UNIVERSIDADE FEDERAL DE SÃO JOÃO DEL-REI

Júlio César Mendes de Resende

Aprendizado por Reforço Profundo: Combinando Técnicas para Aperfeiçoar o Algoritmo FQF

São João del-Rei

2022

UNIVERSIDADE FEDERAL DE SÃO JOÃO DEL-REI

Júlio César Mendes de Resende

Aprendizado por Reforço Profundo: Combinando Técnicas para Aperfeiçoar o Algoritmo FQF

Dissertação apresentada ao Programa de Pós Graduação em Ciência da Computação da UFSJ como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

Orientador: Edimilson Batista dos Santos

Coorientador: Marcos Antonio de Matos Laia

Universidade Federal de São João del-Rei – UFSJ

Programa de Pós Graduação em Ciência da Computação

São João del-Rei

2022

Júlio César Mendes de Resende

Aprendizado por Reforço Profundo: Combinando Técnicas para Aperfeiçoar o Algoritmo FQF

Dissertação apresentada ao Programa de Pós Graduação em Ciência da Computação da UFSJ como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

Trabalho aprovado. São João del Rei, 26 de outubro de 2022:

Dr. Edimilson Batista dos Santos
Universidade Federal de São João del Rei
Orientador

Dr. Marcos Antônio de Matos Laia
Universidade Federal de São João del Rei
Coorientador

Dra. Carolina Ribeiro Xavier
Universidade Federal de São João del Rei

Dr. Dênis Fernando Wolf
Universidade de São Paulo

São João del Rei
2022

Ficha catalográfica elaborada pela Divisão de Biblioteca (DIBIB)
e Núcleo de Tecnologia da Informação (NTINF) da UFSJ,
com os dados fornecidos pelo(a) autor(a)

R433a

Resende, Júlio.
Aprendizado por Reforço Profundo: Combinando Técnicas para Aperfeiçoar o Algoritmo FQF / Júlio Resende ; orientador Edimilson Santos; coorientador Marcos Laia. -- São João del-Rei, 2022.
98 p.

Dissertação (Mestrado - Programa de Pós-Graduação em Ciência da Computação) -- Universidade Federal de São João del-Rei, 2022.

1. Aprendizado por Reforço. 2. Aprendizado Profundo. 3. Aprendizado por Reforço Distributivo.. I. Santos, Edimilson, orient. II. Laia, Marcos, co orient. III. Título.

Agradecimentos

Primeiramente eu gostaria de render agradecimentos especiais à Deus e aos meus pais Tiago e Zélia, por terem me oferecido a oportunidade de cursar uma graduação e agora um mestrado. Infelizmente são poucos os brasileiros que possuem a oportunidade de frequentar uma universidade, então me sinto privilegiado por isso.

Agradeço aos meus irmãos Tássio e Cássia e a toda a minha família pelo apoio, especialmente às minhas sobrinhas Maya e Maria Luísa, que mesmo sem compreenderem, tornaram esse período do mestrado mais alegre. Aos meus amigos, gratidão pela cumplicidade e motivação.

A todos os meus professores da graduação e mestrado, deixo aqui o meu agradecimento. Foram tantos os que contribuíram para a minha formação, que não vou arriscar a citar nomes. Aproveito para agradecer à Universidade Federal de São João del Rei e ao Departamento de Ciência da Computação (DCOMP) por oferecerem um ensino público, gratuito e de qualidade.

Aos meus companheiros de serviço do Sicoob Credivertentes, Bitka Analytics e Stone Pagamentos, agradeço pelo apoio, amizade e principalmente pela compreensão nas manhãs pós estudos em que estive mais cansado do que o normal.

Agradeço aos meus orientadores Edimilson e Marcos pela parceria e por confiarem no meu potencial em desenvolver este trabalho, mesmo quando os experimentos iniciais não pareciam tão promissores.

Agradeço ao professor Denis Wolf pela disponibilidade em participar da banca final deste trabalho. Agradeço também aos professores que estiveram presentes na minha banca de qualificação: Carolina Xavier (também presente na banca final) e Erivelton Nepomuceno. Ambos apresentaram contribuições importantes para o término desta pesquisa.

Agradeço ao DCOMP e ao professor Diego Dias por disponibilizarem duas ótimas máquinas para a execução dos experimentos. Também agradeço ao professor Elverton Fazzion pela atenção e disponibilidade em realizar suporte técnico em uma das máquinas utilizadas.

No fim desse ciclo, olho para o início e faço um agradecimento a todos os professores que eu tive ainda na época de escola. Destaco aqui o Ronaldo, a Alda, a Beatriz, o Francis e o Rodrigo. Essas pessoas fizeram mais do que suas obrigações e permitiram que meu caminho na área de computação, através da robótica, começasse desde a infância. Isso com certeza fez muita diferença na minha carreira acadêmica e profissional.

"Mesmo quando tudo parece desabar, cabe a mim decidir entre rir ou chorar, ir ou ficar, desistir ou lutar; porque descobri, no caminho incerto da vida, que o mais importante é o decidir" - (Cora Coralina)

Resumo

Os algoritmos de aprendizado por reforço permitem que os agentes aprendam com a experiência, sem a necessidade de conhecimento prévio. Por essa razão, eles têm sido amplamente pesquisados e o uso de jogos digitais de baixa e média complexidade como ambientes de benchmark tornou-se uma prática comum. Em 2013, um novo algoritmo, chamado DQN (Deep Q Network), causou grande impacto no meio acadêmico ao obter resultados a nível de humanos em diversos jogos do Atari 2600, utilizando para isso redes neurais artificiais. Consequentemente, novas linhas de pesquisa surgiram e novos algoritmos derivados foram propostos. Dentre esses, destaca-se o FQF (*Fully Parameterized Quantile Function*), um algoritmo que se tornou o estado da arte entre os algoritmos não distribuídos no domínio do Atari 2600. No entanto, o FQF ainda não alcançou os resultados obtidos por um especialista humano em todos os jogos avaliados. Considerando a habilidade da inteligência artificial em detectar padrões imperceptíveis por humanos, isso nos levou a acreditar que resultados melhores do que os atuais ainda poderiam ser obtidos. Portanto, neste trabalho foi realizada uma pesquisa por trabalhos relacionados e foram escolhidas três melhorias que trouxeram sucesso em algoritmos propostos antes do FQF para serem combinadas e avaliadas juntamente com o FQF, buscando assim melhorar o algoritmo. As melhorias aplicadas ao FQF são: a utilização de três passos na diferença temporal, a aplicação da abordagem de Munchausen e o uso da experiência por repetição priorizada. A combinação das três melhorias possibilitou a análise de oito algoritmos, os quais foram avaliados em cinco jogos do Mini Atari. De acordo com as métricas analisadas, a versão do FQF que faz o uso das três melhorias foi melhor que o FQF original em todos os experimentos realizados, disponibilizando portanto uma versão mais promissora do algoritmo para a comunidade científica.

Palavras-chaves: Aprendizado por Reforço, Aprendizado Profundo, Aprendizado por Reforço Distributivo.

Abstract

Reinforcement learning algorithms allow agents to learn from experience, without the need for prior knowledge. For this reason, they have been widely used and the use of low and medium complexity digital games as benchmark environments has become a common practice. In 2013, a new algorithm, called DQN (Deep Q Network), caused a great impact in the academic environment by obtaining human-level results in several Atari 2600 games, using artificial neural networks. Consequently, new lines of research emerged and new derived algorithms were proposed. Among these, the FQF (Fully Parameterized Quantile Function) stands out, an algorithm that has become the state of the art among the non-distributed algorithms in the Atari 2600 domain. However, the FQF has not yet achieved results obtained by a human expert in all evaluated games. Considering the ability of artificial intelligence to detect patterns imperceptible by humans, this led us to believe that better results than the current ones could still be obtained. Therefore, in this work, a search for related works was carried out and three improvements that brought success in algorithms proposed before the FQF were chosen to be combined and evaluated together with the FQF, thus seeking to improve the algorithm. The improvements applied to the FQF are: the use of three steps in temporal difference, the application of the Munchausen approach and the use of prioritized experience replay. The combination of the three improvements made it possible to analyze eight algorithms, which were evaluated in five MinAtar games. According to the analyzed metrics, the version of the FQF that makes use of the three improvements was better than the original FQF in all experiments carried out, thus making a more promising version of the algorithm available to the scientific community.

Key-words: Reinforcement Learning, Deep Learning, Distributional Reinforcement Learning.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

ALE	*Arcade Learning Enviromment*
ANN	Rede Neural Artificial
CDF	Função de distribuição acumulada
CNN	Rede Neural Convolucional
DP	Programação Dinâmica
DQN	*Deep Q Network*
DDQN	*Double Deep Q Network*
DRQN	*Deep Recurrent Q Network*
FQF	*Fully Parameterized Quantile Function*
FQFM	FQF + MRL
FQFn3	FQF + n passos com $n = 3$
FQFn3M	FQF + 3 passos + MRL
FQFn3P	FQF + 3 passos + PER
FQFn3PM	FQF + 3 passos + PER + MRL
FQFP	FQF + PER.
FQFPM	FQF + PER + MRL
GD	Gradiente Descendente
IQM	Média Interquartil
IQN	*Implicit Quantile Network*
IS	Amostragem por Importância
MC	Monte Carlo
MDP	Processo de Decisão de Markov
MRL	*Munchausen Reinforcement Learning*

PER	Experiência por Repetição Priorizada
POMDP	Processo de Decisão de Markov Parcialmente Observável
QR-DQN	*Quantile Regression DQN*
SGD	Gradiente Descendente Estocástico
TD	Diferença Temporal. Quando utilizado TD(0) indica diferença temporal de um passo.

Lista de símbolos

$\doteq$	Relação de igualdade que é verdadeira por definição.
$\overset{D}{=}$	$B \overset{D}{=} C$ indica que as variáveis aleatórias B e C seguem a mesma distribuição.
$Pr\{X = x\}$ ou $p(x)$	Probabilidade de uma variável aleatória X assumir o valor x.
$X \sim p$	Variável aleatória X tem distribuição de probabilidade p.
$\mathbb{E}[X]$	Esperança de uma variável aleatória X, ou seja: $\mathbb{E}[X] = \sum_x p(x)x$
t	Passo de tempo discreto.
T	Último passo de um episódio.
n, N	Variáveis quantitativas utilizadas com diferentes finalidades ao longo do texto.
s, s'	Estados.
$\mathcal{S}$	Conjunto de todos os estados.
S_t	Estado do agente no tempo t.
a	Ação.
A_t	Ação tomada no tempo t.
$\mathcal{A}$	Conjunto de todas ações.
$\mathcal{A}(s)$	Conjunto de todas ações no estado s.
r	Recompensa.
R_t	Recompensa recebida no tempo t.
$\mathcal{R}$	Conjunto de todas as recompensas.
G_t	Soma das recompensas recebidas após o passo t.
$G_{t:t+n}$	Retorno de n passos de $t+1$ a $t+n$.
G_t^λ	Retorno λ.
π	Política.

Π	Conjunto de todas as políticas.
$\pi(s)$	Ação tomada no estado s seguindo a política π.
$\pi(a\|s)$	Probabilidade de se tomar a ação a no estado s seguindo a política π.
α	Fator de aprendizagem.
γ	Fator de desconto.
ε	Probabilidade de se tomar uma ação aleatória em uma política *ε-greedy*.
$v_\pi(s)$	Função de valor do estado s seguindo a política π.
$q_\pi(s, a)$	Função de valor de se tomar a ação a no estado s seguindo a política π posteriormente.
$v_*(s)$	Função de valor do estado s seguindo uma política ótima.
$q_*(s, a)$	Função de valor de se tomar a ação a no estado s seguindo uma política ótima posteriormente.
V	Vetor de estimativas de v_π ou v_*.
Q	Vetor de estimativas de q_π ou q_*.
$\mathbf{w}$	Vetor de pesos
w_i	Componente i do vetor $\mathbf{w}$.
$\widehat{v}(s, \mathbf{w})$	Valor aproximado do estado s dado o vetor $\mathbf{w}$.
$\widehat{q}(s, a, \mathbf{w})$	Valor aproximado do par (s, a) dado o vetor $\mathbf{w}$.
$\nabla f(\mathbf{w})$	Vetor coluna das derivadas parciais de $f(\mathbf{w})$ em relação a $\mathbf{w}$.
$\mathbf{x}(s)$	Vetor de *features* que descrevem o estado s.
$\mathbf{z}$	Traço de elegibilidade.
θ, θ^-	Parâmetros de redes neurais.
Y^{ALG}	Target do algoritmo ALG.
$Z(s, a)$	Distribuição dos retornos obtidos tomando a ação a no estado s.
$\mathcal{T}^\pi$	Operador distributivo de Bellman.
P^π	Operador de transição $P^\pi : Z \to Z$.

Φ	Operador utilizado para alinhar os suportes de distribuições discretas nos algoritmos C51 e Rainbow.
τ_i	Fração de quantil i de uma distribuição.
$\widehat{\tau}_i$	Ponto médio entre τ_i e τ_{i+1}.
ϕ	Rede neural utilizada na DQN e algoritmos sucessores.
ψ	Rede neural utilizada no IQN e FQF para estimar o valor dos quantis.
φ	Rede neural utilizada no FQF para gerar os quantis.
W_1	Métrica de 1-Wassertein

Sumário

1 Introdução

O aprendizado é um conceito compartilhado e explorado por várias áreas de estudo, como, por exemplo, a pedagogia, psicologia, neurociência e ciência da computação. Embora cada área e autor possuam uma definição formal diferente sobre o que é o aprendizado, é comum que o termo seja exemplificado utilizando o processo de desenvolvimento de uma criança, que aprende conforme observa e interage com o seu meio e memoriza (SUBRAMANIAN et al., 2022). Também é possível elucidar a aprendizagem através de animais, como, por exemplo, um cachorro doméstico, que recebe uma recompensa quando tem uma boa atitude ou uma punição quando toma uma ação considerada indevida por seu tutor.

Em ciência da computação, especificamente na área de inteligência artificial, tais comportamentos de seres vivos e fenômenos naturais são frequentemente estudados de forma a obter-se eficientes algoritmos capazes de ensinar máquinas, sendo esses chamados de algoritmos de aprendizado de máquina (MITCHELL, 1997). Embora o aprendizado por interação seja uma referência no assunto, os algoritmos mais estudados de aprendizado de máquina são os de aprendizado supervisionado, em que um modelo preditivo é construído com base em um conjunto de dados previamente rotulado. Existem também os algoritmos chamados de não supervisionados, que, por sua vez, buscam agrupar dados não rotulados. Por fim, um terceiro paradigma engloba os algoritmos de aprendizado por reforço, que são os únicos que de fato aprendem por interação.

Em (MITCHELL, 1997), é definido que "um programa de computador aprende com a experiência E em relação a alguma classe de tarefas T e medida de desempenho P, se seu desempenho em tarefas T, medido por P, melhora com a experiência E". Nesse sentido, os algoritmos de aprendizado por reforço se diferem dos demais pelo fato de eles não necessitarem de uma base de dados e por, principalmente, possuírem um objetivo fixo em cada tarefa, o que envolve desafios de exploração de ambientes e planejamento (SUTTON; BARTO, 2018).

No ano de 2013, um novo algoritmo, chamado de DQN (*Deep Q Network*) (MNIH et al., 2015), revolucionou a literatura por combinar algoritmos clássicos de aprendizado por reforço com técnicas de *deep learning*, que vinham obtendo importantes resultados em algoritmos de aprendizado supervisionado. A DQN foi o primeiro algoritmo a conseguir jogar diversos jogos de Atari 2600 no mesmo nível de um humano, o que impulsionou uma nova linha de pesquisa: a de algoritmos de aprendizado por reforço avaliados em jogos de Atari 2600.

Nesse contexto, é importante ressaltar que os jogos digitais são apenas ambientes

utilizados para estudo dos algoritmos, que posteriormente podem ser aplicados em outros ambientes, como na indústria, através do controle de máquinas (SCHOETTLER et al., 2020); em sistemas de recomendação (MUNEMASA et al., 2018); em aplicações na área da saúde (ABDELLATIF et al., 2021), para auxílio ao médico; ou até mesmo em vias públicas, na condução de veículos autônomos (KENDALL et al., 2019). Além disso, a crescente expansão tecnológica mundial, juntamente com a possibilidade de aplicação de algoritmos de aprendizado por reforço em tarefas complexas, demanda que os mesmos estejam em constante evolução, o que de fato tem acontecido.

Após a publicação da DQN, surgiram diversos algoritmos derivados, vários desses detalhados no Capítulo 4. Dentre os tais se destaca o FQF (*Fully Parameterized Quantile Function*) (YANG et al., 2019), que obteve importantes resultados no domínio do Atari 2600, mas que não foi capaz de superar um humano especialista em todos os jogos em que foi avaliado. Isso leva a crer que resultados melhores que os atuais ainda podem ser obtidos, dada a capacidade da inteligência artificial em detectar padrões que muitas vezes não são percebidos por um ser humano. Além disso, o FQF não incorpora várias outras técnicas que se mostraram bem sucedidas em trabalhos relacionados.

Sendo assim, considerando a relevância das contribuições de vários trabalhos presentes na literatura e a necessidade de um constante aperfeiçoamento dos métodos, este trabalho busca melhorar o algoritmo FQF através da combinação de três melhorias que foram capazes de impulsionar o desempenho de algoritmos propostos antes do FQF. As melhorias aplicadas e avaliadas em conjunto ao FQF neste trabalho são: o uso de três passos na diferença temporal; a aplicação da abordagem de Munchausen; e o uso da experiência por repetição priorizada. Os algoritmos foram avaliados em cinco jogos de um conjunto de ambientes de *benchmark* chamado de Mini Atari. O uso das três melhorias, quando combinadas ao FQF e avaliadas no Mini Atari, produziu um algoritmo substancialmente melhor do que o FQF, oferecendo assim um algoritmo mais promissor para a aplicação em tarefas de maior complexidade, em que análises como as realizadas aqui são muitas vezes inviáveis computacionalmente.

Este documento está organizado da seguinte forma: no Capítulo 2 é apresentada uma introdução ao aprendizado por reforço, tratando dos principais conceitos utilizados ao longo do trabalho. No Capítulo 3 é apresentada uma introdução a Redes Neurais Artificiais, com destaque para as Redes Neurais Convolucionais, que neste trabalho são utilizadas em conjunto com os métodos de aprendizado por reforço. No Capítulo 4 são apresentados vários trabalhos relacionados, como o próprio FQF e outros trabalhos sucessores da DQN. Na metodologia, presente no Capítulo 5, são oferecidos detalhes a respeito da implementação das melhorias no FQF, além de constar também a descrição sobre os ambientes e métricas de avaliação. Os resultados e discussões estão no Capítulo 6. Por fim, no Capítulo 7, estão as conclusões e perspectivas futuras do trabalho aqui desenvolvido.

2 Introdução ao Aprendizado por Reforço

Este capítulo apresenta os fundamentos do aprendizado por reforço. Os conceitos aqui apresentados foram extraídos do livro (SUTTON; BARTO, 2018), cuja a leitura foi fundamental para a compreensão dos fundamentos teóricos, oferecendo assim uma das principais bases de conhecimentos para o desenvolvimento deste trabalho.

2.1 Problema do Bandido Multi Armado

Para introduzir o aprendizado por reforço é comum a apresentação do problema do bandido multi armado, que é uma nomenclatura geral para uma classe de problemas em que existe apenas um estado e múltiplas ações possíveis. Nesse tipo de problema um agente que inicia sem nenhum conhecimento prévio tem que escolher em cada passo t uma ação a para tomar dentre um conjunto de k ações disponíveis.

Após cada ação a tomada no passo t (A_t), o agente recebe uma recompensa numérica R_t, que indica o quão boa foi a ação A_t. O objetivo do agente é maximizar a soma das recompensas recebidas após um período de tempo com várias ações tomadas.

Apesar de possuírem médias diferentes para cada ação, as recompensas são aleatórias. Sendo assim, não basta experimentar apenas uma vez cada ação para saber qual é a melhor. Uma solução para esse problema é utilizar uma função q que retorna a média das recompensas recebidas após cada ação a, conforme representado na Equação 2.1, em que $\mathbb{1}_{predicado}$ é uma variável que assume o valor 1 quando o predicado é verdadeiro e 0 quando é falso.

$$q_t(a) \doteq \frac{\sum_{i=1}^{t-1} R_i \cdot \mathbb{1}_{A_i=a}}{\sum_{i=1}^{t-1} \mathbb{1}_{A_i=a}} \tag{2.1}$$

O cálculo da função q, conforme apresentado na Equação 2.1, torna-se mais custoso computacionalmente de acordo com o aumento do número de passos e demanda de uma estrutura para armazenar todas as recompensas recebidas, o que o torna ineficiente. Uma solução melhor é realizar o cálculo da média de forma iterativa, em que a função passa a ser armazenada em uma tabela. Para simplificar notações, vamos concentrar em uma única ação, em que Q_n (Equação 2.2) denota a estimativa do valor de tal ação depois que a mesma foi selecionada $n-1$ vezes.

$$Q_n \doteq \frac{R_1 + R_2 + \cdots + R_{n-1}}{n-1} \tag{2.2}$$

Dado o valor de Q_n e da mais recente recompensa R_n, é possível encontrar o valor de Q_{n+1} de forma iterativa, conforme demonstrado na Equação 2.3.

$$\begin{aligned} Q_{n+1} &= \tfrac{1}{n} \textstyle\sum_{i=1}^{n} R_i \\ &= \tfrac{1}{n} \left(R_n + \textstyle\sum_{i=1}^{n-1} R_i \right) \\ &= \tfrac{1}{n} \left(R_n + (n-1)\tfrac{1}{n-1} \textstyle\sum_{i=1}^{n-1} R_i \right) \\ &= \tfrac{1}{n} \left(R_n + (n-1) Q_n \right) \\ &= \tfrac{1}{n} \left(R_n + nQ_n - Q_n \right) \\ &= Q_n + \tfrac{1}{n} \left[R_n - Q_n \right] \end{aligned} \tag{2.3}$$

A equação resultante apresenta uma formulação bem utilizada no aprendizado por reforço. No entanto, nos algoritmos é comum que a fração $\frac{1}{n}$ seja substituída por um parâmetro fixo α. Isso é feito pois muitos dos ambientes são considerados não estacionários, ou seja, sofrem alterações ao longo do tempo. Sendo assim, com um parâmetro fixo é possível valorizar mais as últimas recompensas em relação as mais antigas.

A formulação apresentada concede ao agente uma tabela com estimativas de quão boa cada ação é, mas afinal, como esses valores influenciam a escolha da ação pelo agente? Essa é uma atribuição da política do agente. Um tipo de política a se considerar são as políticas do tipo *greedy*, que escolhem sempre a ação mais promissora a cada instante de acordo com o conhecimento do agente.

A escolha de uma política *greedy* apresenta alguns comportamentos não desejados. Por exemplo, imagine um problema com duas ações disponíveis: A e B. A tabela Q inicia zerada para todas as ações, então o agente escolhe a ação A aleatoriamente e recebe uma recompensa de valor 1. Se as recompensas continuarem sendo positivas, nos próximos passos o agente vai sempre tomar a ação A, sem mesmo ter explorado a ação B, que pode ser melhor. Por outro lado, ao escolher uma ação que aparentemente não é boa, a política pode de fato prejudicar o desempenho do agente.

Os questionamentos apresentados fazem parte de um dilema clássico do aprendizado por reforço: o *exploration vs exploitation*. Na língua portuguesa, ambas palavras são comumente traduzidas como exploração. No entanto elas possuem definições diferentes:

- *exploitation:* indica um comportamento do agente em que o mesmo escolhe sempre a melhor ação de acordo com o seu conhecimento, priorizando as recompensas imediatas.

- *exploration:* indica um comportamento do agente em que o mesmo escolhe ações que a princípio não são as melhores com o objetivo de aumentar o seu conhecimento e maximizar recompensas a longo prazo.

Apresentado o dilema, fica a cargo da política encontrar um meio termo entre *exploration* e *exploitation*. Existem várias propostas de políticas que tentam lidar de formas distintas com o desafio, mas esse ainda é um desafio muito explorado por pesquisadores. Uma das políticas mais utilizadas é a ε-*greedy*, em que o agente age de forma gulosa com probabilidade $1 - \varepsilon$ e de forma aleatória com probabilidade ε. O Algoritmo 1 apresenta um pseudocódigo para solucionar o problema do bandido multi armado utilizando uma tabela Q e uma política ε-*greedy*.

Algoritmo 1 Simples algoritmo para o problema do bandido multi armado

para a=1 **até** k **faça**
  $Q(a) \leftarrow 0$
  $C(a) \leftarrow 0$
fim para
enquanto verdadeiro **faça**
  $A \leftarrow \begin{cases} \underset{a}{argmax}\ \text{Q(a) com probabilidade } 1-\varepsilon \\ \text{ação aleatória com probabilidade } \varepsilon \end{cases}$
  $R \leftarrow$ ação(A)
  $C(A) \leftarrow C(A) + 1$
  $Q(A) \leftarrow Q(A) + \frac{1}{C(A)}[R - Q(A)]$
fim enquanto

2.2 Processo de Decisão de Markov Finito

O Processo de Decisão de Markov (MDP) Finito, pode ser visto como a formalização de um problema de aprendizado por reforço. Um problema modelado via MDP é semelhante com o problema do bandido multi armado apresentado. No entanto, em um MDP existem mais de um estado e portanto o agente precisa de aprender os valores de ação para cada estado. Consequentemente, o desempenho do agente é medido por uma sequência de ações e nem sempre os benefícios são imediatos. Entretanto, um MDP segue a propriedade de Markov, que diz que a probabilidade de um agente atingir os estados futuros é dada somente em função do estado e ação atual, desconsiderando os estados passados.

A Figura 1 ilustra a interação do agente com o ambiente em um MDP. Ambos interagem em uma sequência discreta de passo de tempo $t = 0, 1, 2, 3, ...$. Após cada ação $A_t \in \mathcal{A}(s)$ no estado $S_t \in \mathcal{S}$, o agente observa o novo estado S_{t+1} e o sinal $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$, que indica o quão bom é para o agente estar em S_{t+1}. Apesar do sinal também poder

representar uma punição, ele é chamado de recompensa. Essa sequência de interações dá origem a uma trajetória que se inicia com: $S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, \ldots$.

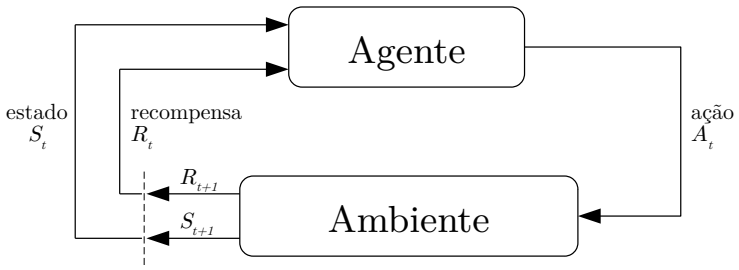


Figura 1 – Interação do agente com o ambiente em um Processo de Decisão de Markov. Extraído e editado de (SUTTON; BARTO, 2018).

Em cada passo t, o objetivo do agente é maximizar o retorno esperado das recompensas recebidas até o passo final T. O retorno é formulado conforme exibido na Equação 2.4.

$$G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T \tag{2.4}$$

Usualmente as recompensas em G_t são ponderadas por um fator de desconto $\gamma \in [0, 1]$, que representa o quanto as futuras recompensas serão consideradas. Com $\gamma = 0$ são consideradas apenas as recompensas imediatas e o agente pode ser considerado "míope". Com $\gamma = 1$ todas as futuras recompensas possuem a mesma importância para o agente e G_t tende ao infinito. A Equação 2.5 apresenta a nova formulação para G_t.

$$\begin{aligned} G_t &\doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \\ &= R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \cdots) \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned} \tag{2.5}$$

Em um MDP Finito os conjuntos de estados, ações e recompensas ($\mathcal{S}$, $\mathcal{A}$ e $\mathcal{R}$) são finitos e as variáveis aleatórias R_t e A_t possuem probabilidades de distribuição que dependem apenas do estado anterior e da ação anterior. A função p na Equação 2.6 representa a dinâmica do MDP, especificamente a probabilidade de se chegar em um estado s' e receber uma recompensa r dado o estado s e a ação a para todo $s', s \in \mathcal{S}$, $r \in \mathcal{R}$ e $a \in \mathcal{A}$.

$$p(s', r|s, a) \doteq Pr\{S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a\} \tag{2.6}$$

Já probabilidade da escolha de cada ação é uma característica do agente, mais especificamente de sua política, e portanto não tem relação com a dinâmica do ambiente. Considerando uma política π, a probabilidade de se tomar uma ação a no estado s é representada por $\pi(a|s)$.

Com a probabilidade de transição de estados p, a definição da política π e do retorno G_t, cuja a esperança é maximizada pelo agente, a função de valor de um estado s sob uma política π é definida pela Equação 2.7, conhecida como equação de Bellman.

$$\begin{aligned} v_\pi(s) &\doteq \mathbb{E}[G_t|S_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma G_{t+1}|S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)[r + \gamma \mathbb{E}[G_{t+1}|S_{t+1} = s']] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')] \end{aligned} \tag{2.7}$$

De forma similar, a definição da função de valor do par estado e ação ocorre conforme posto na Equação 2.8. Pela definição de $q_\pi(s, a)$ observa-se que $v_\pi(s)$ pode ser definida simplesmente por $\sum_a \pi(a|s) q_\pi(s, a)$.

$$\begin{aligned} q_\pi(s, a) &\doteq \mathbb{E}[G_t|S_t = s, A_t = a] \\ &= \mathbb{E}[R_{t+1} + \gamma G_{t+1}|S_t = s, A_t = a] \\ &= \sum_{s'} \sum_r p(s', r|s, a)[r + \gamma \mathbb{E}[G_{t+1}|S_{t+1} = s']] \\ &= \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')] \end{aligned} \tag{2.8}$$

$v_\pi(s)$ e $q_\pi(s, a)$ representam respectivamente a função de valor de estado e estado-ação de acordo com uma política genérica π. Existe também um conceito adicional que diz respeito as políticas ótimas. Uma política π' é dita ótima se $v_{\pi'}(s) \geq v_\pi(s)$ para todo $s \in \mathcal{S}$ e para todo $\pi \in \Pi$, em que Π representa o conjunto das políticas existentes. As Equações 2.9 e 2.10 apresentam a função de valor de estado e de estado-ação para uma política ótima. Elas são conhecidas como as equações de otimalidade de Bellman.

$$v_*(s) = \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')] \tag{2.9}$$

$$q_*(s, a) = \sum_{s',r} p(s', r|s, a)[r + \gamma \max_{a'} q_*(s', a')] \tag{2.10}$$

2.3 Métodos de Monte Carlo

Na seção anterior foi apresentado o Processo de Decisão de Markov, que é utilizado para modelar problemas de aprendizado por reforço. Um problema modelado via MDP pode ser solucionado utilizando-se programação dinâmica (DP). Esses métodos, assim como vários abordados por este trabalho, utilizam tabelas para armazenar os valores de $q_\pi(s, a)$ ou $v_\pi(s)$, os quais são atualizados constantemente. Eles também sempre estimam os valores dos estados atuais baseando-se no valor dos estados sucessores, sendo esse comportamento chamado de *bootstrapping*.

