma(s, \boldsymbol{\theta}) = \boldsymbol{\theta}_\sigma^T \mathbf{x}_\sigma(s). \tag{2.65}$$

Ou ainda, estas funções de valor podem ser calculadas a partir de redes neurais, onde as features de estado correspondem à entrada da rede, e exista uma rede responsável por calcular o valor da média e do desvio padrão correspondentes (ou ainda ser uma rede com duas saídas, dependendo da arquitetura utilizada).

3 Otimização de Portfólio

3.1 Introdução

Neste capítulo faremos uma introdução ao problema de otimização de portfólio, tomando por referência o livro *Optimization Methods in Finance*, de Cornuejols e Tütüncü (CORNUEJOLS; TUTUNCU, 2007).

O problema de otimizar alocação de investimentos em um portfólio de ações envolve encontrar a melhor configuração da carteira de ativos financeiros com base na relação existente entre risco e retorno. O modelo de otimização de média-variância de Markowitz (MVO[1]), amplamente utilizado, apresenta um mecanismo para a seleção destes ativos. Considere ativos $s_1, s_2, ..., s_n$ com retornos aleatórios, e μ_i e σ_i o retorno esperado e o desvio padrão do mesmo para um determinado ativo s_i. Seja ρ_{ij} a correlação existente entre os retornos dos ativos s_i e s_j e $\mu = [\mu_1, \mu_2, ..., \mu_n]^T$ e $\Sigma = (\sigma_{ij})$ a matriz de covariância simétrica com $\sigma_{ii} = \sigma_i^2$ e $\sigma_{ij} = \rho_{ij}\sigma_i\sigma_j$ para $i \neq j$. Assumindo x_i como o valor proporcional alocado para o ativo s_i, o retorno e a variância esperados do portfólio $x = (x_1, x_2, ..., x_n)$ são,

$$G = \mathbb{E}[x] = x_1\mu_1 + x_2\mu_2, ..., x_n\mu_n = \mu^T x, \tag{3.1}$$

$$Var[x] = \sum_{i,j} = \rho_{i,j}\sigma_i\sigma_j x_i x_j = x^T \Sigma x. \tag{3.2}$$

Definimos um conjunto de portfólios admissíveis

$$\mathcal{X} := \{x : Ax = b, Cx \geq d\}, \tag{3.3}$$

onde A é uma matriz $m \times n$, b é um vetor unidimensional com m posições, C é uma matriz $p \times n$ e d é um vetor com p dimensões. m e p são as quantidades de restrições de igual e desigualdade, respectivamente. Uma restrição considerada para qualquer representação é

[1] Do inglês *mean-variance optimization.*

$$\sum_i x_i = 1. \tag{3.4}$$

Quaisquer restrições lineares existentes para a alocação de recursos na carteira podem ser considerados como contidos na representação genérica definida em 3.3.

Um portfólio x é chamado *eficiente* se possui o retorno máximo esperado dentre todos os portfólios de mesma variância, ou se possui a menor variância dentre os portfólios com ao menos um certo retorno esperado. A coleção de portfólios eficientes em todo o espaço de possíveis combinações é chamada de *fronteira de eficiência*. A mesma pode ser representada como um gráfico bidimensional, onde cada ponto possui coordenadas em um eixo relativo à sua esperança, e em outro relativo ao seu desvio padrão.

3.2 Definição do problema de Otimização de Média-Variância

Se assumirmos Σ como sendo positivo-definido, a variância é uma função estritamente convexa das variáveis do portfólio e existe um portfólio único que possui a menor variância. Chamaremos este portfólio de x_{min} e seu retorno $\mu^T x_{min}$ é R_{min}. R_{max} será definido como sendo o máximo retorno esperado para um portfólio.

O problema de otimização de média-variância de Markowitz pode ser formulado de três formas diferentes.

3.2.1 Encontrar o portfólio de menor variância que possua ao menos um retorno esperado desejado.

Esta abordagem pode ser formulada como o seguinte problema de otimização.

$$\begin{aligned} \min_x \frac{1}{2} x^T \Sigma x, & \\ \mu^T x \geq R, & \\ Ax = b, & \\ Cx \geq d. & \end{aligned} \tag{3.5}$$

A primeira restrição indica que o retorno esperado não deve ser menor do que o valor alvo R. Solucionando este problema tomando valores de R indo de R_{min} a R_{max} temos todos os portfólios eficientes. A função objetivo corresponde a metade da variância total do portfólio, o que contribui para a convergência em condições ótimas, mas não afeta a solução encontrada.

3.2.2 Encontrar o portfólio com o maior retorno esperado.

$$\begin{aligned} \max_x \mu^T x, \\ x^T \Sigma x \leq \sigma^2, \\ Ax = b, \\ Cx \geq d, \end{aligned} \tag{3.6}$$

onde σ^2 representa um limite superior para o valor da variância do portfólio.

3.2.3 Encontrar o portfólio que maximiza o retorno de risco ajustado.

$$\begin{aligned} \max_x \mu^T x - \frac{\delta}{2} x^T \Sigma x, \\ Ax = b, \\ Cx \geq d, \end{aligned} \tag{3.7}$$

onde a função objetivo é o *retorno de risco ajustado*, e a constante δ representa uma medida de *aversão ao risco*.

3.3 Métricas de Risco para Portfólios

Existem algumas métricas que podem ser usadas para caracterizar um portfólio, tanto quanto a retorno, como a risco. Nesta sessão, daremos destaque a duas, que podem ser vantajosas para o desenvolvimento do estudo proposto neste trabalho, os chamados *Sharpe ratio* e *Sortino ratio*.

Ambas as métricas podem ser utilizadas como função objetivo de problemas de otimização de portfólio, ao buscar-se maximizar seus valores.

3.3.1 Sharpe Ratio

Considere uma carteira formada por dois ativos, um ativo A, de retornos R_A e mais um ativo livre de risco de retornos r_f. A composição do mesmo, portanto, é:

- w_A alocado no ativo A;
- $w_f = 1 - w_A$ alocado no ativo livre de risco de retornos r_f.

Para o ativo livre de risco, assumimos as seguintes propriedades:

- $\mu_f = E[r_f] = r_f$;
- $var(r_f) = 0$;
- $cov(R_A, r_f) = 0$.

O retorno do portfólio é $R_p = (1 - w_A)r_f + w_A R_A = r_f + w_A(R_A - r_f)$. O retorno esperado é $\mu_p = r_f + w_A(\mu_A - r_f)$, sendo μ_A o retorno esperado para o ativo A. E como o ativo livre de risco é constante, e portanto possui variância zero, toda a variância do portfólio depende apenas do ativo A, e é dada por $\sigma_p^2 = w_A^2\sigma_A^2$. Disso, temos que

$$w_A = \frac{\sigma_p}{\sigma_A}.$$

Substituindo o valor para o peso na equação do retorno esperado do portfólio, temos

$$\mu_p = r_f + \left(\frac{\mu_A - r_f}{\sigma_A}\right)\sigma_p.$$

A inclinação da reta resultante desta equação, chamada de *linha de alocação de capital*, é o chamado *Sharpe ratio*.

$$SR_p = \frac{\mu_p - r_f}{\sigma_p}. \tag{3.8}$$

Na prática, portanto, este índice trás uma métrica que pode ser utilizada para classificar as propriedades de risco e retorno de ativos individuais, e comparar o desempenho de diferentes investimentos.

3.3.2 Sortino Ratio

O *Sharpe ratio* leva em consideração distribuições de retorno normais, e portanto, para portfólios assimétricos, como por exemplo, compostos por inúmeros retornos positivos pequenos, e com raros retorno negativos altos, este índice pode dar uma falsa sensação de segurança. Assim, Frank Sortino e Lee Price (SORTINO; PRICE, 1994) propuseram o chamado *Sortino ratio*, que considera o desvio padrão dos retornos negativos apenas, em relação a algum valor alvo. Assim, definindo a chamada *semivariância* $\overline{\sigma}_p$ como

$$\overline{\sigma}_p = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(min(0, R_{pi} - \tau))^2}, \tag{3.9}$$

onde N corresponde ao número de retornos considerados no período para o cálculo, R_p os retornos observados do portfólio e τ um retorno alvo, como por exemplo r_f, o *Sortino ratio* é calculado de maneira semelhante ao *Sharpe*,

$$S_p = \frac{\mu_p - r_f}{\overline{\sigma}_p}. \tag{3.10}$$

4 Aprendizado por Reforço para Finanças

A utilização de algoritmos de aprendizado de máquina no âmbito de mercado financeiro não é algo novo. Algoritmos conhecidos pela academia, como os descritos no livro *Machine Learning in Finance: From Theory to Practice* (DIXON; HALPERIN; BILOKON, 2020), são utilizados como alternativas às metodologias clássicas como Markowitz, e a outros métodos de otimização, como os listados em MANSINI; OGRYCZAK; SPERANZA, 2015.

Como destaca RUNDO et al., 2019, a automatização progressiva de certos processos, juntamente ao desenvolvimento da tecnologia possibilitou a utilização de algoritmos de aprendizado de máquina no ramo financeiro, onde a capacidade de análise de grandes volumes de dados com uma certa acurácia, aliado à capacidade de reconhecimento de padrões e relações nem sempre tão claras entre os dados são pontos de vantagem destas metodologias comparadas às clássicas. Em sua pesquisa, ainda demonstraram como metodologias de ML possuem melhores resultados em situações práticas.

Dentro desta categoria de abordagens, encontram-se os algoritmos de aprendizado por reforço, como os trabalhos listados em FISCHER, 2018. Existem diversas aplicações para programas deste tipo, tanto para *trading* - como em ZARKIAS et al., 2019, DENG et al., 2017, XIONG et al., 2018, LI; NI; CHANG, 2019 e AZHIKODAN; BHAT; JADHAV, 2018 - quanto para gerenciamento de portfólio - como em LI et al., 2019, LIANG et al., 2018, JIANG; LIANG, 2017 e YU et al., 2019.

O problema de gerenciamento de portfólio, foco deste trabalho, consiste em determinar uma alocação ótima para ativos dentro de uma carteira de ações, de forma a alcançar um objetivo, podendo ser tanto a maximização dos ganhos obtidos, quanto a minimização das perdas, em momentos de crise. A utilização de aprendizado por reforço nesse contexto mostra-se ideal, uma vez que não é necessário que o mesmo possua conhecimento do ambiente. Assim, ao invés de tentar aprender os padrões de comportamento das ações, e tentar predizer como

as mesmas vão se comportar, o agente deve aprender uma estratégia para lidar com elas. Isto muda a abordagem do problema, uma vez que não importa saber como o mundo funciona, contanto que ele consiga lidar com ele da melhor forma que puder. Podemos comparar este tipo de problema com o aprendizado de um robô em um jogo eletrônico, como o AlphaStar (VINYALS et al., 2019), que conseguiu vencer jogadores profissionais de StarCraft II. É impossível para um agente aprender o comportamento de um oponente humano, e obter um modelo das jogadas e táticas que serão realizadas. O que ele aprende é a melhor estratégia dentro do campo limitado de dados da partida que é fornecido a ele, e quais comportamentos possuem maior probabilidade de vitória, considerando o oponente.

As ações que o agente pode tomar apresentam-se de diferentes formas. Porém, a abordagem mais comum encontrada na literatura é modelar as ações como pesos diretos de cada ativo no portfólio, na forma de um vetor $\boldsymbol{w} = [w_1, w_2, ..., w_n]$ para uma composição de n ativos. Seguindo esse raciocínio, a ação do agente é contínua e sua política é determinística (uma vez que a cada instante t, o vetor $\boldsymbol{w}_t$ é tomado).

No artigo LI et al., 2019, os autores propõe uma metodologia chamada de *Adaptive Deep Deterministic Reinforcemente Learning*, *Adaptive DDPG*. Esta abordagem utiliza um framework Actor-Critic, visando um largo espaço de estados, uma rede alvo, a fim de estabilizar o processo de treinamento, e replay de experiência (planejamento), a fim de eliminar a correlação entre as amostras, e melhorar o aproveitamento de dados. Segundo os autores, sua proposta leva em consideração cenários pessimistas (chamados de *bear*, quando ocorre uma recessão ou uma depressão no mercado) e otimistas (chamados de *bull*, quando os valores dos ativos sobem em um ritmo mais alto do que de algum índice de interesse), e com base nos sinais de erro das predições das recompensas (recompensa real menos recompensa estimada), consegue ajustar a amplitude das mudanças das alocações no portfólio.

Em seu artigo, os autores ainda trazem a dificuldade da implementação de algoritmos de aprendizado por reforço para finanças, em comparação com problemas mais tradicionais da área. Eles citam a dificuldade de descrever o ambiente do mercado financeiro, particularmente a definição das *features* necessárias para

se entender um movimento de ações, e a grande presença de ruído, o que torna o estudo das séries temporais de preços uma tarefa complexa. Diferente de outras abordagens tradicionais de gerenciamento de porfólio, as ações aqui são definidas individualmente para cada ativo, como *aumentar peso em k*, *diminuir o peso em k* ou *nenhuma ação* visando ajustar a relação entre os pesos do portfólio da carteira operada, de acordo com o objetivo desejado. Assim, a saída da política do agente determina indiretamente qual vai ser a alocação no próximo período. O valor de k é um parâmetro do modelo. Sendo uma abordagem Actor-Critic utilizando Q-Learning para o aprendizado dos valores dos pares de estado-ação, o diferencial desta abordagem está no parâmetro de taxa de aprendizado durante a atualização da estimativa destes valores das funções Q, onde, dependendo se o agente está operando um mercado em *bull* ou *bear*, o valor utilizado é diferente. Ou seja, sendo uma regra de atualização do valor da função Q, como

$$Q_\pi(s_{t+1}, a_{t+1}) = Q_\pi(s, a) + \alpha\delta(t),$$

para o modelo apresentado, esta correção torna-se

$$Q_\pi(s_{t+1}, a_{t+1}) = Q_\pi(s, a) + \begin{cases} \alpha^+\sigma(t) & \text{se } \delta(t) > 0 \\ \alpha^-\sigma(t) & \text{se } \delta(t) < 0, \end{cases}$$

sendo $\delta(t)$ o valor da recompensa real menos o valor descontado da recompensa estimada,

$$\delta(t) = r(s_t, a_t, s_{t+1}) - Q_\pi(s_t, a_t).$$

Em LIANG et al., 2018, os autores compararam três estratégias típicas de aprendizado por reforço, no âmbito do problema de gerenciamento de portfólio. Foram testados agentes utilizando os algoritmos *Deep Deterministic Policy Gradient*, *Proximal Policy Optimization* e *Policy Gradient*, e ainda foi proposta uma metodologia de treinamento que os autores chamaram de *Adversarial Training method*.

Utilizando *features* com dados de séries históricas de preços de ativos do mercado chinês, e utilizando a metodologia nova proposta, onde um ruído é acrescido nos valores dos ativos no momento de calcular a recompensa do próximo período de tempo, o artigo conclui que, apesar de ambos os algoritmos possuírem usos potenciais para este tipo de problema, o algoritmo de *Policy Gradient* obteve um desempenho melhor do que seus competidores. Outro ponto trazido pelos autores é a capacidade de modelos de aprendizado por reforço profundo de capturar padrões de movimentação de mercado, apesar de visibilidade limitada de informações. Ainda assim, conclui que existe espaço para aprimoramentos, principalmente no que tange a alta sensibilidade de agentes ao ambiente ruidoso do mercado financeiro.

De modo geral, o que esses artigos trazem de ponto comum é a possibilidade de modelagem do mercado financeiro como um problema MDP, o que implica na oportunidade de aplicação de algoritmos de aprendizado por reforço na resolução dos mesmos.

De modo resumido, partindo da definição de MDP dada em SUTTON; BARTO, 2018 - e descrito na seção 2.3.1 -, considere um agente que, a cada instante de tempo, recebe informações sobre o estado atual do mercado, e usa essas informações para decidir qual a próxima alocação para uma carteira de ativos. No próximo instante de tempo, essa alocação representa um retorno financeiro, que é agregado de maneira incremental ao valor do portfólio, e novas informações do estado do mercado são recebidas. Esta formulação caracteriza um processo de decisão Markoviano, e portanto permite a utilização de técnicas de aprendizado por reforço para sua resolução.

4.1 Aplicação de Algoritmos de Aprendizado por Reforço para Gerenciamento de Portfólios

As seções à seguir avaliam a aplicação de uma das abordagens mais recentes presentes na literatura, e as contribuições em termos de modelagem e abordagem do problema.

4.1.1 Revisão Bibliográfica

No artigo de JIANG; XU; LIANG, 2017, os autores propõem a aplicação de uma rotina de aprendizado por reforço para fazer a alocação de pesos em um portfólio de ativos de criptomoedas. Neste caso, o algoritmo proposto leva em consideração custos de transação de compra e venda de ativos, além da evolução do desempenho dos mesmos, ao fazer a decisão da alocação.

O algoritmo foi desenvolvido para operar no mercado de criptomoedas chinês. A decisão dos autores baseou-se nas premissas de que as mesmas são descentralizadas e abertas, o que, segundo os autores, faz desse mercado um bom ambiente para testes. Sem um órgão centralizador, qualquer um pode participar nas negociações sem muito esforço inicial. Já o fato de serem abertas implica que as moedas negociadas são mais acessíveis, com uma maior facilidade de obtenção de dados (uma vez que grande parte das moedas possui interface para acesso a dados de mercado, e sem limite de frequência de transações, segundo os autores destacam no artigo). Estas duas características tornam este tipo de negociação ideal para algoritmos de aprendizado poderem atuar com dados reais e em curtos períodos de tempo.

A base do algoritmo proposto consiste no chamado *Ensemble of Identical Independent Evaluators* (EIIE). Esta estrutura consiste em uma rede neural avaliando o histórico de preços de um ativo, e estimando seu potencial de crescimento, enquanto leva em conta o volume da alteração de peso deste ativo dentro de um portfólio . Este comportamento remete à avaliação dos custos de transação sobre o valor de um portfólio, que influencia na decisão do próximo vetor de pesos.

Considere um portfólio contendo m ativos. Os preços de fechamento de cada ativo, ao final do período de negociação t está contido no vetor de preços $\mathbf{v}_t$. Este período de negociação consiste no intervalo de tempo em que o valor dos ativos serão avaliados, e no qual ao final as transações são realizadas. No artigo, os autores consideraram um intervalo de 30 minutos. Considere também os vetores contendo o valor de máximo e mínimo deste ativo dentro da janela de tempo t, respectivamente $\mathbf{v}_t^{(hi)}$ e $\mathbf{v}_t^{(lo)}$. Na definição dos vetores feita pelos autores, foi considerada a existência de um ativo representando *cash*, que é representação

do montante não aplicado em nenhum ativo. No artigo, o montante representado corresponde ao *Bitcoin*, e os ativos relacionados serão representados a partir do valor relativo ao mesmo. Este ativo corresponde ao primeiro elemento dos vetores descritos, e valerá sempre 1, correspondendo ao valor do *Bitcoin* relativo a ele mesmo.