Os métodos de DP são baseados em modelo, ou seja: dependem das equações da dinâmica do MDP. No entanto, são raros os problemas em que se tem o conhecimento de tais equações, o que torna a aplicação de DP no contexto do aprendizado por reforço muito limitada. Este trabalho aborda uma classe de métodos que é considerada *model-free* e buscam estimar os valores de estados por amostragem, sem a necessidade das equações da dinâmica do MDP. Um desses são os métodos de Monte Carlo (MC), que são descritos nesta seção.

Os métodos de MC são aplicáveis apenas em tarefas que são compostas por episódios. Eles buscam aprender a média dos retornos para cada estado e ação e não realizam *bootstrapping*. A ideia central do método se baseia em uma lista que armazena todas as recompensas recebidas pelo agente no episódio e também os estados e ações pelos quais o agente passou. Sendo assim, ao final do episódio, o algoritmo percorre essa lista de trás para frente somando as recompensas e atualizando a tabela do algoritmo. O Algoritmo 2 exibe um pseudocódigo da aplicação de um método MC que só atualiza a tabela baseado na primeira visita do agente em cada estado e ação no episódio. A Figura 2 apresenta uma ilustração de como ocorre a atualização de um estado utilizando métodos de MC, em que círculos vazios representam estados e círculos preenchidos representam ações.

Algoritmo 2 Método *on-policy* de Monte Carlo para primeiras visitas

Inicialize $Q(s,a) \in \mathbb{R}$ arbitrariamente para todo $s \in \mathcal{S}, a \in \mathcal{A}(s)$
Inicialize $C(s,a) \leftarrow 0$ para todo $s \in \mathcal{S}, a \in \mathcal{A}(s)$
Defina π como uma política soft arbitrária baseada em Q

enquanto verdadeiro **faça**
  Gere um episódio baseado em π: $S_0, A_0, R_1, \cdots, S_{T-1}, A_{T-1}, R_T$
  $G \leftarrow 0$
  para $t = T-1$ **até** 0 **faça**
    $G \leftarrow R_{t+1} + \gamma G$
    se o par (S_t, A_t) não aparece em $(S_0, A_0), (S_1, A_1), \cdots, (S_{t-1}, A_{t-1})$ **então**
      $C(S_t, A_t) \leftarrow C(S_t, A_t) + 1$
      $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{1}{C(S_t, A_t)}[G - Q(S_t, A_t)]$
    fim se
  fim para
fim enquanto

Para a convergência do método é necessário que ou o primeiro estado e ação do episódio sejam aleatórios ou a política seja considerada $soft$, que é um termo geral para políticas em que $\pi(a|s) > 0$ para todo $s \in \mathcal{S}$ e para todo $a \in \mathcal{A}(s)$. Caso contrário o agente não explorará o ambiente de forma ampla e o algoritmo não irá convergir.

Aqui, mais uma vez, surge o dilema de *exploration vs exploitation*. Nós desejamos que o agente aprenda uma política ótima, no entanto, é necessário que ele aja de forma não ótima para explorar o ambiente. Uma forma de minimizar esse dilema é através

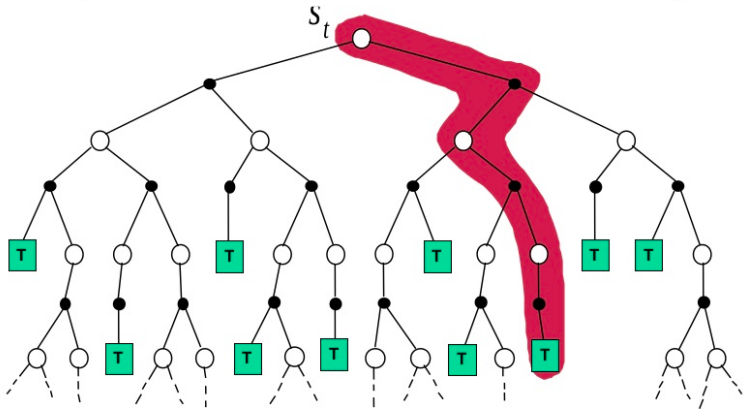


Figura 2 – Diagrama de backup para os Métodos de Monte Carlo. Quando o agente chega ao estado T, inicia-se a atualização retroativa dos estados percorridos anteriormente. Extraído de (SILVER, 2015).

da utilização de algoritmos *off-policy*, que utilizam duas políticas diferentes: uma para aprendizado e outra para comportamento.

Algoritmos *off-policy* possuem maior capacidade de generalização e são aplicáveis em um número maior de tarefas, pois eles podem aprender por dados gerados por um agente que não está em aprendizado, como por exemplo um especialista humano. Entretanto, eles demoram mais tempo para convergir e possuem maior divergência que os algoritmos de política única, chamados de *on-policy*.

A política de aprendizado, doravante representada por π nessa seção, é usualmente determinística e gulosa, buscando se aproximar da política ótima. Já a política de comportamento, representada por b, é estocástica. Para utilizar dados de uma distribuição gerada por uma política b para aprender π é comum a utilização de um termo chamado de importância por amostragem (IS), que calcula a razão entre as probabilidades de se tomar A_t seguindo b e π, conforme demonstrado na Equação 2.11.

$$\rho_{t:T-1} = \prod_{k=t}^{T-1} \frac{\pi(A_k|S_k)}{b(A_k|S_k)} \tag{2.11}$$

Considerando que um estado s pode ser visitado em vários episódios, inclusive com a ação a tomada várias vezes nesse mesmo estado, e considerando uma abordagem que a contagem do passo de tempo t não é zerada a cada novo episódio, utilizando-se de $T(t)$ para representar o último passo de tempo do episódio em que t se enquadra, podemos utilizar o termo $\mathsf{T}(s,a)$ para representar o conjunto de todos os passos de tempos que uma ação a foi tomada no estado s.[1] Logo $\{G_t\}_{t\in\mathsf{T}(s,a)}$ representa o conjunto de retornos do par (s,a) e $\{\rho_{t:T(t)-1}\}_{t\in\mathsf{T}(s,a)}$ suas respectivas importâncias por amostragens, de modo

[1] Na abordagem de primeira visita $\mathsf{T}(s,a)$ seria composto apenas pela primeira ocorrência de s,a em cada episódio.

que a Equação 2.12 representa uma nova forma de estimar $q_\pi(s,a)$ utilizando IS.

$$Q(s,a) \doteq \frac{\sum_{t\in \mathsf{T}(s,a)} \rho_{t:T(t)-1} G_t}{|\mathsf{T}(s,a)|} \tag{2.12}$$

Quando a IS é feita através de uma média aritmética, como apresentado, ela é chamada de amostragem por importância ordinária. Uma outra forma é utilizar a amostragem por importância ponderada, conforme demonstrado na Equação 2.13.

$$Q(s,a) \doteq \frac{\sum_{t\in \mathsf{T}(s,a)} \rho_{t:T(t)-1} G_t}{\sum_{t\in \mathsf{T}(s,a)} \rho_{t:T(t)-1}} \tag{2.13}$$

A abordagem ponderada é considerada enviesada, mas não tem uma variância tão grande pois os pesos estão tanto no numerador quanto no denominador, o que não acontece na abordagem ordinária, que pode possuir alta variância. O Algoritmo 3 exibe o pseudocódigo de um método MC *off-policy* para todas visitas com IS ponderada.

Algoritmo 3 Método *off-policy* de Monte Carlo para todas visitas com amostragem de importância ponderada

Inicialize $Q(s,a) \in \mathbb{R}$ arbitrariamente para todo $s \in \mathcal{S}, a \in \mathcal{A}(s)$
Inicialize $C(s,a) \leftarrow 0$ para todo $s \in \mathcal{S}, a \in \mathcal{A}(s)$
Defina π como uma política gulosa baseada em Q e b como uma política soft arbitrária baseada em Q

enquanto verdadeiro **faça**
  Gere um episódio baseado em b: $S_0, A_0, R_1, \cdots, S_{T-1}, A_{T-1}, R_T$
  $G \leftarrow 0$
  $W \leftarrow 1$
  para $t = T-1$ **até** 0 **faça**
    $G \leftarrow R_{t+1} + \gamma G$
    $C(S_t, A_t) \leftarrow C(S_t, A_t) + W$
    $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)}[G - Q(S_t, A_t)]$
    se $A_t \neq \pi(S_t)$ **então**
      Quebrar loop interno
    fim se
    $W \leftarrow W \frac{1}{b(A_t|S_t)}$
  fim para
fim enquanto

Quando uma ação tomada em conformidade com a política de exploração não possui probabilidade maior do que zero de ser tomada seguindo a política de aprendizado, não é possível seguir com a atualização da tabela Q. Essa característica inviabiliza bastante a utilização de métodos de Monte Carlo *off-policy*, que acabam apenas evoluindo de acordo com as transições que ocorrem no final do episódio.

Além disso, os métodos de MC apenas atualizam as tabelas ao final de cada episódio, o que é um ponto negativo em tarefas com episódios muito longos, pois o agente demora muito para poder aprender com suas experiências. Na próxima seção serão apresentados alguns métodos que buscam distribuir o aprendizado ao longo do episódio.

2.4 Métodos de Diferença Temporal

Assim como os métodos de Monte Carlo e todos os outros vistos daqui para frente, os métodos de Diferença Temporal (TD) não necessitam das probabilidades do MDP. No entanto, eles herdam as características de *bootstrapping* dos algoritmos de DP, atualizando os valores de um estado baseado nos valores dos estados sucessores. Essa característica traz mais agilidade ao algoritmo, pois enquanto nos métodos de MC é necessário aguardar até o fim do episódio para atualizar um estado, com TD basta apenas aguardar o próximo passo.

Os métodos de TD se baseiam na Equação 2.5, que define $G_t = R_{t+1} + \gamma G_{t+1}$. O Algoritmo 4 exibe um pseudocódigo do Sarsa (RUMMERY; NIRANJAN, 1994), que é um método de Diferença Temporal *on-policy*. Aqui, como mencionado na Seção 2.1, o cálculo da média é aproximado por um parâmetro $\alpha \in (0, 1]$.

Algoritmo 4 Sarsa

Inicialize Q(s,a) $\in \mathbb{R}$ arbitrariamente para todo $s \in \mathcal{S}, a \in \mathcal{A}(s)$, exceto que Q(terminal,.) = 0
Defina π como uma política baseada em Q (e.g., ε-greedy)

para cada episódio **faça**
  Inicialize S
  $A \leftarrow \pi(S)$
  repita para cada passo do episódio
    Tome a ação A e observe R, S'
    $A' \leftarrow \pi(S')$
    $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma Q(S', A') - Q(S, A)]$
    $S \leftarrow S'$
    $A \leftarrow A'$
  até que S seja terminal
fim para

Um outro famoso algoritmo de TD é o *Q-learning* (WATKINS, 1989), que por sua vez é *off-policy*. Esse é um algoritmo eficiente e simples, que tem a vantagem de não demandar o uso da amostragem por importância. O pseudocódigo do *Q-learning* está exibido no Algoritmo 5.

Embora o Sarsa e o *Q-learning* sejam bem parecidos, a única diferença não está apenas no uso do operador máximo no *Q-learning*, que o torna *off-policy*. Note que o Sarsa

Algoritmo 5 *Q-learning*

Inicialize Q(s,a) $\in \mathbb{R}$ arbitrariamente para todo $s \in \mathcal{S}, a \in \mathcal{A}(s)$, exceto que Q(terminal,.) = 0
Defina π como uma política baseada em Q (e.g., ε-greedy)

para cada episódio **faça**
  Inicialize S
  repita para cada passo do episódio
    $A \leftarrow \pi(S)$
    Tome a ação A e observe R, S'
    $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma max_a Q(S',a) - Q(S,A)]$
    $S \leftarrow S'$
  até que S seja terminal
fim para

precisa de escolher a ação A' antes de atualizar o valor do par (S, A), o que não acontece no *Q-learning*, que escolhe A' após a atualização. Um terceiro algoritmo TD é o *Expected Sarsa* (JOHN, 1994). Esse algoritmo é construído da mesma forma que o *Q-learning*, exceto que a atualização dos valores ocorre conforme exibido na Equação 2.14.

$$Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \sum_a \pi(a|S')Q(S',a) - Q(S,A)] \tag{2.14}$$

O *Expected Sarsa* pode ser construído tanto para ser um algoritmo *on-policy* quanto *off-policy*, a depender se as políticas utilizadas para comportamento e aprendizado são diferentes ou não. Ele tem esse nome pois em sua versão *on-policy* caminha na mesma direção que o Sarsa, mas é considerado mais estável por não ser tão sensível a ações com baixa probabilidade de serem escolhidas.

Um ponto em comum desses 3 algoritmos é que todos eles utilizam o operador máximo na política, que é muito sensível a ruídos. Uma forma de mitigar esses ruídos é utilizar a técnica de *Double Learning*, que busca aprender duas estimativas independentes: Q_1 e Q_2. O pseudocódigo no Algoritmo 6 apresenta o *Double Q-learning* (HASSELT, 2010). A técnica de *Double Learning* pode ser também aplicada em outros algoritmos.

Os algoritmos apresentados nessa seção podem ser descritos como Métodos *model-free* tabulares de Diferença Temporal de um passo. Todos métodos de TD de um passo são também chamados de TD(0).

2.5 Métodos de Diferença Temporal de *n* Passos

Os métodos de Monte Carlo apenas atualizam suas estimativas de estado ao final de cada episódio. Por outro lado, os métodos de TD(0) atualizam os estados logo após

Algoritmo 6 *Double Q-learning*

Inicialize $Q_1(s,a)$ e $Q_2(s,a) \in \mathbb{R}$ arbitrariamente para todo $s \in \mathcal{S}, a \in \mathcal{A}(s)$, exceto que $Q(terminal,.) = 0$
Defina π como uma política baseada em $Q_1 + Q_2$ (e.g., ε-greedy)

para cada episódio **faça**
  Inicialize S
  repita para cada passo do episódio
    $A \leftarrow \pi(S)$
    Tome a ação A e observe R, S'
    se moeda aleatória = cara **então**
      $Q_1(S,A) \leftarrow Q_1(S,A) + \alpha[R + \gamma Q_2(S', argmax_a(Q_1(S',a)) - Q_1(S,A)]$
    se não
      $Q_2(S,A) \leftarrow Q_2(S,A) + \alpha[R + \gamma Q_1(S', argmax_a(Q_2(S',a)) - Q_2(S,A)]$
    fim se
    $S \leftarrow S'$
  até que S seja terminal
fim para

cada ação. Nesta seção são descritos os métodos de Diferença Temporal de n passos, que atuam como um ponto de equilíbrio entre MC e TD(0).

Anteriormente vimos que G_t pode ser definido como:

$$G_t = R_{t+1} + \gamma G_{t+1} \tag{2.15}$$

No entanto, podemos representar mais recompensas de forma explícita, como na Equação 2.16 que representa dois passos explícitos, e na Equação 2.17 que representa n passos. Essas transformações na forma de representar o retorno, com uma postergação do *bootstrapping*, são a diferença entre os métodos de TD de um passo para os de n passos.

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 G_{t+2} \tag{2.16}$$

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n G_{t+n} \tag{2.17}$$

A Figura 3 representa bem a diferença prática entre métodos MC, TD(0) e TD de n passos. É exibido um ambiente tabular, em que cada quadrículo representa um estado. Consideremos que todos estados estão inicialmente zerados e que as recompensas são sempre zero em todos os estados se não aquele marcado por G, onde o episódio se encerra e o agente recebe uma recompensa positiva. À esquerda é exibido o caminho traçado pelo agente. Considerando um método de MC, todos os estados pelo qual o agente passou seriam atualizados de acordo com a recompensa recebida ao fim do episódio. Com TD(0), apenas o último estado seria atualizado. Uma eventual diferença temporal de dez passos, conforme representado à direita na figura, atualizaria os valores dos últimos dez estados

percorridos, aproveitando mais do conhecimento obtido no episódio, mas sem utilizar o método de MC, que como visto tem pontos negativos significantes.

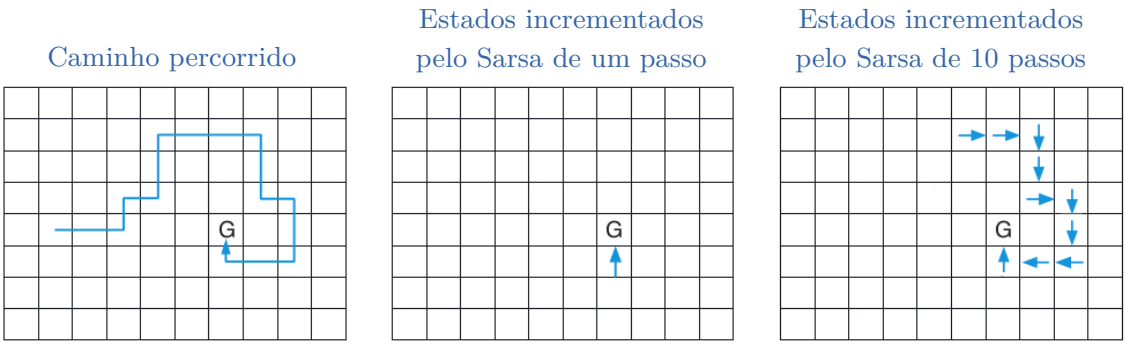


Figura 3 – Comparação entre métodos de diferença temporal de um e dez passos em um ambiente inicialmente zerado com recompensa apenas no último estado. Extraído e editado de (SUTTON; BARTO, 2018).

Na Figura 4 são apresentados três diagramas de backup para algoritmos de TD com quatro passos. Nos diagramas os pontos pretos se referem a ações, os círculos vazios a estados e as setas representam transições. O primeiro algoritmo apresentado é uma versão do Sarsa com n passos. Esse algoritmo pode ser representado tanto na forma *on-policy* quanto *off-policy*, sendo que na última opção é necessário utilizar a amostragem por importância, conforme apresentado nos métodos de Monte Carlo. O terceiro algoritmo é a versão n passos do *Expected Sarsa*, em que nota-se que no último passo são contabilizadas todas as ações possíveis no estado.

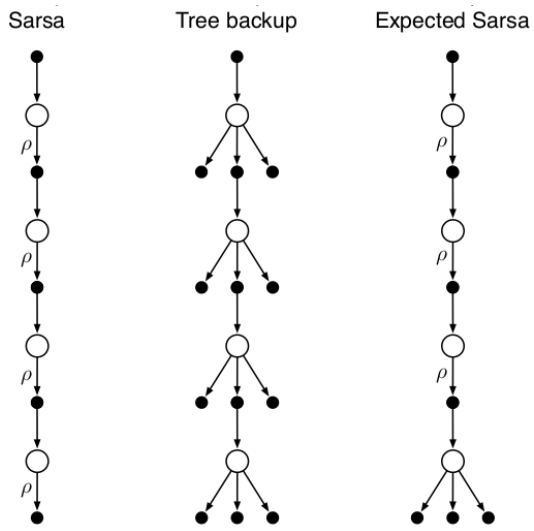


Figura 4 – Diagramas de backup para 3 algoritmos de diferença temporal de $n = 4$ passos. Extraído e editado de (SUTTON; BARTO, 2018).

O segundo diagrama presente na Figura 4 representa o algoritmo *Tree Backup* (PRECUP et al., 2000), que considera o valor de todas as ações nos estados pelo qual o agente passou e não demanda o uso da amostragem por importância. Esse algoritmo

realiza *bootstrapping* em cada ação não tomada e contabiliza as demais através das recompensas. O retorno com apenas um passo ($G_{t:t+1}$) para o *Tree Backup* ocorre da mesma forma como no algoritmo *Expected Sarsa*:

$$G_{t:t+1} \doteq R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q_t(S_{t+1}, a) \tag{2.18}$$

Com dois passos, o retorno do *Tree Backup* é definido como:

$$\begin{aligned} G_{t:t+2} \doteq \; & R_{t+1} + \gamma \textstyle\sum_{a \neq A_{t+1}} \pi(a|S_{t+1}) Q_{t+1}(S_{t+1}, a) \\ & + \gamma \pi(A_{t+1}|S_{t+1}) (R_{t+2} + \gamma \textstyle\sum_a \pi(a|S_{t+2}) Q_{t+1}(S_{t+2}, a)) \\ = \; & R_{t+1} + \gamma \textstyle\sum_{a \neq A_{t+1}} \pi(a|S_{t+1}) Q_{t+1}(S_{t+1}, a) + \gamma \pi(A_{t+1}|S_{t+1}) G_{t+1:t+2} \end{aligned} \tag{2.19}$$

e assim recursivamente. Do mesmo modo que o *Expected Sarsa*, o *Tree Backup* pode ser tanto *off-policy* quanto *on-policy*

2.6 Métodos Baseados em Aproximadores de Função

Os métodos apresentados até o momento são chamados de métodos tabulares, pois utilizam tabelas para armazenar os valores dos estados e ações. Esses métodos nos fornecem conceitos importantes, mas são aplicáveis apenas em problemas de pequena dimensão, pois na medida que o número de estados e ações cresce, fica inviável armazenar e preencher toda a tabela. Além disso, em muitos problemas do mundo real a definição de estado não é muito clara, pois o que o agente tem contato é com um conjunto de *features* (i.e. características do estado), geralmente extraído através de sensores e/ou câmeras.

Uma alternativa para esses problemas é utilizar aproximadores de função, em que se tem um conjunto de pesos $\mathbf{w} \in \mathbb{R}^d$ que se associam às *features* do estado para estimar o valor do mesmo. Esses pesos precisam ser ajustados, de forma que se tem $\hat{v}(s, \mathbf{w}) \approx v_\pi(s)$ e $\hat{q}(s, a, \mathbf{w}) \approx q_\pi(s, a)$, em que $\hat{v}$ e $\hat{q}$ são os aproximadores, que podem ser por exemplo uma função linear, uma rede neural ou qualquer outro aproximador de função que seja adequado para aprendizado online.

Usualmente o número de pesos do modelo é bem menor que o número de estados que se deseja estimar, de forma que o ajuste de um único peso modifica a estimativa de vários estados. Essa característica torna o ajuste dos pesos uma tarefa difícil, mas também permite que o modelo tenha capacidade de generalização, pois consegue estimar o valor de estados sem mesmo ter visitado-os antes.

Existem diversas formas de se ajustar o vetor de pesos, sendo que uma delas é através do Gradiente Descendente Estocástico (SGD), que são métodos de Gradiente Descendente em que o ajuste dos pesos ocorre amostra por amostra. Podemos definir

$\mathbf{w} \doteq (w_1, w_2, \cdots, w_d)^T$ e $\hat{v}(s, \mathbf{w})$ como uma função diferenciável de $\mathbf{w}$, que vai sendo atualizado a cada passo t, sendo portanto utilizada a notação $\mathbf{w}_t$. A atualização dos pesos pode ser feita através da Equação 2.20:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha[v_\pi(S_t) - \hat{v}(s, \mathbf{w})]\nabla\hat{v}(s, \mathbf{w}) \tag{2.20}$$

em que $\nabla f(\mathbf{w})$ representa o vetor coluna das derivadas parciais da função f em relação aos componentes do vetor $\mathbf{w}$, conhecido como o gradiente de f em relação a $\mathbf{w}$, conforme exibido na Equação 2.21.

$$\nabla f(\mathbf{w}) \doteq \left(\frac{\partial f(\mathbf{w})}{\partial w_1}, \frac{\partial f(\mathbf{w})}{\partial w_2}, \cdots, \frac{\partial f(\mathbf{w})}{\partial w_d}\right)^T \tag{2.21}$$

No processo de aprendizagem o valor de $v_\pi(S_t)$ é geralmente desconhecido e o que se tem é uma estimativa a cada passo, como por exemplo G_t nos métodos de Monte Carlo e $R + \gamma\hat{v}(s', \mathbf{w})$ em TD(0). Nos métodos que utilizam *bootstrapping*, como os de Diferença Temporal, existe uma particularidade pois a estimativa depende também de $\mathbf{w}$, motivo esse que faz com que eles sejam considerados semi gradiente.

Dentre os modelos aproximadores de função, os mais clássicos são as funções lineares. Considerando um vetor de *features* do estado $\mathbf{x}(s) \doteq (x_1(s), x_2(s), \cdots, x_d(s))^T$, de mesma dimensão que $\mathbf{w}$, a estimativa do estado por um modelo linear pode ser realizada conforme demonstrado na Equação 2.22. No caso da utilização de SGD com modelos lineares, temos também que $\nabla\hat{v}(s, \mathbf{w}) = \mathbf{x}(s)$.

$$\hat{v}(s, \mathbf{w}) \doteq \mathbf{w}^T\mathbf{x}(s) = \sum_{i=1}^{d} w_i x_i(s) \tag{2.22}$$

Funções lineares são simples e oferecem garantias de convergência, conforme demonstrado em (SUTTON; BARTO, 2018). No entanto, elas possuem baixa capacidade de relacionamento das *features*. Por esse motivo, é necessário que seja realizado um pré-processamento das mesmas antes da aplicação na função. Isso pode ser realizado através da aplicação de bases polinomiais, bases de fourrier (KONIDARIS et al., 2011), *coarse coding* (HINTON, 1984), *tile coding* (WATKINS, 1989), dentre outras.

Nas aplicações de controle, em que é necessário estimar $q_\pi(s, a)$, existem duas alternativas para os modelos lineares: ou o número da ação é inserido na entrada do modelo, de forma que é necessário processar a saída várias vezes modificando a ação de entrada, ou os vetores $\mathbf{w}$ e $\mathbf{x}$ são empilhados $|\mathcal{A}|$ vezes, de forma que o modelo terá uma estimativa para cada estado, o que é equivalente a treinar um modelo para cada ação. A segunda alternativa é considerada mais adequada. O Algoritmo 7 apresenta um pseudocódigo do Sarsa utilizando funções lineares.

Outros aproximadores de função também podem ser utilizados. Através de redes neurais com funções de ativação não lineares, por exemplo, o pré processamento é menos

Algoritmo 7 Sarsa Semi Gradiente.

Inicialize $\mathbf{w}$ arbitrariamente (e.g., $\mathbf{w} = 0$)
Defina Uma função de estado ação diferenciável $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \to \mathbb{R}$
Defina π como uma política baseada em $\hat{q}$ (e.g., ε-greedy)

para cada episódio **faça**
  Inicialize S
  $A \leftarrow \pi(S)$
  repita para cada passo do episódio
    Tome a ação A e observe R, S'
    se S' é terminal **então**
      $\mathbf{w} = \mathbf{w} + \alpha[R - \hat{q}(S, A, \mathbf{w})]\nabla\hat{q}(S, A, \mathbf{w})$
    se não
      $A' \leftarrow \pi(S')$
      $\mathbf{w} = \mathbf{w} + \alpha[R + \gamma\hat{q}(S', A', \mathbf{w}) - \hat{q}(S, A, \mathbf{w})]\nabla\hat{q}(S, A, \mathbf{w})$
      $A \leftarrow A'$
    fim se
    $S \leftarrow S'$
  até que S seja terminal
fim para

relevante e também não é necessário realizar o empilhamento das *features*, pois o próprio modelo já oferece a possibilidade de múltiplas saídas, conforme detalhado no Capítulo 3. Além disso, também é possível utilizar algoritmos *off-policy* com aproximadores de função. No entanto, eles podem gerar algoritmos muito instáveis, principalmente se aplicados métodos que utilizam *bootstrapping*, formando a combinação chamada em (SUTTON; BARTO, 2018) por "O Tríade Mortal". Alguns algoritmos apresentados no Capítulo 4 realizam tal combinação e aplicam outras técnicas para minimizar a variância do modelo.

2.7 Traços de Elegibilidade

Na Seção 2.5 os Métodos de Diferença Temporal de n passos foram apresentados como pontos de equilíbrio entre TD(0) e MC. Nesta seção é apresentado uma classe de métodos chamados de TD(λ) que possuem o mesmo propósito, mas que são considerados mais eficientes computacionalmente e também mais elegantes.

O TD(λ) busca estimar um retorno representado por G_t^λ, que corresponde a uma média ponderada por um parâmetro $\lambda \in [0, 1]$ de todos os possíveis retornos de n passos calculáveis de t até o fim do episódio. A Equação 2.23 apresenta uma formulação de G_t^λ e a Figura 5 o diagrama de backup do algoritmo TD(λ). É importante destacar que é o parâmetro λ que define o quanto o algoritmo se aproxima de um método Monte Carlo ou do TD(0). Com $\lambda = 0$ o algoritmo é equivalente à Diferença Temporal de um passo, motivo esse que esses métodos são conhecidos como TD(0). Com $\lambda = 1$ o algoritmo é

equivalente a um Método de Monte Carlo.

$$G_t^{\lambda} = (1-\lambda) \sum_{n=1}^{T-t-1} \lambda^{n-1} G_{t:t+n} + \lambda^{T-t-1} G_t \tag{2.23}$$

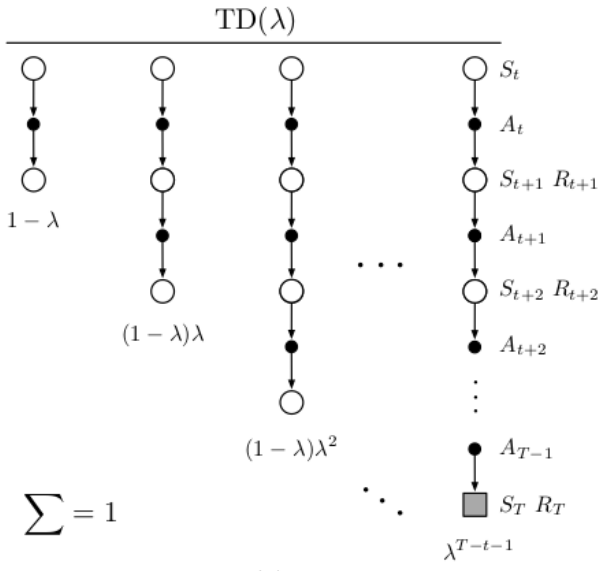


Figura 5 – Diagrama de Backup para o TD(λ). Extraído de (SUTTON; BARTO, 2018).

O algoritmo que aplica a Equação 2.23 é chamado de *Offline* λ-*return*, pois realiza a atualização do modelo apenas no final do episódio, assim como nos Métodos MC. Já o TD(λ) busca aproximar G_t^{λ} por uma visão que leva sempre em consideração os estados passados, sem a necessidade de postergar muito o aprendizado.

O elemento chave do TD(λ) é um vetor $\mathbf{z} \in \mathbb{R}^d$ chamado de traço de elegibilidade, que acumula a soma dos vetores gradientes ao longo do tempo. O Algoritmo 8 exibe o pseudocódigo do TD(λ) semi gradiente, que estima $\widehat{v} \approx v_{\pi}$. Também existem as versões de controle, em que se estima q_{π}.