O artigo define, a partir disso, o chamado *vetor de preços relativos*, que será utilizado pelo agente para tomada de decisão. A utilização de valores relativos minimiza o efeito da discrepância de valores absolutos entre os ativos, evitando que o agente dê mais peso a moedas com um maior valor em módulo. Além disso, a utilização deste vetor auxilia no cálculo da variação de preço de um portfólio, abstraindo o seu valor inicial. Sendo assim, o vetor de preços relativos corresponde a

$$\boldsymbol{y}_t := \frac{\boldsymbol{v}_t}{\boldsymbol{v}_{t-1}} = \left(1, \frac{v_{1,t}}{v_{1,t-1}}, \frac{v_{2,t}}{v_{2,t-1}}, \ldots, \frac{v_{m,t}}{v_{m,t-1}}\right). \tag{4.1}$$

E o valor do portfólio, ao final do período t, passa a ser de

$$p_t = p_{t-1}\boldsymbol{y}_t \cdot \boldsymbol{w}_{t-1}, \tag{4.2}$$

sendo o *vetor do portfólio* $\mathbf{w}_{t-1}$ o valor do peso dos ativos no portfólio durante a janela do período t. Um vetor de portfólio $\mathbf{w}_t = (1, 0, 0, \ldots, 0)^T$, por exemplo, indica todo o capital aplicado na moeda, ao invés de investido em ativos. Ainda temos a *taxa de retorno* do portfólio, calculada como

$$\rho_t := \frac{p_t}{p_{t-1}} - 1 = \boldsymbol{y}_t \cdot \boldsymbol{w}_{t-1} - 1, \tag{4.3}$$

com seu valor logarítmico calculado como

$$r_t := \ln \frac{p_t}{p_{t-1}} = \ln(\boldsymbol{y}_t \cdot \boldsymbol{w}_{t-1}). \tag{4.4}$$

Cada operação realizada em um cenário real de mercado possui um custo associado. Algumas modelagens desprezam os mesmos para fins de simplificação da

lógica aplicada, obtendo assim uma estimativa aproximada do comportamento de um portfólio negociado período a período. Porém, por não corresponder à situação vista no mundo real ao rebalancear os pesos dos ativos na carteira, uma estratégia que inclua estes gastos pode mostrar-se muito mais otimizada e eficiente do que a aproximação comum obtida. Partindo desse princípio, o artigo procura modelar a lógica de negociação de ativos na presença dos custos de transação, e obter uma estimativa da carteira muito mais realista. Esta torna-se a base dos cálculos da recompensa do algoritmo de aprendizado por reforço.

Seja $\mathbf{w}_{t-1}$ o vetor de pesos de portfólio no início de uma janela de transação t. Devido à movimentações de preço dos ativos ao longo do tempo, ao final desta janela os pesos dentro da carteira tornam-se

$$\boldsymbol{w}'_t = \frac{\boldsymbol{y}_t \odot \boldsymbol{w}_{t-1}}{\boldsymbol{y}_t \cdot \boldsymbol{w}_{t-1}}, \tag{4.5}$$

sendo $\odot$ uma multiplicação elemento a elemento. O valor do portfólio ao final do período, agora, é p'_t. O agente irá decidir qual a nova alocação de ativos para a próxima janela, $t+1$, denominado $\boldsymbol{w}_t$, desta vez considerando que, passar de $\boldsymbol{w}'_t$ para $\boldsymbol{w}_t$ terá custos que devem ser pagos, e que diminuirão o valor do portfólio, ao final dessas transações, para um valor p_t, por um fator μ_t, chamado de *fator de resíduo de transação*,

$$p_t = \mu_t p'_t, \tag{4.6}$$

alterando os cálculos das equações 4.3 e 4.4 para considerar essa movimentação,

$$\rho_t := \frac{p_t}{p_{t-1}} - 1 = \frac{\mu_t p'_t}{p_{t-1}} - 1 = \mu_t \boldsymbol{y}_t \cdot \boldsymbol{w}_{t-1} - 1, \tag{4.7}$$

e

$$r_t := \ln \frac{p_t}{p_{t-1}} = \ln \frac{\mu_t p'_t}{p_{t-1}} = \ln(\mu_t \boldsymbol{y}_t \cdot \boldsymbol{w}_{t-1}). \tag{4.8}$$

O fator μ_t, por sua vez, é calculado levando em consideração tanto os valores dos custos de transação, em porcentagem, c_p de compra e c_s de venda, quanto a

diferença entre os valores dos pesos dos ativos a cada passagem de tempo. Dessa forma, o cálculo é feito de maneira incremental, a partir de uma função definida como

$$f(\mu) := \frac{1}{1 - c_p w_{t,0}} \left[1 - c_p w'_{t,0} - (c_s + c_p - c_s c_p) \sum_{i=1}^{m} (w'_{t,i} - \mu w_{t,i})^+ \right], \tag{4.9}$$

sendo $w'_{t,0}$ e $w_{t,0}$ o valor alocado no ativo *cash* ao final do período corrente $t-1$ após as movimentações de mercado, e seu próximo valor calculado para o período t, respectivamente. A operação $(\cdot)^+$ corresponde a $ReLU(\cdot)$. Esta função é utilizada para aproximar iterativamente o termo μ a partir de uma estimativa inicial $\mu_\odot$, da forma

$$\mu_t^{(k)} = f(\mu_t^{(k-1)}), \tag{4.10}$$

para $\mu_t^0 = \mu_\odot$. No artigo é possível ver uma demonstração da convergência deste método. A velocidade da mesma depende do erro da estimativa inicial do valor final da solução, $|\mu_t - \mu_\odot|$. Os autores utilizam o seguinte cálculo para a estimativa inicial do valor do fator, válido quando $c_s = c_p = c$,

$$\mu_\odot = c \sum_{i=1}^{m} |w'_{t,i} - w_{t,i}|. \tag{4.11}$$

O fator k utilizado na equação 4.10 indica uma iteração do cálculo. Em geral, um valor fixo para o número de iterações pode ser utilizado em um algoritmo, e na implementação do artigo o cálculo era feito em 15 iterações.

A Figura 11 ilustra a lógica considerada nos cálculos do modelo.

A representação de estado utilizada durante a modelagem do tema como um problema de aprendizado por reforço consiste em uma junção de variáveis históricas de preço de ativos, e da última configuração de pesos do portfólio, sendo,

$$\boldsymbol{s}_t = (\boldsymbol{X}_t, \boldsymbol{w}_{t-1}), \tag{4.12}$$

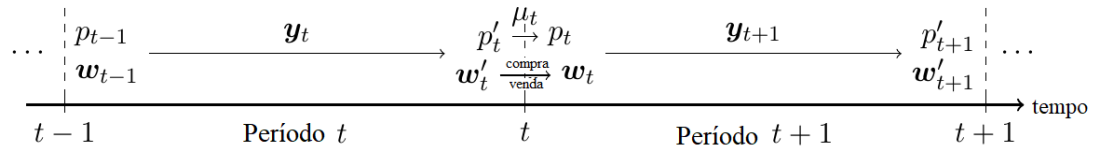


Figura 11 – Ilustração de um período de negociação dos ativos e ajuste do portfólio. Adaptado de JIANG; XU; LIANG, 2017.

em que $\boldsymbol{X}_t$ é um tensor contendo o histórico das últimas $n = 50$ entradas do histórico de preços, chamado de *tensor de preços*. É composto da forma:

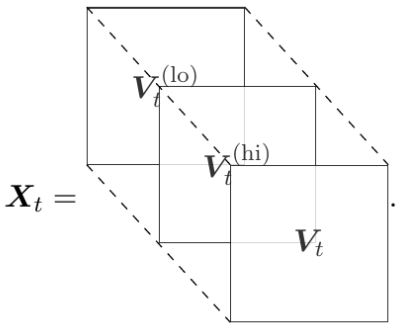


Figura 12 – Tensor de preços

com

$$\boldsymbol{X}_t = [\boldsymbol{V}_t | \boldsymbol{V}_t^{(hi)} | \boldsymbol{V}_t^{(lo)}], \tag{4.13}$$

$$\boldsymbol{V}_t = \left[\boldsymbol{v}_{t-n+1} \oslash \boldsymbol{v}_t | \boldsymbol{v}_{t-n+2} \oslash \boldsymbol{v}_t | \ldots | \boldsymbol{v}_{t-1} \oslash \boldsymbol{v}_t | \mathbf{1}\right], \tag{4.14}$$

$$\boldsymbol{V}_t^{(hi)} = \left[\boldsymbol{v}_{t-n+1}^{(hi)} \oslash \boldsymbol{v}_t | \boldsymbol{v}_{t-n+2}^{(hi)} \oslash \boldsymbol{v}_t | \ldots | \boldsymbol{v}_{t-1}^{(hi)} \oslash \boldsymbol{v}_t | \boldsymbol{v}_t^{(hi)} \oslash \boldsymbol{v}_t\right], \tag{4.15}$$

$$\boldsymbol{V}_t^{(lo)} = \left[\boldsymbol{v}_{t-n+1}^{(lo)} \oslash \boldsymbol{v}_t | \boldsymbol{v}_{t-n+2}^{(lo)} \oslash \boldsymbol{v}_t | \ldots | \boldsymbol{v}_{t-1}^{(lo)} \oslash \boldsymbol{v}_t | \boldsymbol{v}_t^{(lo)} \oslash \boldsymbol{v}_t\right]. \tag{4.16}$$

Ou seja, uma combinação das matrizes de preços de fechamento, alta e baixa de cada um dos ativos considerados, normalizada pelo último valor de fechamento de cada ativo, em uma operação item a item, indicada pelo símbolo $\oslash$.

A definição da recompensa é feita tendo como base o objetivo do agente em si, que é maximizar o valor final de um portfólio, dado pelo retorno acumu-

lado de um período de transação (episódio), a partir dos retornos imediatos de cada transição de estado (cada configuração de valores de ativos e pesos dos mesmos na carteira). Portanto, considerando um episódio qualquer de duração t_f, a recompensa utilizada é dada por

$$R(\boldsymbol{s}_1, \boldsymbol{a}_1, \ldots, \boldsymbol{s}_{t_f}, \boldsymbol{a}_{t_f}, \boldsymbol{s}_{t_f+1}) := \frac{1}{t_f} \ln \frac{p_f}{p_0} = \frac{1}{t_f} \sum_{t=1}^{t_f+1} \ln(\mu_t \boldsymbol{y}_t \cdot \boldsymbol{w}_{t-1}) = \frac{1}{t_f} \sum_{t=1}^{t_f} r_t. \tag{4.17}$$

Segundo o artigo, os ativos selecionados para execução pelo agente deveriam possuir um alto volume de transação, o que na prática permitiria aproximar a premissa de que as negociações feitas pelo agente não iriam interferir no valor de mercado das moedas. Desta forma, a recompensa dada pela equação 4.17 permite calcular o valor exato do retorno para cada estado ou ação, uma vez que os valores das moedas serão os mesmos independente da ação (configuração de pesos da carteira) escolhida. Isto significa que tanto um histórico de preços pode ser utilizado para retreino ou planejamento do agente mesmo que este escolha diferentes configurações de peso, quanto que não existe a necessidade de considerar exploração, uma vez que o valor exato das ações pode ser calculado, e não há a necessidade de avaliar se outras configurações de peso, dado um preço, teriam um efeito melhor, considerando o retorno acumulado. Assim, o agente pode focar em escolher a ação com melhor retorno imediato a cada etapa de tempo.

A estratégia abordada para a resolução do problema consiste em Gradiente de Política Determinístico (DPG[1]) (SILVER et al., 2014), onde a ação é determinada de maneira direta a partir do estado, utilizando uma função ou estrutura parametrizada, como uma rede neural, e ajustando os parâmetros seguindo o gradiente de uma métrica de performance $J(\boldsymbol{\theta}_t)$, como descrito na seção 2.5.3. Alinhado com o objetivo geral do problema, que é otimizar o valor final de um portfólio em um período de negociação, a métrica utilizada pode ser definida, portanto, como

$$J(\boldsymbol{\theta}_t) = R(\boldsymbol{s}_1, \pi_{\boldsymbol{\theta}}(\boldsymbol{s}_1), \ldots, \boldsymbol{s}_{t_f}, \pi_{\boldsymbol{\theta}}(\boldsymbol{s}_{t_f}), \boldsymbol{s}_{t_f+1}), \tag{4.18}$$

[1] Sigla derivada da expressão em inglês *Deterministic Policy Gradient*

onde o treinamento é realizado utilizando batches de períodos do histórico de preços, ao invés de todo o histórico disponível, para otimizar eficiência. Esse, por sua vez, é amostrado do histórico conhecido considerando uma probabilidade distribuída geometricamente, onde períodos mais recentes do histórico possuem uma chance maior de serem escolhidos do que períodos mais antigos. A justificativa por trás desta abordagem é que a correlação entre dois eventos de preço de ativos no mercado decresce exponencialmente com a distância temporal entre eles HOLT, 2004, KIRKPATRICK; DAHLQUIST, 2006. Portanto, dado um ponto no tempo t_b, dentro do tempo contemplado no histórico de dados disponível, e seus subsequentes n_b intervalos, e ainda considerando que o histórico possui t entradas totais, a probabilidade deste ponto $t_b \leq t - n_b$ ser escolhido dentre todos é de

$$Pr_\beta(t_b) = \beta(1-\beta)^{(t-n_b-t_b)}.$$

Os autores utilizaram três tipos de implementação para a rede neural responsável pela política, sendo uma CNN, uma rede LSTM e uma arquitetura RNN. Em ambos os casos, a rede é construída de forma que o fluxo de dados é avaliado de maneira independente para cada ativo do portfólio, porém os parâmetros das redes são compartilhados entre eles. Desta forma consegue-se uma generalização da rede para os ativos avaliados, mas com a garantia de que cada decisão envolvendo um ativo leva em conta apenas a parte do estado respectiva a ela. Esses fluxos são chamados de Avaliadores Idênticos Independentes (IIE[2]), enquanto a topologia é chamada de Conjunto de Avaliadores Idênticos Independentes(EIIE[3]), que como citado anteriormente, é a base da implementação. A forma como a topologia é realizada varia com a arquitetura, mas para visualização do que ela significa na prática, podemos citar o caso de uma rede CNN. A topologia EIIE pode ser entendida como uma cadeia de convoluções com kernels de altura 1, onde, assim, durante a operação de convolução, apenas os dados do ativo - linha de cada matriz do tensor de entrada - é considerado. A Figura 13 ilustra essa implementação.

Nos testes realizados com esta metodologia, os autores observaram uma

[2] Do inglês *Identical Independent Evaluators*
[3] Do inglês *Ensemble of Identical Independent Evaluators*

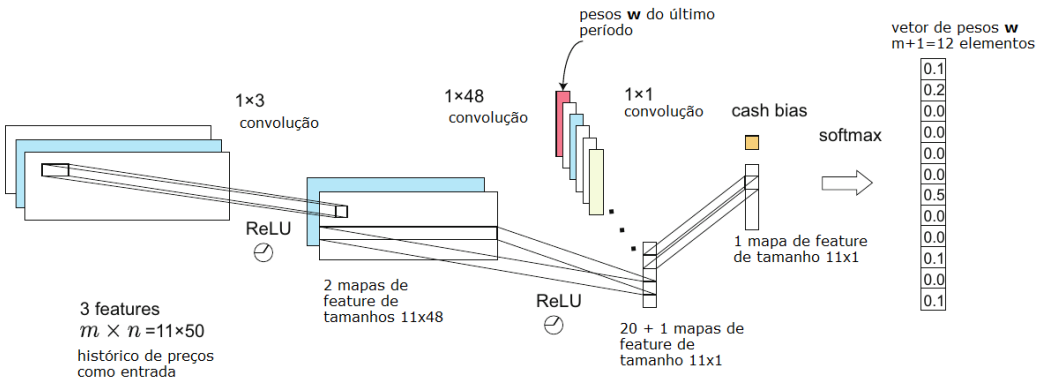


Figura 13 – Implementação do EIIE como uma CNN. Figura adaptada de JIANG; XU; LIANG, 2017.

melhora nos desempenhos dos algoritmos propostos, quando comparados com metologias mais comuns utilizadas no problema de gerenciamento de portfólio, considerando as métricas observadas, como o valor acumulado final dos portfólios. Em particular, as redes construídas com CNN e RNN obtiveram um desempenho melhor do que a arquitetura com LSTM, sendo redes CNN mais eficientes em dois dos três backtests realizados. Como os autores destacaram, apesar de haver ainda amplo espaço para melhorias tanto da lógica empregada, quanto da exploração de diferentes cenários, o uso da arquitetura proposta indica um caminho promissor para a implementação de algoritmos de aprendizado por reforço em cenários de gerenciamento de portfólio.

Apesar do ganho apresentado pela arquitetura proposta, a representação dos ativos no estado ainda apresentava um comportamento bastante incerto, mesmo incorporando informações temporais dos preços. A partir desta linha de raciocínio, Yunan Ye et al. propuseram uma implementação buscando ampliar o conhecimento do agente sobre o ambiente que o mesmo se relaciona, ao inserir informações extras que podem ser úteis para a estimativa do comportamento do mercado. A implementação recebe o nome de *Framework de Aprendizado por Reforço com Espaço Aumentado*[4], ou *SARL* (YE et al., 2020).

[4] Livre tradução do nome original, *State-Augmented RL framework*.

A premissa para esta abordagem é a de que os valores de ativos no mercado são bastante ruidosos, diversos e não balanceados, além de possuírem bastante incerteza, o que pode levar a mudanças nas distribuições dos dados de treinamento e teste. Assim, levar em consideração apenas o histórico de preços pode minar a capacidade de aprendizado de um modelo. A proposta dos autores para mitigar este problema consiste em adicionar às informações de estado dados extras sobre os ativos, que podem auxiliar o agente na tomada de decisão da alocação dos pesos em uma carteira. Estes dados podem vir de diferentes fontes, como por exemplo embeddings de notícias que podem oferecer um impacto no mercado financeiro, ou predições de preços ou movimentações nos valores de ativos, fornecidos por modelos externos, mesmo que não tão acurados.

Tomando por base uma modelagem dos vetores de preço, $\boldsymbol{y}_t$, e peso $\boldsymbol{w}_t$ similar à utilizada pelo artigo de JIANG; XU; LIANG, 2017, o artigo propõe uma representação de estados composta por

$$s = (s^*, \delta), \tag{4.19}$$

sendo s^* um vetor com os preços atuais dos ativos sendo considerados, o que compõe a informação interna do ambiente, e δ um componente externo que pode ser tanto informação oriunda das características internas do sistema (como rótulo de subida/descida obtida através de um preditor separado), quanto informações de fontes externas (como embeddings de notícias).

O agente utiliza um algoritmo Gradiente de Política Determinístico, DPG[5], para determinar as ações a serem realizadas. Este algoritmo utiliza de uma rede neural para aprender uma política de forma não linear, como explicado na seção 2.5.3. A função objetivo aprendida pela rede consiste no retorno logarítmico do portfólio, considerando também os custos de transação, assim como feito no artigo de JIANG; XU; LIANG, 2017.

[5] Do inglês *Determinístic Policy Gradient*

4.1.2 Considerações e Replicação de Resultados

O artigo propõe utilizar estados passados para treinamento do agente, tal qual atividade de planning (sec. 2.4.4). Para isso, usa o histórico de preços, e um histórico de pesos usados com esses preços, salvo em memória, para treinamento, como se fosse um episódio. O artigo sugere que, à medida que os parâmetros do agente convergem à valores ótimos para o comportamento desejado, estes pesos salvos em memória também convergem. Do ponto de vista de aprendizado por reforço, esta abordagem equivale a considerar ocorrência de estados aleatórias como se tivessem ocorrido em sequência, como se fossem um episódio. Assim, ao final, o que está sendo fornecido pelo modelo não é um episódio completo, onde o mesmo aprenderá a obter a melhor recompensa ao final do mesmo, mas sim uma série de estados, onde o modelo estará otimizando a média das recompensas das decisões para cada estado sozinho, e aprendendo a tomar a melhor decisão instantânea. Ou seja, estará aprendendo a tomar a melhor decisão para apenas o estado atual, sem considerar o todo - já que não possui informações do *valor do estado*, por ser uma abordagem de gradiente de política, e não considera o retorno acumulado correto, apenas uma soma de recompensas individuais desconexas. Esta abordagem não é ótima, do ponto de vista de RL, haja visto que, para obter a melhor recompensa para um episódio, o modelo às vezes precisa tomar decisões que não são ótimas para o estado atual, mas que levem a recompensas melhores ao final. O problema do *mountain car* (fig. 1), já citado, é um bom exemplo disso. A Figura 14 ilustra o problema dessa abordagem.

Há quem possa argumentar que mesmo assim esta abordagem seja vantajosa, afinal tomar as melhores decisões a cada passo poderia significar um melhor retorno ao final do episódio. Porém, considerando que o problema contém perdas através de custos de transação, por exemplo, se um agente não souber balancear com cuidado as alocações de pesos ao longo do episódio, de forma a evitar compras e vendas exageradas, pode acabar fazendo escolhas que levem a perdas no geral, e portanto não necessariamente obter um retorno ótimo. É importante que o decisor tenha visibilidade e entenda como o ambiente se comporta, e como suas decisões, desde o começo, são influentes, para poder chegar a um bom comportamento frente ao problema visto. E, mesmo nessa abordagem, ainda assim um agente poderia

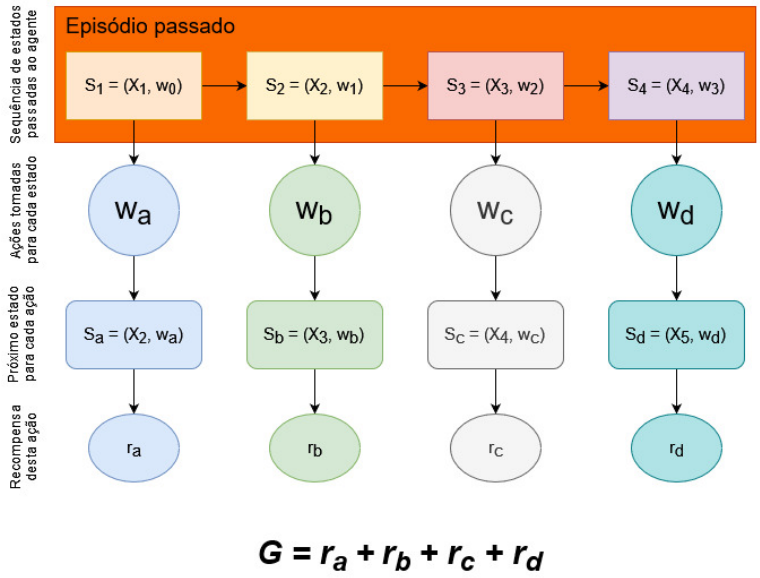


Figura 14 – Imagine que uma sequência de estados, selecionada da memória, seja utilizada para *planning*. A sequência já possui nas informações de estados as decisões tomadas pelo agente a cada passo, representadas pelos vetores de peso $\boldsymbol{w} = \{\boldsymbol{w}_1, \boldsymbol{w}_2, \boldsymbol{w}_3, \boldsymbol{w}_4\}$. $\boldsymbol{w}_0$ é a alocação inicial do portfólio. Porém, a cada intervalo apresentado para o agente, ao invés do mesmo tomar a mesma decisão que está registrada na memória, ele toma outra - por exemplo, ao invés de $\boldsymbol{w}_1$, selecionar $\boldsymbol{w}_a$ - o que é esperado, afinal o agente aprendeu de lá para cá. Assim, o próximo estado da sequência é diferente do que o episódio registrou. Apesar disso, o que é passado para o modelo, como próximo passo, é o estado oriundo da memória, e não o estado visto nesta nova interação com o ambiente. Finalmente, dessa forma, as recompensas obtidas (r) representam apenas as recompensas instantâneas desta passagem de estado única. Ao somá-las, o que consideramos ser o retorno do episódio, G, na realidade é a soma das recompensas instantâneas não relacionadas.

acabar aprendendo a tomar apenas as melhores decisões para cada estado visto, caso isso signifique um melhor resultado ao final.

Essa abordagem, ao final das contas, assemelha-se mais à amostrar uma série de episódios com apenas dois passos de duração (o primeiro passo sendo os ativos no tempo t, e o segundo os ativos no tempo $t + 1$, sendo este o estado

terminal), e aprender maximizando a função $J(\theta)$ para o conjunto de episódios (através da média dos retornos dos portfólios). Em um outro contexto de problema, uma abordagem como esta, porém, poderia ser vantajosa.

Além disso, ao verificar o algoritmo utilizado pelos autores, disponível no github, os mesmos optaram por utilizar um treinamento em batches para o modelo, com a intenção de acelerar o tempo de aprendizado do agente. A base para esta implementação foi justamente a utilização dos dados em memória dos estados, onde cada item do batch corresponde a um estado do episódio. No artigo os autores ainda justificam que, devido às considerações realizadas para o problema - de que os valores dos ativos não se alteram devido às transações deste agente, pelo alto volume de transações do mercado -, os valores dos estados não dependeriam da decisão tomada no passo anterior. Porém, como a informação de um estado *precisa* da decisão tomada anteriormente, os mesmos também precisam ser sequenciais, o que inviabilizaria um treinamento em batch, como o realizado.

Avaliando o código escrito, existiam ainda alguns *bugs* na implementação, como no valor do cálculo da recompensa durante o treinamento. Além disso, algumas lógicas ainda estavam diferentes do que mencionado anteriormente no artigo, o que pode indicar também que os autores tentaram de abordagens diferentes após a escrita do texto. Assim, foi tomado como base apenas o texto do artigo, e a partir disso feita uma interpretação que procurou ser mais coerente com a temática de aprendizado por reforço, e da forma como os algoritmos desse tipo são elaborados.

O modelo adaptado utilizado para teste realizava o treinamento e planejamento considerando tomadas de decisões sequenciais. Assim, tendo como informação de entrada uma sequência do componente $\boldsymbol{X}$ dos estados s, e uma alocação de pesos iniciais, o agente tomava suas decisões a cada informação da sequência, e compunha o próximo estado como sendo o próximo valor de $\boldsymbol{X}_t$ e o vetor de pesos anterior ($\boldsymbol{w}_{t-1}$). O treinamento foi feito por 80.000 épocas, onde cada uma consistia em uma amostra aleatória de um período in-sample de 50 subamostras.

O backtest realizado com o período out-of-sample, porém, obteve o resultado mostrado na Figura 15. Como pode ser visto, o agente não conseguiu obter um desempenho satisfatório ao longo do período, e o portfólio apresentou perdas.

Como comparativo, implementamos a solução proposta pelo artigo YE et al., 2020. Esta não possuía código público disponível para referência, portanto o modelo partiu das descrições apresentadas pelo texto.

Como o mesmo não citou a arquitetura da rede construída, e também utilizava de série histórica de preços de ativos como parâmetro de entrada, e também como citou o artigo de Jiang et al. como referência, utilizamos a mesma arquitetura construída com o modelo anterior, com algumas diferenças. A primeira é a utilização de uma camada de normalização em batch após cada saída de rede. O modelo anterior não utilizava esta estrutura, mas por ser uma técnica conhecida na literatura para acelerar o processo de treinamento - ao trazer as saídas dos neurônios de uma camada para uma mesma escala - decidimos utilizá-la também, para esta nova implementação. A outra diferença das arquiteturas é na camada EIIE do modelo anterior. O artigo citava a utilização de uma matriz com a série histórica dos últimos 30 dias de preços para cada ativo como informação de entrada. Como as informações aumentadas de estado consistiam em um vetor de uma dimensão, havia a necessidade de ajustar na arquitetura do modelo essas duas estruturas de formatos diferentes. Portanto, aproveitamos a camada EIIE da rede anterior, para embutir este vetor de informação extra, onde antes era introduzida a informação de pesos passados do portfólio.

Como informação de estado aumentado, foi implementado um preditor simples, que emitia como saída um valor binário indicando se, para o próximo período, o preço do ativo seria maior ou menor do que o atual. Para simular esse modelo, o mesmo recebia o valor relativo dos ativos deste próximo período, em relação ao anterior, com 55% de chance de acertar as predições, e outros 45% de devolver um valor aleatório. Este, por sua vez, consistia em uma probabilidade de 50% de chance para 1 ou 0, para cada ativo. Como estamos testando em um ambiente controlado, e possuímos todos os dados out-of-sample, o objetivo é verificar a hipótese descrita pelo artigo de que uma informação extra, mesmo que com uma precisão baixa, possa ser vantajosa para a tomada de decisão do agente.

O agente foi treinado da mesma forma que o modelo anterior, sob os mesmos dados in-sample, e o seu desempenho com o conjunto de testes pode ser visto na Figura 15.

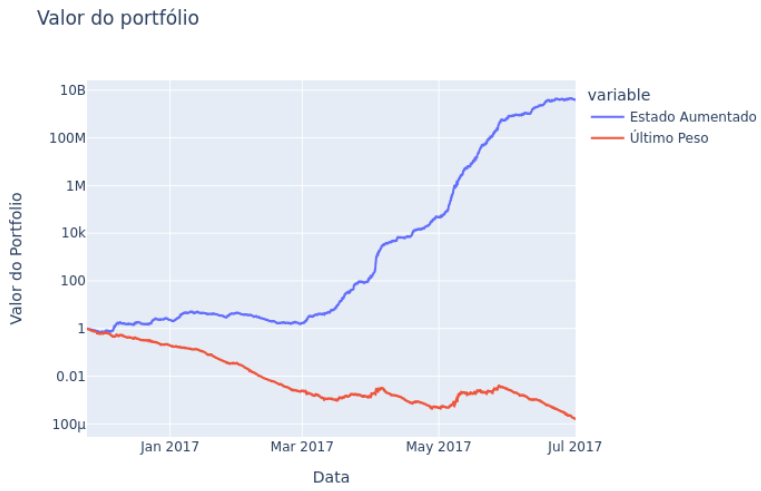


Figura 15 – Desempenho do agente com estado aumentado, e considerando último peso, sobre os dados do período out-of-sample.

Comparando os dois resultados, é possível ver como a implementação mais recente acaba tendo um desempenho melhor do que a anterior. Podemos entender que, para um agente realizando operações com ativos, é mais vantajoso estender o conhecimento sobre os ativos operados, do que conhecer o estado anterior do portfólio, para prever perdas por transação. Assim, entendemos que a decisão tomada agora é mais assertiva, e acaba oferecendo ganhos maiores do que eventuais perdas. Este ponto está em acordo com o que foi observado no artigo original, e os resultados experimentais nos possibilitaram observar isso. Uma forma de generalizar este conceito, e entender melhor à respeito, poderia ser aplicar a mesma abordagem a outros mercados, considerando diferentes composições de carteira.

Quanto ao primeiro artigo, apesar de não termos conseguido reproduzir os resultados do modelo apresentado, a partir de versões adaptadas do apresentado no texto, é importante reconhecer avanços trazidos pelo mesmo para a modelagem de problemas envolvendo ativos. Podemos citar a utilização do vetor de preços relativos, e a maneira como aborda os custos de transação, modelando os mesmos de forma que o agente possa ter uma visão mais próxima da realidade das transações, inspirando abordagens como em HUANG; ZHOU; SONG, 2021.

5 Estudo de Caso

Com a revisão realizada, aplicamos técnicas de aprendizado por reforço em um problema de gerenciamento de portfólio. O objetivo é observar como diferentes abordagens influenciam os resultados obtidos, e trazer uma linha de raciocínio sobre como diferentes aspectos da modelagem interferem no comportamento desejado por parte do agente. Assim, vamos aplicar a teoria em um problema simplificado de otimização de portfólio, a fim de demonstrar com um exemplo como este tipo de problema pode ser modelado como um problema de aprendizado por reforço, e como uma possível linha de raciocínio de resolução pode ser desenvolvida. A situação abordada considera uma carteira simples, de apenas 3 ativos, que permita observar como um agente realiza suas escolhas de modo mais fácil do que analisando uma carteira com dezenas de ativos.

5.1 Descrição do Problema

O problema abordado consiste na otimização da alocação de uma carteira de ações, composta por três ativos do mercado financeiro americano. O primeiro ativo, SPY, é uma ETF que replica o comportamento do índice S&P500, que é uma média ponderada das 500 principais empresas americanas no momento atual (chamadas *blue chips*, são as empresas com valor de mercado alto. Também compõe a carteira as *midcaps*, empresas com tamanho médio (SUNO, 2021).). Uma ETF é um fundo de investimento que é atrelado a um índice de referência - como o S&P500 ou o IBOVESPA - constituído com o objetivo de replicar a rentabilidade deste indicador. Ao adquirir cotas deste fundo, o investidor passa a deter de maneira indireta todas as ações do mesmo (INVESTIDOR, 2022).

O segundo corresponde a um ativo de proteção, referenciado como *hedge*, que na maior parte do tempo (especialmente nos momentos de crise) possui uma correlação negativa com o mercado. Nesse caso específico, usamos TLT que representa Títulos do tesouro americano com vencimento de 20 anos ou mais com o

índice do mercado (*20+ treasure bonds*). Ver ETF.COM, 2022, INVESTOPEDIA, 2020.

O terceiro ativo é uma ETF que representa o setor de tecnologia do índice S&P500, representado pela sigla XLK. Sua carteira de investimentos inclui companhias das indústrias de hardware, armazenamento e periféricos, software, equipamentos de comunicação, semicondutores, serviços de TI e equipamentos, instrumentos e componentes eletrônicos (SPDR, 2022). O *beta*[1] desta ETF é de 1.05, o que indica que este ativo varia praticamente em linha com o mercado (ou seja, se o índice S&P crescer 1%, esta ETF irá crescer 1.05%).

Utilizamos dados diários dos ativos, indo de 2002 à meados de 2021, divididos em dois datasets. O primeiro, utilizado para treinamento da rede do agente, chamado período *in-sample*, e o segundo, utilizado para *backtesting*, chamado *out-of-sample*. Os gráficos de desempenho (retorno cumulativo) dos ativos, dia à dia, podem ser vistos nas Figuras 16 e 17.

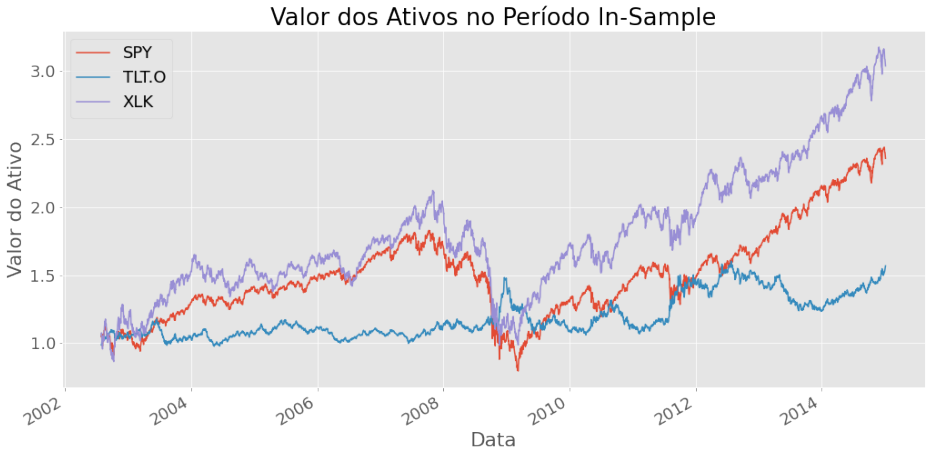


Figura 16 – Evolução dos valores dos ativos no período in-sample.

É interessante notar que nos dois períodos considerados estão presentes momentos de queda do índice SPY e XLK, e que nesses momentos o ativo de

[1] *Beta* é uma métrica estatística comumente utilizada em análises de investimento. Avalia a sensibilidade de um ativo em relação ao mercado, e pode ser uma boa medida de volatilidade de um ativo (INVESTOPEDIA, 2021; DATABASE, 2016).

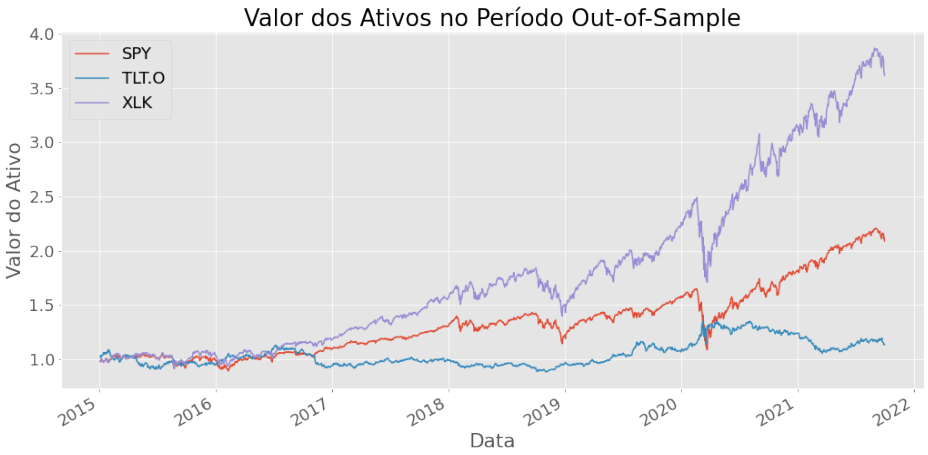


Figura 17 – Evolução dos valores dos ativos no período out-of-sample.

proteção TLT.O, ou possui alta, ou mantém-se estável. A grosso modo, o objetivo deste ativo é proteger a carteira justamente durante essas situações, e espera-se que um gestor consiga aproveitar-se desse benefício nesses períodos. Porém, como o ativo hedge não possui um retorno muito grande, no restante do tempo o investimento deve estar com maior foco nas outras duas ações. Ainda, o ativo XLK apresenta recuperações pós-quedas mais rápidas do que o mercado, portanto espera-se também um melhor aproveitamento do mesmo nos momentos de pull-back.

Um conceito abordado por autores diversos é o de *regimes de mercado* (DIEBOLD; LEE; WEINBACH, 1994; ANG; TIMMERMANN, 2012), caracterizados por períodos onde os ativos apresentam um comportamento ou tendência diferente de períodos anteriores e subsequentes - como momentos de alta volatilidade - que destoam do comportamento apresentado até então. Esses períodos podem estar relacionados com momentos de crise ou incertezas, e detectá-los e entendê-los é importante para guiar o comportamento de um investidor, e assim obter bons desempenhos em investimentos. Existem alguns trabalhos que concentram-se em detectar e predizer esses regimes, como CHEN; TSANG, 2018 e ZHU, 2020, e assim obter uma camada de informação à mais para decisões de investimentos. Utilizando um Modelo Oculto de Markov (GAGNIUC, 2017) para determinar regimes

de mercado através dos valores de fechamento do ativo SPY, podemos observar os diferentes períodos presentes na base de dados de teste, como na Figura 18 (ANGELIS; PAAS, 2013). Na figura, estão representados dois regimes distintos, obtidos pelo modelo. O regime identificado por *Regime 0* consiste em um comportamento mais "estável"do mercado, associado a períodos de baixa volatilidade. O regime identificado como *Regime 1*, por sua vez, consiste em períodos de alta volatilidade do mercado, alinhado com momentos de crise.