Observa-se no algoritmo que o parâmetro γ define não só o fator de desconto, como também um fator de esquecimento, que estabelece o quanto as experiências passadas influenciam nas modificações do vetor $\mathbf{w}$. Também é interessante destacar o parâmetro λ e seu papel de equilibrar o quanto o algoritmo de aproxima de MC ou TD(0). Com $\lambda = 1$, por exemplo, obtemos uma versão online do Método de Monte Carlo.

Existem versões mais avançadas do TD(λ), que podem ser encontradas em (SUTTON; BARTO, 2018).

Algoritmo 8 TD(λ) Semi Gradiente para estimar $\widehat{v} \approx v_\pi$.

Entrada: Uma política π a ser avaliada.
Defina Uma função de estado diferenciável $\widehat{v} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \to \mathbb{R}$
Inicialize $\mathbf{w}$ arbitrariamente (e.g., $\mathbf{w} = 0$)

para cada episódio **faça**
    Inicialize S
    $\mathbf{z} \leftarrow \mathbf{0}$
    repita para cada passo do episódio
        $A \leftarrow \pi(S)$
        Tome a ação A e observe R, S'
        $\mathbf{z} \leftarrow \gamma\lambda\mathbf{z} + \nabla\widehat{v}(S, \mathbf{w})$
        se S' é terminal **então**
            $\delta \leftarrow R - \widehat{v}(S, \mathbf{w})$
        se não
            $\delta \leftarrow R + \gamma\widehat{v}(S', \mathbf{w}) - \widehat{v}(S, \mathbf{w})$
        fim se
        $\mathbf{w} \leftarrow \mathbf{w} + \alpha\delta\mathbf{z}$
        $S \leftarrow S'$
    até que S seja terminal
fim para

3 Introdução a Redes Neurais Artificiais

As Redes Neurais Artificiais (ANNs) são estruturas de aprendizado que foram criadas baseadas no sistema nervoso de um animal, sendo uma modelagem da forma como um cérebro processa determinada função (HAYKIN, 2009). Elas possuem esse nome pois são compostas por unidades interligadas chamadas de neurônios.

Os neurônios em uma ANN são ativados a partir de um vetor de entradas $\mathbf{x}$, que é ponderado por um vetor de pesos $\mathbf{w}$. Feito isso os produtos são somados com um valor de limiar, denominado de bias (c), e depois passam por uma função de ativação $g(u)$. A Figura 6 ilustra a realização desse processo.

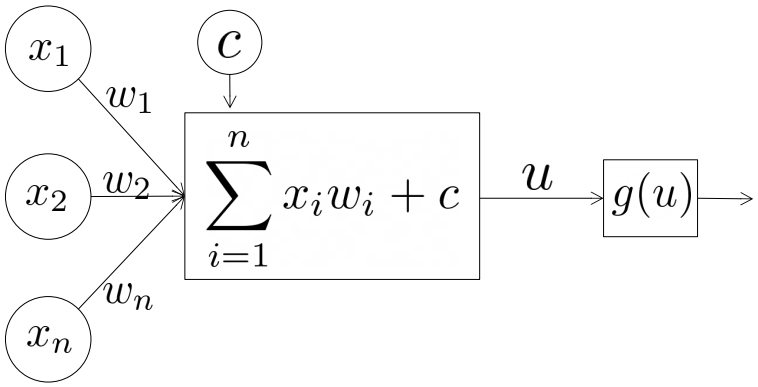
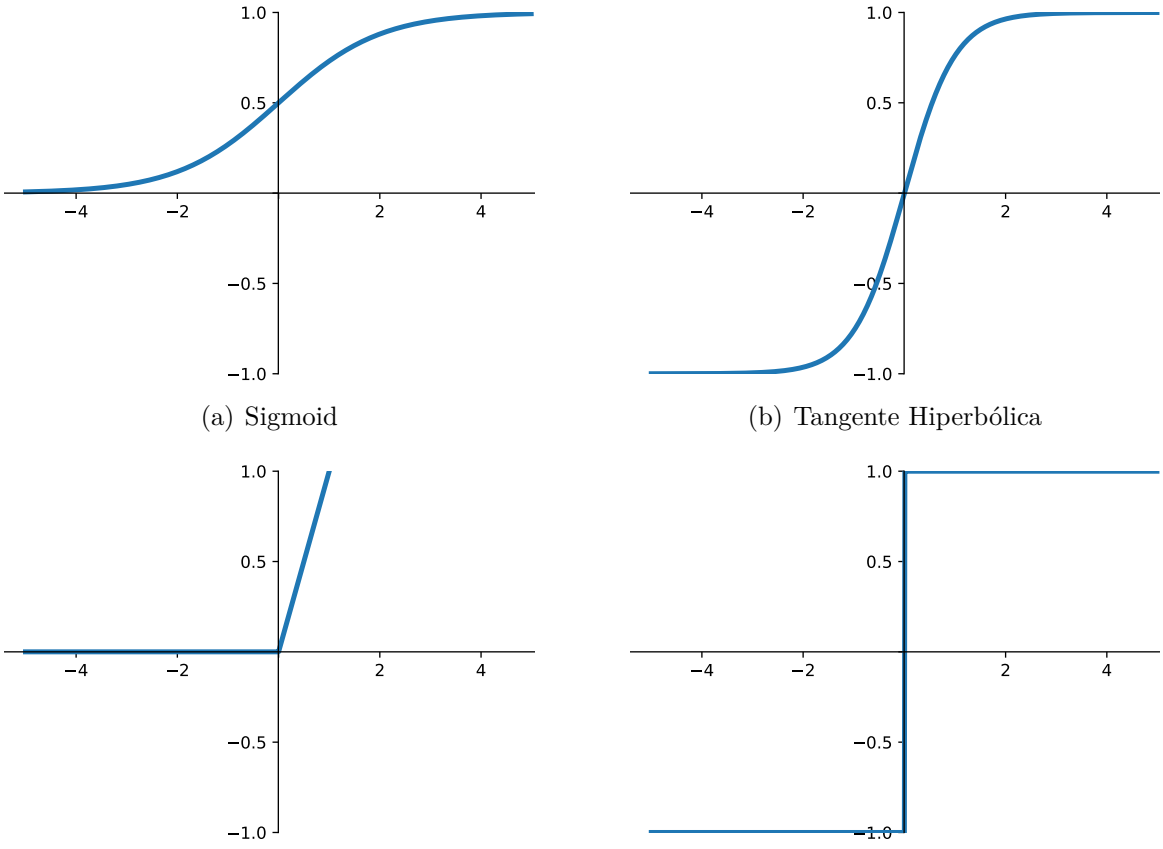


Figura 6 – Representação de um neurônio artificial.

Utilizando diferentes funções $g(u)$ é possível gerar diferentes modelos. Algumas das funções mais utilizadas estão representadas na Figura 7. As funções sigmoid e tangente hiperbólica são funções não lineares com curvas achatadas no início e no final, o que indica que o valor de saída de grandes valores de entrada tendem a ser bem próximos. Já a ReLU (Unidade Linear Retificada) gera uma função linear para valores de entrada maiores do que zero, enquanto para o valor zero e demais valores negativos, o resultado da função é sempre zero. A ReLU tem sido bastante utilizada em redes com camadas convolucionais (RUSSELL; NORVIG, 2021), que são apresentadas posteriormente.

Quando a rede é composta por um único neurônio utilizando uma função de ativação de *Threshold*, como a representada na Figura 7(d), a rede é chamada de *perceptron* (MITCHELL, 1997). Com o devido ajuste dos pesos, um perceptron é capaz de representar funções booleanas simples como AND, OR, NAND e NOR. Porém, não é capaz de representar o XOR, por exemplo. Isso acontece pois o XOR é uma função que não é linearmente separável (MITCHELL, 1997).

Para representar problemas não lineares é necessário utilizar multi camadas. Nesse



(a) Sigmoid (b) Tangente Hiperbólica

(c) ReLU (d) Threshold

Figura 7 – Representação de quatro modelos de funções de ativação.

modelo, os vários neurônios de uma camada se interligam com os neurônios da camada seguinte, aumentando assim o espaço de hipóteses que a rede pode representar. A Figura 8 ilustra uma Perceptron de Multi Camadas (MLP), em que cada círculo representa um neurônio.

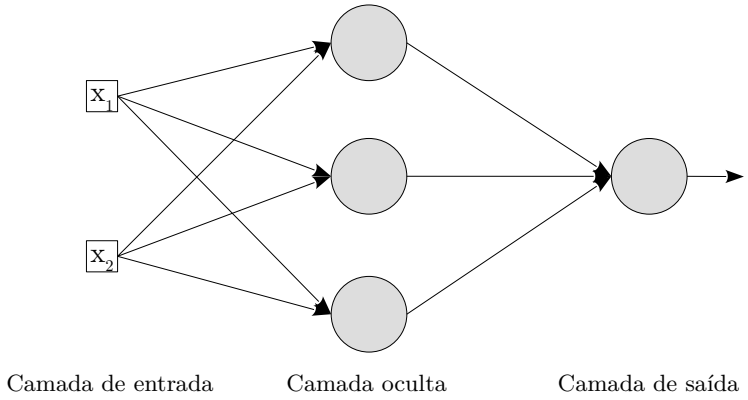


Figura 8 – Representação de uma perceptron de multi camadas, com camada oculta de três unidades.

Quando são utilizadas várias camadas, a rede neural é considerada como um método de *Deep Learning* (RUSSELL; NORVIG, 2021). Essa classe de métodos começou a se

destacar quando foi utilizada em redes neurais convolucionais para o reconhecimento de dígitos manuscritos na década de 90 (LECUN et al., 1995). No entanto, a partir do ano de 2011, que esses métodos ganharam mais projeções, sendo utilizados para o reconhecimento visual de objetos e de áudio (RUSSELL; NORVIG, 2021).

3.1 Redes Neurais Convolucionais

Uma MLP pode ser utilizada para vários tipos de entradas, inclusive imagens. No entanto, uma imagem não pode ser interpretada simplesmente como um vetor unidimensional de pixels, uma vez que a adjacência dos pixels é muito importante para a detecção de padrões nesse tipo de dado. Sendo assim, para utilizar uma MLP em aplicações que oferecem imagens, é necessário extrair características das mesmas inicialmente, o que pode ser feito através de técnicas de processamento digital de imagens. Embora essa combinação possa gerar ótimos resultados, ela apresenta uma limitação, pois para que sejam extraídas as características da imagem, é necessário que o pesquisador possua um bom conhecimento do ambiente de aplicação, assim como ocorreu em (TESAURO, 1995) no jogo TD-Gammon.

Uma forma de fazer com que a própria rede neural seja capaz de extrair características de cada imagem - que nada mais é que uma matriz de pixels - é através do uso de camadas convolucionais, formando assim uma Rede Neural Convolucional (CNN). A convolução em uma imagem pode ser definida como o processo de aplicação de um kernel (aqui chamado de filtro), sobre vários subconjuntos de pixels da imagem (RUSSELL; NORVIG, 2021), sendo que a aplicação ocorre através do produto escalar. Nesse caso, a rede deve ser capaz de aprender os filtros aplicados na imagem, de forma a extrair características relevantes.

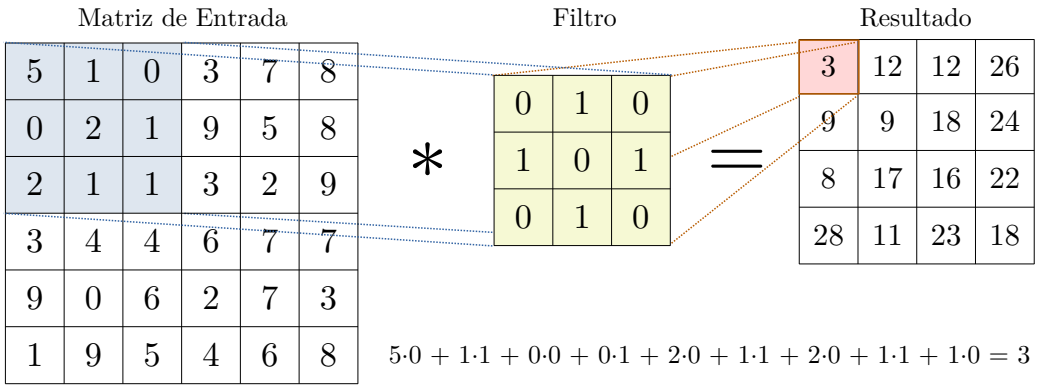


Figura 9 – Ilustração da aplicação de uma convolução em uma matriz de dimensão 6x6, com filtro de dimensão 3x3, *stride* = 1 e *padding* = 0.

Na Figura 9 é apresentada a ilustração de uma operação de convolução em uma matriz de duas dimensões. Note que a matriz de saída é menor que a matriz de entrada.

Isso acontece em função de dois parâmetros: o *stride* (na Figura 9 definido com valor um), determina o tamanho do pulo na navegação do filtro pela imagem. Já o *padding* (na Figura 9 definido como zero), determina a largura de uma possível borda de pixels extras que pode ser adicionada na matriz de entrada. No exemplo da Figura 9, se fosse utilizada uma camada de *padding*, a matriz de saída teria a mesma dimensão da matriz de entrada, uma vez que o tamanho do *stride* é igual a um. Essa situação é ilustrada na Figura 10.

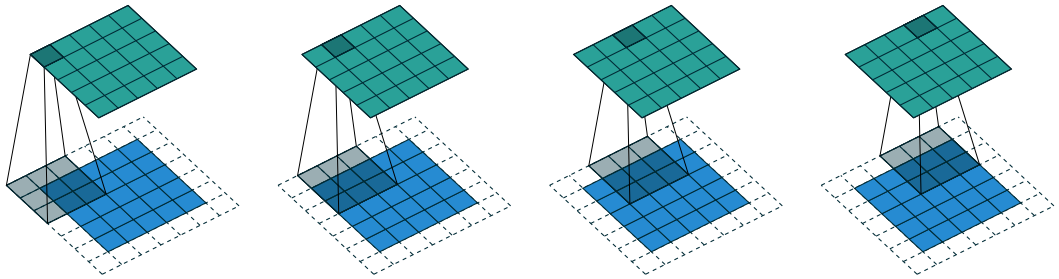


Figura 10 – Representação do deslocamento de um filtro de dimensão 3x3 em uma matriz de dimensão 5x5 utilizando $stride = 1$ e $padding = 1$. Extraído de (DUMOULIN; VISIN, 2016).

Camadas convolucionais podem ainda serem seguidas por camadas de *pooling*. Diferentemente da convolução, nesse tipo de operação não é necessário aprender pesos para os filtros, pois eles são definidos por medidas agregadoras estáticas, como por exemplo a aplicação de filtros de média, máximo e mínimo. Ao utilizar um *stride* igual ao tamanho do filtro na dimensão em questão, a camada de *pooling* pode ser utilizada para reduzir a dimensionalidade dos dados (RUSSELL; NORVIG, 2021). A Figura 11 exibe um exemplo de arquitetura de rede convolucional, que possui duas camadas ocultas com convolução e *pooling* cada uma, seguidas por mais uma camada oculta, mas com neurônios altamente conectados.

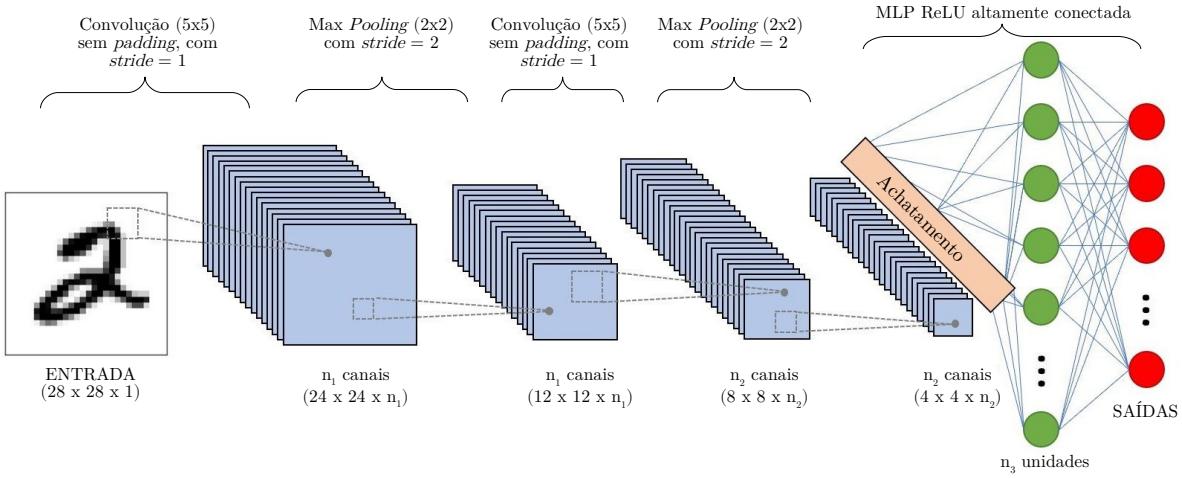


Figura 11 – Representação da arquitetura de uma rede neural convolucional utilizada para classificar dígitos. Extraída e modificada de (SAHA, 2018).

3.2 Treinamento de Redes Neurais

Neste capítulo foram apresentados os conceitos elementares relacionados à composição de uma rede neural artificial, desde o neurônio até a arquitetura de uma CNN. No entanto, independente da arquitetura da rede neural, é necessário que seja aplicado um algoritmo para ajustar os pesos da rede, que é chamado de otimizador. Vamos iniciar apresentando o algoritmo de Gradiente Descendente (GD) para perceptrons de única camada sem função de ativação, o que equivale a um aproximador linear simples.

No GD, o vetor de pesos $\mathbf{w}$ é iniciado aleatoriamente e o ajuste dos valores ocorre de acordo com uma função de erro entre o valor estimado pela rede (o_d) e o valor desejado (p_d) sobre um conjunto de D pares de amostras. A Equação 3.1 exibe um exemplo de função de erro que é bastante utilizada (MITCHELL, 1997).

$$E(\mathbf{w}) = \frac{1}{2} \sum_{d \in D} (p_d - o_d)^2 \tag{3.1}$$

A equação do erro é dada em função de $\mathbf{w}$, pois são os pesos que determinam o valor de saída da rede. Dada a formulação, o ajuste dos pesos pode ser feito pelo gradiente do erro, conforme exibido na Equação 3.2, em que η é a taxa de aprendizado e o símbolo ∇ indica o gradiente. Note que o gradiente é multiplicado por um sinal negativo, tendo em vista que o objetivo é minimizar o erro.

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla E(\mathbf{w}) \tag{3.2}$$

Considerando a Equação 2.21, que permite encontrar o gradiente como um vetor de derivadas parciais da função em relação ao vetor, podemos reescrever a Equação 3.2 por componentes, conforme exibido abaixo:

$$w_i \leftarrow w_i - \eta \frac{\partial E}{\partial w_i} \tag{3.3}$$

Aplicando as derivadas parciais, conforme demonstrado na Equação 3.4, é possível representar a atualização dos pesos por componente como exibido na Equação 3.5. Note que a derivação aplicada considera a saída da rede dada por $\mathbf{w}^T\mathbf{x}$, que é o que ocorre em neurônios com funções de ativação que apenas repassam a entrada. Logo, para outros tipos de função de ativação, a equação de atualização dos pesos é diferente.

$$\begin{aligned}
\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i}\frac{1}{2}\sum_{d\in D}(p_d - o_d)^2 \\
&= \frac{1}{2}\sum_{d\in D}\frac{\partial}{\partial w_i}(p_d - o_d)^2 \\
&= \frac{1}{2}\sum_{d\in D}2(p_d - o_d)\frac{\partial}{\partial w_i}(p_d - o_d) \\
&= \sum_{d\in D}(p_d - o_d)\frac{\partial}{\partial w_i}(p_d - \mathbf{w}^T\mathbf{x}) \\
\frac{\partial E}{\partial w_i} &= \sum_{d\in D}(p_d - o_d)(-x_{id})
\end{aligned} \tag{3.4}$$

$$w_i \leftarrow w_i + \eta\sum_{d\in D}(p_d - o_d)x_{id} \tag{3.5}$$

O GD é um algoritmo eficiente, mas ele pode ser lento devido à necessidade de se realizar atualizações dos pesos várias vezes até eles convergirem, além de ser sensível a mínimos locais (MITCHELL, 1997). Um outro algoritmo também aplicável a redes neurais de única camada é o Gradiente Descendente Estocástico (SGD), que foi apresentado na Seção 2.6 [1]. O SGD realiza as atualizações baseadas em cada par de amostras, podendo ser utilizado tanto para o aprendizado *online* quanto *offline*. Esse algoritmo é mais rápido e mais resiliente a mínimos locais, mas possui a desvantagem de produzir alta variância nas atualizações. Uma forma de combinar as vantagens dos dois métodos é utilizar batchs, em que os pares de amostras são separados em grupos menores (chamados de batchs), permitindo que o algoritmo GD seja aplicado dentro de cada batch, e não em todo o conjunto de amostras.

3.2.1 Algoritmo *Backpropagation*

Para redes com múltiplas camadas é necessário realizar modificações nos algoritmos GD e SGD de forma que o erro seja propagado para as camadas ocultas. Considerando que em redes multi camadas a rede neural pode ter múltiplas saídas, é necessário reescrever a função de erro para que a mesma contabilize as diferentes saídas, conforme consta na Equação 3.6.

$$E(\mathbf{w}) = \frac{1}{2}\sum_{d\in D}\sum_{k\in \text{Saídas}}(p_{kd} - o_{kd})^2 \tag{3.6}$$

No Algoritmo 9 é exibido um pseudo código do SGD extraído e editado de (MITCHELL, 1997) em uma versão com *backpropagation*, em que o cálculo do gradiente foi

[1] Note que na Seção 2.6 o cálculo de erro é baseado na diferença temporal.

realizado baseando-se em funções de ativação do tipo sigmoid. No algoritmo, cada amostra é um par da forma $(\mathbf{x}, \mathbf{p})$, em que $\mathbf{x}$ é o vetor de entradas da rede e $\mathbf{p}$ é o vetor de saídas que se deseja obter. A entrada de um neurônio j proveniente de um neurônio i é dada por x_{ji}, com seu respectivo peso representado por w_{ji}. No Algoritmo 9 aparece também o símbolo δ, que representa o gradiente local de um determinado neurônio.

Algoritmo 9 Versão do SGD com Backpropagation para redes com função sigmoid.

função BACKPROPAGATION($amostras, \eta$)
  repita
    para cada $(\mathbf{x}, \mathbf{p}) \in$ amostras **faça**
      Propague a entrada $\mathbf{x}$ pela rede e compute a saída o de cada neurônio.

      para cada saída k da rede **faça**
        Calcule o erro de k dado por $\delta_k \leftarrow o_k(1 - o_k)(p_k - o_k)$
      fim para

      para cada neurônio oculto h da rede, de trás para frente, **faça**
        Calcule o erro de h dado por $\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in Saídas} w_{kh}\delta_k$
      fim para

      para cada peso w_{ji} da rede **faça**
        $w_{ji} \leftarrow w_{ji} + \eta\delta_j x_{ji}$
      fim para
    fim para
  até que a condição de término seja alcançada
fim função

A função descrita no Algoritmo 9 atualiza os pesos a cada amostra, conforme propõe o SGD. Para calcular o verdadeiro gradiente do erro, conforme realizado no algoritmo GD, seria necessário somar todos os valores referentes à $\delta_j x_{ji}$, para somente depois alterar os pesos. Outra característica da atualização dos pesos presente no Algoritmo 9, é o ciclo de repetição externo, que faz com que as atualizações dos pesos sejam realizadas várias vezes. Esse ciclo corresponde ao número de épocas do treinamento de rede e pode ser tanto um valor fixo quanto um valor dependente de uma condição de parada, que pode ser em função do erro, por exemplo.

3.2.2 *Adicionando Momentum*

Para aumentar o desempenho do treinamento existe a possibilidade de aumentar a taxa de aprendizagem. No entanto, com uma taxa de aprendizagem mais alta, aumenta também a variação dos parâmetros, o que pode gerar instabilidade no modelo. Uma alternativa para esses casos é o uso de uma técnica chamada de *momentum*. Essa técnica incorpora na atualização dos pesos de cada iteração, uma parte das correções que foram aplicadas nas iterações anteriores, o qual é controlada pela constante de *momentum*. Dessa

forma a atualização tende a ficar mais estável, mesmo com taxas de aprendizagem maiores (HAYKIN, 2009).

3.2.3 AdaGrad

O AdaGrad (Gradiente Adaptativo) é um método apresentado em (DUCHI et al., 2011) que propõe que a taxa de aprendizado seja adaptativa para cada dimensão no espaço de busca de parâmetros. Isso é realizado com base na raiz quadrada da soma dos quadrados dos gradientes de iterações anteriores. Dessa forma, as correções podem ser mais bruscas em direções em que o vetor gradiente é mais íngreme e mais leves em dimensões onde a inclinação do vetor de gradiente é menos acentuada. Essa modificação permite que o algoritmo seja menos sensível à taxa de aprendizagem definida inicialmente. No entanto, existe um problema conhecido em que esse parâmetro pode ser reduzido drasticamente, fazendo com que o modelo não consiga convergir.

3.2.4 Adadelta, RMSProp e Adam

Buscando corrigir o problema do AdaGrad, que pode diminuir muito a taxa de aprendizagem, em (ZEILER, 2012) é proposto o Adadelta. Esse otimizador define uma janela móvel que limita os gradientes do passado que são considerados para ajustar a taxa de aprendizagem, não considerando todo o histórico. De forma paralela e independente, em um curso online foi proposto um algoritmo bem próximo chamado de RMSprop (TIELEMAN; HINTON, 2012). Já em (KINGMA; BA, 2015) foi proposto um novo otimizador chamado de Adam, que é similar à uma versão do RMSProp com *momentum*, sendo bastante utilizado nos trabalhos mais recentes de *deep learning*.

4 Trabalhos Relacionados

Em (TESAURO, 1995) foi criado um programa chamado de TD-Gammon. Esse programa faz o uso de uma rede neural artificial do tipo *multilayer perceptron* treinada contra si mesma para aprender o jogo backgammon, em uma versão não linear do algoritmo TD(λ) (Seção 2.7). A entrada da rede, além de contar com posições das peças, também foi composta por métricas bem elaboradas sobre a dinâmica do jogo, que são usualmente chamadas de *features*. O programa ficou conhecido por conseguir superar campeões mundiais de backgammon e também por descobrir novas jogadas não exploradas na época, sendo um marco da inteligência artificial nos anos 90.

Apesar do sucesso do TD-Gammon, as técnicas utilizadas nesse trabalho não obtiveram êxito em outros jogos como Xadrez, Go e Damas (MNIH et al., 2015). Em (POLLACK; BLAIR, 1996) foi realizado um estudo para se averiguar o motivo pelo qual TD-Gammon foi um sucesso. Após conseguirem atingir resultados bem próximos com técnicas mais simples e sem *features* muito bem elaboradas no *backgammon*, os autores concluíram que o êxito de TD-Gammon não teve relação forte com o algoritmo de diferença temporal em conjunto com o *backpropagation*. Segundo os autores a técnica utilizada de self-play é problemática pois ela pode fazer com que os jogadores continuem explorando sempre os mesmos tipos de jogos repetidamente, explorando apenas uma parte do espaço de busca. No entanto a hipótese levantada foi de que a dinâmica do jogo backgammon colaborou para que ocorresse justamente o inverso, o que não aconteceu em outros jogos.

4.1 DQN

Duas décadas após o TD-Gammon, em (MNIH et al., 2015) foi proposto um algoritmo chamado de DQN (*Deep Q Network*) que teve como objetivo aprender a jogar jogos de Atari 2600 através do ALE (*Arcade Learning Enviromment*) (BELLEMARE et al., 2013). Ao contrário de (TESAURO, 1995), a DQN faz o uso de uma CNN e portanto tem como representação do estado uma imagem, não sendo necessário realizar extração de *features* do estado. Esse trabalho abriu as fronteiras do aprendizado por reforço profundo e impulsionou o ALE e seus jogos de Atari como um importante conjunto de ambientes de *benchmark* para os algoritmos, o que faz com que o mesmo seja utilizado em vários trabalhos posteriores. Embora o ALE tenha sido lançado também como um desafio, é importante que o leitor não compreenda-o apenas como uma aplicação que necessita ser solucionada, mas principalmente como um conjunto de ambientes que possuem várias particularidades do mundo real e por isso é ideal para testes e comparativos.

Os autores da DQN também apresentaram uma importante contribuição que foi a

de aplicar a técnica de experiência por repetição juntamente com mini batchs no aprendizado por reforço. A experiência por repetição consiste em manter em um *buffer* o registro das n últimas transições de estado. Sendo assim a atualização dos pesos na rede é baseado em um batch que é formado através de amostras escolhidas em cada passo de atualização de forma uniformemente aleatória dentre as n armazenadas no buffer. Essa é uma importante característica do algoritmo pois treinar a rede baseado em amostras consecutivas pode ser ineficiente devido à forte correlação entre as amostras. Na DQN o objetivo da rede é aproximar os valores de estado e ação seguindo o algoritmo *Q-Learning*, que como já visto é um algoritmo *off-policy*. Segundo os autores, a escolha de um algoritmo *off-policy* para ser aplicado em conjunto com a experiência por repetição é necessária, uma vez que os parâmetros da rede no momento da atualização são diferentes daqueles utilizados na escolha da ação.

Por utilizar um algoritmo de diferença temporal, a DQN se enquadraria nos algoritmos que são considerados semi gradiente e consequentemente possuem grande variação. Para amenizar esse problema os autores da DQN contam com uma outra rede neural chamada de *target network*. A rede *target* possui a mesma estrutura que a rede principal, no entanto seus pesos θ^- apenas são atualizados a cada k passos, como cópia dos pesos da rede original. Com uma rede *target*, os valores alvo em cada passo na rede original são atualizados conforme a Equação 4.1. Essa rede auxiliar foi importante para reduzir a variância na DQN proposta em (MNIH et al., 2015).

$$Y_t^{DQN} = R_{t+1} + \gamma \underset{a}{max}\ \hat{q}(S_{t+1}, a; \theta_t^-) \tag{4.1}$$

4.2 DDQN

Em (HASSELT et al., 2016) foi aplicado uma pequena modificação na DQN em busca de aproximá-la ao algoritmo *Double Q-Learning*, apresentado na Seção 2.4, mas sem perder a essência da modelagem inicial da DQN. Essa alteração foi realizada apenas no valor alvo da rede, que passou a ser conforme exibido na Equação 4.2, em que os pesos θ^- continuam sendo uma cópia dos pesos θ da rede principal a cada k passos. A nova rede foi batizada de *Double DQN* (DDQN) e embora tenha uma diferença bem sutil em relação à DQN, obteve resultados significativamente superiores de acordo com as análises dos autores.

$$Y_t^{DDQN} = R_{t+1} + \gamma \hat{q}(S_{t+1}, \underset{a}{argmax}\ \hat{q}(S_{t+1}, a; \theta_t), \theta_t^-) \tag{4.2}$$

4.3 DRQN

Ainda no ano de 2015, outras novas variantes da DQN surgiram, como a *Deep Recurrent Q-Network - DRQN*: uma variação da DQN que faz o uso de uma rede neural recorrente. Como já descrito, a lei geral de uma MDP estabelece que a decisão tomada deve ser sempre baseada apenas no estado atual. No entanto, muitas vezes o ambiente é considerado parcialmente observável (POMDP), que é quando não é possível extrair todas as informações de um estado. Isso acontece, por exemplo, em alguns ambientes em que a direção de objetos importantes não é extraível diretamente do estado, sendo necessária a observação de estados anteriores pelo modelo. A DQN lida com essa questão incorporando sempre na entrada da rede os últimos 4 *frames*, sendo assim a entrada é uma imagem concatenada. Embora essa solução tenha funcionado bem, o trabalho (HAUSKNECHT; STONE, 2015) propôs a DRQN como uma alternativa, que por ser recorrente, só toma como entrada um *frame*. Os resultados mostraram que a DRQN é tão boa quanto a DQN na maioria dos jogos de atari, apenas apresentando uma superioridade mais robusta em jogos nos quais os elementos estão frequentemente desaparecendo da tela (piscando).

4.4 Experiência por Repetição Priorizada

Em (SCHAUL et al., 2016) foi proposto uma melhoria na DQN e na DDQN, que alterou a forma como ocorre a experiência por repetição. Enquanto nas duas redes as amostras para formarem o mini batch são retiradas de forma uniformemente aleatória dentre as últimas armazenadas no buffer, o novo trabalho propõe a criação de uma estrutura de dados para armazenar prioridades, em que as amostras de transição que possuem uma diferença temporal maior possuem maior probabilidade para serem selecionadas. A motivação para o uso dessa técnica é que muitas das vezes transições importantes para a evolução do agente deixam de ser selecionadas e consequentemente aprendidas também. No entanto, segundo os autores, a introdução da experiência por repetição priorizada (PER) insere um bias porque muda a distribuição com que as atualizações foram obtidas, o que pode levar ao sobre-ajuste. A solução adotada foi mais uma vez utilizar amostragem por importância, mas com uma formulação diferente.

Tanto na aplicação voltada para algoritmos *off-policy* quanto na PER, o objetivo da IS é aproveitar dados gerados por uma distribuição em outra, atuando como se fosse um conversor. No entanto, especificamente na aplicação da experiência por repetição, é natural que surja um questionamento: se já é possível retirar as transições do *buffer* de forma uniforme, por que retirar de forma priorizada para depois ser necessário ponderar os impactos na rede simulando uma retirada uniforme? Selecionar transições mais importantes com maior probabilidade é uma boa escolha, mas se elas forem selecionadas frequentemente, é grande a chance de sobre-ajuste. Ao adicionarmos o IS, o impacto da

transição é reduzido, mas ela ainda terá maior probabilidade de ser selecionada, ao contrário do uso de uma distribuição uniforme sem IS, em que todas transições possuem probabilidades iguais.

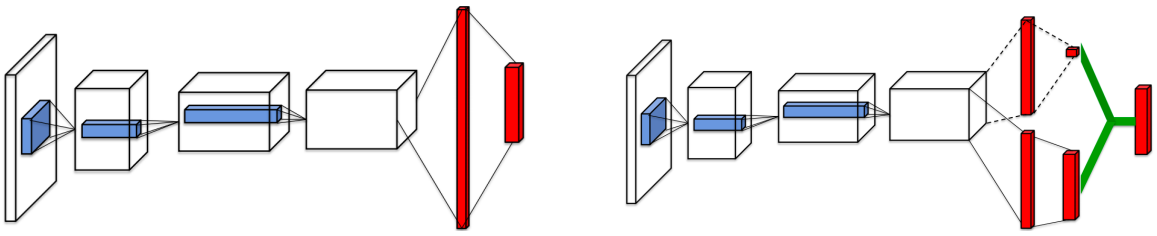
A nova versão da DDQN que faz o uso de experiência por repetição priorizada demonstrou-se ser mais eficiente que a DQN em 41 dos 48 jogos de Atari analisados pelos autores, além de acelerar o aprendizado por um fator 2, tornando-se então o estado da arte até o momento da publicação.

4.5 Rede de Duelo

Em (WANG et al., 2016) foi proposta uma nova arquitetura de rede, novamente baseada na DQN e suas variações. A nova rede, que possui uma arquitetura de Duelo, utiliza o conceito da função de vantagem $a_\pi(s, a)$, representada na Equação 4.3, que tem o intuito de destacar o quanto uma ação é boa em relação às demais viáveis no estado.

$$a_\pi(s, a) = q_\pi(s, a) - v_\pi(s) \tag{4.3}$$

Com $v_\pi(s)$, é possível obter $q_\pi(s, a)$ pela soma de $v_\pi(s)$ e $a_\pi(s, a)$. Sendo assim, a nova rede possui dois fluxos paralelos: um para estimar $v_\pi(s)$ e outro para estimar $a_\pi(s, a)$, que são combinados na camada final de modo a se obter a estimativa de $q_\pi(s, a)$. A Figura 12 exibe uma comparação entre a arquitetura da DQN com a da Rede de Duelo.



(a) DQN (b) Rede de Duelo

Figura 12 – Arquitetura das redes DQN e de Duelo. Extraída de (WANG et al., 2016).

Se os neurônios da última camada da Rede de Duelo realizarem uma simples operação de soma entre $\widehat{v}(s, \theta)$ e $\widehat{a}(s, a, \theta)$ (estimativas de $v_\pi(s)$ e $a_\pi(s, a)$), não é garantida a representação independente dos termos, uma vez que um pode compensar o outro e a operação de soma produzirá o mesmo resultado. Evitando esse comportamento, os autores propuseram obter $\widehat{q}(s, a, \theta)$ seguindo a Equação 4.4. Dessa forma a arquitetura garante que no caso da ação mais valorosa, a função de vantagem será igual a zero, forçando assim que $\widehat{v}(s, \theta)$ se aproxime mais de $v_\pi(s)$.

$$\widehat{q}(s, a; \theta) = \widehat{v}(s; \theta) + \widehat{a}(s, a; \theta) - \max_{a' \in |\mathcal{A}|} \widehat{a}(s, a'; \theta) \tag{4.4}$$

Embora exista uma perda semântica, os criadores da rede mostraram que trocar a função *max* pela média na Equação 4.4 garante mais estabilidade aos modelos. Mas afinal, qual a vantagem de se utilizar uma arquitetura de duelo? Segundo os autores da rede, em muitos estados as ações possuem resultados muito similares, principalmente nos ambientes em que existem várias ações possíveis. Sendo assim, ao estimar $q_\pi(s, a)$ baseado na estimativa de $v_\pi(s)$, a arquitetura da rede força com que uma única transição gerada por uma ação a em um estado s, melhore também as estimativas das demais ações possíveis no estado, o que acelera o processo de aprendizagem. Os testes descritos em (WANG et al., 2016) mostraram que a combinação da Rede de Duelo com o target proposto pela DDQN e a experiência por repetição priorizada configurou o novo estado da arte no domínio dos jogos do Atari 2600.

4.6 A3C

Embora a DQN juntamente com todas as suas melhorias representem importantes avanços para a pesquisa em aprendizado de reforço, esses algoritmos ainda possuem alguns pontos críticos, como o tempo necessário para a convergência da rede, o que pode gerar dependência de hardwares especializados em algumas tarefas e também a necessidade de se utilizar algoritmos *off-policy*, devido à experiência por repetição. Vale lembrar que a técnica de experiência apenas é utilizada pois as atualizações sequenciais são fortemente correlatas, podendo causar sobre-ajuste.

Baseado nas ideias apresentadas, em (MNIH et al., 2016) foi proposto um novo *framework* para o aprendizado de reforço profundo que funciona de forma assíncrona utilizando CPUs. A ideia central do *framework* é executar vários agentes em paralelo em instâncias diferentes de um mesmo ambiente, mas compartilhando uma mesma rede neural. Como os agentes são estocásticos, é esperado que cada agente explore uma sub-parte do problema. Sendo assim, com vários agentes explorando ambientes paralelos e realizando atualizações de tempos em tempos em uma mesma rede central, é possível dispensar a experiência por repetição, uma vez que a rede não será submetida apenas à atualizações baseadas em amostras sequenciais de um único agente. Essa contribuição abre possibilidades para a combinação eficiente de algoritmos *on-policy* com redes neurais, além de diminuir o tempo necessário para a convergência da rede devido ao paralelismo. Além disso, caso seja de interesse do pesquisador ou programador utilizar um algoritmo *off-policy*, uma boa técnica é utilizar uma política de exploração para cada agente, pois isso aumentará ainda mais a cobertura dos agentes sobre o problema (MNIH et al., 2016).

Assim como na DQN, o *framework* abordado também faz o uso de uma rede *target*. Os gradientes de cada agente são acumulados e a atualização na rede pode ocorrer em intervalos sincronizados, o que elimina a concorrência entre as threads. Em (MNIH et

al., 2016) foram realizados vários testes com diferentes algoritmos no *framework*. Dentre todos se destacou o A3C (*asynchronous advantage actor-critic*), um algoritmo do tipo *Actor-Critic*, de n passos e que faz o uso da função de vantagem $a_\pi(s, a)$.

O A3C foi treinado por 16 agentes paralelos executando em CPUs de uma única máquina e foi capaz de superar significativamente os algoritmos anteriores tanto em pontuação quanto em tempo de convergência no domínio do Atari 2600. Além disso, o A3C foi aplicado no jogo TORCS 3D, que possui gráficos mais desafiadores, e em ambientes que demandam ações contínuas, obtendo resultados satisfatórios em ambas situações. Vale ressaltar que na comparação no ambiente do Atari, não foi informado se a versão da Rede de Duelo com o DDQN utilizou a experiência por repetição priorizada, o que impede a constatação se de fato o A3C se tornou o novo estado da arte.

Outro ponto sobre o A3C é que ele também foi avaliado em alguns ambientes tomando como entrada da rede características do estado, tendo se comportado de forma satisfatória. Embora esse resultado já seja esperado, essa é uma importante constatação, uma vez que em ambientes reais ou digitais complexos graficamente, o uso de imagens como entrada da rede pode tornar o aprendizado muito demorado.

4.7 C51 - Uma perspectiva distributiva

Em (BELLEMARE et al., 2017) foi apresentado um estudo que introduziu a representação de distribuições de maneira explícita nas redes de aprendizado por reforço profundo. Antes de introduzir o algoritmo, é importante compreender alguns conceitos, como por exemplo o da distribuição Z do valor do par estado e ação, que está definida através da Equação 4.5, e o da função $q(s, a)$, já conhecida, que pode ser também definida como a esperança de tal distribuição, como representado na Equação 4.6.

$$Z(s, a) \overset{D}{=} R + \gamma Z(s', a') \tag{4.5}$$

$$q(s, a) = \mathbb{E}[Z(s, a)] \tag{4.6}$$

Os autores fazem o uso de alguns operadores de Bellman para simplificar as notações. Dentre eles o operador P^π (aplicado na Equação 4.7) que gera o valor da função em que é aplicado para o próximo par estado-ação, e o operador $\mathcal{T}^\pi$ (demonstrado na Equação 4.8) que atualiza o valor da função em que é aplicado.

$$P^\pi Z(s, a) :\overset{D}{=} Z(s', a') \tag{4.7}$$

$$\mathcal{T}^\pi Z(s, a) :\overset{D}{=} R + \gamma P^\pi Z(s, a) \tag{4.8}$$

A Figura 13(a) ilustra uma determinada distribuição, que por sua vez é discretizada através de átomos. Na Figura 13(b) a distribuição foi multiplicada por $0 < \gamma < 1$. Nesse caso a multiplicação ocorre em cada átomo z_i da distribuição, aproximando-o de 0. Logo, baseando-se na representação gráfica, a Figura 13(b) assume que a probabilidade central da distribuição refere-se ao valor 0. Já na Figura 13(c) foi aplicado um deslocamento na distribuição devido à soma de uma recompensa positiva.

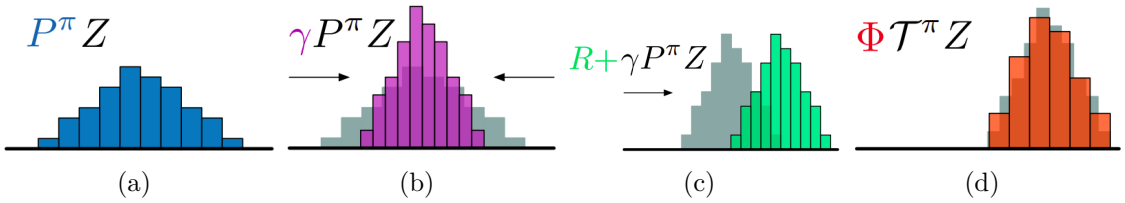


Figura 13 – Representação dos operadores de Bellman aplicados à distribuições. Extraída de (BELLEMARE et al., 2017).

Uma das contribuições de (BELLEMARE et al., 2017) foi apresentar uma prova de que o operador distributivo de Bellman ($\mathcal{T}^\pi$) é uma contração da métrica de Wasserstein na forma máxima entre distribuições de probabilidades. A métrica de Wassertein W_p, também conhecida como métrica de Mallows ou métrica de Kantorovich, é definida na Equação 4.9, em que F e G são duas CDFs e o ínfimo é calculado sobre todos os pares de variáveis (U, V). Já o teorema da contração está na Definição 1.

$$W_p(F, G) := \inf_{U,V} \|U - V\|_p \tag{4.9}$$

Definição 1 *Uma contração em um espaço métrico (M, d) é uma função f de M se existe algum número real não negativo $0 \leq k < 1$ tal que $d(f(x), f(y)) \leq kd(x, y)\ \forall x, y \in M$.*

Sendo assim, o que foi provado em (BELLEMARE et al., 2017) é que existe k tal que $d_p(\mathcal{T}^\pi Z_1, \mathcal{T}^\pi Z_2) \leq kd_p(Z_1, Z_2)$. Isso indica que aplicar o operador $\mathcal{T}^\pi$ em duas distribuições diferentes diminui a distância entre elas, tomando como medida a métrica de Wassertein. Além disso, de acordo com o teorema de ponto fixo de Banach (BANACH, 1922), os autores provaram que a sequência Z_k converge para Z^π tomando como base a métrica de distância de Wassertein (W_p).

Apesar das contribuições teóricas, os autores não conseguiram propor um algoritmo prático que represente em totalidade as ideias por eles apresentadas, uma vez que a métrica de Wassertein não pode ser utilizada para aprendizado baseado em amostras de transição (BELLEMARE et al., 2017). Portanto foi proposto um algoritmo distributivo que não possui garantias teóricas de convergência, chamado de C51.

A proposta do C51 segue a linha de não limitar a representação à média, e sim obter uma representação discretizada de $Z(s, a)$, utilizando como suporte um conjunto de

$N = 51$ átomos chamados de z_i, em que cada átomo representa um retorno do par estado e ação, cuja a probabilidade de ocorrência de tal retorno é dada por p_i.

Para a definição da distribuição é preciso escolher um intervalo $[V_{min}, V_{max}] \in R$ para o retorno. Portanto é possível definir $z_i = V_{min} + i\Delta z$, sendo $0 \leq i < N$ e $\Delta z = \frac{V_{max} - V_{min}}{N-1}$. A probabilidade $p_i(s,a)$ de ocorrência de z_i em $Z(s,a)$ é representada pela Equação 4.10, em que $\theta_i(s,a)$ é a saída da rede neural referente a z_i e à ação a no estado s. Baseado em z_i e p_i é possível estimar $q(s,a)$ por $\sum_i z_i p_i(s,a)$;

$$p_i(s,a) = \frac{e^{\theta_i(s,a)}}{\sum_j e^{\theta_j(s,a)}} \tag{4.10}$$

Para a definição da distribuição *target*, o algoritmo faz o uso dos operadores de Bellman, ilustrados na Figura 13. É importante destacar que após multiplicar cada átomo da distribuição z_i por γ e deslocá-la horizontalmente devido a soma de uma recompensa, existe grande probabilidade que os suportes de Z e $\mathcal{T}^\pi Z$ sejam disjuntos. Sendo assim o operador Φ, ilustrado na Figura 13(d) aplica transformações na distribuição $\mathcal{T}^\pi Z$ de modo a produzir uma distribuição similar, mas com os mesmos suportes de Z. Essa transformação distribui as probabilidades de cada átomo z_j para os dois vizinhos mais próximos, considerando o suporte ideal definido previamente.

Com a distribuição Z obtida inicialmente pela rede neural e a distribuição alvo $\Phi\mathcal{T}^\pi Z$ possuindo os mesmos suportes, a tarefa de ajustes dos pesos ocorre como em uma rede de multi classificação, frequentemente utilizada no aprendizado supervisionado. Sendo assim para calibrar os pesos da rede foi aplicado o gradiente descendente baseado na divergência entre $Z(s,a)$ e $\Phi\mathcal{T}^\pi Z(s,a)$.

O C51 conta com a estrutura das camadas iniciais da rede de forma semelhante a DQN, inclusive utilizando uma rede target e experiência por repetição. Mesmo sem garantias de convergência e sem incorporar as demais melhorias realizadas sobre a DQN nos trabalhos anteriores, o C51 obteve resultados importantes e superou os algoritmos: DQN, DDQN e Rede de Duelo com experiência por repetição priorizada. Infelizmente não foram realizadas comparações com o algoritmo A3C, o que enfraquece a informação de que o C51 se tornou o novo estado da arte no domínio do Atari 2600.

4.8 Redes Ruidosas

A DQN e suas variantes listadas até o momento utilizam políticas de exploração ε-*greedy*, que é um dos tipos de políticas mais clássicas presentes na literatura. Existem outros tipos de políticas que oferecem melhores resultados, como a UCB (*Upper-Confidence-Bound*), mas que geralmente estão limitadas a problemas com poucos estados e ações e a algoritmos tabulares (SUTTON; BARTO, 2018). Sendo assim em (FORTUNATO et

al., 2018) foi proposto uma forma de melhorar a exploração dos agentes baseados na DQN. Essa alteração adiciona ruídos nas camadas altamente conectadas da rede neural, de forma a aumentar a capacidade de exploração do agente no ambiente. Considerando que os valores de estado e ação providos pela rede neural já possuem ruídos, não existe a necessidade de executar uma política ε-*greedy*, então basta escolher a melhor ação em todos os momentos que o agente já vai estar sendo induzido a explorar o ambiente.

Os autores aplicaram a melhoria na DQN, Rede de Duelo e A3C e avaliaram os algoritmos nos jogos do Atari 2600. Em todos três algoritmos houveram melhorias significativas. Um ponto relevante a se destacar é que o A3C, ao contrário do que descrito em (MNIH et al., 2016), apresentou desempenho inferior à DQN tanto nas versões originais dos algoritmos quanto nas versões que utilizaram redes ruidosas.

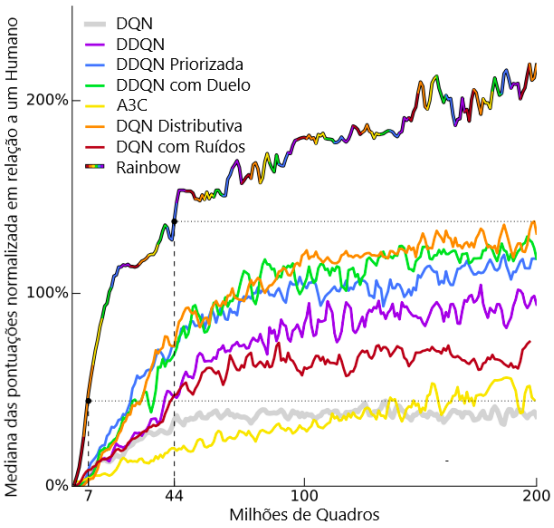
4.9 Rainbow

É notável que várias melhorias foram aplicadas à DQN. No entanto, observa-se que principalmente a partir da publicação do A3C, as melhorias não foram sincronizadas umas com as outras, provavelmente devido ao curto prazo entre a publicação de um trabalho e o outro. Sendo assim em (HESSEL et al., 2018) foi realizado um estudo que buscou combinar todas as variantes da DQN listadas até o momento, com exceção da DRQN, que apenas apresentou contribuições mais robustas para uma classe muito pequena de problemas.

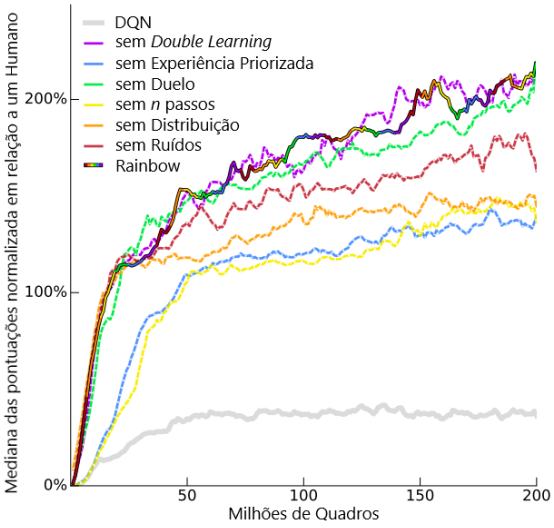
A combinação de todas as variantes foi denominada de Rainbow, um poderoso algoritmo que se tornou o estado da arte no domínio do Atari 2600. O Rainbow busca estimar distribuições assim como no C51, mas utiliza também a target da DDQN, a experiência por repetição priorizada, a arquitetura de duelo e também as redes ruidosas. Das contribuições do A3C o Raibow utilizou apenas o aprendizado de n passos, em que foi definido $n = 3$.

A Figura 14(a) exibe uma comparação em função do número de frames observados do Rainbow em relação à DQN e outras variações, em que é possível notar que com uma experiência de 44 milhões de frames o Rainbow conseguiu equiparar-se ao C51 quando treinado em 200 milhões de frames. Em quantidades iguais de frames, observa-se uma clara superioridade do Rainbow em relação aos demais algoritmos.

Na Figura 14(b) os autores compararam o Rainbow com versões defasadas do mesmo, retirando-se em cada versão uma das evoluções que compõem o Rainbow. Pelo resultado observa-se que os componentes mais importantes do algoritmo são a experiência por repetição priorizada, o algoritmo de n passos e a representação da distribuição em átomos. Os componentes que apresentaram contribuições menos relevantes para o Rainbow foram a arquitetura de duelo e a target baseada no DDQN.



(a) Comparação do Rainbow com outros algoritmos.



(b) Comparação do Rainbow com versões incompletas do próprio algoritmo.

Figura 14 – Mediana das pontuações normalizada em relação a um jogador humano. Métrica avaliada em função do número de quadros em 57 jogos de Atari. Extraída e editada de (HESSEL et al., 2018)

Um ponto curioso sobre o Rainbow é que ele utilizou uma adaptação do *Q-Learning* para n passos que dependeria da amostragem por importância, mas que não foi utilizada. A ausência do IS no algoritmo de n passos não é relatada no artigo do Rainbow e portanto não foi explicada. Em um fórum sobre ciência de dados essa questão foi levantada. Segundo uma das respostas (SLATER, 2019), os autores podem ter ignorado a IS por fatores como o tamanho de $n = 3$ ser muito pequeno em relação aos milhões de passos utilizados no treinamento e também ao fato do valor de ε da política de exploração ser pequeno também (0,01 ou 0,001) o que faz com que o algoritmo seja quase *on-policy*.

4.10 QR-DQN

Em (BELLEMARE et al., 2017) os autores apresentaram importantes contribuições teóricas sobre a aplicação da métrica de Wassertein em distribuições de valor de estado e ação, mas deixaram uma questão em aberto, já que propuseram um algoritmo que, apesar de eficiente, não representa em totalidade as teorias discutidas: o C51.

Sendo assim em (DABNEY et al., 2018b) foi proposto um novo algoritmo distributivo, batizado de QR-DQN, que minimiza a métrica de Wassertein. Enquanto o C51 utiliza valores fixos para as distribuições e busca estimar as probabilidades de cada valor, o QR-DQN transpõe essa parametrização, considerando probabilidades uniformes para a distribuição e estimando os valores com que cada probabilidade se refere.

O que o QR-DQN faz é realizar uma regressão quantílica sobre a estrutura da

DQN, estimando os N quantis da distribuição. Cada quantil τ_i pode ser definido como $\frac{i}{N}$ para $i = 1, .., N$. Portanto a rede neural tem na última camada $N|\mathcal{A}|$ neurônios, com cada neurônio correspondendo a o valor de um quantil τ_i de uma determinada ação.

O algoritmo utiliza uma versão quantílica da função de perda de Huber (HUBER, 1964) para o treinamento da rede neural. Embora a métrica de Wassertein não seja utilizada diretamente no algoritmo os autores mostraram que ela é minimizada de forma indireta.

Essa é uma solução interessante em relação ao C51 pois não sofre de suportes disjuntos, uma vez que os quantis são ajustados pela rede neural. Consequentemente, também não é necessário limitar a distribuição por um valor máximo e mínimo e a construção da distribuição target não depende de se aplicar o operator Φ utilizado no C51. A versão do QR-DQN que estima os valores de 200 quantis se demonstrou muito eficiente, superando o algoritmo C51. Não foram realizadas comparações do QR-DQN com o Rainbow.

4.11 IQN

Em (DABNEY et al., 2018a) foi proposto um novo algoritmo chamado de IQN (*Implicit Quantile Network*), que apresenta uma versão implícita do algoritmo QR-DQN. Ao invés de ter uma saída para cada quantil τ_i da distribuição, o IQN traz o quantil τ_i como entrada da rede (além do estado). Dessa forma para montar uma distribuição é necessário acionar uma das funções da rede neural várias vezes com diferentes valores de τ_i na entrada em um mesmo estado.

A Figura 15 ilustra um comparativo entre alguns dos algoritmos abordados e pode ajudar no entendimento. Note que na representação da IQN existe uma função a mais (ϕ), que é responsável por processar a entrada τ_i e é formada por uma rede neural rasa. A outra função, ψ, é idêntica à DQN. Por fim existe uma função f que combina ϕ com ψ, gerando como saída um valor para o quantil τ_i de cada ação no estado. É importante ressaltar que, embora seja necessário acionar a função ϕ várias vezes para formar a distribuição em um mesmo estado, em todas essas vezes a imagem de entrada é a mesma e a rede neural é determinística. Portanto não é necessário processar todos os fluxos da rede nos vários acionamentos de cada estado.

Em (DABNEY et al., 2018a) os autores inserem na rede 32 quantis, que são selecionados de forma aleatória dentre uma distribuição de intervalo $[0, 1]$ e depois estimam o valor de $q(s, a)$ baseado na média dos valores de cada quantil. Como existe uma aleatoriedade na seleção de quais quantis serão estimados, não é necessário utilizar uma política ε-*greedy*. O treinamento da rede é feito também utilizando a função de perda de Huber e o gradiente é calculado para outros oito quantis obtidos aleatoriamente, podendo a correção da rede ser realizada por mini batchs.

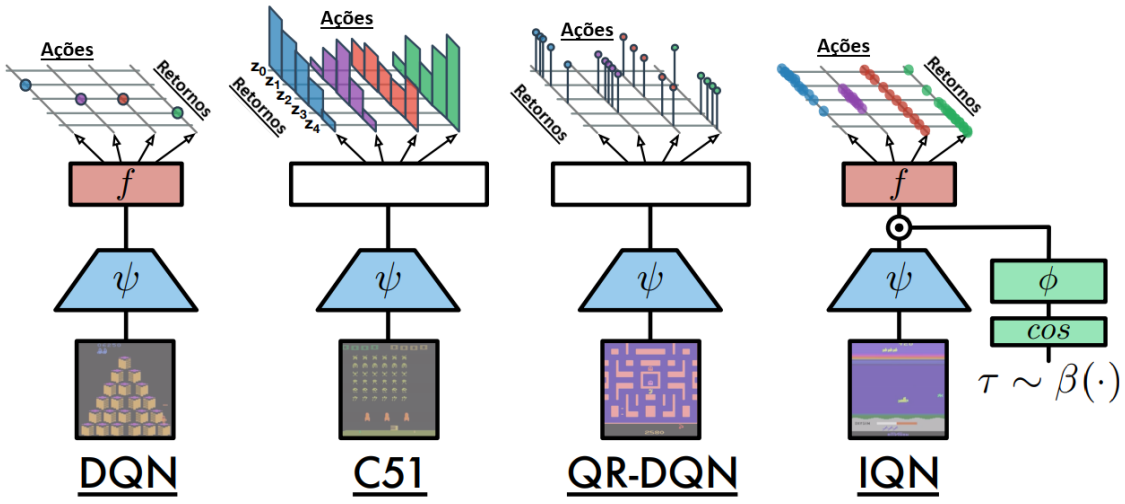


Figura 15 – Arquitetura da DQN e de recentes algoritmos distributivos de aprendizado por reforço. Extraído e editado de (DABNEY et al., 2018a).

Nos trabalhos anteriores sobre aprendizado por reforço distributivo (C51, Rainbow e QR-DQN) sempre houve a motivação de que apenas a média não é suficiente para representar um estado e por isso surgiu a ideia de se estimar distribuições, assim como foi feito. No entanto após obtida a distribuição, é calculada a média para se tomar a decisão de qual ação a ser tomada, o que não aproveita todo o esforço da representação. No IQN a política utilizada busca explorar mais sobre a distribuição. Isso é realizado no momento de seleção dos quantis. Ao invés de selecioná-los dentre uma distribuição uniforme, o que levaria a representação de $q(s, a)$ após o cálculo da média, os quantis são selecionados de distribuições não uniformes.

A IQN obteve ótimos resultados no domínio do Atari 2600 e conseguiu superar o QR-DQN, mas não foi capaz de superar o Rainbow. Apesar disso os autores destacaram que a IQN não incorpora nenhuma das melhorias realizadas sobre a DQN e que por isso um novo algoritmo que substitui o C51 pelo IQN no Rainbow, poderia gerar um novo estado da arte.

4.12 Munchausen R.L.

Em 2020 uma nova e simples melhoria aplicada na DQN surgiu e foi batizada de Munchausen R.L. (MRL) (VIEILLARD et al., 2020). Baseando-se em conhecimentos elementares do aprendizado por reforço, essa abordagem parte do princípio que de forma indireta a política está sendo sempre evoluída e que portanto ela pode emitir um reforço no aprendizado do agente. Se considerarmos uma política ótima, temos que o log da probabilidade de se tomar a melhor ação sempre será zero, enquanto das outras ações será $-\infty$. O que o MRL faz é adicionar esse sinal na recompensa, substituindo R_t por $R_t + \alpha_{mrl} ln(\pi(A_t|S_t))$, em que α_{mrl} é um parâmetro ajustável.