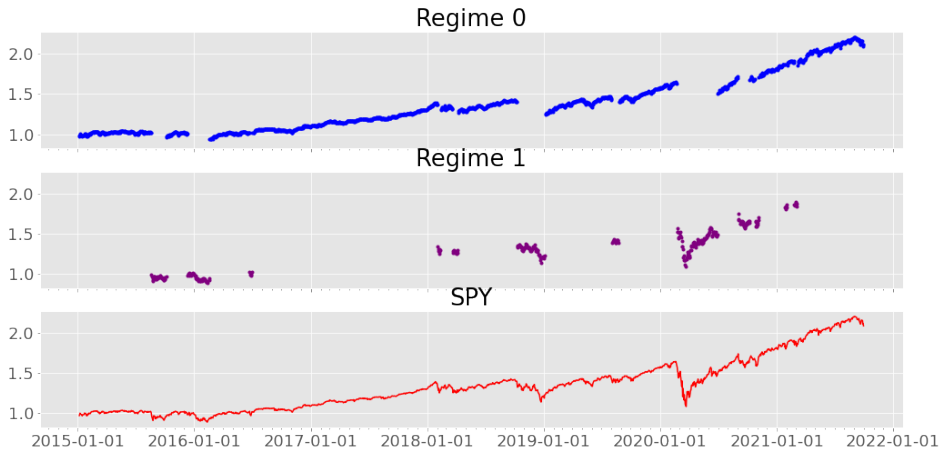


Figura 18 – Regimes de Mercado detectados a partir do ativo SPY.

É esperado que um bom investidor consiga mapear a composição de ativos com maior probabilidade de ganho (ou não perda) a cada janela de tempo (out-of-sample), mantendo uma carteira adequada para diferentes regimes de mercado. Essa premissa ajuda a alinhar as expectativas sobre o que desejamos de desempenho dos algoritmos testados.

5.2 Modelagem do Problema

A resolução do problema foi feita de forma incremental, onde o raciocínio era aplicar abordagens simples para depois avaliar o desempenho com implementações mais complexas. A proposta é comparar a aplicação de algoritmos com

técnicas mais usuais, como Markowitz, e observar como técnicas de aprendizado por reforço se comportam neste tipo de problema.

Como baseline, consideramos a ETF SPY. A proposta do agente é fornecer uma configuração de portfólio que seja a mais indicada para cada momento (estado).

Se modelássemos isso como uma saída discreta, digamos, de possíveis configurações que o peso de um determinado ativo poderia assumir, acabaríamos com um universo de possibilidades muito grande para lidar. Por exemplo, digamos que o peso de um ativo pudesse assumir 11 valores possíveis,

$$\boldsymbol{W} = [0, 0.1, 0.2, 0.3, \ldots, 0.9, 1.0]. \tag{5.1}$$

Neste tipo de situação, considerando três ativos, as ações possíveis que um agente poderia empregar seriam

$$A = \{(w_1, w_2, w_3) | w_1, w_2, w_3 \in \boldsymbol{W} \wedge \sum_{i=1}^{3} w_i = 1\}. \tag{5.2}$$

Se cada peso pode assumir 11 valores, e respeitando a restrição de $\sum_{i=1}^{3} w_i = 1$ isso dá um total de 186 ações discretas diferentes possíveis, para apenas 3 ativos. Com um espaço de ações definido desta forma, fica inviável escalar o problema. Afinal, abordagens estocásticas - que considerassem que todas as ações poderiam possuir simultaneamente uma probabilidade não nula de serem escolhidas - poderiam culminar em um modelo mais complexo, com maiores tempos de processamento e utilização de recursos computacionais. Veja, para um modelo de política estocástico conseguir tomar uma decisão, este teria que conhecer a probabilidade de todas as 1331 ações possíveis. Além de aumentar a complexidade de uma rede para determinar essas probabilidades devido ao tamanho do espaço de ações, diversas ações teriam probabilidades próximas o suficiente para gerarem problemas de convergência dos resultados. Isso vale até para abordagens onde apenas as funções de valor são aproximadas, como *Deep Q-Learning*, onde mesmo uma política ϵ-greedy ainda necessita conhecer as probabilidades de todas as ações.

Uma abordagem mais óbvia e simples do que esta é tratar o espaço de ações como contínuo, e utilizar abordagens com políticas determinísticas para obter os pesos dos ativos a cada estado. Assim, de maneira similar à empregada na literatura revisada na seção 4.1, iremos empregar arquiteturas que possuam um modelo cuja saída seja um vetor de números contínuos, sendo um item do vetor para cada ativo da carteira.

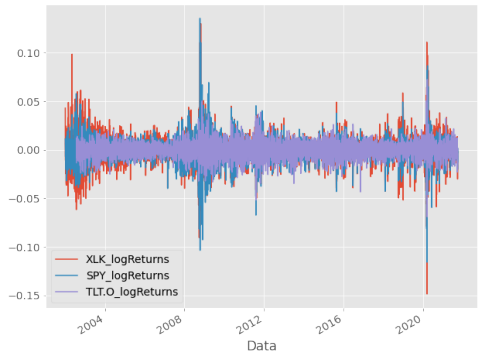
A princípio foram empregados algoritmos simples de aprendizado por reforço, usando modelos igualmente simples. Como *features*, foram usados valores obtidos a partir dos dados temporais de retorno dos ativos:

- Valores de retorno logarítmico dos ativos considerando o valor do dia atual e do dia anterior (fig. 19a);
- Valores da volatilidade anualizada de cada ativo, considerando uma janela deslizante dos últimos 66 dias, para cada dia do período (fig. 19b);
- Valores de curtose para cada ativo, e para o índice VIX[2], considerando uma janela deslizante dos últimos 125 dias (fig. 19c);
- Informação de momento probabilístico[3], aplicado entre pares de ativos, considerando a janela deslizante dos últimos 60 dias (fig. 19d).

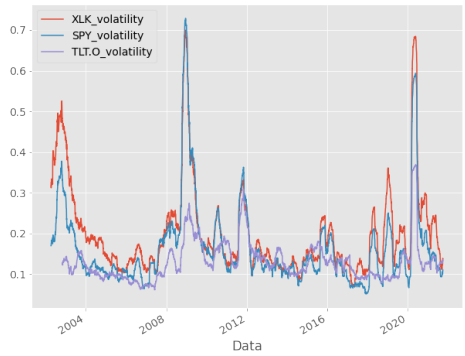
A decisão do agente será diária, uma vez que assim a quantidade de amostras disponíveis tanto para o treinamento do agente, quanto para as tomadas de decisão, é maior. Porém, a variabilidade de preço dentro de um período da série temporal com valores diários é maior do que caso fossem utilizadas agregações semanais, portanto o agente também deverá ser capaz de lidar com este comportamento volátil do ambiente. O conjunto de dados utilizado para treinamento compreende o período de tempo que vai de 29 de setembro de 2002 até 04 de janeiro de 2015, enquanto o utilizado para *backtest* vai de 05 de janeiro de 2015 a 30 de setembro de 2021. Nesta implementação, é considerado um problema episódico,

[2] CBOE Volatility Index, é uma estimativa da volatilidade do índice S&P500 considerando a janela dos próximos 30 dias (INDICES, 2022).

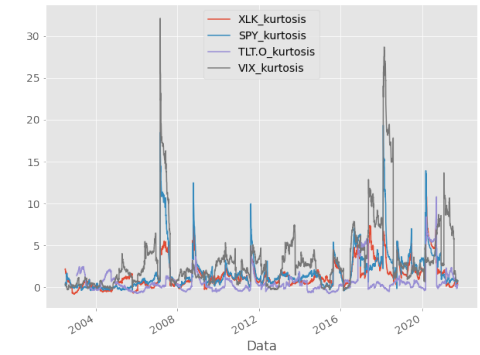
[3] (MURPHY, 1999)



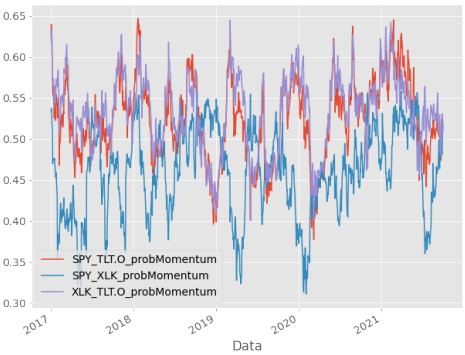
(a) Retornos logarítmicos dos ativos.



(b) Volatilidade dos ativos.



(c) Curtoses dos ativos.



(d) Momentos probabilísticos.

Figura 19 – Evolução dos ativos no período in-sample.

onde o mesmo consiste de 100 dias de dados, e a recompensa é calculada ao final desse tempo como o valor logarítmico do portfólio após esse período, normalizada pela duração do episódio, como realizada nos artigos avaliados no último capítulo, e visto na equação 4.17,

$$R(\boldsymbol{s}_1, \boldsymbol{a}_1, \ldots, \boldsymbol{s}_{t_f}, \boldsymbol{a}_{t_f}, \boldsymbol{s}_{t_f+1}) := \frac{1}{t_f} \ln \frac{p_f}{p_0} = \frac{1}{t_f} \sum_{t=1}^{t_f+1} \ln(\boldsymbol{y}_t \cdot \boldsymbol{w}_{t-1}) = \frac{1}{t_f} \sum_{t=1}^{t_f} r_t,$$

com a diferença de que, desta vez, não foram considerados os custos de transação ($\mu = 0$). Ou seja, o valor implícito de cada estado é atualizado retroativamente

após o final do episódio, e não a cada passagem de tempo. O objetivo do agente é obter a combinação de parâmetros para a função da política que melhor apresente um resultado para o portfólio em questão ao final dos 100 dias.

Para começar a abordagem do problema, optamos por testar duas metodologias de aprendizagem, uma baseada em *Gradiente de Política Profundo* - DPG [4], e outra em Ator-Crítico, sendo DPG um algoritmo mais simples - onde existe apenas um modelo para a política, que procura fornecer as ações diretamente a partir das *features* de estado - e o Ator-Crítico uma abordagem com a utilização de outro modelo para as funções de valor. Infelizmente, os resultados com o algoritmo Ator-Crítico não foram satisfatórios, de forma que, na nossa avaliação, este modelo não foi capaz de aprender o comportamento que esperávamos para o agente. Optamos então por seguir com o DPG, e apresentar os resultados do Ator-Crítico mais ao final deste Capítulo.

As implementações ocorreram de forma incremental, e seguiram a seguinte ordem:

- Modelo atuando em uma representação do ambiente que não utiliza *features* temporais;
- Modelo atuando em uma representação do ambiente utilizando *features* temporais;
- Utilização de métricas de risco para modelagem da recompensa;
- Alteração da estrutura do modelo utilizado.

Outras execuções e experimentos foram realizados, e seus resultados podem ser encontrados no repositório do código no Github <http://github.com/victormmp/rl_mestrado>. No repositório podem ser encontrados experimentos com diferentes combinações de hiperparâmetros.

[4] Deep Policy Gradient.

5.2.1 Aprendizado utilizando *features* não-temporais

No primeiro teste realizado, os agentes não lidaram com análises de séries temporais, trabalhando apenas com os valores mais recentes de cada *feature*, ao invés de receber uma série histórica de valores. Por isso, uma rede neural simples, profunda e densa foi usada como aproximador da política. A intenção é ver como o agente se comporta com informações de apenas uma dimensão nos estados, evitando usar algoritmos mais complexos logo no começo, como CNNs ou LSTMs, que consideram uma sequência na entrada ao invés de um simples valor. Assim, a alocação de pesos na carteira de cada dia será considerada utilizando as *features* do dia anterior. É importante garantir que nenhuma informação do dia corrente seja passada ao modelo até que o mesmo faça a escolha dos pesos para o próximo dia. Uma abordagem para este problema pode ser iterar na lista de dias, em sequência. O valor da recompensa do dia é calculado a partir dos valores atuais dos ativos, e dos pesos calculados no dia anterior, seguindo a fórmula da equação 4.17. O algoritmo 22 contém o pseudocódigo desta implementação.

Algoritmo 22 Pseudocódigo para implementação do algoritmo DPG Simples.

Inicializa modelo π_θ para a política, com parâmetros θ.
Define taxa de aprendizado para o aprendizado do modelo π.
Inicializa número de épocas de treinamento.
for *cada época* **do**
    Inicializa um episódio $s_1, s_2, ..., s_{t_f}$
    Obtém o estado inicial, correspondente ao primeiro dia, $S \leftarrow s_1$
    Inicializa valor do portfólio $p \leftarrow 0$
    $t_f \leftarrow$ duração do episódio em dias.
    Obtém uma alocação de pesos w inicial (pode ser igualmente distribuído).
    for *cada dia* s_t *do episódio, sendo* $t = 2, ..., t_f$ **do**
        Obtém os retornos y dos ativos a partir dos retornos logarítmicos contidos em S.
        $p \leftarrow p + \ln(y \cdot w)$
        $S \leftarrow s_t$
        Obtém novo vetor de pesos w a partir de S.
    end for
    Calcula retorno do episódio $R \leftarrow \frac{p}{t_f}$
    Corrige os parâmetros θ do modelo da política buscando maximizar R.
end for

O agente continha uma rede neural de 3 camadas, com 64 e 32 neurônios, respectivamente, além da camada de saída. Todos os neurônios utilizando Leaky-ReLU como função de ativação (MAAS; HANNUN; NG, 2013). O esquema pode ser visto na Figura 20. Realizando uma série de treinamentos com diferentes valores de *taxa de aprendizado* (*learning rate*), obtemos os resultados da Figura 21.

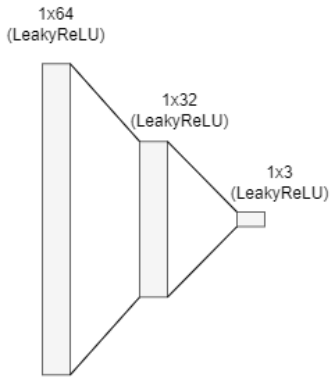


Figura 20 – Esquemático da rede DPG utilizada.

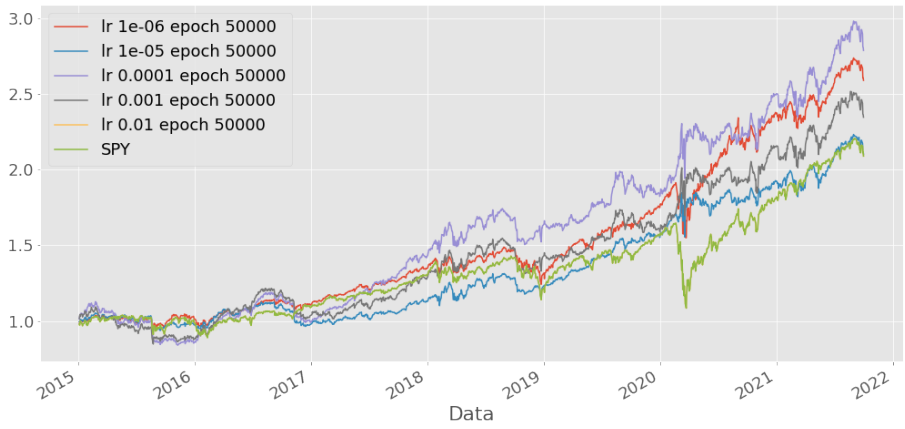


Figura 21 – Resultado do treino com DPG simples. O modelo com taxa de aprendizado de 0.01 teve o mesmo desempenho do ativo SPY.

Observando a evolução do valor do portfólio, vemos que para alguns valores de taxa de aprendizado o resultado foi melhor do que o SPY. Além disso, durante os períodos de crise (como na Figura 18) o portfólio apresentou uma queda menor

do que cada ativo separadamente (como pode ser visto na Figura 22). Quando observamos a alocação de pesos realizada pelo agente (Figura 23), vemos que o mesmo havia feito a troca para o ativo de proteção durante esses períodos, apesar de a mesma ter sido feita ao longo de um período maior do que o de queda dos índices.

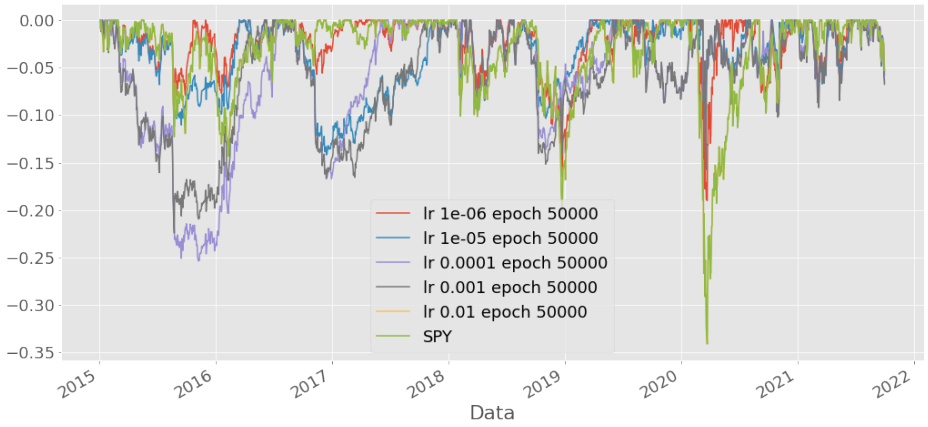


Figura 22 – Curvas de drawdown para o modelo DPG simples.

5.2.2 Aprendizado utilizando *features* temporais

Os testes realizados até o momento levavam em consideração que as *features* coletadas conseguiam trazer informação em apenas uma dimensão do comportamento temporal dos valores dos ativos. O agente em si não tinha visibilidade da evolução dos preços, diretamente.

A fim de possibilitar ao agente acesso direto às informações de valor dos ativos temporalmente, o conjunto de dados foi alterado. Agora, as *features* utilizadas até o momento trariam seus valores históricos no estado. Isto é, onde antes tínhamos os valores de retorno logarítmico, volatilidade, curtose, etc., calculados para o dia corrente, agora temos ainda os valores históricos dessas variáveis para uma janela de dias anteriores em relação ao dia atual. A hipótese por trás desta alteração no conjunto de dados é a de que ao possibilitarmos ao agente ter visibilidade da evolução temporal desses valores, ele será mais apto a identificar as

(a) Learning rate de 1e-06.

(b) Learning rate de 1e-05.

(c) Learning rate de 0.0001.

(d) Learning rate de 0.001.

Figura 23 – Evolução dos pesos para o modelo DPG simples com diferentes *learning rates*.

correlações e padrões existentes entre os ativos, e conseguirá tomar decisões de maneira mais assertiva. A janela temporal consta, portanto dos últimos 60 dias de dados de cada série temporal.

Os primeiros testes com as *features* temporais foram feitos utilizando um algoritmo DPG, porém agora utilizando uma rede convolucional para o ator, tomando por inspiração a rede utilizada por JIANG; XU; LIANG, 2017. Assim como no artigo citado, foram usadas duas camadas convolucionais, usando filtros de uma unidade na dimensão das *features*, o que significava um filtro com visão apenas de

cada *feature* por vez. Além disso, como cada *feature* é independente das outras, e a ordem em que as mesmas se organizam no conjunto de dados não é importante, um filtro configurado desta forma parece bem indicado. O esquemático da rede pode ser visto na Figura 24. Diferentes variações do modelo foram treinadas, e os resultados podem ser vistos a seguir.

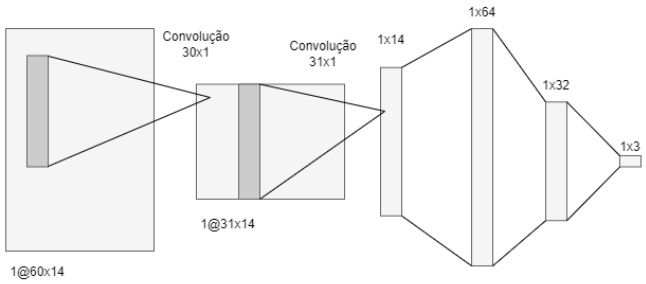


Figura 24 – Rede neural utilizada para o ator, levando em conta *features* temporais.

A Figura 25 mostra o resultado para o modelo quando o mesmo inclui uma camada de normalização em batch após cada camada de convolução. A rede foi treinada com 50.000 épocas, e com diferentes valores de taxa de aprendizado. Já a Figura 26 traz as curvas de drawdown para as mesmas.