O nome da abordagem foi inspirado em um livro chamado "As Aventuras Do Barão em Munchausen", em que o barão se arranca de um pântano puxando seu próprio cabelo. Tanto o acontecimento do livro, quanto a abordagem de Munchausen rememoram um conceito base do aprendizado por reforço chamado de Iteração de Política Generalizada (GPI), que diz que a política está sendo sempre melhorada baseada na função de valor e a função sempre sendo melhorada baseada na política (SUTTON; BARTO, 2018).

Um problema inicial da aplicação dessa abordagem na DQN é que a rede não aprende políticas estocásticas, e sim a política ótima, o que é um empecilho para a aplicação do log, que sempre resultaria em zero ou $-\infty$. Antes de introduzir a abordagem de Munchausen vamos reescrever a *target* da DQN (*Q-Learning*) da mesma forma como ocorre no algoritmo Expected Sarsa, mas lembrando que a política de aprendizado da DQN é gulosa e determinística:

$$Y_t^{DQN} = R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) \widehat{q}(S_{t+1}, a; \theta_t^-) \tag{4.11}$$

A primeira alteração realizada é trocar a política ótima por uma política *softmax* com fator de temperatura τ. Também é adotado no algoritmo um termo para maximização de entropia, conforme realizado em (HAARNOJA et al., 2018), ficando a *target* como:

$$Y_t^{S-DQN} = R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) (\widehat{q}(S_{t+1}, a; \theta_t^-) - \tau ln \pi(a|S_{t+1})) \tag{4.12}$$

com $\pi = soft\_max(\frac{\widehat{q}(S_{t+1}, a; \theta_t^-)}{\tau})$

Por fim é adicionado o fator de munchausen ($\alpha_{mrl} \tau ln(\pi(A_t|S_t))$), ficando a *target* como na Equação 4.13, ainda considerando uma política estocástica *softmax* ajustada por um fator de temperatura τ. Fator de munchausen é o nome dado ao termo adicionado logo após a recompensa na Equação 4.13.

$$Y_t^{M-DQN} = R_{t+1} + \alpha_{mrl} \tau ln(\pi(A_t|S_t)) + \gamma \sum_a \pi(a|S_{t+1}) (\widehat{q}(S_{t+1}, a; \theta_t^-) - \tau ln \pi(a|S_{t+1})) \tag{4.13}$$

Para evitar uma grande variância, os autores também implementaram um limite superior e inferior para o valor do fator de munchausen. A versão adaptada da DQN foi batizada de M-DQN e conseguiu ótimos resultados, superando por exemplo o algoritmo C51, o que tornou o M-DQN a ser o primeiro algoritmo não distributivo a superar um distributivo.

Os autores também aplicaram a abordagem de Munchausen no IQN e foram analisadas duas versões: uma com diferença temporal de um passo e outra com três passos. Ambas versões conseguiram superar o Rainbow, sendo que a versão com três passos se mostrou superior, o que a tornou o estado da arte até o momento de sua publicação.

4.13 FQF

Até o momento foram abordados três algoritmos distributivos: o C51 que mantém valores fixos e estima probabilidades, o QR-DQN que fixa quantis e estima os valores desses quantis e o IQN, que estima valores baseados em quantis aleatoriamente amostrados para formar uma distribuição. Em (YANG et al., 2019) os autores reconhecem a importância do IQN, o último algoritmo proposto, mas destacam que como o número de quantis é finito e pequeno (32), a escolha de quais quantis vão ser utilizados para estimar a distribuição é essencial, uma vez que a seleção dos mesmos está diretamente ligada com o sucesso da predição. Dessa forma, em (YANG et al., 2019) é proposto um novo algoritmo chamado de FQF (*Fully Parameterized Quantile Function*), que busca estimar tanto os quantis quanto seus respectivos valores.

A Figura 16 ilustra como é a arquitetura do FQF. Comparando com o IQN, ilustrado na Figura 15, observa-se que houve a adição de uma nova função na rede, representada por φ. Essa função tem o objetivo de estimar $N = 32$ frações de quantis para cada ação disponível no estado. Posteriormente cada fração de quantil τ_i estimado por φ é inserido na rede ϕ, que segue o padrão da IQN.

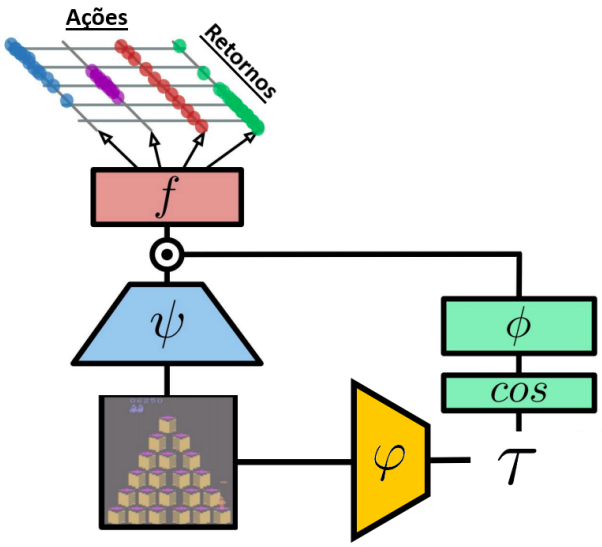


Figura 16 – Representação da arquitetura do FQF. Extraído e editado da apresentação de (YANG et al., 2019).

Para entender melhor o funcionamento do FQF e como o algoritmo consegue obter uma estimativa de $q_\pi(s, a)$, é importante entender a relação do quantil com a função de distribuição acumulada (CDF). A CDF de uma distribuição Z pode ser definida conforme na Equação 4.14, em que, dado um valor z, a função retorna a probabilidade p de uma variável aleatória amostrada da distribuição assumir um valor menor a z. Já a função de quantil pode ser definida como o inverso da CDF, ou seja, dada uma probabilidade p, encontrar z tal que, ao ser aplicado em $F_Z(z)$, retorne a menor probabilidade maior ou

igual a p, conforme definido na Equação 4.15.

$$F_Z(z) := Pr(Z < z) \tag{4.14}$$

$$F_Z^{-1}(p) := inf\{z \in \mathbb{R} : p \leq F_Z(z)\} \tag{4.15}$$

Possuindo N frações de quantis, é possível obter probabilidades de ocorrência de certos intervalos através da diferença entre duas frações de quantis consecutivos. Sendo assim, a Equação 4.16 demonstra como obter uma estimativa de $q_\pi(s, a)$ utilizando N frações de quantis τ_i, sendo que $\tau_i < \tau_{i+1}$, $\tau_0 = 0$, $\tau_1 = 1$ e $\widehat{\tau}_i = \frac{\tau_{i+1} - \tau_i}{2}$.

$$Q(s, a) = \sum_{i=0}^{N-1} (\tau_{i+1} - \tau_i) F_Z^{-1}(\widehat{\tau}_i) \tag{4.16}$$

A Equação 4.16 é utilizada pelo FQF e também, com algumas variações, para os algoritmos QR-DQN e IQN [1]. O QR-DQN utiliza valores fixos para o conjunto de frações de quantis τ_i e o algoritmo IQN escolhe frações para serem estimadas aleatoriamente. No entanto, a escolha de quais frações vão ser utilizadas pode ser essencial para melhorar as estimativas. A Figura 17 exibe duas aproximações da mesma função de quantil F_Z^{-1}, mas que fazem o uso de dois conjuntos distintos de frações de quantis τ. Na Figura 17(a) os valores de τ foram escolhidos de forma a minimizar a métrica de 1-Wassertein, representada na Equação 4.17. Já na Figura 17(b) os valores de τ foram escolhidos aleatoriamente, conforme ocorre no IQN. Observa-se, portanto, que escolher quais frações de quantis são utilizadas teve impacto significativo na redução do erro da estimativa.

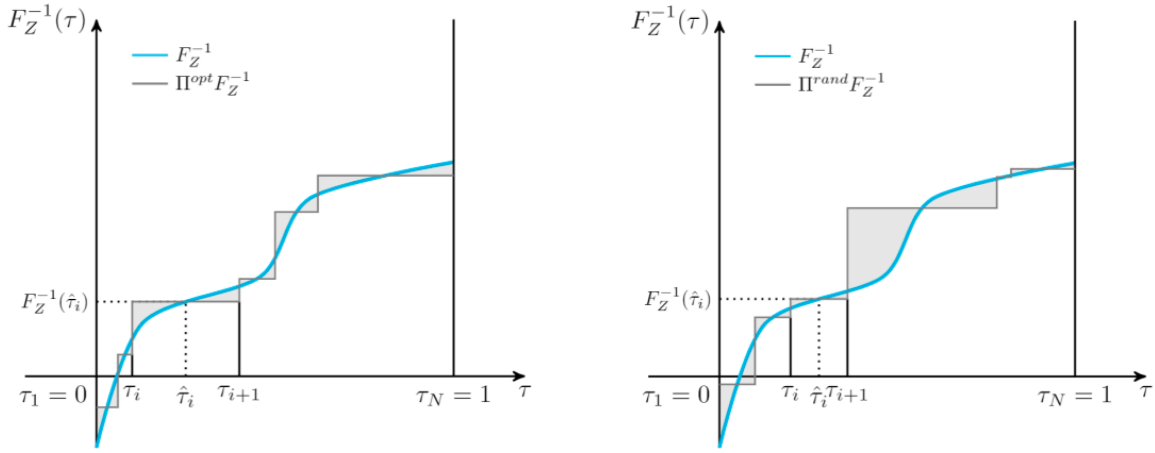
$$W_1(Z, \tau) = \sum_{i=0}^{N-1} \int_{\tau_i}^{\tau_{i+1}} |F_Z^{-1}(\omega) - F_Z^{-1}(\widehat{\tau}_i)| d\omega \tag{4.17}$$

O treinamento da FQF acontece em duas partes: primeiramente é ajustada a rede φ por uma função baseada na métrica de 1-Wassertein e depois acontece o treinamento das redes ϕ e ψ, de forma semelhante como ocorre na IQN[2]. Tendo em vista que não existe aleatoriedade na escolha dos quantis, como ocorre na IQN, no FQF a política de exploração volta a ser ε-*greedy*.

O FQF conseguiu superar os algoritmos anteriores, incluindo o IQN, mas demonstrou ser 20% mais lento que a IQN considerando a versão do FQF que também utiliza 32 quantis. A Tabela 1 exibe os resultados do FQF apresentados em (YANG et al., 2019) e também de outros algoritmos já abordados. Um ponto interessante é que nas análises do

[1] QR-DQN e IQN realizam uma média simples de F_Z^{-1}, desconsiderando a probabilidade de ocorrência de cada intervalo dada por $\tau_{i+1} - \tau_i$

[2] Embora φ também tenha como entrada as saídas das redes convolucionais, o ajuste de seus pesos não é propagado para tais camadas



(a) (b)

Figura 17 – Duas aproximações da mesma função de quantil, utilizando dois conjuntos de tamanho N=6, mas com diferentes valores para cada τ. A área sombreada corresponde à métrica de 1-Wassertein. Figuras extraídas de (YANG et al., 2019).

autor o IQN foi inferior ao Rainbow, ao contrário do que foi apresentado em (DABNEY et al., 2018a).

	Média	Mediana	>Humano	>DQN
DQN	221%	79%	24	0
PRIOR.	580%	124%	39	48
C51	701%	178%	40	50
RAINBOW	1213%	227%	42	52
QR-DQN	902%	193%	41	54
IQN	1112%	218%	39	54
FQF	**1426%**	**272%**	**44**	**54**

Tabela 1 – Média e mediana das pontuações recebidas em 55 jogos de Atari 2600, medidos como porcentagens de ganho sobre uma base de pontuações humanas. Pontuações agregadas sobre três execuções com sementes diferentes. Resultados extraídos de (YANG et al., 2019).

Destaca-se nos resultados que o FQF foi melhor que os demais algoritmos tanto na média quanto na mediana. Além disso, ele foi melhor que um humano em 44 dos 55 jogos avaliados e foi melhor do que a DQN em 54 dos 55 jogos. O FQF se tornou o melhor algoritmo não distribuído no domínio do Atari 2600.

4.14 Mini Atari

O ALE com seus jogos de Atari 2600 se consolidou como um dos principais ambientes de *benchmark* para os algoritmos de aprendizado por reforço, sendo utilizado em vários trabalhos. Seus jogos trazem dois principais desafios: o aprendizado de *features* e

o aprendizado de comportamento. Inicialmente o aprendizado de *features* era um desafio interessante. No entanto, observa-se que após o surgimento da DQN, todas as evoluções estão focadas no aprendizado de comportamento, pois as camadas convolucionais das redes não têm recebido muita atenção. Apesar disso, o aprendizado de *features* é a parte mais custosa computacionalmente dos algoritmos, podendo ser considerada até um empecilho para quem deseja explorar mais o comportamento, pois o alto custo computacional impede que sejam realizados experimentos mais diversificados (Young; Tian, 2019).

Em (CERON; CASTRO, 2021) os autores estimam que uma avaliação do algoritmo Rainbow nos 57 jogos de Atari utilizando 5 sementes levaria aproximadamente 1425 dias para ser concluída caso fosse realizada de forma sequencial, baseando-se em um tempo de 5 dias de treinamento para cada jogo que é estimado para execuções na GPU Nvidia Tesla P100.

Considerando que o aprendizado de *features* é um problema que já demonstrou ser solucionável com o devido ajuste de camadas convolucionais, em (Young; Tian, 2019) é proposto um novo conjunto de ambientes chamado de Mini Atari. Esse ambiente é uma versão miniaturizada de 5 jogos do Atari 2600 (*Seaquest, Breakout, Asterix, Freeway* e *Space invaders*) e busca simplificar o frame dos jogos ao máximo para que os pesquisadores possam focar nos algoritmos de aprendizado por reforço. A Figura 18 exibe os jogos explorados.

No Mini Atari os jogos são compostos por uma matriz de 10×10 pixels com n canais binários, em que cada canal é responsável por representar um objeto. Essa separação foi realizada para que as redes neurais não precisem de aprender as cores dos objetos. Também foram adicionados alguns rastros dos objetos na imagem para diminuir a parcialidade da observação, além da adição de aleatoriedade nos jogos. No *Seaquest*, por exemplo, os inimigos surgem de locais aleatórios, o que acelera a aprendizagem.

Os autores do Mini Atari também propuseram uma configuração da DQN para ser aplicada aos jogos, que consiste de uma única camada convolucional seguida de uma camada oculta totalmente conectada. Vários outros parâmetros foram reduzidos em dez vezes, incluindo o *buffer* de repetição, que agora armazena as últimas 100 mil transições, e não mais o último milhão, o que reduz significativamente a necessidade de memória da máquina no treinamento.

4.15 Revisitando o Rainbow

Na mesma tendência de críticas ao alto custo computacional dos trabalhos, em (CERON; CASTRO, 2021) os autores buscaram descobrir se seria possível obter resultados próximos aos do Rainbow obtidos em (HESSEL et al., 2018), mas com um conjunto menor de ambientes, que por sua vez também são mais simples.

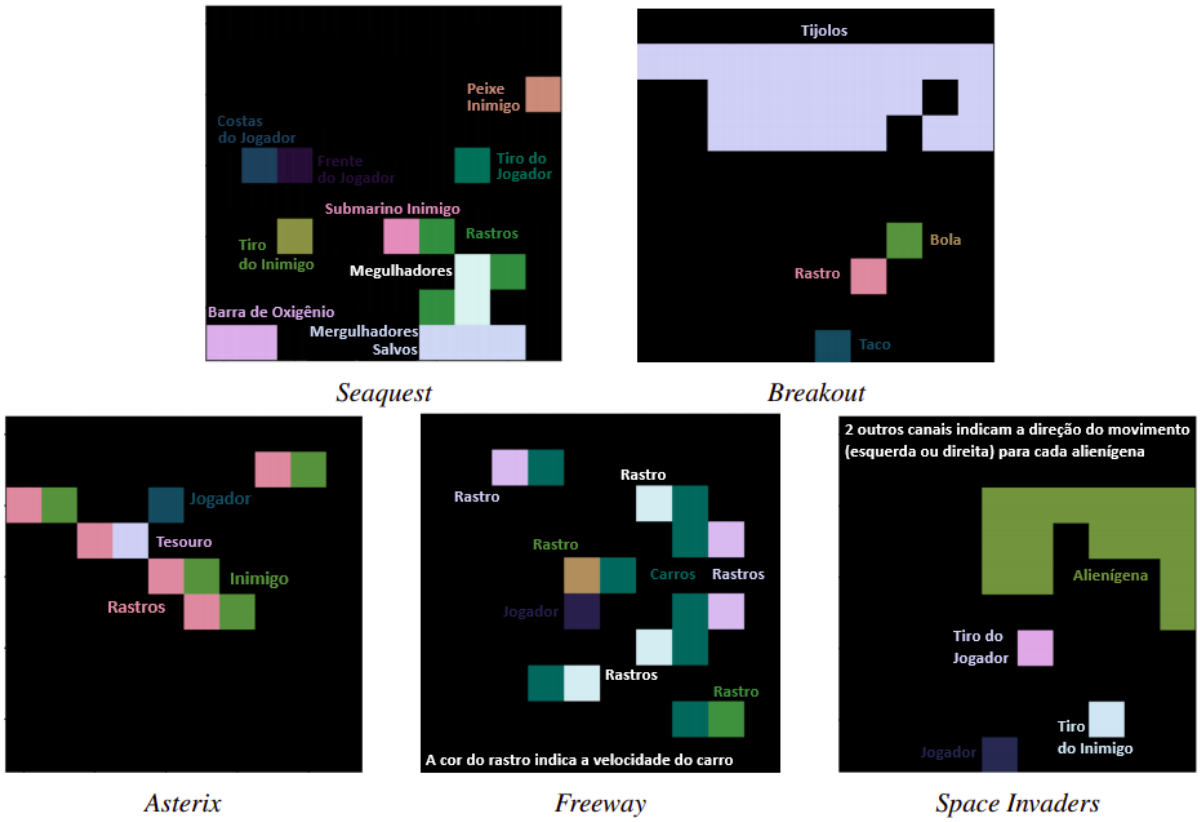


Figura 18 – Visualização dos ambientes do Mini Atari. Para melhor visualização foi adicionada uma cor para cada canal das imagens. Extraído e editado de (Young; Tian, 2019).

Os primeiros testes ocorreram nos clássicos ambientes: CartPole, Acrobot, LunarLander e MountainCar, que são ambientes que fornecem *features* para o estado, e portanto não há a necessidade de camadas convolucionais. Nesses ambientes os autores notaram que a adição do aprendizado distributivo na DQN (i.e. C51) não trouxe grandes benefícios. Essa constatação não foi encontrada em trabalhos anteriores. No entanto, na apresentação do IQN na Conferência Internacional de Aprendizado de Máquina (ICML)[3], o pesquisador Will Dabney, ao ser questionado sobre o porquê de se utilizar o Atari como benchmark, respondeu que acredita que existe uma forte relação entre os algoritmos distributivos e o aprendizado de *features* através de imagens.

O segundo conjunto de testes realizados em (CERON; CASTRO, 2021) ocorreu utilizando cinco jogos do Mini Atari. Nesses ambientes os resultados encontrados em (CERON; CASTRO, 2021) se mostraram em conformidade com os encontrados em (HESSEL et al., 2018), com 57 jogos do Atari 2600. Essa é uma importante constatação pois mostrou que também é possível realizar comparações interessantes dos algoritmos utilizando ambientes de menor escala. Além disso, os autores avaliaram versões do Rainbow com o QR-DQN e com o IQN. A combinação com o IQN, chamada de IRainbow, se mostrou promissora, sendo ainda melhorada com a adição da abordagem de Munchausen, o que

[3] Disponível em: <https://vimeo.com/312260406>

resultou no algoritmo chamado de M-IRainbow.

4.16 Considerações Gerais

A publicação da DQN foi de fato uma revolução no aprendizado por reforço moderno. A partir desse algoritmo surgiram novas linhas de pesquisas, todas relacionadas ao aprendizado por reforço profundo. Aqui foram apresentados os trabalhos mais relevantes do grupo de *Deep Q-Learning*, a qual pertence a DQN, e também dos trabalhos relacionados ao aprendizado por reforço distributivo, em que se destacam os algoritmos C51, Rainbow, QR-DQN, IQN e o FQF. Além desses também estão presentes na literatura versões recentes de *policy gradient*, como o TRPO (SCHULMAN et al., 2015); algoritmos do tipo *actor critic*, como o DDPG (LILLICRAP et al., 2016), que são muito utilizados para controle contínuo; métodos baseados em modelo; além de algoritmos distribuídos, que seguem a linha de raciocínio do A3C com múltiplos agentes aprendendo em instâncias diferentes de um mesmo ambiente.

Dentre os algoritmos distributivos abordados os trabalhos da literatura evidenciaram que C51 < QR-DQN < IQN < FQF. Em análises no Mini Atari foi encontrado que as versões melhoradas do QR-DQN e IQN também sobrepõem o Rainbow (melhoria do C51) na maioria dos jogos. Esses resultados forneceram uma forte evidência de que uma versão melhorada do FQF poderia produzir um algoritmo ainda mais poderoso, fato esse que foi explorado por este trabalho.

Arquiteturas distribuídas, como a Ape-X (HORGAN et al., 2018), ofereceram resultados surpreendentemente melhores nos jogos de Atari, pois mais agentes estão aprendendo, permitindo a criação de algoritmos mais eficientes, como o R2D2 (KAPTUROWSKI et al., 2019) e o Agent57 (BADIA et al., 2020). No entanto, essas técnicas não estão sendo abordadas neste trabalho. Os algoritmos distribuídos são gerados a partir de modificações nos algoritmos síncronos. Portanto, acreditamos que os algoritmos aqui propostos podem ser explorados futuramente para se tornarem ainda mais eficientes.

5 Metodologia

De acordo com os trabalhos abordados no Capítulo 4, é notável que houve grandes evoluções desde o surgimento da DQN. Dentre as tais, se destaca o FQF, um poderoso algoritmo distributivo. No entanto, também é notório que o FQF ainda pode ser melhorado, pois não foi capaz de superar um humano em todos os jogos avaliados do Atari 2600, o que nos motiva a encontrar um algoritmo ainda melhor, tendo em vista a capacidade da inteligência artificial em reconhecer padrões que muitas vezes não são percebidos por humanos. Considerando que técnicas como a PER e o uso de n passos foram capazes de melhorar o algoritmo C51 consideravelmente, neste trabalho é avaliado a adição de tais melhorias no FQF, que isoladamente apresentou melhores resultados que o algoritmo C51. Além desses dois métodos, é avaliado também a adição da abordagem de Munchausen, pois essa foi uma evolução que se demonstrou promissora e não foi considerada no Rainbow. Assim, pode-se listar os algoritmos avaliados da seguinte forma:

- FQF
- FQFn3: FQF + n passos com $n = 3$
- FQFP: FQF + PER.
- FQFM: FQF + MRL
- FQFn3P: FQF + 3 passos + PER
- FQFn3M: FQF + 3 passos + MRL
- FQFPM: FQF + PER + MRL
- FQFn3PM: FQF + 3 passos + PER + MRL

5.1 Implementação dos Agentes

Os algoritmos avaliados neste trabalho foram implementados utilizando a linguagem Python, reaproveitando códigos do framework Rljax[1], que é um framework para aprendizado por reforço com implementação de vários algoritmos, incluindo o FQF. O FQF, tal como todas as melhorias abordadas, foi introduzido nos capítulos anteriores. Nas próximas subseções são apresentadas diretrizes de implementação dos algoritmos.

[1] Os códigos originais do Rljax podem ser encontrados em <https://github.com/ku2482/rljax>. A versão do framework adaptada para este trabalho pode ser encontrada em <https://github.com/julio-cmdr/rljax>.

5.1.1 Treinamento das Redes do FQF

Na Seção 4.13 foi apresentada uma introdução ao FQF, contendo a definição da função de quantil $F_Z^{-1}(p)$, a forma com que essa função é utilizada para obter uma estimativa de $q_\pi(s,a)$, além de exibir os ganhos do FQF em relação a outros algoritmos, utilizando um exemplo que tem como base a métrica W_1. Esta Seção traz mais detalhes sobre o algoritmo, apresentando como ocorre o treinamento das redes.

Na Seção 4.13 também foi relatado que o FQF possui três redes: φ, ϕ e ψ. A rede φ é a responsável por gerar as frações de quantis. Para adotar o padrão de símbolos deste trabalho, os parâmetros de φ no FQF são chamados de θ_1 daqui em diante. Já a rede ψ é bem similar às redes da DQN e QR-DQN e a rede ϕ é o fluxo extra proposta pelo IQN. Na prática, essas duas últimas redes são acopladas e treinadas em conjunto. Por esse motivo, os parâmetros das duas redes são sintetizados apenas como θ_2 daqui em diante.

O pseudocódigo da função que calcula a diferença temporal para o FQF pode ser visualizado no Algoritmo 10. Inicialmente são geradas frações de quantis para os pares s,a e s,a' através da função P_{θ_1} que utiliza a rede φ. Logo depois, são geradas estimativas da função q_π para cada $a' \in \mathcal{A}(s)$, conforme a Equação 4.16, sendo encontrado, posteriormente, a ação que maximiza a função. Note que, assim como a DQN, o FQF também possui uma rede target para θ_2, sendo utilizada a representação de θ_2^-. O algoritmo calcula a diferença temporal (representada por δ) para todos os pares de quantis i,j entre a distribuição do estado e ação tomada ($Z(s,a)$) e a distribuição da ação mais promissora do próximo estado ($Z(s',a^*)$). A variável N_τ indica o número de quantis utilizados.

Algoritmo 10 Cálculo da diferença temporal do algoritmo FQF

função GET_TD_ERROR(s,a,s',r)
  $\tau \leftarrow P_{\theta_1}(s,a)$
  para $a' \in \mathcal{A}(s')$ **faça**
    $\tau^{a'} \leftarrow P_{\theta_1}(s',a')$
    $\widehat{q}(s',a';\theta_2^-) \leftarrow \sum_{i=0}^{N_\tau-1}(\tau_{i+1}^{a'} - \tau_i^{a'})F^{-1}_{Z(s',a'),\theta_2^-}(\widehat{\tau}_i^{a'})$
  fim para

  $a^* \leftarrow \underset{a'}{argmax}\widehat{q}(s',a';\theta_2^-)$

  para $i = 0$ **até** N_τ **faça**
    para $j = 0$ **até** N_τ **faça**
      $\delta_{ij} \leftarrow r + \gamma F^{-1}_{Z(s',a^*),\theta_2^-}(\widehat{\tau}_i) - F^{-1}_{Z(s,a),\theta_2}(\widehat{\tau}_j)$
    fim para
  fim para

  retorna τ, δ
fim função

O treinamento do algoritmo ocorre em batchs que são formados através de tran-

sições selecionadas aleatoriamente de um buffer, assim como na DQN. No entanto, pelo fato do FQF possuir duas redes, o treinamento ocorre em duas etapas: primeiramente é ajustado θ_1 por uma função que minimiza W_1 utilizando o otimizador RMSProp e posteriormente é atualizado θ_2 por uma versão quantílica da função de perda de Huber (HUBER, 1964) que foi proposta inicialmente em (DABNEY et al., 2018b), utilizando o otimizador Adam.

A função de perda de Huber, representada na Equação 5.1, é usada como uma alternativa menos sensível a ruídos do que o erro quadrático. Essa função utiliza um parâmetro κ e apresenta um comportamento quadrático quando o erro u é menor ou igual a κ e um comportamento linear quando o erro é maior do que κ. A versão quantílica da função de perda de Huber está representada na Equação 5.2, em que $\mathbb{1}_{predicado}$ é uma variável que assume o valor um quando o predicado é verdadeiro e zero quando é falso.

$$\mathcal{L}_\kappa(u) = \begin{cases} \frac{1}{2}u^2 & \text{se } |u| \leq \kappa \\ \kappa(|u| - \frac{1}{2}\kappa) & \text{se não} \end{cases} \tag{5.1}$$

$$\rho_\tau^\kappa(u) = |\tau - \mathbb{1}_{\{u<0\}}|\frac{\mathcal{L}_\kappa(u)}{\kappa} \tag{5.2}$$

No Algoritmo 11 está exibido um pseudocódigo do FQF, com ênfase no treinamento das redes. O fluxo principal do algoritmo segue como na DQN, com a escolha de transições de um buffer e o uso de uma rede target, que é atualizada de tempos em tempos. O ciclo mais interno do código é composto por uma estrutura que realiza k repetições, sendo k o tamanho do mini batch. Dentro dessa estrutura, o algoritmo seleciona uma transição do buffer de forma uniformemente aleatória, calcula a diferença temporal utilizando a função representada no Algoritmo 10 e posteriormente calcula a função de perda para as redes θ_1 e θ_2, respectivamente.

O ajuste de θ_2 ocorre utilizando a Equação 5.2, utilizando a diferença temporal como medida de erro. Já o ajuste de θ_1 representa uma forma indireta de minimizar a métrica 1-Wassertein[2], assim como na Figura 17. No entanto, como não se sabe o verdadeiro valor da função de quantil F_z^{-1} no momento do cálculo da função de perda, é utilizada a estimativa gerada por θ_2. Note que como θ_1 não é treinada em função de si mesma, não é utilizada uma rede target θ_1^-, como acontece com θ_2.

Diferentemente do pseudocódigo do Algoritmo 11, na implementação utilizada não existe uma estrutura de repetição para computar o mini batch, pois são utilizadas operações em matrizes multidimensionais, que são paralelizadas através da biblioteca Jax[3]. Portanto calcula-se a diferença temporal para todas as k transições primeiramente e depois a função de perda para todas transições.