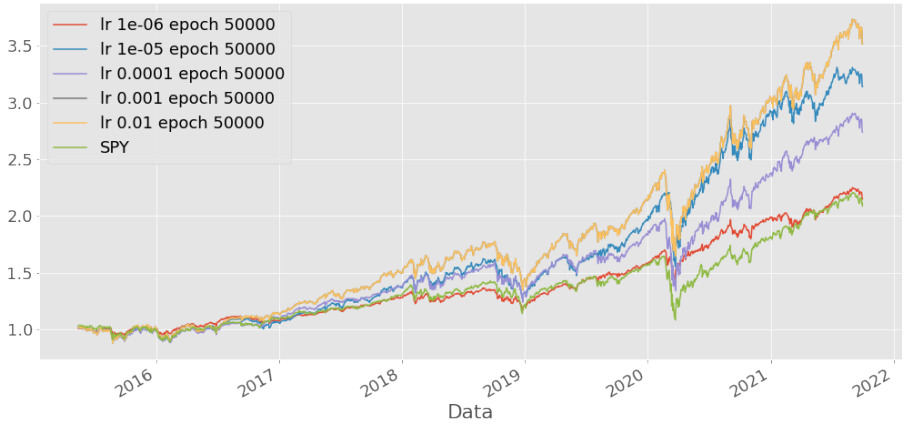


Figura 25 – Resultado para algoritmo DPG com camadas de normalização em batch.

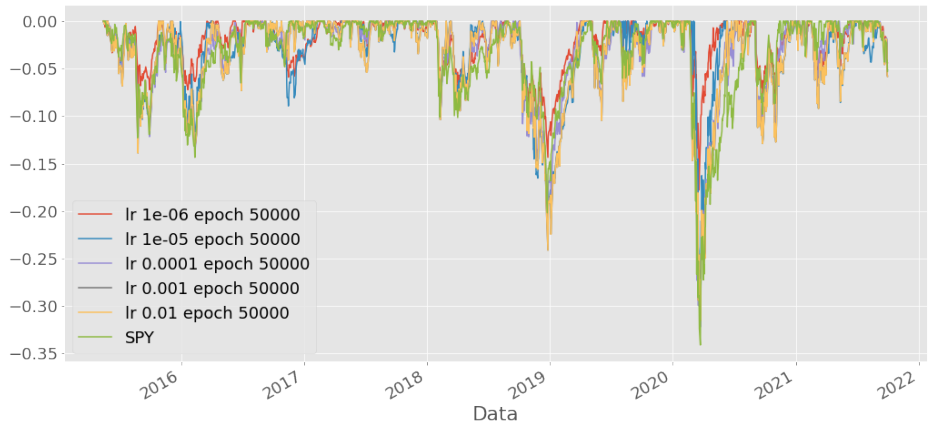
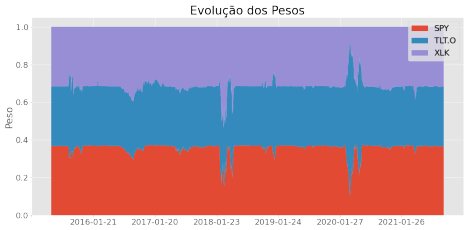
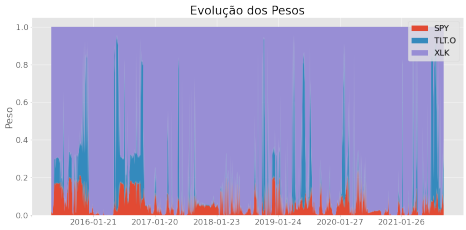


Figura 26 – Curvas de drawdown para o algoritmo DPG com camadas de normalização em batch.

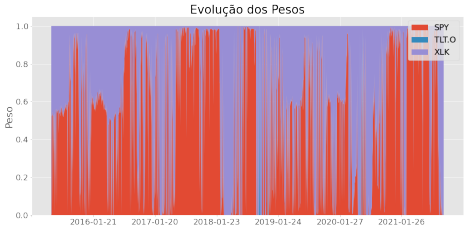
A evolução da alocação de pesos a cada instante de tempo, por sua vez, pode ser vista nas Figuras 27a, 27b, 27c e 27d, para cada valor de aprendizado.



(a) Learning rate de 1e-06.



(b) Learning rate de 1e-05.



(c) Learning rate de 0.0001.

(d) Learning rate de 0.001.

Figura 27 – Evolução dos pesos para o modelo DPG usando normalização em batch com diferentes *learning rates*.

É possível notar que para um valor mais baixo de *learning rate*, considerando o mesmo período de treinamento, o agente fez a alocação de maneira igualitária entre os ativos. Enquanto isso, para um valor mais alto, o mesmo acabou por concentrar sua alocação um apenas um ativo - neste caso, no ativo com maior retorno a longo prazo. Como poderá ser visto nos próximos resultados, estas tendências irão repetir-se mais vezes.

Foi testado uma variação da implementação deste modelo sem as camadas de normalização em batch. Porém, o resultado obtido foi inferior, e por isso desconsiderado.

5.2.3 Utilizando métricas de risco como recompensa

Os testes realizados até aqui utilizavam o retorno cumulativo do portfólio após um período de dias como métrica a ser otimizada pelo modelo. Porém, como visto pela evolução dos pesos, havia uma tendência do modelo em investir com maior foco no ativo de maior retorno a longo prazo, o XLK. Isso indica que o modelo acaba por aceitar alguns resultados negativos em alguns períodos com a promessa de que o retorno ao final será maior dessa forma. O comportamento desejado, porém, é de que o modelo consiga identificar períodos em que o ativo com maior retorno não vá apresentar um resultado tão bom, e consiga trocar para algum outro que apresente um resultado melhor, mais notoriamente durante as grandes quedas ocorridas ao final de 2018 e 2020. Em outras palavras, queremos um modelo que se arrisque menos do que o que vêm sendo feito, e opte por caminhos mais seguros, se isso significar proteção durante períodos de crise. Assim, foi feito um novo teste utilizando a métrica *Sharpe ratio* (ver seção 3.3.1) como objetivo de otimização.

A Figura 28 mostra os resultados para o teste com o modelo DPG, utilizando normalização em batch e sharpe ratio como métrica de recompensa do episódio.

Os gráficos da Figura 29 mostram como o agente fez a alocação dos pesos dos ativos para diferentes valores de *learning rate*.

A utilização do índice como recompensa levou o agente a fazer uma alocação

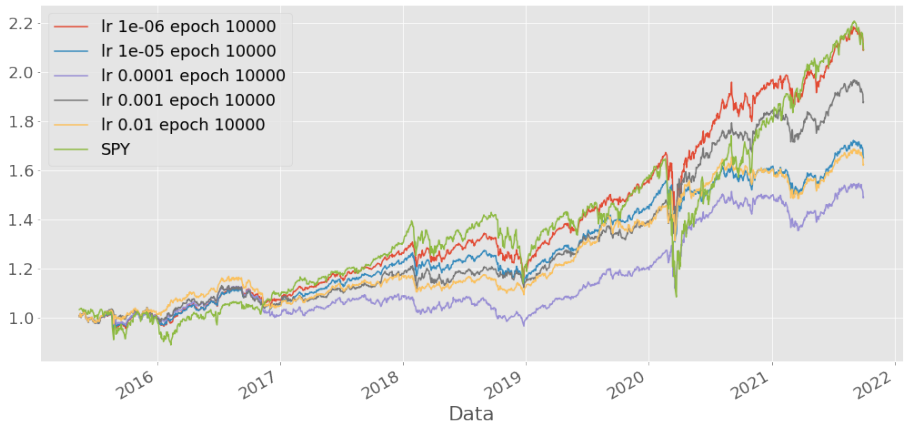
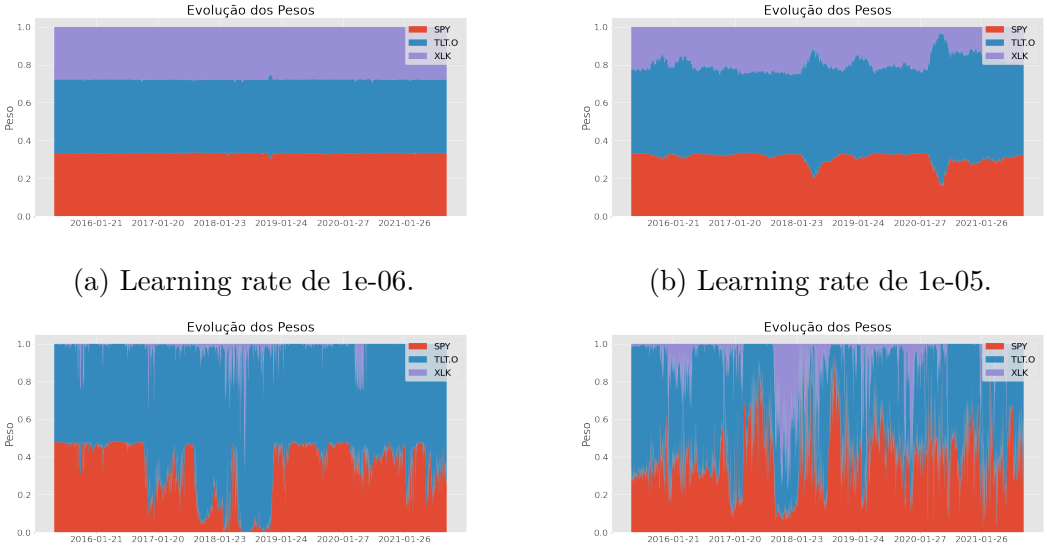


Figura 28 – Resultado para algoritmo DPG utilizando Sharpe ratio como recompensa do episódio.



(a) Learning rate de 1e-06.

(b) Learning rate de 1e-05.

(c) Learning rate de 0.0001.

(d) Learning rate de 0.001.

Figura 29 – Evolução dos pesos para o modelo DPG usando sharpe ratio como recompensa, para diferentes *learning rates*.

mais igualitária entre os ativos, mas ao contrário do que vinha sendo visto nos testes anteriores, agora a tendência é alocar maior peso ao ativo de proteção, para *learning rates* mais altos. Considerando que essa métrica é maior quanto melhor

for a relação média - variância (retorno/risco) dos retornos, esse comportamento faz sentido. Além disso, um maior foco no ativo de proteção trouxe uma queda menos significativa durante os períodos de crise, como por exemplo 2020.

Essa abordagem aponta um caminho interessante, uma vez que utilizar métricas que levam em conta o risco parece ser uma boa solução. Apesar disso, pelo fato do *Sharpe* considerar toda e qualquer variância prejudicial, inclusive quando ela é positiva, ele pode penalizar demais ativos que apresentam um bom crescimento. Assim, talvez métricas que foquem mais em minimizar *quedas* de retorno, como o *Sortino ratio* podem ser opções boas de serem utilizadas.

As próximas execuções procuraram explorar melhor o comportamento de diferentes métricas de risco sobre as decisões do agente. Assim, além da métrica Sharpe, foram testados agentes usando Sortino ratio (ver capítulo 3.3.2) como recompensa. Considerando que esta métrica também apresenta valores mais baixos apenas para retornos negativos (ou abaixo do valor de retorno de um ativo livre de risco), é interessante avaliar se o agente conseguirá aprender a tendência de alta de determinados ativos, e irá decidir alocar nos mesmos, mesmo que sua variância seja mais alta, como por exemplo é o caso do ativo XLK.

A Figura 30 mostra os resultados para os primeiros testes comparativos, para diferentes *learning rates*. A arquitetura da rede de convolução é a mesma utilizada anteriormente, e as execuções aconteceram com 10.000 épocas.

Na Figura31 são mostradas as curvas de drawdown para os algoritmos testados.

As Figuras 32 e 33 mostram as alocações feitas pelo agente no *backtest*, quando utilizando Sharpe e Sortino para o cálculo da recompensa do episódio, respectivamente.

Em termos gerais, o desempenho para ambas as métricas foi similar. Para valores mais baixos de *learning rate*, ambos fizeram alocações mais equilibradas entres os três ativos, e realizaram maiores trocas à medida que a taxa de aprendizado se elevava.

Considerando os resultados vistos até o momento, vimos que utilizar métri-

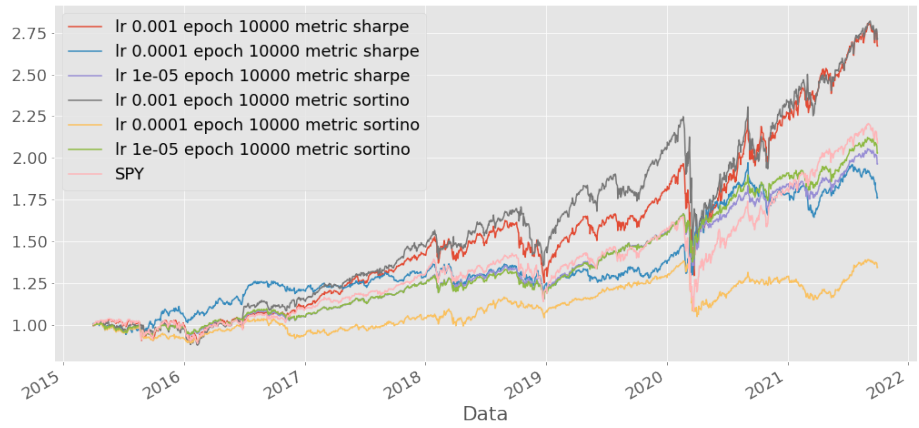


Figura 30 – Resultado para testes comparativos do algoritmo DPG com diferentes métricas para a recompensa, e com diferentes *learning rates*.

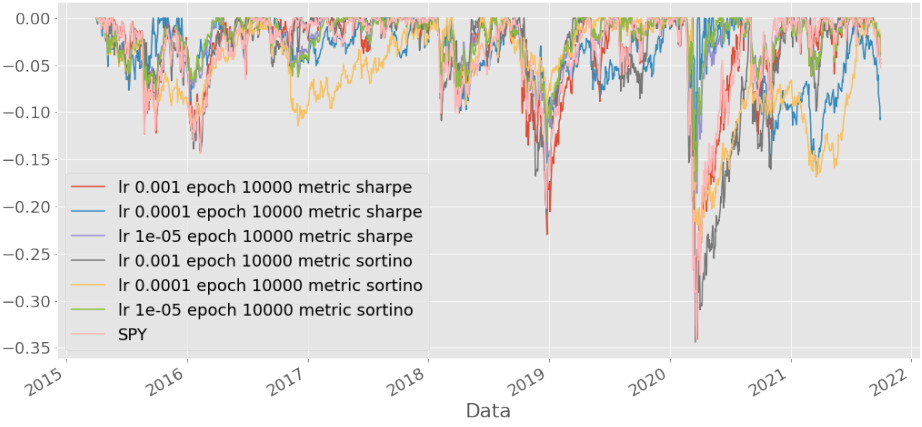
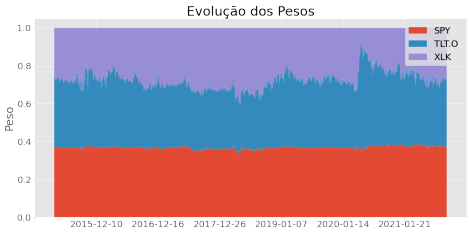
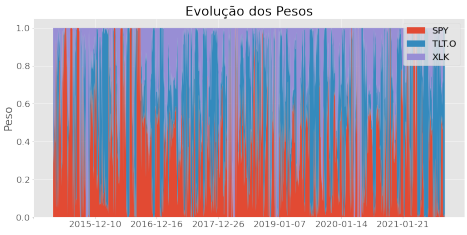


Figura 31 – Curvas de drawdown para o algoritmo DPG com diferentes métricas para a recompensa, e com diferentes *learning rates*.

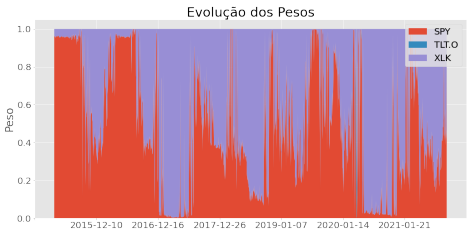
cas de risco como recompensa para episódios fez com que o agente tomasse melhores decisões em termos de proteção da carteira durante momentos de crise. Não foi notada nenhuma diferença evidente entre utilizar Sharpe ou Sortino, entretanto.



(a) Learning rate de 1e-05.

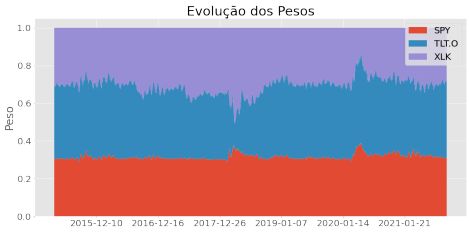


(b) Learning rate de 0.0001.

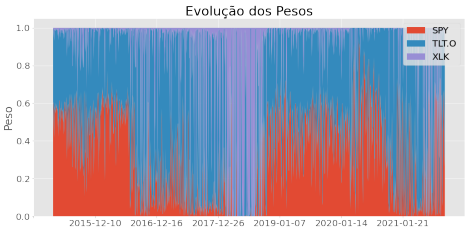


(c) Learning rate de 0.001.

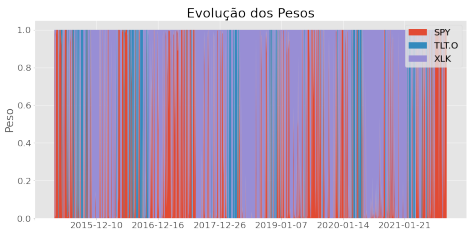
Figura 32 – Evolução dos pesos para o modelo DPG utilizando Sharpe como métrica da recompensa.



(a) Learning rate de 1e-05.



(b) Learning rate de 0.0001.



(c) Learning rate de 0.001.

Figura 33 – Evolução dos pesos para o modelo DPG utilizando Sortino como métrica da recompensa.

5.2.4 Alterando a estrutura do modelo

Nesta etapa de testes, adicionou-se como variável de entrada a média móvel dos valores de retorno, para uma janela dos últimos 66 dias, como *features* no conjunto de dados, e avaliando a resposta do agente a isso. A expectativa é que, com essa informação, o agente possa fazer uma melhor alocação focando em retorno também, e balancear com a mudança de comportamento que foi inserida ao utilizar as métricas de risco, onde o modelo passou a prezar a segurança.

Além disso, uma nova estrutura de rede foi proposta. Até o momento, a definição do estado utilizada era de usar valores temporais para *todas as features obtidas*, passando uma janela fixa de dias históricos, a partir das métricas do dia anterior. Porém, apenas as *features* de retorno logarítmico são "amostras brutas"de cada dia, como medições diárias do comportamento dos ativos. Por outro lado, as outras *features* incluídas já eram, de certa forma, análises temporais dos ativos, obtidos a partir da análise de uma janela deslizante também dos dias anteriores a cada medição - como descrito no começo do capítulo. Assim, existe a possibilidade de que uma convolução nestas *features* poderia não trazer qualquer benefício, uma vez que já são avaliações de dados temporais. Esse fato ganha destaque também ao lembrarmos que a arquitetura da rede prevê uma convolução sendo realizada por um filtro de 1 dimensão, realizando a convolução de cada *feature* de forma isolada, e sendo compartilhado por todas as *features* do estado. Assim, considerando que o filtro da camada de convolução da rede CNN é compartilhado, e vai tentar ajustar seu valor para obter o melhor desempenho no processamento de cada *feature*, pode ser que este valor calculado não seja o ideal para extrair melhores informações de cada coluna da matriz de *features* de entrada. Tomando por exemplo o filtro da primeira camada de convolução, segundo o esquemático da Figura 24. Uma vez que, o filtro é composto por uma matriz de uma coluna apenas e 30 linhas, é feita uma convolução em cada *feature* (coluna) separadamente, considerando uma janela de 30 dias. O valor dos pesos utilizado nessa convolução vai ser ajustado para que, ao final, as convoluções em todas as *features* levem ao melhor resultado. Uma vez que as informações das variáveis que não são as de retorno logarítmico já são calculadas a partir de janelas de tempo, o valor dos pesos do filtro pode estar tentando ajustar-se para *features* com comportamentos muito diferentes, e por isso chegando em um

valor que entregaria um resultado final sub-ótimo, se comparado com o caso onde a convolução é realizada apenas nas *features* de retorno logarítmico.

Assim, a modificação realizada na rede do ator teve como objetivo fazer com que apenas os retorno logarítmicos necessitem de uma janela temporal de *features* como entrada, enquanto as outras informações trarão apenas o valor calculado para o dia corrente, para cada dia do episódio. Dessa forma, a convolução acontece apenas nas primeiras, enquanto as outras serão incluídas pouco antes das camadas densas. A Figura 34 ilustra essa nova arquitetura, enquanto a Figura 35 mostra o resultado obtido para esta arquitetura de rede.

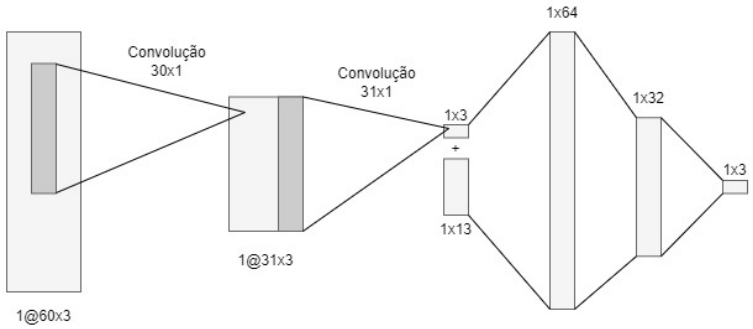


Figura 34 – Arquitetura alterada da rede do ator. Para as convoluções, a entrada consiste de uma matriz com 60 dias de dados históricos para cada um dos 3 ativos da carteira. Antes do dado passar para a camada densa, são adicionadas as demais *features*.

Na Figura 36 é possível ver como foi o drawdown para os algoritmos avaliados.

As Figuras 37 e 38 mostram como foi a alocação de pesos para as métricas Sharpe e Sortino, respectivamente.

Já é possível notar pelas alocações que o modelo fez que, comparado com os testes das seções anteriores, houve maiores períodos de tempo com maior prevalência de alocação em um dos ativos. Também foi possível notar que os agentes conseguiram um controle muito maior do valor do portfólio durante os períodos de crise, onde quase não é possível ver alguma queda, acompanhando a evolução do valor do portfólio pela Figura 35.

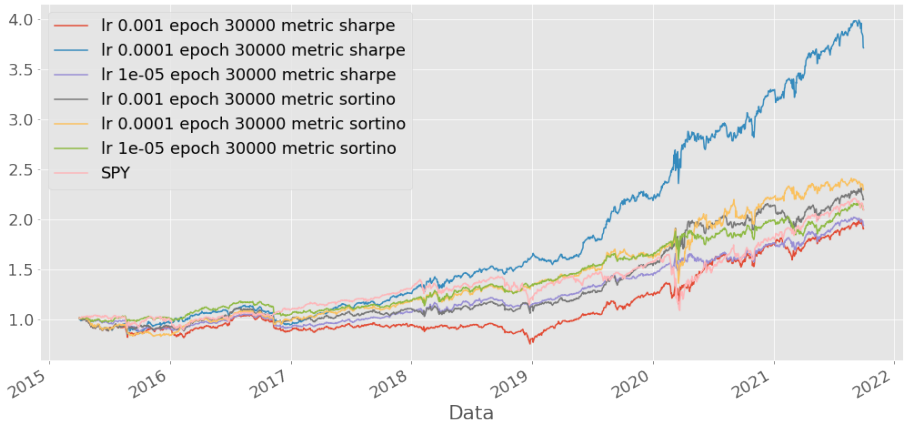


Figura 35 – Resultado para algoritmo DPG com nova arquitetura para rede do ator.

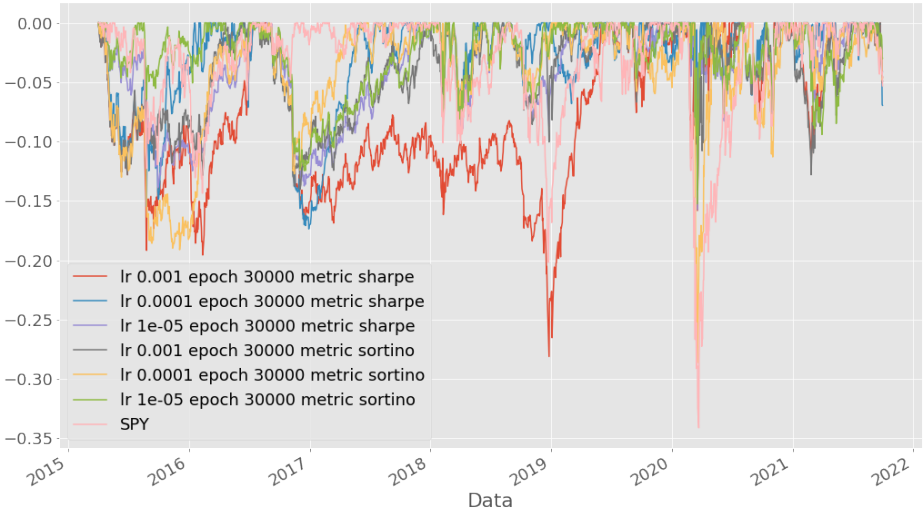
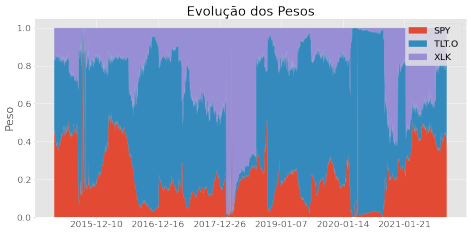
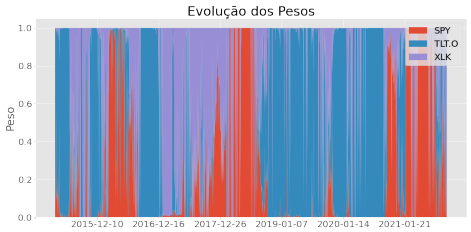


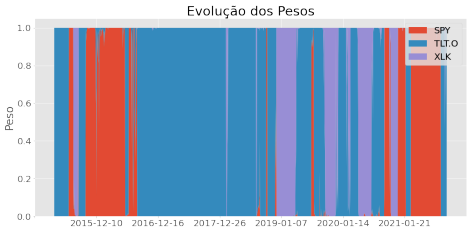
Figura 36 – Curvas de drawdown para algoritmo DPG com nova arquitetura para rede do ator.



(a) Learning rate de 1e-05.

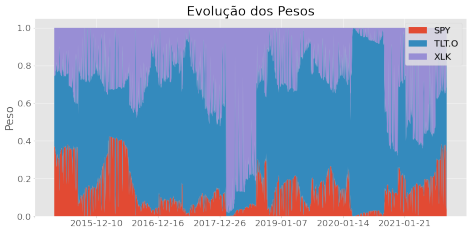


(b) Learning rate de 1e-04.

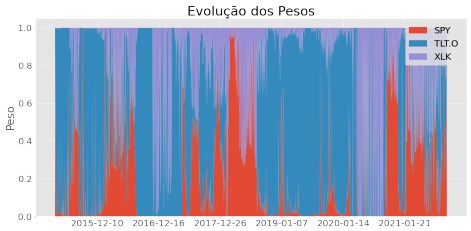


(c) Learning rate de 1e-03.

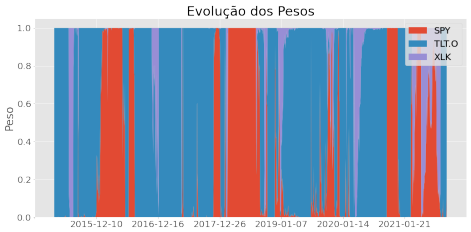
Figura 37 – Evolução dos pesos para o modelo DPG utilizando Sharpe como métrica de recompensa para nova arquitetura da rede do ator.



(a) Learning rate de 1e-05.



(b) Learning rate de 1e-04.



(c) Learning rate de 1e-03.

Figura 38 – Evolução dos pesos para o modelo DPG utilizando Sortino como métrica de recompensa para nova arquitetura da rede do ator.

5.2.5 Implementação com Ator-Crítico

Conforme observamos na Figura 21, os modelos implementados apenas com uma função para aproximar a política (ator) já apresentam um desempenho melhor do que o SPY. A ideia deste teste é usar um crítico é acelerar o aprendizado do agente, além de estabilizar o processo de treinamento e ter um melhor aproveitamento das amostras (SUTTON; BARTO, 2018; LILLICRAP et al., 2016).

Uma rede neural simples, de mesma arquitetura da empregada para o ator, foi implementada para o crítico. O aprendizado da função de valor se deu utilizando SARSA, em um aprendizado on-policy online determinístico. Assim como David Silver explica em seu artigo *Deterministic Policy Gradient Algorithms* (SILVER et al., 2014), os parâmetros $\boldsymbol{w}$ do ator e $\boldsymbol{\theta}$ do crítico são atualizados conforme as seguintes equações,

$$\delta_t = r_t + \gamma Q^w(s_{t+1}, a_{t+1}) - Q^w(s_t, a_t) \tag{5.3}$$

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha_w \delta_t \nabla_w Q^w(s_t, a_t) \tag{5.4}$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha_\theta \nabla_\theta \mu_\theta(s_t) \nabla_a Q^w(s_t, a_t)|_{a=\mu_\theta(s)}, \tag{5.5}$$

sendo r_t a recompensa imediata calculada no tempo t, γ a taxa de desconto, Q^w a função de valor de ação de acordo com os parâmetros $\boldsymbol{w}$, α_w a taxa de aprendizado da função do crítico ($\alpha_w \in (0, 1]$), α_θ a taxa de aprendizado da função da política, e μ_θ a função da política determinística, de acordo com os parâmetros $\boldsymbol{\theta}$.

Três testes foram realizados verificando como diferentes parâmetros influenciam no desempenho do algoritmo. A Figura 39 mostra o desempenho para diferentes taxas de aprendizado, a Figura 40 mostra para diferentes durações de épocas, e a Figura 41 para diferentes valores de γ, o que interfere o quanto de informações passadas são consideradas para atualizar os parâmetros a cada tempo t.

Comparando os desempenhos nas Figuras 39, 40 e 41 com o comportamento dos ativos sozinhos na Figura 17, vemos que o agente tendeu a seguir o resultado de apenas um dos ativos a cada teste, o que se reflete nos desempenhos dos portfólios serem idênticos ao de alguma ação. Seguir apenas um ativo não é um compor-

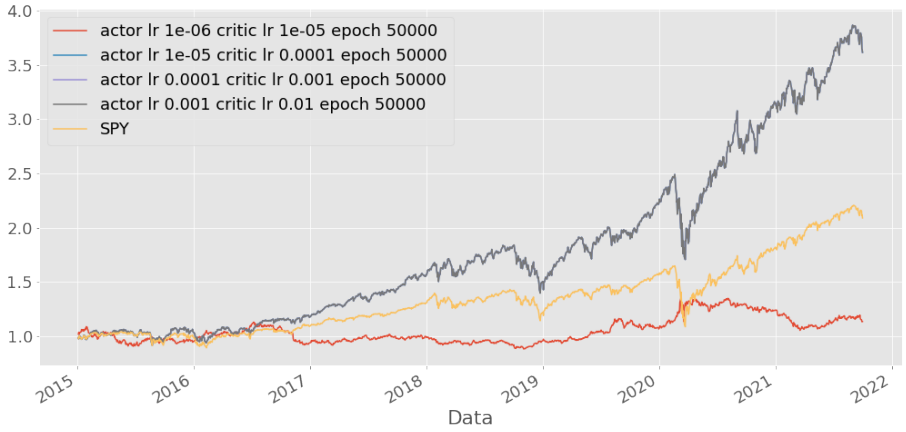


Figura 39 – Desempenho de um modelo ator-crítico para diferentes *learning rates*. Os modelos com taxa de aprendizado do ator de 1e-05, 1e-04 e 1e-03 tiveram o mesmo resultado, alocando o peso totalmente para o ativo XLK.

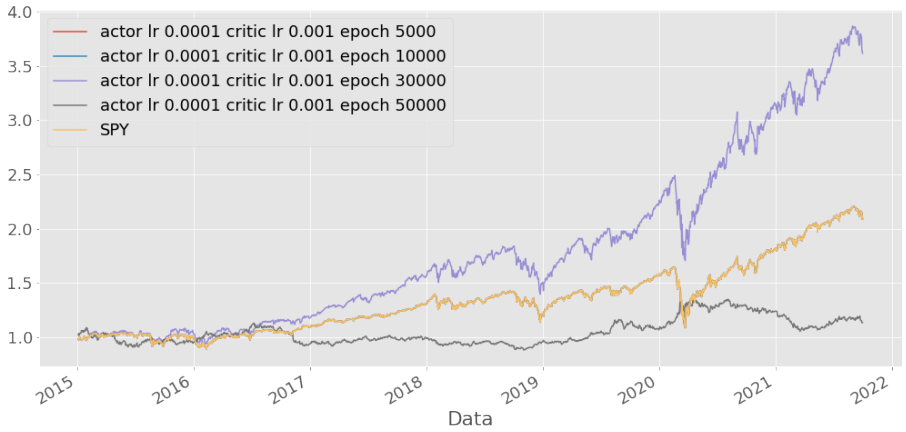


Figura 40 – Desempenho de um modelo ator-crítico para diferentes épocas. Os modelos com 5000 e 10000 épocas de aprendizado obtiveram o mesmo resultado, alocando todo o peso para o ativo SPY.

tamento que desejamos do agente. Esta implementação, mesmo as variações dos parâmetros, não obteve uma alocação esperada, diversificando o investimento ao longo do tempo.

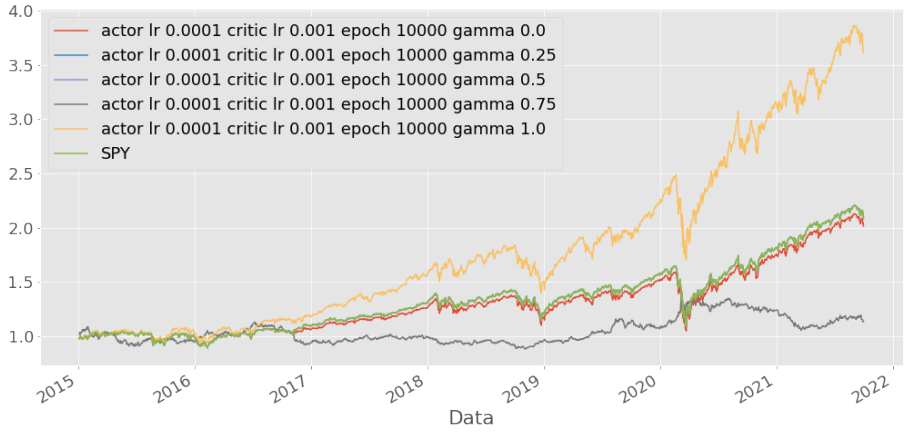


Figura 41 – Desempenho de um modelo ator-crítico para diferentes valores de γ. Os modelos com γ igual a 0.25 e 0.5 obtiveram o mesmo resultado, alocando todo o peso no ativo SPY.

Algumas hipóteses podem ser levantadas sobre este resultado. Uma delas é de que o modelo não conseguiu aprender a tendência de crescimento de ativos com maiores valores de retorno, como o XLK, e não conseguiu aprender a correlação negativa entre ele e o ativo de proteção. Uma das causas é de que o modelo pode não ser complexo o suficiente para conseguir extrair esses padrões das *features*, ou então as *features* não trazem informação suficiente para o agente conseguir aprender. Antes de realizar alterações nas *features*, existem algumas modificações no próprio algoritmo que podem ser testadas.

A fim de testar a primeira hipótese, o próximo treinamento realizado utilizou uma rede mais complexa para o crítico. A rede agora teria 4 camadas, uma com 128 neurônios, seguida por uma com 64, outra com 32 e a camada de saída. O resultado do treinamento pode ser visto na Figura 42.

Aumentar um pouco a complexidade do modelo não trouxe um resultado diferente dos observados anteriormente, porém.

Outras modificações na rotina foram abordadas, a fim de verificar o efeito no resultado final do agente. As Figuras 43 e 44 acompanham o efeito de adicionar *planning* à rotina, para diferentes valores de γ e para diferentes valores de *learning*

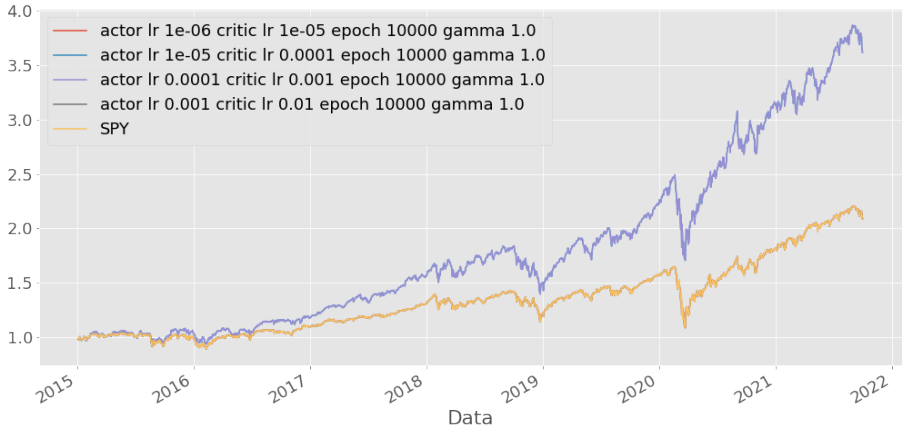


Figura 42 – Desempenho do Ator-Crítico om uma rede mais complexa para o crítico. Os modelos com taxa de aprendizado do ator de 1e-06 e 0.001 obtiveram o mesmo resultado, alocando todo o peso no ativo SPY, enquanto os modelos restantes alocaram todo o peso no ativo XLK.

rate, respectivamente.

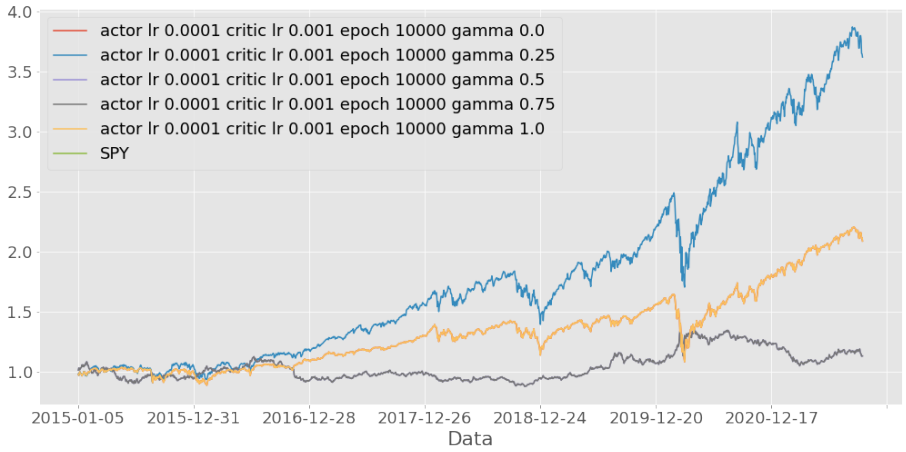


Figura 43 – Desempenho do modelo Ator-Crítico com planning, para diferentes valores de γ. Os modelos com γ igual a 0.5 e 0.75 tiveram o mesmo desempenho, alocando todo o peso no ativo TLT.O.