[2] Provas matemáticas da equivalência podem ser encontradas nos apêndices de (YANG et al., 2019)
[3] Mais informações em <https://jax.readthedocs.io/en/latest/index.html>

Algoritmo 11 FQF com Experiência por Repetição Não Priorizada

Entrada: Um minibatch de tamanho k, um buffer de tamanho $N_{\mathcal{H}}$, frequência de atualização K das redes
Inicialize o buffer $\mathcal{H} = \emptyset$
Observe S_0 e escolha $A_0 = \pi_\theta(S_0)$

para $t = 1$ **até** T **faça**
  Observe S_T e R_T
  Armazene a transição $(S_{t-1}, A_{t-1}, R_t, S_t)$ em $\mathcal{H}$
  se t mod $K == 0$ **então**
    $\mathcal{L} = \frac{\partial W_1}{\partial \tau_i} = 0$
    para $l = 0$ **até** k **faça**
      Escolha uma transição $x \sim P(x) = 1/N_{\mathcal{H}}$
      $s, a, s', r \leftarrow x$
      $\tau, \delta \leftarrow GET\_TD\_ERROR(s, a, s', r)$
      $\mathcal{L} = \mathcal{L} + \frac{1}{k}\left(\frac{1}{N_\tau}\sum_{i=0}^{N_\tau - 1}\sum_{j=0}^{N_\tau - 1}\rho^1_{\widehat{\tau}_j}(\delta_{ij})\right)$
      $\frac{\partial W_1}{\partial \tau_i} = \frac{\partial W_1}{\partial \tau_i} + \frac{1}{k}\left(2F^{-1}_{Z(s,a),\theta_2}(\tau_i) - F^{-1}_{Z(s,a),\theta_2}(\widehat{\tau}_i) - F^{-1}_{Z(s,a),\theta_2}(\widehat{\tau}_{i-1})\right)$
    fim para
    Atualize θ_1 com $\frac{\partial W_1}{\partial \tau_i}$ e θ_2 com $\nabla\mathcal{L}$
    De tempos em tempos, atualize $\theta_2^- \leftarrow \theta_2$
  fim se
  Escolha $A_t = \pi_\theta(S_t)$
fim para

5.1.2 FQF com Experiência por Repetição Priorizada

A Experiência por Repetição Priorizada, proposta em (SCHAUL et al., 2016) e introduzida na Seção 4.4, conta com uma estrutura de dados que armazena valores utilizados para calcular a probabilidade de cada transição ser selecionada, utilizando para isso a diferença temporal (TD). Inicialmente, toda transição é inserida no buffer com prioridade máxima, mas quando ela é selecionada, ocorre a modificação de sua prioridade de seleção, de acordo com o módulo da TD. No caso do FQF, foi utilizada a média do módulo das diferenças temporais geradas por cada fração de quantil. O Algoritmo 12 exibe um pseudocódigo da combinação do FQF com a abordagem proporcional da PER proposta em (SCHAUL et al., 2016), em que consta também a ponderação de IS, ainda não apresentada.

No Algoritmo 12 a implementação do buffer de transições é abstraída. Mas entendê-la é importante para a compreensão da PER. O buffer pode ser definido como um vetor de tamanho fixo contendo, em cada posição, uma transição de estado. A transição de estado é composta por cinco variáveis: estado atual, ação tomada pelo agente no estado, estado do agente no próximo passo, recompensa obtida pelo agente e mais uma variável de controle, para indicar se o agente chegou em um estado final ou não. Na implementação é utilizado um contador para marcar qual posição o algoritmo deve inserir a transição em

Algoritmo 12 FQF com Experiência por Repetição Priorizada

Entrada: Um minibatch de tamanho k, um buffer de tamanho $N_{\mathcal{H}}$, expoentes α_{per} e β_{per}, frequência de atualização K das redes
Inicialize o buffer $\mathcal{H} = \emptyset$, $p_0 = 1$
Observe S_0 e escolha $A_0 = \pi_\theta(S_0)$

para $t = 1$ **até** T **faça**
  Observe S_T e R_T
  Armazene a transição $(S_{t-1}, A_{t-1}, R_t, S_t)$ em $\mathcal{H}$ com max. prioridade $p_t = max_{i<t} p_i$
  se t mod $K == 0$ **então**
    $\mathcal{L} = \frac{\partial W_1}{\partial \tau_i} = 0$
    para $l = 0$ **até** k **faça**
      Escolha uma transição $x \sim P(x) = p_j / \sum_i p_i$
      Compute a ponderação da IS $\varrho = (N \cdot P(x))^{-\beta_{per}} / max_i \varrho_i$
      $s, a, s', r \leftarrow x$
      $\tau, \delta \leftarrow GET\_TD\_ERROR(s, a, s', r)$
      Atualize a prioridade de seleção de x com $\frac{1}{N_\tau} \sum_{i=0}^{N_\tau - 1} \sum_{j=0}^{N_\tau - 1} |\delta_{ij}|^{\alpha_{per}}$
      $\mathcal{L} \leftarrow \mathcal{L} + \frac{1}{k} \varrho \left(\frac{1}{N_\tau} \sum_{i=0}^{N_\tau - 1} \sum_{j=0}^{N_\tau - 1} \rho^1_{\widehat{\tau}_j}(\delta_{ij}) \right)$
      $\frac{\partial W_1}{\partial \tau_i} \leftarrow \frac{\partial W_1}{\partial \tau_i} + \frac{1}{k} \left(2F^{-1}_{Z(s,a),\theta_2}(\tau_i) - F^{-1}_{Z(s,a),\theta_2}(\widehat{\tau}_i) - F^{-1}_{Z(s,a),\theta_2}(\widehat{\tau}_{i-1}) \right)$
    fim para
    Atualize θ_1 com $\frac{\partial W_1}{\partial \tau_i}$ e θ_2 com $\nabla \mathcal{L}$
    De tempos em tempos, atualize $\theta_2^- \leftarrow \theta_2$
  fim se
  Escolha $A_t = \pi_\theta(S_t)$
fim para

cada passo. Esse contador inicia na posição zero e vai sendo incrementado de um em um, a cada passo do agente. Quando o contador atinge o limite do buffer, o valor é reiniciado, de forma que o algoritmo passa a substituir as transações mais antigas, presentes no início do buffer.

Na versão não priorizada, a cada passo de treinamento, são selecionadas amostras de forma uniformemente aleatória para formarem um mini batch. Na versão priorizada, é necessário uma outra estrutura de dados, que deve conter, para cada transição, um valor, aqui chamado de indicador de probabilidade, que será utilizado para calcular a probabilidade da transição ser selecionada. Assim, para selecionar uma transição, é utilizado o método de roleta, ou seja: é gerado um número aleatório pertencente ao intervalo de zero até a soma de todos os indicadores de probabilidade. Para encontrar a posição da transição, é necessário percorrer a estrutura de dados, realizando a soma dos indicadores de probabilidade.

Como esse processo é realizado várias vezes, é necessário que seja utilizada uma estrutura de dados que permita a soma dos indicadores de probabilidade de forma eficiente. Por esse motivo, é utilizado na implementação da PER uma árvore de segmentos para

armazenar os indicadores de probabilidade. Essa estrutura é uma árvore binária em que todos os valores se encontram nas folhas, sendo que os nós são responsáveis por armazenar as somas (ou outra operação) dos filhos. Na Figura 19 são apresentadas duas ilustrações de uma árvore de segmentos com oito elementos e operação de soma. Na primeira ilustração os dados estão organizados em árvore e na segunda ilustração, os dados estão dispostos em um vetor, conforme ocorre na implementação da estrutura.

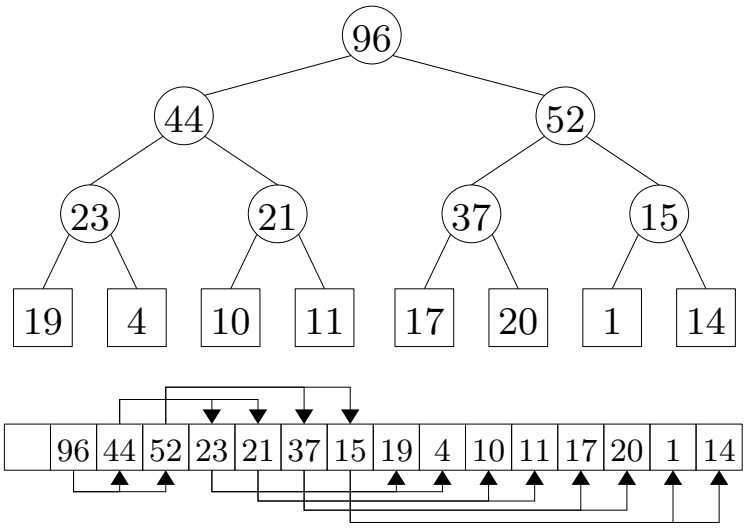


Figura 19 – Exemplo de uma árvore de segmentos com 8 elementos e operação de soma. Ilustração no formato de árvore e também no formato vetorial.

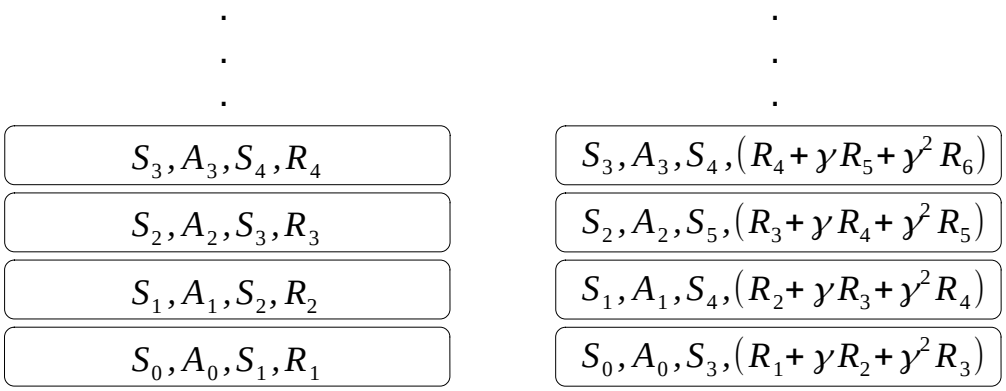
Utilizando uma árvore de segmentos, a complexidade para alterar o valor de um elemento é $O(logn)$, pois é necessário propagar o valor do elemento para os nós superiores da árvore, enquanto em uma estrutura linear simples a modificação de elementos ocorre em $O(1)$. No entanto, para selecionar um valor da árvore baseado no método da roleta, a estrutura linear simples tem complexidade de $O(n)$, enquanto a árvore de soma permanece com complexidade $O(logn)$, o que a torna mais vantajosa (SCHAUL et al., 2016).

5.1.3 FQF com n Passos

Embora o uso de n Passos com algoritmos de TD tenha sido proposto no ano de 1989 (WATKINS, 1989), essa ainda é uma técnica que tem sido capaz de melhorar vários algoritmos mais novos, como constatado em (HESSEL et al., 2018) e (CERON; CASTRO, 2021). Na Seção 2.5 o uso de n passos foi introduzido como um ponto de equilíbrio entre TD(0) e MC, elencando os benefícios da técnica. Aqui nesta seção é descrito como implementá-la para o FQF. No entanto, como vai ser possível perceber, essa é uma implementação genérica para qualquer algoritmo que usa experiência por repetição, pois a implementação ocorre no buffer de transições.

Utilizando TD(0), o módulo responsável por gerir o buffer apenas recebe uma transição e a armazena, contendo: estado atual, ação tomada pelo agente no estado,

estado do agente no próximo passo, recompensa obtida pelo agente e mais uma variável de controle, para indicar se o agente chegou em um estado final ou não. Quando se usa n passos, a gestão do buffer precisa acumular n passos antes de salvar a transição no buffer, o que pode ser feito com uma fila de tamanho n. A Figura 20 apresenta a comparação entre dois buffers, sendo que no primeiro é utilizado um passo e no segundo três passos. Para a ilustração, foi removida a variável de controle. Note que no buffer de três passos as recompensas dos próximos três passos já são inseridas somadas no buffer, com a devida ponderação do fator de desconto. Da mesma forma, o estado a ser utilizado para calcular o target da rede também já é o estado postergado com n passos. Portanto, com as transições sendo salvas já preparadas no buffer, a única alteração na implementação do agente, é que o fator de desconto na elaboração do target da rede passa a ser γ^n, ao invés de apenas γ.



(a) Transições para TD com um passo (b) Transições para TD com três passos

Figura 20 – Ilustrações dos buffers utilizados na experiência por repetição, para um e três passos respectivamente.

Essa descrição do buffer foi realizada de acordo com o formato que foi implementado no framework Rljax, que originalmente já possui suporte ao uso de n passos para todos algoritmos do framework, sendo essa a implementação utilizada neste trabalho. No entanto, é perceptível na Figura 20 que os estados aparecem duplicados nos buffers (tanto com um quanto com n passos), pois um mesmo estado faz parte de duas transições. Dessa forma, em aplicações em que o estado é computacionalmente pesado (como imagens de alta resolução), vale a pena adicionar os estados em um vetor diferente e fazer com que as transições apenas façam referência para a posição dos estados em outro buffer.

5.1.4 FQF com Munchausen

Na Seção 4.12 a abordagem de Munchausen (MRL) foi apresentada através de sua junção na DQN. Nesta seção é apresentado como ocorreu a junção de MRL com o FQF. Mais uma vez, para exemplificar essa combinação foi utilizado um pseudocódigo, exibido no Algoritmo 13. No pseudocódigo é apresentado uma modificação da função *GET_TD_ERROR()*, exibida no Algoritmo 10. Note que a função agora fica um pouco

maior, pois é necessário calcular a política de aprendizagem π tanto para s quanto para s'. Além disso, aparecem três novos parâmetros: α_{mrl}, que realiza uma ponderação do fator de munchausen; o fator de temperatura τ utilizado na política *softmax* e um limitante inferior l_0 para o fator de munchausen.

Algoritmo 13 Cálculo da diferença temporal do algoritmo FQF utilizando MRL

Entrada: Fator de temperatura τ, parâmetro α_{mrl}, limitante inferior l_0
função GET_MUNCHAUSEN_TD_ERROR(s, a, s', r)
  para $a' \in \mathcal{A}$ **faça**
    $\tau^{s,a'} \leftarrow P_{\theta_1}(s, a')$
    $\tau^{s',a'} \leftarrow P_{\theta_1}(s', a')$
    $\widehat{q}(s', a'; \theta_2^-) \leftarrow \sum_{i=0}^{N_\tau - 1}(\tau_{i+1}^{s',a'} - \tau_i^{s',a'}) F^{-1}_{Z(s',a'),\theta_2^-}(\widehat{\tau}_i^{s',a'})$
    $\widehat{q}(s, a'; \theta_2) \leftarrow \sum_{i=0}^{N_\tau - 1}(\tau_{i+1}^{s,a'} - \tau_i^{s,a'}) F^{-1}_{Z(s,a'),\theta_2}(\widehat{\tau}_i^{s,a'})$
  fim para

  $\pi(a'|s') \leftarrow softmax(\frac{\widehat{q}(s',a';\theta_2^-)}{\tau})$ # para cada $a' \in \mathcal{A}$
  $\pi(a|s) \leftarrow softmax(\frac{\widehat{q}(s,a';\theta_2)}{\tau})$

  para $i = 0$ **até** N_τ **faça**
    $Y_i \leftarrow r + \alpha_{mrl}\left[\tau ln(\pi(a|s))\right]_{l_0}^{0} + \gamma \sum_{a'} \pi(a'|s') \left(F^{-1}_{Z(s',a'),\theta_2^-}(\widehat{\tau}_i^{s,a}) - \tau ln(\pi(a'|s')) \right)$
    para $j = 0$ **até** N_τ **faça**
      $\delta_{ij} \leftarrow Y_i - F^{-1}_{Z(s,a),\theta_2}(\widehat{\tau}_j^{s,a})$
    fim para
  fim para

  retorna $\tau^{s,a}, \delta$
fim função

Um detalhe importante na implementação do MRL, é que a função *softmax* em sua versão original pode sofrer com problemas de *overflow*. Por isso é necessário adotar uma versão mais estável da implementação, fazendo conforme na Equação 5.3. Da mesma forma, a aplicação direta do logaritmo natural em uma função *softmax* também é considerada problemática, e por isso para calcular $\tau ln(\pi(a|s))$ é utilizado o cálculo demonstrado na Equação 5.4.

$$softmax_i(x, \tau) = \frac{e^{\frac{x_i - max(x)}{\tau}}}{\sum_j e^{\frac{x_j - max(x)}{\tau}}} \tag{5.3}$$

$$\tau ln(\pi(a|s)) \approx q(s,a) - \max_{a'} q(s,a') - \tau ln(\sum_{a'} e^{\frac{q(s,a) - max(q(s,a'))}{\tau}}) \tag{5.4}$$

5.1.5 O Agente integrado

Todas as três melhorias descritas podem ser combinadas, sem a necessidade de ajustes extras. O uso de n passos e a PER demandam alterações no buffer, mas elas não conflitam, uma vez que a repetição priorizada adiciona uma estrutura extra para armazenar indicadores de probabilidades e a técnica de n passos apenas altera a forma com que as transições são adicionadas no buffer, acumulando-as antes. Da mesma forma, MRL pode ser combinado com ambas melhorias, sendo apenas necessário utilizar γ^n no fator de desconto quando usado n passos, assim como também é necessário alterar na versão original do FQF.

5.2 Ambientes de Avaliação

Neste trabalho, os algoritmos foram avaliados nos cinco jogos do Mini Atari, que são ambientes de menor complexidade gráfica, mas que demonstraram ser bons para testes em (CERON; CASTRO, 2021). As subseções abaixo listam características da modelagem de cada um dos jogos.

5.2.1 Asterix

Asterix é um jogo em que surgem inimigos e tesouros pelas laterais, os quais andam horizontalmente e deixam sempre um rastro. O jogador pode se mover livremente ao longo das 4 direções cardeais e uma recompensa de +1 é dada sempre que o jogador pega um tesouro. No entanto, se o jogador tiver contato com um inimigo o episódio é encerrado. No jogo a dificuldade é aumentada periodicamente através de acréscimos na velocidade e na taxa de geração de inimigos e tesouros (Young; Tian, 2019).

5.2.2 Breakout

No breakout existe uma bola que viaja ao longo das diagonais, refletindo em cada objeto em que ocorre a colisão. O jogador controla uma raquete na parte inferior da tela utilizada para rebater a bola e quebrar um ou mais tijolos de uma parede representada na parte superior do jogo. Uma recompensa de +1 é dada para cada tijolo quebrado pela bola. Quando todos os tijolos são limpos, outras três linhas são adicionadas. Os episódios terminam quando o jogador não é capaz de rebater uma bola. A direção da bola é indicada por um canal de trilha (Young; Tian, 2019).

5.2.3 Freeway

O desafio do freeway consiste em fazer com que o jogador atravesse uma rodovia movimentada, com carros que movimentam horizontalmente. O jogador começa na parte

inferior da tela e o movimento é restrito a subir e descer. A velocidade do jogador também é restrita de tal forma que o jogador só pode se mover a cada três quadros. Sempre que o jogador consegue terminar a travessia da rodovia, ele recebe uma recompensa de +1, momento em que o jogador retorna ao fundo da tela. Quando atingido por um carro, o jogador também retorna para a parte inferior da tela, mas não recebe recompensa e nem é considerado fim de episódio, pois nesse jogo os episódios possuem duração fixa de 2.500 quadros. A direção e a velocidade dos carros são indicadas por cinco canais de trilha. A localização da trilha representa a direção, enquanto o canal que ela está indica a frequência com que o carro se move (de uma vez a cada quadro até uma vez a cada cinco quadros). Sempre que o jogador atinge com sucesso o topo da rodovia, as velocidades dos carros são redefinidas aleatoriamente (Young; Tian, 2019).

5.2.4 Seaquest

No jogo Seaquest o jogador controla um submarino composto por duas posições alinhadas horizontalmente, frontal e traseira, o que permite que a direção do submarino seja determinada. Os inimigos consistem em submarinos e peixes, que são distinguidos pelo fato de que os submarinos disparam balas e os peixes não. O jogador também pode disparar balas da frente do submarino e receber uma recompensa de +1 sempre que atingir um inimigo, momento em que o inimigo é removido. Existem também no jogo mergulhadores em que o jogador pode se mover para pegá-los, o que incrementa uma barra indicada por outro canal na parte inferior da tela, mas o submarino comporta no máximo seis mergulhadores ao mesmo tempo. O submarino do jogador tem um suprimento limitado de oxigênio, que é indicado por uma barra representada em um outro canal da barra de mergulhadores. O oxigênio se degrada ao longo do tempo e é reabastecido sempre que o jogador se move para a superfície do mar, desde que o jogador tenha pelo menos um mergulhador resgatado a bordo. Se o jogador emergir do mar com menos de seis, apenas um mergulhador é liberado. Ao emergir com seis, todos os mergulhadores são liberados e uma recompensa é dada para cada célula ativa na barra de oxigênio. Sempre que o submarino sobe até a superfície, a dificuldade é incrementada através do aumento da velocidade e da taxa de geração dos inimigos. O episódio é encerrado quando o jogador é atingido por um peixe inimigo, submarino ou bala; quando o oxigênio do submarino esgota ou então quando o jogador tenta emergir sem mergulhadores resgatados. Assim como acontece nos outros jogos, as direções dos objetos também são indicadas através de um canal que indica a posição anterior dos objetos para reduzir a observabilidade parcial (Young; Tian, 2019).

5.2.5 Space Invaders

No jogo Space Invaders o jogador controla um canhão que navega horizontalmente na parte inferior da tela e pode disparar balas para cima, com o objetivo de eliminar um dos vários alienígenas que ficam na parte superior do jogo. Os alienígenas se movem pela tela até que um deles atinja a borda, momento em que todos se movem para baixo e mudam de direção. A direção atual do alienígena é indicada por dois canais (um para a esquerda e outro para a direita), dessa forma apenas um dos dois canais está ativo na localização de cada alienígena. Uma recompensa de +1 é dada cada vez que um alienígena é baleado, momento em que é removido. Os alienígenas também atiram em direção ao jogador, que deve ser capaz de desviar. Quando restam poucos alienígenas, a velocidade deles começa a aumentar até o ponto em que resta apenas um alienígena, que moverá uma célula por quadro. Quando um grupo de alienígenas for totalmente eliminado, um novo grupo mais ágil surgirá novamente ao topo da tela. O episódio termina quando o jogador tem contato ou é baleado por um alienígena (Young; Tian, 2019).

5.2.6 Considerações dos Ambientes

Com exceção do jogo Breakout, que possui apenas um nível de complexidade e os objetos possuem velocidade fixa, todos os outros jogos considerados são parcialmente observáveis (Young; Tian, 2019). O Seaquest é um jogo que apresenta vários desafios ao agente, pois possui muitas regras e detalhes a serem aprendidos, um número maior de ações possíveis e as recompensas são bem espaçadas. O Freeway apresenta uma particularidade que é o fato de o episódio não terminar quando o jogador é atingido por um carro, mas sim quando um número fixo de episódios ocorre, o que o torna mais tolerante a falhas. No entanto, assim como ocorre no Astrix e Space Invaders, o Freeway também possui um acréscimo de dificuldade ao longo do episódio, o que traz desafios de aprendizagem para o agente.

5.3 Métricas de Avaliação

Os oito algoritmos foram avaliados nos cinco jogos do Mini Atari em um processo contínuo de treinamento, de modo que é possível ver a evolução do desempenho dos mesmos. Apesar de ser um processo de aprendizado sem fim, é necessário estipular um número máximo de passos a serem observados.

Nos testes realizados, os algoritmos foram avaliados durante o processo contínuo de treinamento, por 20 milhões de passos, os quais foram agrupados em 20 iterações, sendo que cada iteração, com 1 milhão de passos, representa vários episódios. Note que nesse tipo de agrupamento por número de passos, o número de episódios em cada iteração varia de acordo com o desempenho do agente. É normal que no início o agente tenha um baixo

desempenho, o que faz com que os episódios sejam mais curtos e portanto mais numerosos em uma mesma iteração (exceto em ambientes com episódios de duração fixa). O inverso acontece quando o agente já está mais maduro. Nesse caso, é esperado que ele sobreviva por mais tempo nos episódios, acumulando, assim, mais recompensas. Por esse motivo, medidas de agregação em relação ao retorno dos episódios representam boas alternativas para avaliações dos agentes.

O processamento dos dados para a análise pode realizar até três aplicações de medidas de agregação sequenciais, que são: 1) sumarização dos retornos de uma determinada iteração de uma execução; 2) agregação dos retornos sumarizados de diferentes execuções em um mesmo ambiente de avaliação e 3) agrupamento dos valores gerados pela aplicação do algoritmo em diferentes tarefas. Em muitos trabalhos, como em (YANG et al., 2019) e (DABNEY et al., 2018a) tem sido utilizado a média e a mediana como medidas de agregação. No entanto, medidas como média e mediana sozinhas não são capazes de representar a diversidade das amostras, o que pode acabar ocultando comportamentos importantes dos algoritmos. Além disso, conforme relatado em (AGARWAL et al., 2021), o alto custo computacional faz com que muitos pesquisadores utilizem um número baixo de sementes, como três, cinco ou, quando muito, dez sementes. Sendo assim, em (AGARWAL et al., 2021) é apresentado um guia com várias sugestões de como analisar e comparar o desempenho de algoritmos de aprendizado por reforço profundo, considerando para tal a conhecida limitação do número de sementes utilizadas por ambientes que existe na linha de pesquisa.

A média é uma medida que considera todos os valores de um conjunto de amostras, mas é sensível a ruídos. Esse comportamento não acontece na mediana, que é uma medida de tendência central. No entanto, se um determinado conjunto de amostras possuir quase metade dos valores zerados, por exemplo, a mediana pode não ser afetada. Uma forma de combinar os benefícios das duas métricas, sugerido em (AGARWAL et al., 2021), é o uso da Média Interquartil (IQM). Essa medida realiza a média das amostras presentes no intervalo interquartil, descartando 50% dos dados para o cálculo da média. A IQM é utilizada neste trabalho no primeiro nível de agregação, que sumariza os retornos de uma mesma iteração. A equação Equação 5.5 exibe o cálculo do IQM para um vetor ordenado x de tamanho n, em que x_i representa o i-ésimo componente do vetor.

$$IQM_x = \frac{2}{n} \sum_{i=\frac{n}{4}+1}^{\frac{3n}{4}} x_i \tag{5.5}$$

Nas análises principais dos algoritmos deste trabalho, são utilizadas cinco sementes para cada algoritmo em cada jogo. Logo, como o número de sementes não é alto, para agregar o IQM dos retornos das execuções provenientes de cada semente em um determinado jogo é utilizada média simples. O intervalo de confiança para um nível de 95%

também é avaliado através de gráficos. Para comparar a performance geral dos algoritmos sobre todos os cinco jogos, é necessário que os dados sejam padronizados, pois, como apresentado na Seção 5.2, cada ambiente do Mini Atari tem uma modelagem única.

Em outros conjuntos de ambientes, como o ALE, as pontuações dos algoritmos são calculadas usualmente em relação a uma base de dados com pontuações de jogadores humanos por jogos. No caso do Mini Atari, não existe na literatura um consenso para ser possível agrupar as métricas extraídas através da aplicação em diversos jogos e não foram encontradas bases com pontuações de humanos. Portanto, neste trabalho é utilizada a padronização *z score* para as execuções referentes a cada jogo. Considerando que são oito algoritmos, avaliados com cinco sementes em cada jogo, essa transformação ocorre nos grupos de 40 amostras (IQM de retornos) de cada jogo.

A padronização *z score* é uma transformação muito utilizada para pré processamento de dados a serem utilizados em algoritmos de aprendizado supervisionado e não supervisionado. Essa transformação, exibida na Equação 5.6, faz com que a distribuição tenha média zero e desvio padrão um. Logo, no contexto deste trabalho, ao serem unidas as execuções referentes a diferentes jogos, a padronização *z score* oferece uma confiança de que, no geral, os resultados provenientes de um jogo não vão se sobressair em relação aos demais.

$$z_{x_i} = \frac{x_i - média(x)}{DesvPad(x)} \tag{5.6}$$

Com os dados padronizados, é possível unir os resultados de cada jogo. Portanto, considerando que são utilizadas cinco sementes em cinco jogos, cada algoritmo possui 25 amostras de desempenho. Esse número maior permite que análises mais detalhadas sejam feitas, como por exemplo o uso de um box plot e de perfis de performance, como é sugerido em (AGARWAL et al., 2021). Outra sugestão de análise no trabalho citado é o cálculo da probabilidade de um algoritmo X ser melhor do que outro algoritmo Y. Considerando que são utilizados M jogos, com N execuções em cada jogo, a probabilidade de X ser maior do que Y em um jogo m pode ser encontrada conforme exibido na Equação 5.7, que faz o uso da Estatística U de Mann-Whitney (MANN; WHITNEY, 1947). Já a probabilidade de X ser melhor do que Y em um contexto geral dos M jogos pode ser encontrada através da Equação 5.8.

$$P(X_M > Y_m) = \frac{1}{N^2} \sum_{i=0}^{N} \sum_{j=0}^{N} S(x_{m,i}, y_{m,i}), \text{ sendo } S(x,y) = \begin{cases} 1 & \text{se } y < x \\ \frac{1}{2} & \text{se } y = x \\ 0 & \text{se } y > x \end{cases} \tag{5.7}$$

$$P(X > Y) = \frac{1}{M} \sum_{m=0}^{M} P(X_m > Y_m) \tag{5.8}$$

6 Experimentos e Resultados

Neste capítulo são apresentados os resultados dos experimentos realizados. Os experimentos foram executados em duas máquinas: a máquina A é composta por um processador Intel(R) Xeon(R) W-2155 CPU @ 3.30GHz, com 64Gb de memória ram e uma GPU Quadro P1000 de 4Gb. Já a máquina B é composta por um processador Intel(R) Core(TM) i7-4960X CPU @ 3.60GHz, 64Gb de memória ram e duas GPUs GeForce GTX TITAN X, com 12Gb cada uma. Em todas as execuções houve o cuidado de definir a flag *TF_CUDNN_DETERMINISTIC=1*, com o objetivo de garantir que o comportamento dos algoritmos que realizam cálculos matriciais nas GPUs seja determinístico, o que faz com que o comportamento estocástico do algoritmo seja definido apenas pela semente fixada. No entanto, durante testes notou-se que mesmo fixando a semente, pode haver distinção dos resultados de um mesmo experimento quando utilizadas máquinas distintas. Por esse motivo, as execuções que utilizam uma mesma semente não foram divididas em máquinas diferentes, com o objetivo de garantir que os algoritmos comparados estão sendo avaliados sob a mesma condição.

Nas análises realizadas a evolução dos algoritmos foi medida em função do número de passos e o tempo de execução real não foi alvo de investigações. No entanto, para que o leitor possua uma noção do custo computacional desses algoritmos, uma única execução do FQF para 20 milhões de passos na máquina A leva em torno de 10 horas para ser executada. A máquina B, possivelmente por ter uma CPU inferior que a máquina A, oferece um desempenho pior. No entanto, a máquina B possui duas ótimas GPUs, o que permitiu que fossem executadas múltiplos experimentos em paralelo, de modo a obter um desempenho similar ao da máquina A.

6.1 Ajustes de Hiperparâmetros

Assim como vários dos algoritmos citados neste trabalho, o FQF também possui um grande conjunto de hiperparâmetros, que foram propostos originalmente baseados principalmente na parametrização proposta para a DQN no ALE. Não foi encontrado na literatura avaliações do FQF no Mini Atari. Portanto, neste trabalho foi utilizado como base os parâmetros propostos em (CERON; CASTRO, 2021), que avaliou diversos algoritmos no Mini Atari, com exceção do FQF. Parâmetros que são particulares do FQF, como a taxa de aprendizado da rede geradora de frações de quantis, foram inicialmente mantidos conforme proposto originalmente em (YANG et al., 2019).

No entanto, em testes preliminares no Mini Atari realizados durante esta pesquisa, especialmente no jogo Space Invaders (exemplo na Figura 21), o FQF apresentou

características não desejadas, que foram identificadas como o problema do esquecimento catastrófico, ou o problema da inferência catastrófica. Esse é um problema clássico de redes neurais em aprendizado contínuo, em que o ajuste da rede para uma nova experiência pode fazer com que ela esqueça aprendizados adquiridos anteriormente (MCCLOSKEY; COHEN, 1989), gerando uma queda brusca do desempenho do agente. No entanto, embora seja um problema comum, esse comportamento não foi relatado nos trabalhos anteriores consultados, possivelmente devido ao uso das várias técnicas propostas originalmente na DQN, como por exemplo a experiência por repetição. Por esse motivo, identificamos que esse comportamento poderia ter sido ocasionado por hiperparâmetros mal ajustados, o que motivou a realização de ajustes de hiperparâmetros.

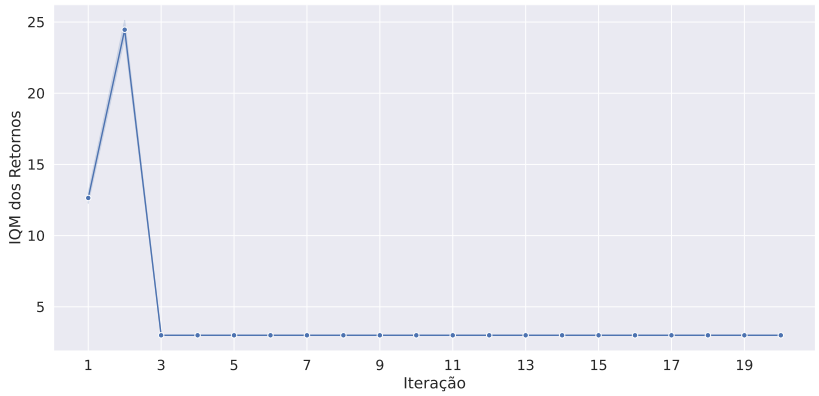


Figura 21 – Exemplo de execução em que ocorreu um esquecimento catastrófico. Resultados provenientes de testes preliminares do FQF no jogo Space Invaders.

Fazer o ajuste de um grande número de hiperparâmetros pode resultar facilmente em uma explosão combinatória, o que é agravado quando se trata de *deep learning*, que são algoritmos computacionalmente custosos. Sendo assim, foi priorizada a escolha para o ajuste de três hiperparâmetros que estão altamente relacionados à capacidade de aprendizado do algoritmo: a taxa de aprendizado da rede de frações de quantis (lr_frac_net), que não foi ajustado para o Mini Atari; a taxa de aprendizado da rede principal (lr) e o tamanho do mini batch (batch_size).

Foi realizada uma avaliação combinacional desses três hiperparâmetros, sendo que em cada um, através de alterações no expoente, foi adicionada uma alternativa maior e outra menor que o valor utilizado inicialmente (extraídos de (CERON; CASTRO, 2021) e (YANG et al., 2019)), o que resultou em 27 combinações. Para o batch_size, usualmente definido como potência de dois, foram avaliados os valores 16, 32 e 64. No caso do lr foram avaliados os valores 2,5e-05; 2,5e-04 e 2,5e-03. Para o lr_frac_net avaliados os valores 2,5e-10; 2,5e-09 e 2,5e-08. A avaliação ocorreu com nove sementes distintas no jogo Space Invaders, que foi o ambiente em que, nos testes preliminares, ocorreu o esquecimento catastrófico do algoritmo.

O objetivo dessa avaliação é encontrar um conjunto de hiperparâmetros que não levem o algoritmo a ter uma queda brusca de seu desempenho, pois esse é um comportamento prejudicial. Portanto, foi adotada uma heurística de poda entre as sementes. Inicialmente todas as 27 configurações do FQF foram avaliados na semente zero. Posteriormente na semente um, somente os algoritmos que não sofreram queda brusca na semente anterior e assim sucessivamente, até a última semente.

O resultado da avaliação pode ser visto na Figura 22, que mostra a evolução da média dos retornos dos algoritmos por iteração. As execuções foram interrompidas sempre que o IQM dos retornos de uma iteração foi menor ou igual a 50% ao maior IQM obtido pelo algoritmo na semente em questão. Nos gráficos, observa-se que após cinco execuções, apenas um conjunto de hiperparâmetros se manteve, que é o conjunto que faz o uso de batch_size = 16, lr_frac_net = 2,5e-10 e lr = 2,5e-05, que foi o único hiperparâmetro que manteve o valor inicial. Como apenas um conjunto de hiperparâmetros restou, não foi necessário comparar as métricas de análises nesse passo.

Realizados os ajustes do FQF, chega o momento de verificar os hiperparâmetros que são introduzidos com cada melhoria. A repetição priorizada introduz dois novos hiperparâmetros: α_{per} e β_{per}, ambos pertencentes ao intervalo $[0, 1]$. O hiperparâmetro α_{per} é utilizado no expoente do cálculo do indicador de probabilidade. Isso indica que se α_{per} for definido igual a zero, a escolha dos hiperparâmetros vai seguir uma distribuição uniforme. Já o hiperparâmetro β_{per} é utilizado no expoente do cálculo da IS, que também pode ser ignorado ao utilizar $\beta_{per} = 0$.

Em (SCHAUL et al., 2016) β_{per} é definido inicialmente igual a 0,4 com seu valor sendo incrementado linearmente até atingir 1 ao final do treinamento, e o hiperparâmetro α_{per} é definido como 0,6 em todo treinamento. Já em (HESSEL et al., 2018) é utilizado a mesma estratégia de valores para β_{per}, mas α_{per} é definido como 0,5. Tendo em vista a divergência entre o valores para α_{per}, foi feita uma análise do comportamento do FQF utilizando os valores 0,5; 0,55; e 0,6 para α_{per}. Mais uma vez foi escolhido o jogo Space Invaders, mas nessas execuções não houve quedas bruscas de desempenho do algoritmo, uma vez que já estão sendo utilizados os hiperparâmetros ajustados anteriormente.

O resultado pode ser visto na Figura 23, que exibe a evolução da média e da mediana dos retornos para as três variações do FQFP, com valores agregados sobre cinco sementes. Pela sobreposição das áreas do intervalo de confiança dos algoritmos, observa-se que a variação no valor de α_{per} não causou impactos significativos. No entanto, é notável que quando utilizado $\alpha_{per} = 0{,}5$ o FQFP apresentou grande variação tanto na média quanto na mediana das iterações intermediárias, o que é um ponto negativo. Em relação a utilização dos valores $0,55$ e $0,6$, o comportamento dos algoritmos foi muito similar, podendo ser utilizado ambos valores. No entanto, como o algoritmo que utilizou 0,55 foi levemente superior tanto na média quanto na mediana da última iteração, esse foi o valor

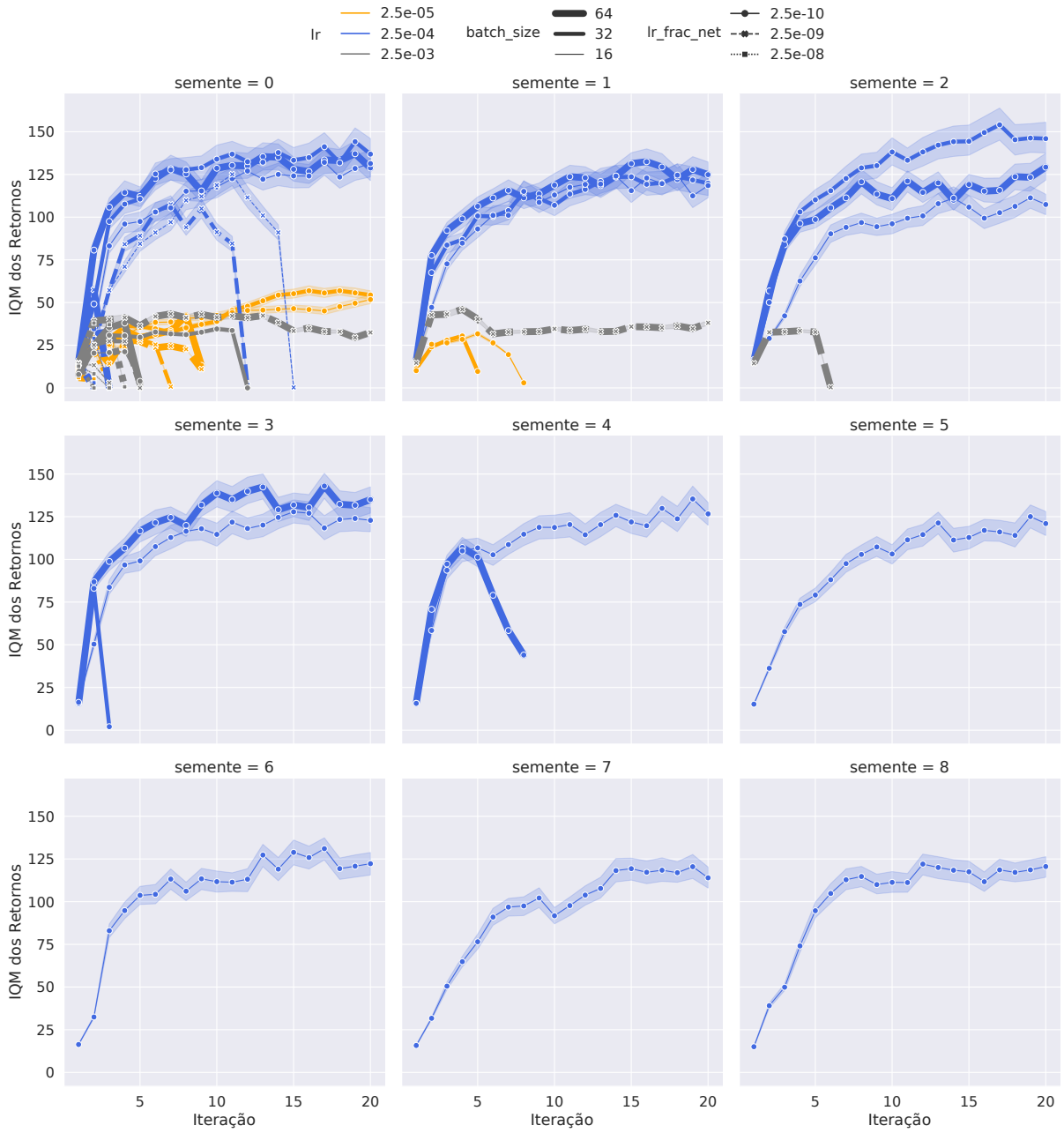


Figura 22 – Análise sequencial de hiperparâmetros do algoritmo FQF aplicado no jogo Space Invaders com 9 sementes. Cada vez que um conjunto de hiperparâmetros ocasionou na queda brusca do desempenho de um algoritmo, ele foi eliminado das próximas execuções.

escolhido para utilizar neste trabalho em todos os algoritmos que utilizam a experiência por repetição priorizada.

Para a aplicação do MRL, são introduzidos três novos parâmetros, em que houve consenso dos valores em (VIEILLARD et al., 2020), (CERON; CASTRO, 2021) e até mesmo em (SIU et al., 2021), que realizou algumas modificações na equação. Por esse motivo os parâmetros de MRL não foram ajustados. A Tabela 2 apresenta uma listagem

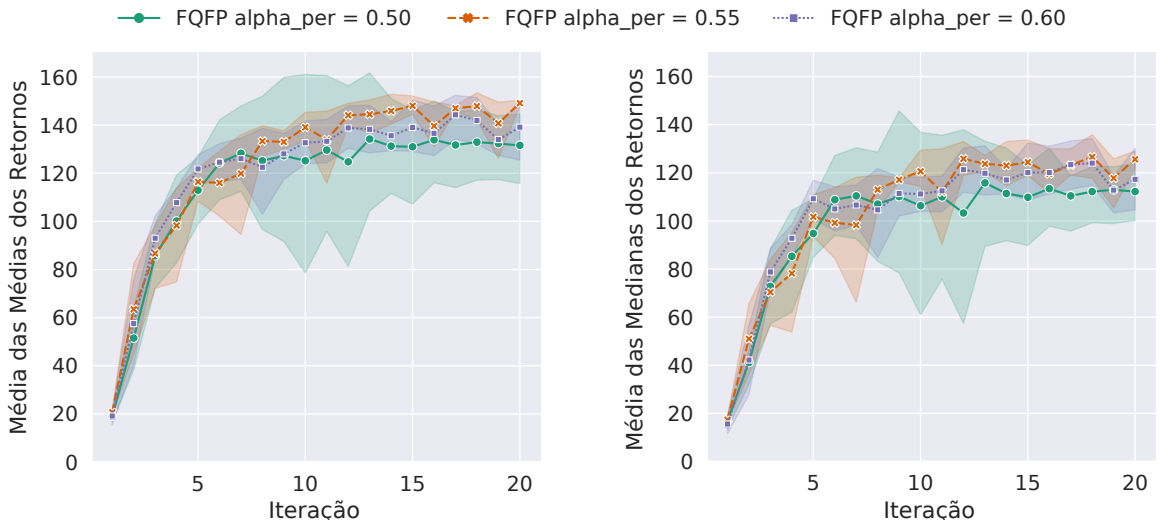


Figura 23 – Agregação sobre 5 sementes da média e mediana dos retornos obtidos pelo algoritmo FQFP no jogo Space Invaders. Foram utilizadas 3 opções de valores para o hiperparâmetro alpha_per.

dos valores dos principais hiperparâmetros utilizados para os algoritmos avaliados neste trabalho.

Hiperparâmetro	Valor	Descrição
output_channels	16	Número de canais de saída da convolução.
kernel_shape	(3,3)	Dimensão da máscara de convolução utilizada pela rede.
stride	1	Tamanho do pulo na aplicação da convolução.
units	128	Número de neurônios da camada densa oculta da rede.
batch_size	16	Tamanho do mini batch.
lr	2,5e-05	Taxa de aprendizagem utilizada no fluxo principal do FQF.
adam_eps	0,01/batch_size	Termo adicionado no denominador do otimizador adam.
lr_frac_net	2,5e-10	Taxa de aprendizagem utilizada na rede responsável por gerar as frações de quantis.
buffer_size	100000	Número de transições comportadas pelo buffer.
start_steps	1000	Número de passos iniciais necessários para a rede começar a ser treinada.
update_interval	4	Intervalo de passos entre dois ajustes da rede.
interval_target	1000	Intervalo de passos utilizados para a atualização da rede target.
gamma	0,99	Fator de desconto γ.
eps	0,01	Probabilidade de o agente escolher uma ação aleatoriamente.
initial_eps	1	Valor inicial para eps.
eps_decay_steps	250000	Número de passos em que o valor de initial_eps decai linearmente até atingir eps.
num_quantiles	32	Número de quantis utilizados na regressão quantílica.
num_cosines	64	Número de cossenos utilizados no fluxo proposto pelo IQN.
n_step	1 ou 3	Número de passos da diferença temporal.
tau_mrl	0,03	Fator de temperatura utilizado na política do munchausen.
alpha_mrl	0,9	Peso do fator de munchausen.
l0_mrl	-1	Limitante inferior utilizado no fator de munchausen.
alpha_per	0,55	Expoente do cálculo do indicador de probabilidade da PER.
beta_per	0,4→1	Expoente do cálculo da IS da PER.

Tabela 2 – Hiperparâmetros utilizados no FQF e algoritmos derivados.

6.1.1 Comparação do FQF com Demais Algoritmos

Os resultados do FQF após a avaliação de hiperparâmetros no jogo Space Invaders mostram que o conjunto de hiperparâmetros produziu um modelo estável do ponto

de vista do esquecimento catastrófico. No entanto, considerando que melhorias vão ser avaliadas juntamente com FQF, é importante que o algoritmo não seja apenas estável, mas que ele produza também agentes que apresentem bons resultados no conjunto dos ambientes avaliados. Para certificar da eficiência do algoritmo, o FQF com o conjunto de hiperparâmetros ajustado foi aplicado a todos os cinco jogos do Mini Atari e os resultados comparados com os resultados de diversos algoritmos avaliados em (CERON; CASTRO, 2021).

Os experimentos realizados em (CERON; CASTRO, 2021) utilizaram 10 sementes, utilizando 10 iterações com 1 milhão de passos cada. Nesses experimentos os algoritmos foram avaliados pela média das médias das recompensas de cada execução e os resultados disponibilizados em gráficos. Como não foram disponibilizadas as fontes de dados referentes aos retornos de cada episódio, foi necessário utilizar a mesma metodologia de avaliação adotada pelos autores, além de ter sido necessário também extrair de forma aproximada os dados a serem comparados, traçando linhas nos gráficos. A comparação entre os algoritmos pode ser vista na Tabela 3.

	Asterix	Breakout	Freeway	Seaquest	Invaders
DQN	28	7,5	60	5	34
QR-DQN	44	8,2	61	5	50
IQN	33	8	61	4,5	44
Rainbow	42	11,3	57	10	91
QRainbow	**55**	13,5	50	17	85
IRainbow	50	11,6	49,5	6,5	77
M-Rainbow	50	11,3	56	17,5	80,5
M-IRainbow	53	14,2	57,6	5	90
FQF	25	**26,4**	**62,1**	**18,6**	**192,2**

Tabela 3 – Comparação da média das médias das recompensas da décima iteração de 10 execuções. Com exceção do FQF, todos os outros resultados foram obtidos de forma aproximada a partir dos gráficos de (CERON; CASTRO, 2021).

Na Tabela 3 são comparados os resultados dos algoritmos DQN, QR-DQN, IQN, Rainbow, QRainbow (QR-DQN + melhorias do Rainbow), IRainbow (IQN + melhorias do Rainbow), M-Rainbow (Rainbow + MRL), M-IRainbow (IRainbow + MRL) e FQF, cujos os resultados são provenientes deste trabalho. Os dados das tabelas mostram que o FQF produziu os melhores resultados em quatro dos cinco jogos avaliados, superando inclusive algoritmos que já possuem várias melhorias incorporadas, como o M-IRainbow. O FQF apenas produziu resultados inferiores no Asterix. Mas na mesma tabela é possível observar que essa variação dos algoritmos entre os jogos é normal. O QRainbow, por exemplo, obteve o melhor desempenho no Asterix, mas ficou entre os dois piores no Breakout.

Os dados da Tabela 3 mostram que o FQF, utilizando os parâmetros da Tabela 2

obteve, no geral dos cinco jogos, resultado superior do que os modelos gerados por outros algoritmos em outro trabalho, o que significa que serão aplicadas melhorias em cima de um algoritmo que já produz resultados bastantes satisfatórios nos ambientes avaliados, afinal, a metodologia de avaliação foi a mesma. No entanto, os dados da Tabela 3 não podem ser utilizados para serem realizadas comparações entre as abordagens dos algoritmos, pois os experimentos não utilizaram os mesmos valores para os hiperparâmetros compartilhados entre os algoritmos. Além disso, em (CERON; CASTRO, 2021) foi utilizado o *framework* Dopamine (CASTRO et al., 2018), enquanto aqui foi utilizado o Rljax[1].

6.2 Comparação das Variações do FQF por Ambiente

Na Figura 24 é possível ver a evolução das oito variações do FQF avaliadas neste trabalho, de acordo o IQM dos retornos por iteração, cujos valores foram agregados por média simples das cinco execuções de cada algoritmo em cada ambiente[2]. A área sombreada indica o intervalo de confiança para o nível de 95%.

Inicialmente analisando os algoritmos no Asterix, observa-se que todos eles convergiram muito rápido. O FQF é o algoritmo que obteve as menores pontuações. Um pouco melhor colocado aparece o FQFP, que apresentou valores mais altos inicialmente, mas apresentou uma leve tendência de queda de desempenho ao longo do treinamento. No topo, com os melhores desempenhos, aparecem o FQFn3PM e o FQFn3M.

No Breakout, os algoritmos FQF, FQFP, FQFn3 e FQFn3P aparecem bem próximos com as pontuações mais baixas, sendo notável uma alta sobreposição do intervalo de confiança. No topo aparecem os algoritmos FQFn3PM, FQFPM e FQFM, no qual o FQFPM aparece com um largo intervalo de confiança, o que indica uma grande variância das pontuações do algoritmo no jogo.

No jogo Freeway, ao analisar os valores do eixo Y, observa-se que a diferença entre os algoritmos é bem pequena, sendo o melhor algoritmo final o FQFn3M, com um IQM médio de aproximadamente 66, enquanto o FQF, que apresentou o menor desempenho, tem a pontuação de aproximadamente 62. No entanto, é perceptível também que todos algoritmos no Freeway geraram um intervalo de confiança muito estreito, o que é um ponto positivo por indicar uma pequena variância dos algoritmos. Esse comportamento pode ser explicado pelo fato de o Freeway ter sido modelado com episódios de duração fixa. Logo, uma ação errada do agente, que pode ser gerada por uma escolha aleatória, por exemplo, não prejudica o desempenho de todo o episódio, o que deixa os retornos de diferentes episódios mais próximos. Da mesma forma com que essa modelagem do ambiente agrega com intervalos de confiança mais precisos, ela também pode impedir que

[1] Mesmo que os algoritmos de ambos frameworks sejam implementados fielmente como proposto nos artigos, o uso de bibliotecas distintas pode gerar divergência nos resultados de um mesmo algoritmo.

[2] Resultados individuais de cada execução podem ser encontrados na Figura 29 dos Apêndices.

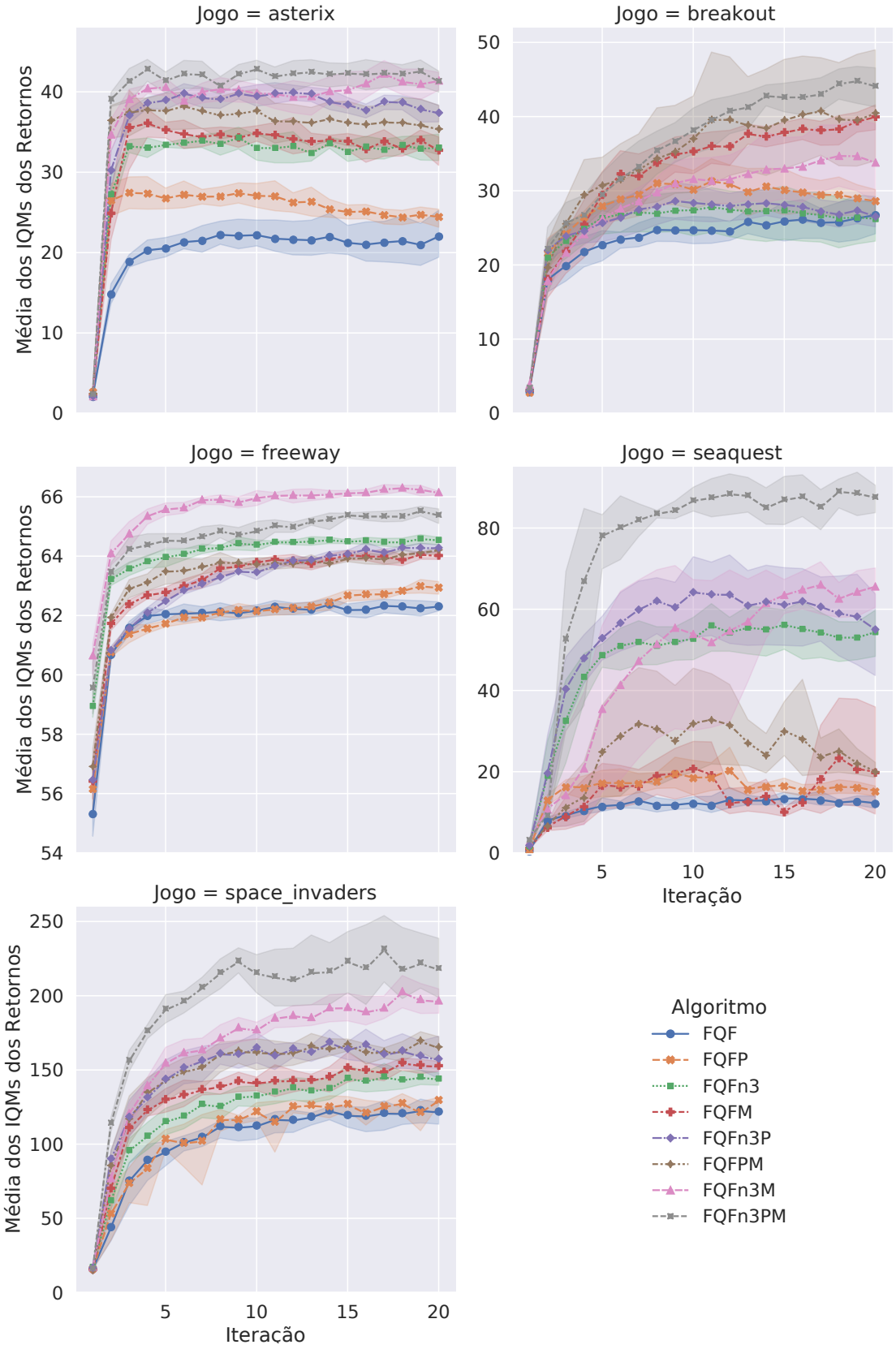


Figura 24 – Evolução da média sobre 5 execuções dos IQMs dos retornos. A área sombreada indica o intervalo de confiança para 95%.

algoritmos melhores se destaquem, justamente pelo ambiente ser mais tolerável a erros que os demais.

No Seaquest repete-se o padrão do FQFn3PM no topo e o FQF e FQFP na base, com pontuações mais baixas. No entanto, dessa vez nota-se que a pontuação do FQFn3PM foi aproximadamente sete vezes maior do que a do FQF na última iteração. Como já discutido, o Seaquest é o jogo mais complexo do Mini Atari, pois possui um número maior de ações disponíveis, além de regras difíceis de se aprender, como por exemplo a necessidade de o agente (submarino) subir à superfície do mar para pegar oxigênio. Aprender tais comportamentos demanda uma alta capacidade de exploração do agente e a discrepância entre os resultados do FQF e FQFn3PM é um forte indício de que o FQFn3PM foi capaz de aprender padrões de jogo distintos do FQF.

Por fim, no jogo Space Invaders, nota-se novamente o FQF e FQFP na base e o FQFn3PM no topo, com uma divergência um pouco maior nas últimas iterações, o que gera uma sobreposição com o intervalo de confiança do FQFn3M. Assim como ocorreu em outros jogos, no Space Invaders os algoritmos FQFPM, FQFn3P, FQFM e FQFn3 aparecem com pontuações intermediárias, bem próximas uma das outras.

Os dados utilizados para gerar os gráficos da Figura 24 foram utilizados também para calcular a porcentagem de ganho dos algoritmos em relação ao FQF, considerando a última iteração (LI) e a área sobre a curva (AUC). O resultado pode ser visualizado na Tabela 4, em que o melhor algoritmo de acordo com cada ambiente e métrica é destacado em negrito.

	Asterix		Breakout		Freeway		Seaquest		Space Invaders	
	AUC	LI	AUC	LI	AUC	LI	AUC	LI	AUC	LI
FQFP	24.68%	10.91%	19.16%	7.12%	0.26%	0.96%	40.26%	24.39%	3.53%	6.34%
FQFn3	56.54%	50.00%	8.83%	-1.87%	3.66%	3.53%	316.19%	342.28%	18.68%	18.20%
FQFM	60.60%	48.64%	39.21%	49.44%	2.25%	2.73%	31.77%	61.79%	27.59%	25.37%
FQFn3P	82.39%	69.55%	11.03%	-0.75%	2.07%	3.21%	369.06%	347.15%	41.82%	29.16%
FQFPM	74.75%	60.45%	45.88%	51.31%	2.53%	3.05%	107.53%	63.41%	41.95%	35.67%
FQFn3M	90.04%	**87.73%**	23.91%	26.59%	**6.17%**	**6.10%**	314.14%	433.33%	59.34%	61.37%
FQFn3PM	**99.23%**	**87.73%**	**52.85%**	**65.17%**	4.67%	4.98%	**556.50%**	**611.38%**	**89.34%**	**79.24%**

Tabela 4 – Porcentagem de ganho em relação ao FQF dos algoritmos com melhorias avaliados. Os valores utilizados para a normalização correspondem à média sobre 5 execuções por jogo do IQM dos Retornos para a última iteração (LI) e área sobre a curva (AUC).

6.3 Comparação Geral das Variações do FQF

Realizadas as análises individuais por jogos, agora é analisado o desempenho geral dos algoritmos, em que foi aplicada a padronização do IQM dos retornos gerados pelos algoritmos em cada jogo avaliado, de forma a avaliar todas as amostras em conjunto. O gráfico da Figura 25 mostra o resultado da padronização por cada iteração. Note que, como a padronização ocorreu por iteração, as curvas perdem a tendência de crescimento

gerada pela evolução do algoritmo, pois todos algoritmos tendem a evoluir em conjunto. Nesse caso, qualquer tendência de queda ou crescimento representa um distanciamento do algoritmo em relação à média dos demais.

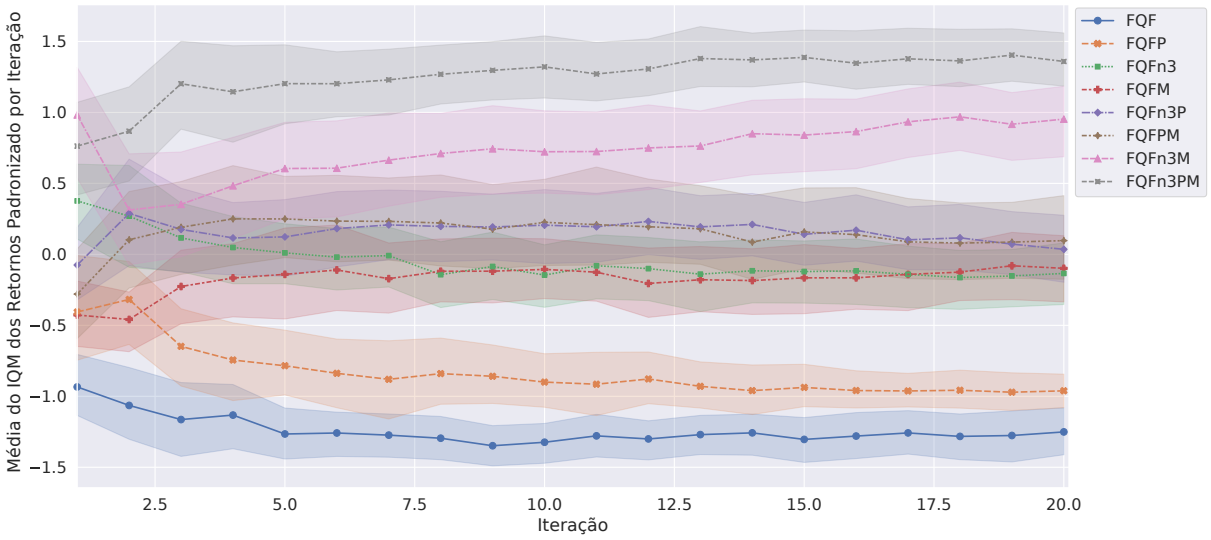


Figura 25 – Representação da média dos IQMs padronizados dos retornos por iteração. A área sombreada representa o intervalo de confiança para um nível de 95%.