O objetivo de adicionar esta etapa de *planning* era tentar aproveitar mais

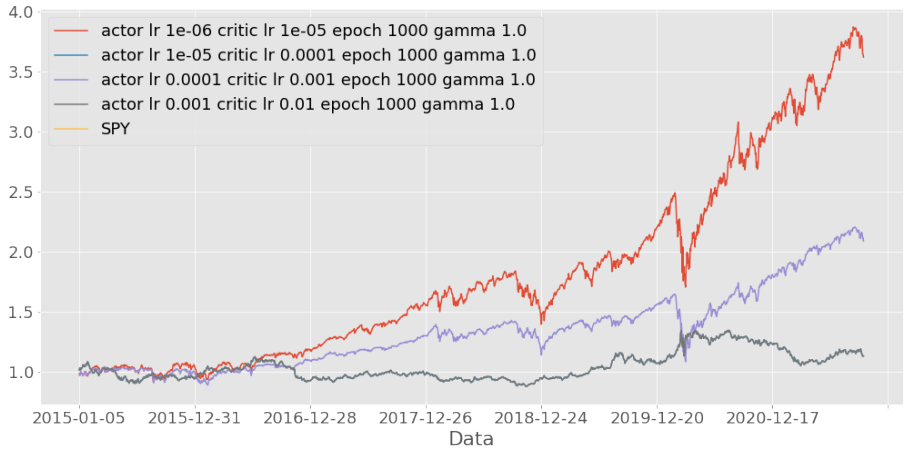


Figura 44 – Desempenho do modelo Ator-Crítico com planning, para diferentes valores de *learning rate*. Os modelos com taxa de aprendizado para o crítico de 1e-05 e 0.01 obtiveram os mesmos resultados, apontando todo o peso para o ativo TLT.O.

dos estados já visitados. Assim, a expectativa era de que dessa forma houvesse o reforço das informações de correlação entre os ativos, e diminuíssemos a tendência do agente de focar em apenas um ativo. Como visto pelos resultados, não houve efeito nesta abordagem que diferenciasse do resultado inicial para esta arquitetura.

Os próximos testes procuravam verificar o efeito do aprendizado online sobre o resultado oferecido pelo agente. Ou seja, o quanto do aprendizado de diferenças temporais, a cada instante de tempo, levava a uma visão "míope"do agente sobre o período de transações. A técnica aplicada para este teste consistia em utilizar traços de elegibilidade no ajuste dos pesos da rede do crítico. De maneira resumida, o algoritmo utiliza uma variável z para manter salvo o efeito dos pesos no gradiente da função de valor, a cada passagem de tempo. Através de um parâmetro de regulagem $\lambda \in [0, 1]$, o algoritmo podia assumir um comportamento mais próximo de um aprendizado por diferenças temporais (para $\lambda = 0$), ou mais próximo de um aprendizado por Monte Carlo (quando $\lambda = 1$), este último com a diferença de poder ser usado para tarefas contínuas (SUTTON; BARTO, 2018). A rotina pode ser vista no algoritmo 23.

Algoritmo 23 Semi-Gradiente TD(λ) para estimar $\hat{v} \approx v_\pi$ usando Traços de Elegibilidade.

Inicializa uma função de valor diferenciável $\hat{v} : \mathcal{S} \times \mathbb{R}^d \to \mathbb{R}$
Inicializa uma função π para a política.
Insere parâmetro de passo $\alpha > 0$ e um parâmetro de taxa decaimento de traço $\lambda \in [0, 1]$
Inicializa vetor de pesos $\mathbf{w} \in \mathbb{R}^d$ arbitrariamente
for *cada episódio* **do**
    Obtém um estado inicial, S
    $\boldsymbol{z} \leftarrow \mathbf{0}$
    for *cada passo do episódio, até* S' *terminal* **do**
        Seleciona uma ação $A \sim \pi(\cdot|S)$
        Executa a ação A e observa R, S'
        $\mathbf{z} \leftarrow \gamma\lambda\mathbf{z} + \nabla\hat{v}(S, \mathbf{w})$
        $\delta \leftarrow R + \gamma\hat{v}(S', \mathbf{w}) - \hat{v}(S, \mathbf{w})$
        $\mathbf{w} \leftarrow \mathbf{w} + \alpha\delta\mathbf{z}$
        $S \leftarrow S'$
    end for
end for

O teste foi realizado alterando-se o valor de λ, a fim de explorar o espaço de comportamentos possíveis. Os resultados podem ser vistos na Figura 45.

Novamente, assim como os outros testes realizados com o algoritmo Ator-Crítico, o agente tendeu a seguir algum dos ativos ao longo de toda a execução.

Nos testes realizados com *features* simples, a utilização de um ator-crítico não se mostrou eficaz, mesmo com variações na metodologia. Este modelo teve desempenho inferior ao DPG, tendendo a seguir um ativo da carteira, e não obtendo melhor visibilidade dos diferentes momentos do mercado, como desejado. Assim, como o modelo DPG obteve melhores desempenhos com uma complexidade menor, optamos por prosseguir com as variações de implementação com o mesmo.

Apesar disso, conforme mencionado em FISCHER, 2018, abordagens utilizando Ator-Crítico ainda são pouco exploradas, e apresentam vantagens em potencial que podem ser melhores abordadas em trabalhos futuros.

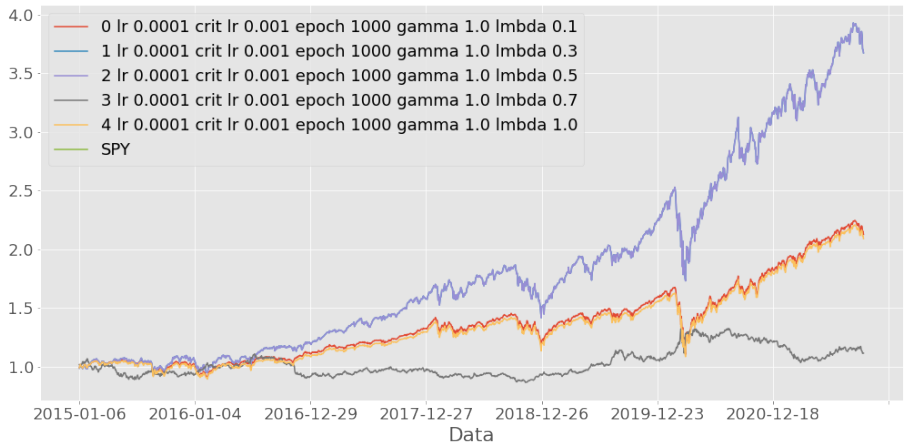


Figura 45 – Resultado para modelo Ator-Crítico com traços de elegibilidade. O modelo com λ igual a 0.3 e o com λ igual a 0.7 obtiveram os mesmos resultados, alocando todo o peso no ativo TLT.O.

5.2.6 Comparativo dos Resultados

Nesta Seção, selecionamos os melhores resultados obtidos em cada experimento e os comparamos ao baseline (SPY) e também a dois benchmarks MinVol e MaxSharpe. Podemos ver os resultados sumarizados na Figura 48. Na tabela 1 é possível realizar a comparação entre os modelos através de diferentes métricas de desempenho, sendo elas:

- **Retorno Total**: valor percentual de retorno financeiro do portfólio, após o período considerado.
- **Sortino Diário**: métrica sortino considerando dados diários de valor do portfólio.
- **Sharpe Diário**: métrica sharpe considerando dados diários de valor do portfólio.
- **Max Drawdown**: máximo valor de drawdown observado na série histórica.
- **Sortino Mensal**: métrica sortino considerando dados mensais de valor do portfólio.

- **Sharpe Mensal**: métrica sharpe considerando dados mensais de valor do portfólio.
- **CAGR**: *Compound Annual Growth Rate*, ou taxa de crescimento anual composto, é uma métrica que indica qual seria a taxa de retorno do investimento até seu valor final, caso a mesma fosse constante ao longo de todo o período considerado. É algo como "qual o retorno diário observado caso o mesmo fosse constante".

Realizamos a comparação também com duas abordagens de otimização de carteira, sendo elas a minimização da volatilidade do portfólio, e a maximização do índice Sharpe. O desempenho de ambas, comparado à evolução do ativo SPY pode ser vista na Figura 46.

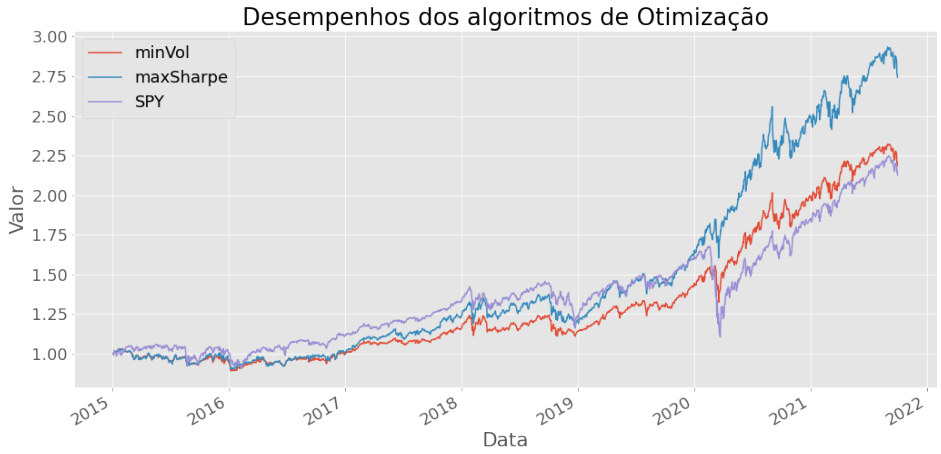


Figura 46 – Desempenho de algoritmos de otimização de portfólio.

Estas abordagens constituem-se em heurísticas modificas para fazer uso da informação de momento linear para melhorar a otimização das métricas consideradas. Assim, não apenas aplicando a lógica de otimização similar às descritas na seção 3.2, sua lógica passa por algo como o descrito nas equações 5.6 e 5.7, para a minimização da volatilidade e maximização da métrica Sharpe, respectivamente. Sendo *mp* o momento probabilístico entre as ETFs SPY e TLT.O, considerando o período dos últimos 60 dias,

$$\begin{aligned} \min_w \frac{1}{2} w^T \Sigma w, \\ \sum w = 1 \\ \text{caso } mp \geq 0.5: \\ w_{TLT.O} = 0 \\ w_{XLK} \geq 0.5 \\ \text{caso contrário:} \\ w_{TLT.O} \geq 0.5. \end{aligned} \tag{5.6}$$

$$\begin{aligned} \max_w \frac{\mu_p^T w - r_f}{\sqrt{w^T \Sigma w}}, \\ \sum w = 1 \\ \text{caso } mp \geq 0.5: \\ w_{TLT.O} = 0 \\ w_{XLK} \geq 0.5 \\ \text{caso contrário:} \\ w_{TLT.O} \geq 0.5. \end{aligned} \tag{5.7}$$

Ou seja, caso haja uma probabilidade de que o ativo SPY performe melhor do que o ativo de proteção, nada é alocado no mesmo, e a otimização é feita entre os dois outros ativos da carteira. Existindo uma probabilidade de que o ativo de proteção performe melhor do que o SPY, então parte da composição do portfólio se dedica à aquele, e a otimização da alocação restante é feita com os outros dois ativos da carteira. Esta modificação da abordagem já representa uma melhoria na metodologia básica descrita na seção 3.2, como pode ser visto ao analisar o desempenho das mesmas, quando não consideramos o momento probabilístico, na Figura 47. Uma outra diferença destas abordagens em comparação com os agentes de aprendizado por reforço é que a tomada de decisão é semanal, ao invés de diária, o que deixa as escolhas mais robustas à variabilidade do preço que ocorre ao longo dos dias - pois a decisão semanal age como uma "suavização"da curva para o agente, que ao invés de observar toda a volatilidade dos preços ao longo de um dia, passa a considerar apenas seu resultado agregado ao final da semana

- mas deixa os decisores menos reativos à desempenhos ruins durante a semana - afinal mesmo que a alocação escolhida para o portfólio comece a performar mal, o ajuste dos pesos será feito apenas na próxima semana.

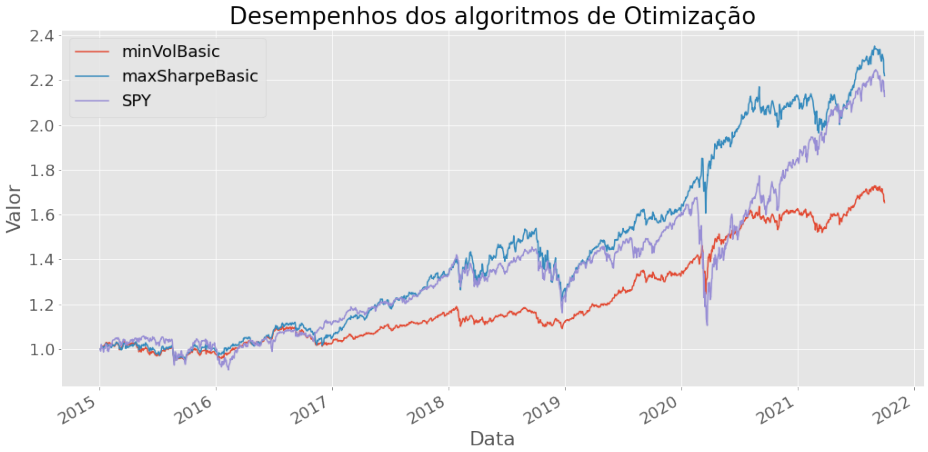


Figura 47 – Desempenho de algoritmos de otimização de portfólio sem considerar momento probabilístico.

Pela tabela é possível observar que os algoritmos de aprendizado por reforço conseguiram trazer um retorno financeiro tão elevado quanto, ou maior do que as abordagens de benchmark, sem comprometer métricas de risco ou drawdown máximo.

A Figura 49 mostra as curvas de drawdown para as abordagens comparadas.

Tabela 1 – Métricas de desempenho para os melhores modelos.

Modelo	Retorno Total	Sortino Diário	Sharpe Diário	Max Drawdown	Sortino Mensal	Sharpe Mensal	CAGR
minVol	121.95%	1.48	0.98	-14.82%	2.56	1.29	13.31%
maxSharpe	177.91%	1.67	1.08	-15.15%	2.79	1.37	17.38%
SPY	102.01%	1.04	0.70	-34.10%	1.42	0.83	11.65%
Deep Policy Gradient	179.98%	1.75	1.07	-17.93%	2.09	1.09	17.51%
DPG (Temporal Features)	112.97%	1.69	1.09	-17.79%	2.37	1.18	12.58%
DPG new Architecture (Temporal Features and Sharpe)	304.12%	2.47	1.48	-17.35%	3.67	1.54	24.47%

O algoritmo DPG, mais comum na literatura, como mencionado no Ca-

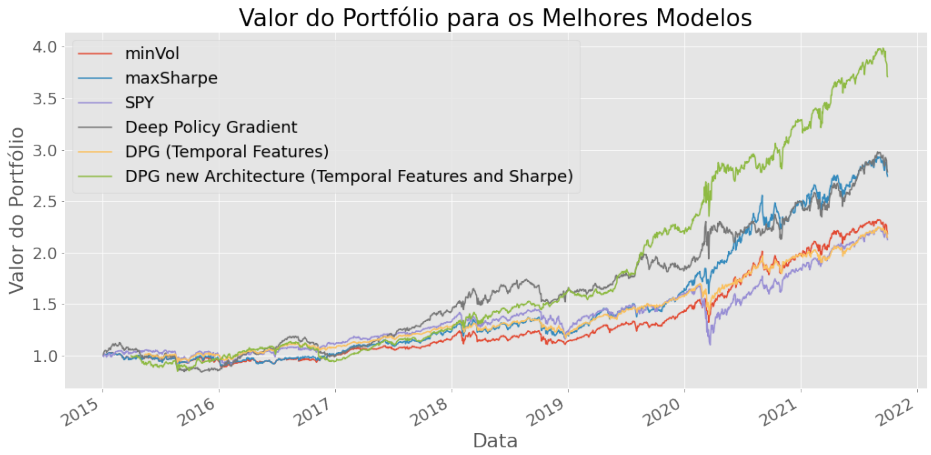


Figura 48 – Valor do portfólio para os melhores modelos.

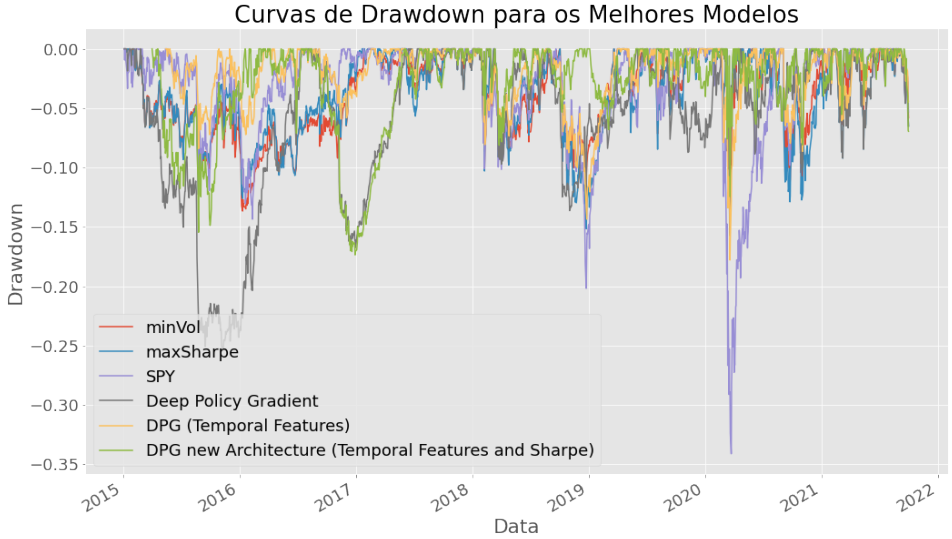


Figura 49 – Curvas de Drawdown para os melhores modelos.

pítulo 4, foi o que obteve bons resultados mesmo em sua implementação mais simples, ainda sem utilizar *features* temporais. As iterações realizadas nesta abordagem, tanto com o intuito de aprofundar a visão do modelo do espaço de estados, quanto para aumentar a complexidade do algoritmo, trouxeram melhorias de al-

guma forma, e contribuíram para tornar o modelo mais robusto na resolução do problema de gerenciamento de portfólio.

Considerando que estas abordagens ainda permitem a utilização de técnicas mais especializadas e complexas para resolver particularidades deste tipo de problema, existe espaço para pesquisas que busquem resultados ainda melhores e mais assertivos. O que este estudo de caso mostra é que é factível e promissor partir para abordagens de aprendizado por reforço para este problema, e também como pode ser desenvolvido o racional por trás da modelagem do mesmo e do desenvolvimento de um algoritmo relacionado.

6 Conclusão

Técnicas de aprendizado por reforço constituem um tópico importante no contexto de algoritmos de aprendizado de máquina, onde não apenas a etapa de aprendizado em si é levada em consideração, como também a construção do contexto e do processo onde este aprendizado ocorre. Fazendo uso de técnicas já conhecidas e comuns em modelagem, como regressões, redes neurais, e técnicas de aprendizado não supervisionado, um sistema de aprendizado por reforço, sumarizado no esquema de um processo de decisão Markoviano, procura dar importância a como o aprendizado ocorre, e como uma interação de um agente atuador com o ambiente pode ser metrificada e traduzida para o agente, tanto na caracterização do ambiente com o qual interage, como na forma como o estímulo da interação é transmitido à entidade em aprendizado.

Este tipo de algoritmo pode ser utilizado para lidar com problemas complexos, como os relativos ao mercado financeiro, onde a modelagem dos dados tende a ser complicada devido à alta volatilidade e dinâmica dos preços. No contexto de gerenciamento de carteiras de ações, onde a atenção dada a cada ativo depende da expectativa em torno do desempenho de cada um deles, a decisão torna-se complexa justamente devido à essa alta variabilidade. Como existem inúmeros fatores que podem influenciar no comportamento de um ativo, e é praticamente impossível predizer como o mercado irá se comportar, a utilização desta metodologia, que não necessita de um modelo do ambiente para poder tomar decisões, mostra-se promissora.

Ao contrário de algoritmos de otimização baseados em heurísticas e regras, a possibilidade de incorporar aprendizado à rotina, e de adequar o comportamento do modelo na busca por melhores resultados constituem vantagens de algoritmos de aprendizado por reforço em comparação com metodologias usuais de otimização de portfólios. Além disso, este tipo de abordagem possibilita ao agente atuar com maior rapidez à alterações nas condições de marcado. Técnicas de otimização de portfólio clássicas tendem a ser mais lentas na percepção das mudanças de corre-

lação, a não ser que tenham alguma informação à priori sobre o retorno esperado dos ativos que compõe a carteira.

Neste trabalho foi mostrado como as técnicas de aprendizado por reforço podem ser utilizadas para atuar em problemas financeiros, revisando a bibliografia existente e os estudos atuais desenvolvidos à respeito do tema. O estudo trouxe ainda um estudo de caso para ilustrar a capacidade de adaptação do agente aos diferentes cenários e regimes que o mercado financeiro pode apresentar.

Partindo da bibliografia revisada, e dos aprendizados empregados durante o estudo de caso, vimos que para problemas de gerenciamento de portfólio o tipo de técnica empregada depende dos objetivos desejados para o problema, e de como queremos atuar no mesmo. Para uma alocação de pesos em ativos dentro de uma carteira, uma metodologia que lide com as ações em um espaço de estados contínuos é mais favorável, uma vez que uma modelagem discreta é complexa e não necessariamente representa o que queremos do agente (como visto na seção 5.2, na descrição da modelagem do espaço de ações). Assim, algoritmos como o *Deep Policy Gradient*, bem como variações do mesmo, são bem indicados, e mais amplamente utilizados na literatura.

Ainda sobre a escolha do algoritmo baseado na modelagem do problema, um ponto em comum na bibliografia encontrada é a utilização de abordagens que não necessitem de um modelo do ambiente para o agente realizar suas escolhas. A explicação passa pela impossibilidade de obter um modelo do comportamento das ações, devido à alta imprevisibilidade que as mesmas possuem dados os fatores externos que podem alterar seu comportamento, e que são complexos de serem enumerados. Dessa forma, abordagens que dependam do fornecimento de uma função de probabilidade para a transição de estados são inviáveis.

Como visto na literatura, e também observado no estudo de caso, arquiteturas de modelos que levem em consideração as features como séries históricas podem ter um desempenho melhor do que modelos feitos para operar com features que não sejam séries temporais (que consideram somente o momento atual). Redes de convolução, apesar de serem popularmente encontradas no contexto de aprendizado supervisionado em problemas relacionados à visão computacional e

processamento de imagens, no âmbito do problema de gerenciamento de portfólio por algoritmos de aprendizado por reforço constituem uma abordagem comum, juntamente com outros algoritmos mais utilizados para atuação com séries temporais, como redes LSTM. A maneira de se modelar dados temporais para serem interpretados por essas redes deve atentar-se à particularidade do problema, como pensar na independência entre as features, e como se espera que os filtros se comportem ao lidar com as mesmas. Mas ainda assim é uma abordagem com bons resultados, conforme a literatura citada no capítulo 4 mostra.

Durante a abordagem do estudo de caso, foi mostrada como alterações na forma de lidar com as features podem representar melhorias no entendimento do problema pelo modelo. Portanto um raciocínio que envolva tanto a escolha da metodologia mais adequada para lidar com o problema quanto a forma como o algoritmo vai interpretar o estado é essencial para obter bons resultados. Podemos citar o exemplo do artigo de JIANG; XU; LIANG, 2017, revisado na seção 4.1, onde a modificação da estrutura do modelo para adicionar a estrutura de EIIE representou uma melhoria na forma como o mesmo lidou com a tarefa que teve que desempenhar.

A área de estudo de algoritmos de aprendizado por reforço para problemas financeiros ainda está em constante desenvolvimento, e novas implementações surgem a todo o tempo. Portanto ainda existe espaço para melhorias de algoritmos, e abordagens que tragam novos olhares sobre o problema, com a perspectiva de melhores resultados. Fica como sugestão de continuação deste trabalho aprofundar-se nas diferentes abordagens existentes para resolver este problema, que não foram citadas. Além disso, é interessante que trabalhos futuros estudem e explorem outras técnicas de aprendizado por reforço que ainda não sejam empregadas no âmbito de mercado financeiro, ou que tenham sido ainda pouco abordadas, como não só algoritmos actor-critic e suas diferentes variações, como técnicas recentes que sejam estado-da-arte na resolução de outros problemas de aprendizado por reforço.

Referências

ANG, A.; TIMMERMANN, A. Regime changes and financial markets. *Annu. Rev. Financ. Econ.*, Annual Reviews, v. 4, n. 1, p. 313–337, 2012. Citado na página 125.

ANGELIS, L. D.; PAAS, L. J. A dynamic analysis of stock markets using a hidden markov model. *Journal of Applied Statistics*, Taylor & Francis, v. 40, n. 8, p. 1682–1700, 2013. Disponível em: <https://doi.org/10.1080/02664763.2013.793302>. Citado na página 126.

AZHIKODAN, A. R.; BHAT, A. G. K.; JADHAV, M. V. Stock trading bot using deep reinforcement learning. In: . [S.l.]: Springer Singapore, 2018. p. 41–49. Citado na página 105.

BARDENET, R. Monte carlo methods. In: . [S.l.: s.n.], 2013. Citado na página 40.

BELLMAN, R. A markovian decision process. *Journal of mathematics and mechanics*, JSTOR, p. 679–684, 1957. Citado na página 35.

BELLMAN, R. Dynamic programming. *Science*, American Association for the Advancement of Science, v. 153, n. 3731, p. 34–37, 1966. Citado na página 40.

CAPIŃSKI, M. J.; KOPP, E. *Portfolio theory and risk management*. [S.l.]: Cambridge University Press, 2014. v. 5. Citado na página 25.

CHEN, J.; TSANG, E. P. Classification of normal and abnormal regimes in financial markets. *Algorithms*, Multidisciplinary Digital Publishing Institute, v. 11, n. 12, p. 202, 2018. Citado na página 125.

CORNUEJOLS, G.; TUTUNCU, R. Optimization Methods in Finance (Mathematics, Finance and Risk). n. January, p. 358, 2007. Disponível em: <http://www.amazon.com/Optimization-Methods-Finance-Mathematics-Risk/dp/0521861705>. Citado 2 vezes nas páginas 25 e 99.

DATABASE, E. *What Is Beta and What Does It Mean for Your ETF Portfolio?* 2016. Disponível em: <https://etfdb.com/portfolio-management/what-is-beta-and-what-does-it-mean-for-your-etf-portfolio/>. Citado na página 124.

DENG, Y. et al. Deep direct reinforcement learning for financial signal representation and trading. *IEEE Transactions on Neural Networks and Learning Systems*, v. 28, n. 3, p. 653–664, 2017. Citado na página 105.

DIEBOLD, F. X.; LEE, J.-H.; WEINBACH, G. C. Regime switching with time-varying transition probabilities. *Business Cycles: Durations, Dynamics, and Forecasting*, Princeton University Press Princeton, NJ, v. 1, p. 144–165, 1994. Citado na página 125.

DIXON, M. F.; HALPERIN, I.; BILOKON, P. *Machine Learning in Finance*. [S.l.]: Springer International Publishing, 2020. ISBN 978-3-030-41067-4. Citado na página 105.

DURRETT, R. *Probability: theory and examples*. [S.l.]: Cambridge university press, 2019. v. 49. Citado na página 37.

ETF.COM. *TLT*. 2022. Disponível em: <https://www.etf.com/TLT#overview>. Citado na página 124.

FARIA, G.; ROMERO, R. A. F. Navegação de robôs móveis utilizando aprendizado por reforço e lógica fuzzi. *Sba: Controle & Automação Sociedade Brasileira de Automatica*, SciELO Brasil, v. 13, n. 3, p. 219–230, 2002. Citado na página 30.

FISCHER, T. G. Reinforcement Learning in Financial Markets. 2018. Citado 3 vezes nas páginas 26, 105 e 151.

GAGNIUC, P. A. *Markov chains: from theory to implementation and experimentation*. [S.l.]: John Wiley & Sons, 2017. Citado na página 125.

GUPTA, A. et al. *Unsupervised Meta-Learning for Reinforcement Learning*. 2018. Citado na página 30.

HOLT, C. C. Forecasting seasonals and trends by exponentially weighted moving averages. *International Journal of Forecasting*, v. 20, n. 1, p. 5–10, 2004. ISSN 0169-2070. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0169207003001134>. Citado na página 115.

HUANG, G.; ZHOU, X.; SONG, Q. *Deep reinforcement learning for portfolio management based on the empirical study of chinese stock market*. 2021. Citado na página 122.

INDICES, S. D. J. *VIX - The CBOE Volatility Index*. 2022. Disponível em: <https://www.spglobal.com/spdji/en/vix-intro/>. Citado na página 128.

INVESTIDOR, P. do. *Fundos de Índice, ETFs*. 2022. Disponível em: <https://www.investidor.gov.br/menu/Menu_Investidor/valores_mobiliarios/ETFs.html>. Citado na página 123.

INVESTOPEDIA. *TLT: An ETF Overview*. 2020. Disponível em: <https://www.investopedia.com/articles/investing/031915/overview-tlt-etf.asp>. Citado na página 124.

INVESTOPEDIA. *Beta*. 2021. Disponível em: <https://www.investopedia.com/terms/b/beta.asp>. Citado na página 124.

JADERBERG, M. et al. *Reinforcement Learning with Unsupervised Auxiliary Tasks*. 2016. Citado na página 30.

JIANG, Z.; LIANG, J. Cryptocurrency portfolio management with deep reinforcement learning. In: *2017 Intelligent Systems Conference (IntelliSys)*. [S.l.: s.n.], 2017. p. 905–913. Citado na página 105.

JIANG, Z.; XU, D.; LIANG, J. A Deep Reinforcement Learning Framework for the Financial Portfolio Management Problem. p. 1–31, 2017. Disponível em: <http://arxiv.org/abs/1706.10059>. Citado 7 vezes nas páginas 13, 109, 113, 116, 117, 134 e 161.

KIRKPATRICK, C.; DAHLQUIST, J. *Technical Analysis: The Complete Resource for Financial Market Technicians*. First. [S.l.]: FT Press, 2006. ISBN 0131531131. Citado na página 115.

LI, X. et al. *Optimistic Bull or Pessimistic Bear: Adaptive Deep Reinforcement Learning for Stock Portfolio Allocation*. 2019. Citado 2 vezes nas páginas 105 e 106.

LI, Y.; NI, P.; CHANG, V. An empirical research on the investment strategy of stock market based on deep reinforcement learning model. In: . [S.l.]: SCITEPRESS - Science and Technology Publications, 2019. ISBN 978-989-758-366-7. Citado na página 105.

LIANG, Z. et al. Adversarial Deep Reinforcement Learning in Portfolio Management. 2018. Disponível em: <http://arxiv.org/abs/1808.09940>. Citado 2 vezes nas páginas 105 e 107.

LILLICRAP, T. P. et al. Continuous control with deep reinforcement learning. *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*, 2016. Citado na página 146.

LUENBERGER, D. G. et al. Investment science. *OUP Catalogue*, Oxford university press, 1997. Citado na página 25.

MAAS, A. L.; HANNUN, A. Y.; NG, A. Y. Rectifier nonlinearities improve neural network acoustic models. *in ICML Workshop on Deep Learning for Audio, Speech and Language Processing*, 2013. Citado na página 132.

MANSINI, R.; OGRYCZAK, W.; SPERANZA, M. G. *Linear and Mixed Integer Programming for Portfolio Optimization*. [S.l.]: Springer International Publishing, 2015. Citado 2 vezes nas páginas 25 e 105.

MARKOV, A. A. et al. *The theory of algorithms*. [S.l.]: Springer, 1960. Citado na página 37.

MOORE, A. W. *Efficient Memory-based Learning for Robot Control*. [S.l.], 1990. Citado 2 vezes nas páginas 13 e 34.

MURPHY, J. J. *Study Guide to Technical Analysis of the Financial Markets: A Comprehensive Guide to Trading Methods and Applications*. [S.l.]: Penguin, 1999. Citado na página 128.

RUNDO, F. et al. Machine learning for quantitative finance applications: A survey. *Applied Sciences*, v. 9, n. 24, 2019. ISSN 2076-3417. Disponível em: <https://www.mdpi.com/2076-3417/9/24/5574>. Citado na página 105.

RUSSELL, S.; NORVIG, P. Artificial intelligence: a modern approach. 2002. Citado na página 29.

SANTOS, M. S.; RUST, J. Convergence properties of policy iteration. *SIAM Journal on Control and Optimization*, v. 42, n. 6, p. 2094–2115, 2004. ISSN 03630129. Citado 2 vezes nas páginas 48 e 51.

SILVER, D. et al. Deterministic policy gradient algorithms. *31st International Conference on Machine Learning, ICML 2014*, v. 1, p. 605–619, 2014. Citado 2 vezes nas páginas 114 e 146.

SORTINO, F. A.; PRICE, L. N. Performance measurement in a downside risk framework. *The Journal of Investing*, Institutional Investor Journals Umbrella, v. 3, n. 3, p. 59–64, 1994. ISSN 1068-0896. Disponível em: <https://joi.pm-research.com/content/3/3/59>. Citado na página 103.

SPDR, S. S. G. A. *The Technology Select Sector SPDR® Fund*. 2022. Disponível em: <https://www.ssga.com/us/en/intermediary/etfs/funds/the-technology-select-sector-spdr-fund-xlk>. Citado na página 124.

SUNO. *SPY: conheça o ETF americano do índice S&P 500*. 2021. Disponível em: <https://www.suno.com.br/artigos/spy/>. Citado na página 123.

SUTTON, R. S.; BARTO, A. G. *Reinforcement learning: an introduction*. Second edi. Cambridge, MA: The MIT Press, 2018. 4153–4153 p. ISBN 9780262039246. Citado 17 vezes nas páginas 13, 27, 29, 30, 32, 48, 52, 53, 71, 73, 76, 83, 93, 94, 108, 146 e 150.

VINYALS, O. et al. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, Springer Science and Business Media LLC, v. 575, n. 7782, p. 350–354, oct 2019. Citado na página 106.

XIONG, Z. et al. *Practical Deep Reinforcement Learning Approach for Stock Trading*. 2018. Citado na página 105.

YE, Y. et al. Reinforcement-learning based portfolio management with augmented asset movement prediction states. *Proceedings of the AAAI Conference on Artificial Intelligence*, v. 34, n. 01, p. 1112–1119, Apr. 2020. Disponível em: <https://ojs.aaai.org/index.php/AAAI/article/view/5462>. Citado 2 vezes nas páginas 116 e 121.

YU, P. et al. Model-based Deep Reinforcement Learning for Dynamic Portfolio Optimization. 2019. Disponível em: <http://arxiv.org/abs/1901.08740>. Citado na página 105.

ZARKIAS, K. S. et al. Deep reinforcement learning for financial trading using price trailing. In: *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. [S.l.: s.n.], 2019. p. 3067–3071. Citado na página 105.

ZHU, F. Market regime forecasting using correlation networks and machine learning classifiers. The University of North Carolina at Chapel Hill University Libraries, 2020. Citado na página 125.

Apêndices

APÊNDICE A – Teoremas e Provas

A.1 Política Ótima a partir de uma Função de Valor ótima

Teorema 1 *Para qualquer Processo de Decisão de Markov:*

- *Existe ao menos uma política ótima* $\pi*$*, onde* $V_{\pi*}(s) \geq V_{\pi}(s)$ *para todas as políticas* π *e todos os estados* $s \in \mathcal{S}$*.*
- *Todas as políticas ótimas possuem a mesma Função de Valor Ótima,* $V_{\pi*}(s) = V_{*}(s)$*, para todos os estados* $s \in \mathcal{S}$*.*
- *Todas as políticas ótimas possuem a mesma Função de Ação-Valor Ótima,* $Q_{\pi*}(s,a) = Q_{*}(s,a)$*, para todos os estados* $s \in \mathcal{S}$ *e ações* $a \in \mathcal{A}$*.*

Prova: Para provar este teorema, partimos para o seguinte lema:

Lema 1 *Para quaisquer duas políticas ótimas* π_1 *e* π_2*,* $V_{\pi_1}(s) = V_{\pi_2}(s)$*, para todo estado* $s \in \mathcal{S}$*.*

Prova: Uma vez que π_1 é uma política ótima, a partir da definição vista na seção 2.3.3, $V_{\pi_1}(s) \geq V_{\pi_2}(s)$ para todo estado $s \in \mathcal{S}$. Da mesma forma, sendo π_2 também uma política ótima, pela definição, $V_{\pi_2}(s) \geq V_{\pi_1}(s)$, para todo estado $s \in \mathcal{S}$. Isto implica em $V_{\pi_1}(s) = V_{\pi_2}(s)$ para todo estado $s \in \mathcal{S}$.

Como consequência, para provar o teorema, precisamos apenas estabelecer uma política ótima π_8, com um valor ótimo para as funções de valor de ação e de estado. Considerando uma política determinística candidata a política ótima, π_*, que seleciona sempre as ações com maior função de valor de ação, a cada estado, ou seja:

$$\pi_*(s) = \underset{a \in \mathcal{A}}{argmax}\ Q_*(s,a) \quad \forall \quad s \in \mathcal{S} \tag{A.1}$$

Primeiramente demonstraremos que a política π_* pode ser considerada como sendo ótima. Considerando a equação A.1, e sendo

$$V_*(s) = \underset{a \in \mathcal{A}}{max}\, Q_*(s, a), \tag{A.2}$$

para todo estado $s \in \mathcal{S}$, então π_* pode ser considerado como ótimo, uma vez que obtém o valor ótimo para a função de valor de estado, V_*, para todos os estados. Ou seja,

$$V_{\pi_*}(s) = V_*(s) \tag{A.3}$$

para todos os estados. Podemos fazer o mesmo raciocínio para provar $Q_{\pi*}(s, a) = Q_*(s, a)$.

Finalmente, podemos provar por contradição que a política π_* é ótima. Assumindo que π_* não seja uma política ótima. Isso significa afirmar que existe uma política π que, em algum estado qualquer s, $V_\pi(s) > V_{\pi_*}(s)$. Como provado que $V_{\pi_*}(s) = V_*(s)$, isto significaria que $V_\pi(s) > V_*(s)$, o que contradiz a definição da equação 2.11, de que $V_* = V_*(s) = \underset{\pi}{max}\, V_\pi(s)$, invalidando a existência de π.