Pela Figura 25 observa-se que no início todos algoritmos avaliados apresentam desempenhos bem próximos, o que vai mudando em iterações mais à direita, com exceção dos algoritmos FQFn3, FQFM, FQFn3P e FQFPM, que aparecem bem próximos uns dos outros. Na Figura 26 as pontuações da última iteração são apresentadas em formato de um box plot, possibilitando observar com mais detalhes as distribuições de cada algoritmo. Pelo box plot mais uma vez percebe-se que os quatro algoritmos citados apresentam desempenhos bem próximos. Porém, chama atenção o fato de nenhuma amostra pertencente ao FQFn3PM ser inferior ao FQF e também ao FQFP, o que representa um ótimo ganho.

Na Figura 27 são exibidos os perfis de performance dos algoritmos. Esse gráfico tem no eixo x o IQM dos Retornos Padronizado (no gráfico abreviado como z) e no eixo y a porcentagem do número de execuções do algoritmo que apresentaram uma pontuação maior do que z. Logo, curvas mais altas são preferíveis. A vantagem dessa visualização é que ela permite detectar com facilidade se um algoritmo domina o outro estocasticamente ou não. A definição matemática de dominância estocástica de primeira ordem consta na Definição 2. Através dos perfis de performance a dominância estocástica pode ser detectada em curvas que não se cruzam.

Definição 2 *Uma variável aleatória X tem dominância estocástica de primeira ordem sobre outra variável aleatória Y se $P(X > z) \geq P(Y > z)$ para todo z e se, para algum z, $P(X > z) > P(Y > z)$.*

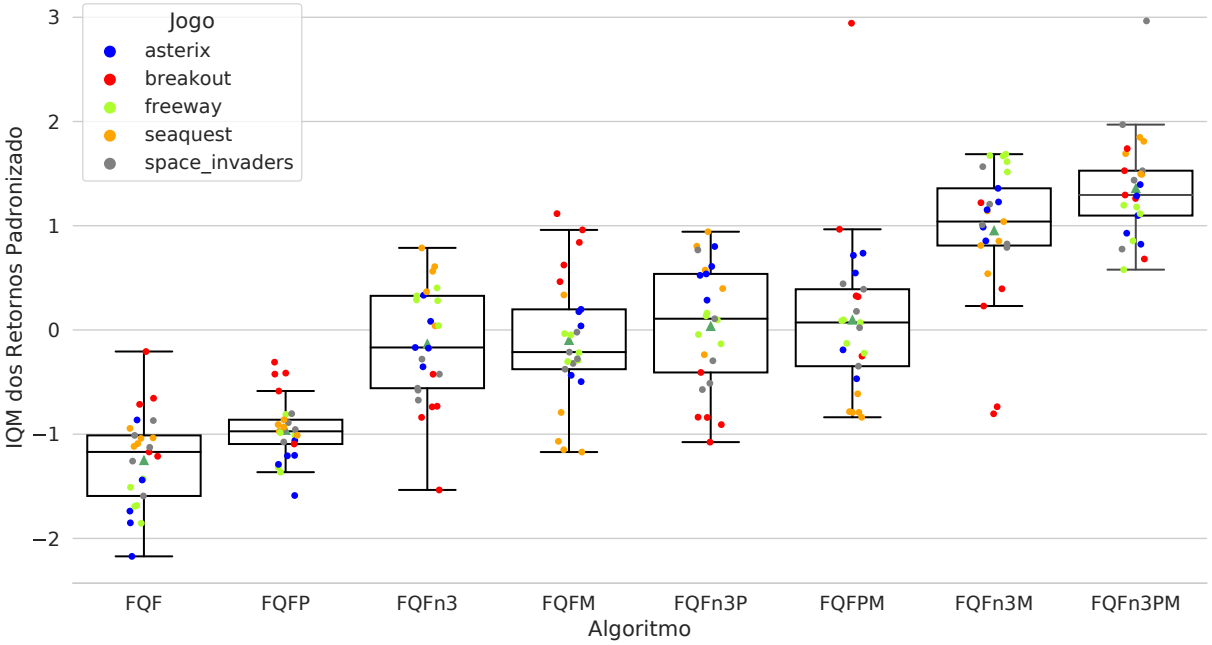


Figura 26 – Box plot do IQM dos retornos da última iteração padronizado.

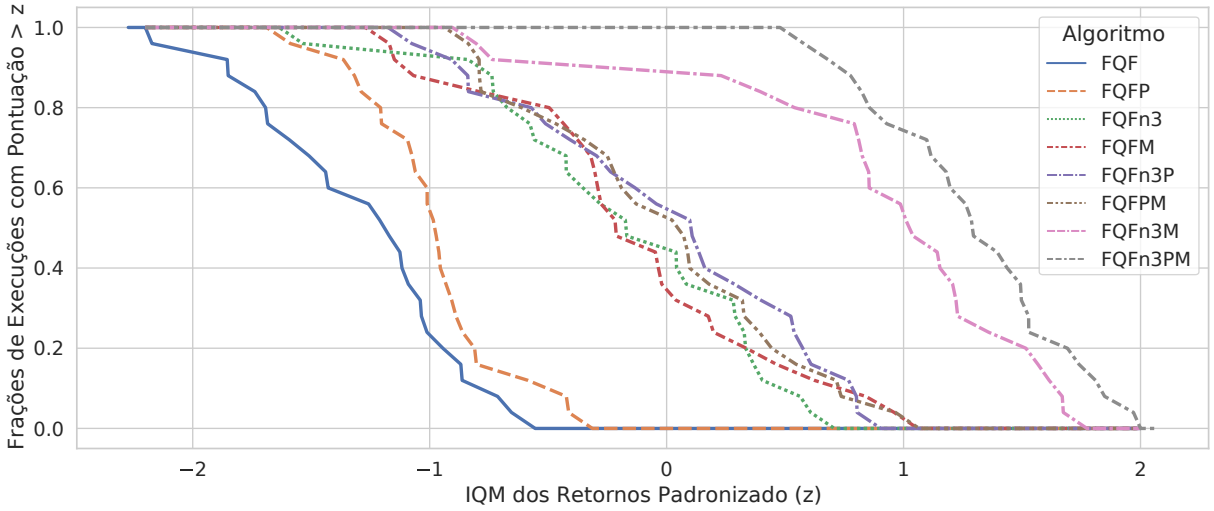


Figura 27 – Perfis de performance da última iteração dos algoritmos avaliados.

O fato de uma variável aleatória X (algoritmo) dominar outra variável aleatória Y, significa que X é preferível a Y, pois oferece retornos melhores (LEVY, 1992). Logo, pelo gráfico da Figura 27 conclui-se que o FQFn3PM é preferível a todos os outros algoritmos avaliados. O FQFn3M é preferível a todos os outros, com exceção do FQFn3PM. Nada se conclui da relação entre FQFn3, FQFM, FQFn3P e FQFPM, mas eles são preferíveis ao FQFP e ao FQF. O algoritmo FQFP é preferível ao FQF.

Por fim na Figura 28 é apresentada uma última análise, que é baseada na probabilidade de um algoritmo ser melhor do que o outro, utilizando para tal a Equação 5.8. Note que a equação utiliza métricas baseadas em rank. Logo, os gráficos de dispersão exibidos anteriormente atuam como complemento para a análise das probabilidades encontradas.

Além disso, diferentemente das análises anteriores, a Equação 5.8 subdivide as amostras em grupos de jogos e depois calcula a média dos resultados de cada jogo.

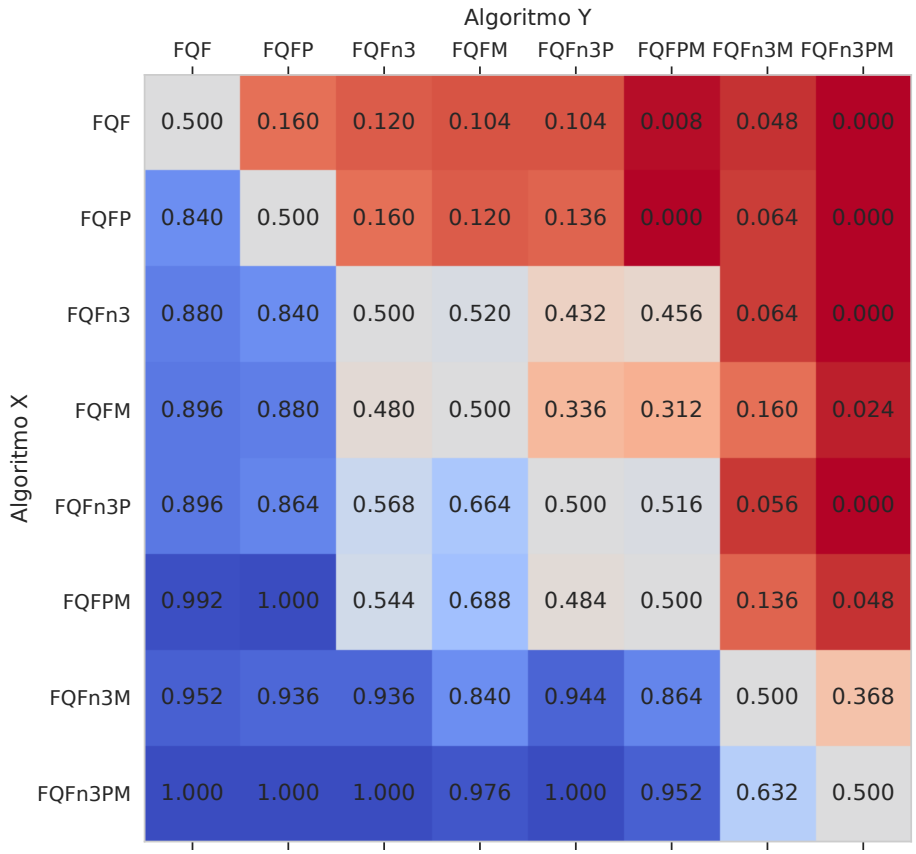


Figura 28 – Probabilidade de X ser melhor do que Y, baseado no IQM dos Retornos da última iteração.

As probabilidades de superioridade da Figura 28 confirmam várias das conclusões já relatadas anteriormente, principalmente em relação à superioridade do algoritmo FQFn3PM aos demais. No entanto, por apresentar uma comparação que é realizada inicialmente nos resultados de cada jogo, a Figura 28 apresenta uma nova informação que é um ganho dos algoritmos FQFPM e FQFn3P em relação ao FQFM, sendo que nas análises anteriores em que não houve a agregação por jogos, não foi possível notar uma distinção entre os resultados dos três algoritmos.

Interpretando cada um dos algoritmos como uma combinação de melhorias adicionadas ao FQF (sendo essas MRL, PER e três passos), é possível chegar a conclusões sobre o impacto de cada melhoria no FQF. É notório que todas elas foram capazes de gerar impactos positivos nos resultados. O uso do MRL e de três passos na diferença temporal trouxeram ganhos consideráveis quando aplicadas isoladamente. Além disso, quando somadas, o ganho foi ainda mais relevante, formando o algoritmo aqui chamado de FQFn3M. O uso da PER também gerou impactos positivos, mas em menor escala que as outras duas melhorias. Essa constatação fica clara quando comparado nas Figuras 26 e 27 a proximidade dos resultados gerados por um algoritmo com PER e o algoritmo correspondente

sem a PER. Porém, vale ressaltar que a adição da PER no algoritmo FQFn3M resultou em um algoritmo (FQFn3PM) que no geral de todos os jogos se demonstrou superior, com o destaque para os resultados obtidos no Seaquest, um jogo de dificuldade elevada em que o FQFn3PM foi aproximadamente sete vezes melhor do que o FQF.

Os resultados obtidos confirmam a hipótese inicial de que a combinação de melhorias utilizadas em algoritmos anteriores ao FQF poderia gerar um algoritmo ainda melhor. Contudo, vale ressaltar que as constatações sobre a superioridade de uma melhoria em relação a outra levam como base o uso dos mesmos valores dos parâmetros compartilhados, e ignora a relação da melhoria aplicada com tais parâmetros. O uso da PER, por exemplo, pode demandar ajustes na taxa de aprendizado da rede para produzir resultados mais relevantes, assim como foi realizado inicialmente em (SCHAUL et al., 2016).

7 Conclusão e Trabalhos Futuros

O FQF é um algoritmo distributivo publicado no ano de 2019, tendo superado vários algoritmos anteriores no domínio dos jogos do Atari 2600. Apesar dos bons resultados, esse algoritmo ainda não foi capaz de superar um especialista humano em todos os jogos do Atari 2600. Isso nos levou a acreditar que uma versão melhor desse algoritmo ainda poderia ser construída, tendo em vista a habilidade da inteligência artificial em detectar padrões que muitas vezes não são perceptíveis por humanos. Sendo assim, neste trabalho realizamos uma pesquisa por trabalhos relacionados e escolhemos três melhorias que se mostraram promissoras quando aplicadas em algoritmos propostos anteriormente ao FQF, sendo essas: o uso de três passos na diferença temporal; a aplicação da abordagem de Munchausen e o uso da PER. Essas melhorias foram combinadas no FQF e avaliadas no domínio do Mini Atari, que é um ambiente de menor escala gráfica ideal para trabalhos como este em que se deseja focar no aprendizado de comportamento, mas sem eliminar o aprendizado de *features*.

Todas as combinações das melhorias no FQF foram avaliadas, de forma que foram considerados oito algoritmos nas análises principais, sendo esses o FQF, FQFn3, FQFP, FQFM, FQFn3P, FQFn3M, FQFPM e FQFn3PM, cujos os nomes ajudam a identificar quais melhorias foram aplicadas. Os algoritmos foram avaliados tanto no desempenho individual em cada jogo, quanto no aspecto geral dos cinco jogos. A adição isolada no FQF de todas as três melhorias abordadas já foi capaz de apresentar ganhos significantes nos jogos avaliados. No entanto, a adição das três técnicas no FQF, que gerou o algoritmo aqui chamado de FQFn3PM, foi mais impactante, sendo melhor do que o FQF em 100% dos experimentos realizados.

Embora esses resultados sejam provenientes de um conjunto de jogos de menor escala que o Atari 2600, onde o FQF foi apresentado, eles são de extrema relevância, pois atuam como um direcionamento para trabalhos posteriores que desejam fazer aplicações em ambientes de maior complexidade gráfica, em que, devido a alta demanda de poder computacional, a realização de comparações como as realizadas aqui podem se tornar inviáveis para muitos pesquisadores. Além disso, destaca-se o fato de o FQFn3PM ter sido aproximadamente sete vezes melhor do que o FQF no jogo Seaquest, considerando como métrica o IQM dos retornos da última iteração. Conforme já descrito, o Seaquest é o jogo que oferece mais desafios aos agentes, pois contém muitas regras a serem aprendidas, com recompensas bem espaçadas. Os ganhos em tarefas mais desafiadoras merecem uma maior atenção, pois é nesse tipo de ambiente que existem mais possibilidades dos melhores algoritmos se sobressaírem. Por esse motivo, espera-se que em ambientes de maior complexidade, como o Atari 2600, a adição das três melhorias seja capaz de impulsionar os

resultados do FQF de forma tão significativa quanto ocorreu nos experimentos utilizando o Seaquest. Isso nos leva acreditar que tais melhorias são capazes de aumentar os ganhos do FQF no Atari 2600 em relação a um humano especialista, o que pode ser confirmado em trabalhos futuros.

Outra proposta para trabalhos futuros é uma investigação mais detalhada sobre as quedas bruscas de desempenho que o FQF estava sofrendo em testes preliminares. Esse problema foi solucionado neste trabalho com um ajuste de hiperparâmetros realizado por uma heurística de poda, que utilizou nove sementes. Embora tenha funcionado, essa estratégia não garante que o algoritmo não irá falhar em uma execução utilizando outra semente não utilizada. Além disso, foi exibido que depois que ocorre a queda de desempenho, o algoritmo não consegue mais voltar a aprender. Isso pode ter acontecido devido ao fato de o FQF, assim como a DQN, utilizar uma política de comportamento ε-*greedy* em que a taxa de aleatoriedade vai diminuindo nos passos iniciais, até se estabilizar. Nesse sentido, quando ocorre a queda de desempenho, o agente pode já estar com uma taxa de aleatoriedade baixa o suficiente ao ponto dele não conseguir mais explorar o ambiente. Essa hipótese levanta alguns questionamentos para trabalhos futuros: Como seria o desempenho do FQF em ambientes não estacionários? Apenas aumentar a taxa de aleatoriedade da política de comportamento seria suficiente, ou seria necessário adotar outros tipos de políticas? Em trabalhos futuros pode ser estudado também métodos para transformar o FQF em um algoritmo distribuído, o que pode melhorar o desempenho do algoritmo ainda mais.

Um outro ponto a ser explorado futuramente tem relação com o uso de n passos. Na Seção 2.5 foi relatado que o uso de algoritmos *off-policy* com métodos de diferença temporal de n passos demanda o uso de IS. Neste trabalho, assim como no Rainbow, a IS não foi aplicada para esse fim, por um entendimento de que um número de passos igual a três é um valor pequeno o suficiente a ser possível ignorar essa correção. No entanto, essa é uma hipótese que merece ser confirmada em um trabalho futuro, podendo ser avaliado o desempenho do algoritmo Tree Backup, que dispensa o uso da IS.

Referências

ABDELLATIF, A. A. et al. Reinforcement learning for intelligent healthcare systems: A comprehensive survey. *CoRR*, abs/2108.04087, 2021. Citado na página 19.

AGARWAL, R. et al. Deep reinforcement learning at the edge of the statistical precipice. *Advances in Neural Information Processing Systems*, v. 34, 2021. Citado 2 vezes nas páginas 75 e 76.

BADIA, A. P. et al. Agent57: Outperforming the Atari human benchmark. In: III, H. D.; SINGH, A. (Ed.). *Proceedings of the 37th International Conference on Machine Learning*. PMLR, 2020. (Proceedings of Machine Learning Research, v. 119), p. 507–517. Disponível em: <https://proceedings.mlr.press/v119/badia20a.html>. Citado na página 63.

BANACH, S. Sur les operations dans les ensembles abstraits et leur applications aux equations integrals. In: *Fundamenta Mathematicae*. [S.l.: s.n.], 1922. p. 133–181. Citado na página 51.

BELLEMARE, M. G.; DABNEY, W.; MUNOS, R. A distributional perspective on reinforcement learning. In: PRECUP, D.; TEH, Y. W. (Ed.). *Proceedings of the 34th International Conference on Machine Learning*. PMLR, 2017. (Proceedings of Machine Learning Research, v. 70), p. 449–458. Disponível em: <http://proceedings.mlr.press/v70/bellemare17a.html>. Citado 4 vezes nas páginas 8, 50, 51 e 54.

BELLEMARE, M. G. et al. The arcade learning environment: An evaluation platform for general agents. *J. Artif. Int. Res.*, AI Access Foundation, El Segundo, CA, USA, v. 47, n. 1, p. 253–279, may 2013. ISSN 1076-9757. Citado na página 45.

CASTRO, P. S. et al. Dopamine: A research framework for deep reinforcement learning. *CoRR*, abs/1812.06110, 2018. Disponível em: <http://arxiv.org/abs/1812.06110>. Citado na página 83.

CERON, J. S. O.; CASTRO, P. S. Revisiting rainbow: Promoting more insightful and inclusive deep reinforcement learning research. In: MEILA, M.; ZHANG, T. (Ed.). *Proceedings of the 38th International Conference on Machine Learning*. PMLR, 2021. (Proceedings of Machine Learning Research, v. 139), p. 1373–1383. Disponível em: <https://proceedings.mlr.press/v139/ceron21a.html>. Citado 10 vezes nas páginas 10, 61, 62, 69, 72, 77, 78, 80, 82 e 83.

DABNEY, W. et al. Implicit quantile networks for distributional reinforcement learning. In: DY, J.; KRAUSE, A. (Ed.). *Proceedings of the 35th International Conference on Machine Learning*. PMLR, 2018. (Proceedings of Machine Learning Research, v. 80), p. 1096–1105. Disponível em: <http://proceedings.mlr.press/v80/dabney18a.html>. Citado 5 vezes nas páginas 8, 55, 56, 60 e 75.

DABNEY, W. et al. Distributional reinforcement learning with quantile regression. In: . AAAI Press, 2018. (AAAI'18/IAAI'18/EAAI'18). ISBN 978-1-57735-800-8. Disponível em: <https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17184>. Citado 2 vezes nas páginas 54 e 66.

DUCHI, J.; HAZAN, E.; SINGER, Y. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, v. 12, n. 61, p. 2121–2159, 2011. Disponível em: <http://jmlr.org/papers/v12/duchi11a.html>. Citado na página 44.

DUMOULIN, V.; VISIN, F. *A guide to convolution arithmetic for deep learning*. arXiv, 2016. Disponível em: <https://arxiv.org/abs/1603.07285>. Citado 2 vezes nas páginas 8 e 40.

FORTUNATO, M. et al. Noisy networks for exploration. In: *International Conference on Learning Representations*. [s.n.], 2018. Disponível em: <https://openreview.net/forum?id=rywHCPkAW>. Citado na página 53.

HAARNOJA, T. et al. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In: DY, J.; KRAUSE, A. (Ed.). *Proceedings of the 35th International Conference on Machine Learning*. PMLR, 2018. (Proceedings of Machine Learning Research, v. 80), p. 1861–1870. Disponível em: <http://proceedings.mlr.press/v80/haarnoja18b.html>. Citado na página 57.

HASSELT, H. Double q-learning. In: LAFFERTY, J. et al. (Ed.). *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2010. v. 23. Disponível em: <https://proceedings.neurips.cc/paper/2010/file/091d584fced301b442654dd8c23b3fc9-Paper.pdf>. Citado na página 29.

HASSELT, H. v.; GUEZ, A.; SILVER, D. Deep reinforcement learning with double q-learning. In: *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*. [S.l.]: AAAI Press, 2016. (AAAI'16), p. 2094–2100. Citado na página 46.

HAUSKNECHT, M. J.; STONE, P. Deep recurrent q-learning for partially observable mdps. In: *AAAI Fall Symposium Series*. [s.n.], 2015. Disponível em: <https://www.aaai.org/ocs/index.php/FSS/FSS15/paper/view/11673>. Citado na página 47.

HAYKIN, S. S. *Neural networks and learning machines*. Third. Upper Saddle River, NJ: Pearson Education, 2009. Citado 2 vezes nas páginas 37 e 44.

HESSEL, M. et al. Rainbow: Combining improvements in deep reinforcement learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, Association for the Advancement of Artificial Intelligence (AAAI), v. 32, n. 1, abr. 2018. Disponível em: <https://doi.org/10.1609/aaai.v32i1.11796>. Citado 7 vezes nas páginas 8, 53, 54, 61, 62, 69 e 79.

HINTON, G. E. Distributed representations. In: *Technical Report CMU-CS-84-157*. [S.l.]: Department of Computer Science, Carnegie-Mellon University, Pittsburgh, PA., 1984. Citado na página 33.

HORGAN, D. et al. Distributed prioritized experience replay. In: *International Conference on Learning Representations*. [s.n.], 2018. Disponível em: <https://openreview.net/forum?id=H1Dy---0Z>. Citado na página 63.

HUBER, P. J. Robust Estimation of a Location Parameter. *The Annals of Mathematical Statistics*, Institute of Mathematical Statistics, v. 35, n. 1, p. 73 – 101, 1964. Disponível em: <https://doi.org/10.1214/aoms/1177703732>. Citado 2 vezes nas páginas 55 e 66.

JOHN, G. H. When the best move isn't optimal: Q-learning with exploration. In: *Proceedings of the Twelfth National Conference on Artificial Intelligence*. [S.l.]: AAAI Press, 1994. p. 1464–1464. Citado na página 29.

KAPTUROWSKI, S. et al. Recurrent experience replay in distributed reinforcement learning. In: *International Conference on Learning Representations*. [s.n.], 2019. Disponível em: <https://openreview.net/forum?id=r1lyTjAqYX>. Citado na página 63.

KENDALL, A. et al. Learning to drive in a day. In: *2019 International Conference on Robotics and Automation (ICRA)*. [S.l.: s.n.], 2019. p. 8248–8254. Citado na página 19.

KINGMA, D. P.; BA, J. Adam: A method for stochastic optimization. In: BENGIO, Y.; LECUN, Y. (Ed.). *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. [s.n.], 2015. Disponível em: <http://arxiv.org/abs/1412.6980>. Citado na página 44.

KONIDARIS, G.; OSENTOSKI, S.; THOMAS, P. Value function approximation in reinforcement learning using the fourier basis. In: *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*. [S.l.]: AAAI Press, 2011. (AAAI'11), p. 380–385. Citado na página 33.

LECUN, Y. et al. Comparison of learning algorithms for handwritten digit recognition. In: *INTERNATIONAL CONFERENCE ON ARTIFICIAL NEURAL NETWORKS*. [S.l.: s.n.], 1995. p. 53–60. Citado na página 39.

LEVY, H. Stochastic dominance and expected utility: Survey and analysis. *Management Science*, INFORMS, v. 38, n. 4, p. 555–593, 1992. ISSN 00251909, 15265501. Disponível em: <http://www.jstor.org/stable/2632436>. Citado na página 87.

LILLICRAP, T. P. et al. Continuous control with deep reinforcement learning. In: BENGIO, Y.; LECUN, Y. (Ed.). *ICLR*. [s.n.], 2016. Disponível em: <http://dblp.uni-trier.de/db/conf/iclr/iclr2016.html#LillicrapHPHETS15>. Citado na página 63.

MANN, H. B.; WHITNEY, D. R. On a test of whether one of two random variables is stochastically larger than the other. *The Annals of Mathematical Statistics*, Institute of Mathematical Statistics, v. 18, n. 1, p. 50–60, mar 1947. Disponível em: <https://doi.org/10.1214%2Faoms%2F1177730491>. Citado na página 76.

MCCLOSKEY, M.; COHEN, N. J. Catastrophic interference in connectionist networks: The sequential learning problem. In: BOWER, G. H. (Ed.). Academic Press, 1989, (Psychology of Learning and Motivation, v. 24). p. 109–165. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0079742108605368>. Citado na página 78.

MITCHELL, T. M. *Machine Learning*. New York: McGraw-Hill, 1997. ISBN 978-0-07-042807-2. Citado 4 vezes nas páginas 18, 37, 41 e 42.

MNIH, V. et al. Asynchronous methods for deep reinforcement learning. In: BALCAN, M. F.; WEINBERGER, K. Q. (Ed.). *Proceedings of The 33rd International Conference on Machine Learning*. New York, New York, USA: PMLR, 2016. (Proceedings of Machine Learning Research, v. 48), p. 1928–1937. Disponível em:

<http://proceedings.mlr.press/v48/mniha16.html>. Citado 3 vezes nas páginas 49, 50 e 53.

MNIH, V. et al. *Human-level control through deep reinforcement learning*. Springer Science and Business Media LLC, 2015. 529–533 p. Disponível em: <https://doi.org/10.1038/nature14236>. Citado 3 vezes nas páginas 18, 45 e 46.

MUNEMASA, I. et al. Deep reinforcement learning for recommender systems. In: *2018 International Conference on Information and Communications Technology (ICOIACT)*. [S.l.: s.n.], 2018. p. 226–233. Citado na página 19.

POLLACK, J. B.; BLAIR, A. D. Why did td-gammon work? In: *NIPS*. [s.n.], 1996. p. 10–16. Disponível em: <http://papers.nips.cc/paper/1292-why-did-td-gammon-work>. Citado na página 45.

PRECUP, D.; SUTTON, R. S.; SINGH, S. P. Eligibility traces for off-policy policy evaluation. In: *Proceedings of the Seventeenth International Conference on Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2000. (ICML '00), p. 759–766. ISBN 1558607072. Citado na página 31.

RUMMERY, G.; NIRANJAN, M. *On-Line Q-Learning using connectionist systems*. [S.l.], 1994. Citado na página 28.

RUSSELL, S.; NORVIG, P. *Artificial intelligence: A modern approach, global edition*. 4. ed. London, England: Pearson Education, 2021. Citado 4 vezes nas páginas 37, 38, 39 e 40.

SAHA, S. *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way*. 2018. Disponível em: <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>. Citado 2 vezes nas páginas 8 e 40.

SCHAUL, T. et al. Prioritized experience replay. In: BENGIO, Y.; LECUN, Y. (Ed.). *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*. [s.n.], 2016. Disponível em: <http://arxiv.org/abs/1511.05952>. Citado 5 vezes nas páginas 47, 67, 69, 79 e 89.

SCHOETTLER, G. et al. Deep reinforcement learning for industrial insertion tasks with visual inputs and natural rewards. In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. [S.l.: s.n.], 2020. p. 5548–5555. Citado na página 19.

SCHULMAN, J. et al. Trust region policy optimization. In: BACH, F.; BLEI, D. (Ed.). *Proceedings of the 32nd International Conference on Machine Learning*. Lille, France: PMLR, 2015. (Proceedings of Machine Learning Research, v. 37), p. 1889–1897. Disponível em: <https://proceedings.mlr.press/v37/schulman15.html>. Citado na página 63.

SILVER, D. *Lectures on Reinforcement Learning*. 2015. URL: <https://www.davidsilver.uk/teaching/>. Citado 2 vezes nas páginas 8 e 26.

SIU, C.; TRAISH, J. M.; XU, R. Y. D. Dual behavior regularized reinforcement learning. *CoRR*, abs/2109.09037, 2021. Disponível em: <https://arxiv.org/abs/2109.09037>. Citado na página 80.

SLATER, N. *Off-policy n-step learning with DQN*. 2019. <https://datascience.stackexchange.com/questions/46245/off-policy-n-step-learning-with-dqn>. Data Science Stack Exchange. Citado na página 54.

SUBRAMANIAN, A.; CHITLANGIA, S.; BATHS, V. Reinforcement learning and its connections with neuroscience and psychology. *Neural Netw.*, Elsevier Science Ltd., GBR, v. 145, n. C, p. 271–287, jan 2022. ISSN 0893-6080. Disponível em: <https://doi.org/10.1016/j.neunet.2021.10.003>. Citado na página 18.

SUTTON, R. S.; BARTO, A. G. *Reinforcement Learning: An Introduction*. Second. The MIT Press, 2018. Disponível em: <http://incompleteideas.net/book/the-book-2nd.html>. Citado 10 vezes nas páginas 8, 18, 20, 23, 31, 33, 34, 35, 52 e 57.

TESAURO, G. Temporal difference learning and TD-gammon. *Communications of the ACM*, Association for Computing Machinery (ACM), v. 38, n. 3, p. 58–68, mar. 1995. Disponível em: <https://doi.org/10.1145/203330.203343>. Citado 2 vezes nas páginas 39 e 45.

TIELEMAN, HINTON, G. *Leitura 6.5 - RMSProp: Divide the Gradient by a Running Average of Its Recent Magnitude*. [S.l.]: COURSERA: Neural Networks for Machine Learning, 2012. Citado na página 44.

VIEILLARD, N.; PIETQUIN, O.; GEIST, M. Munchausen reinforcement learning. In: LAROCHELLE, H. et al. (Ed.). *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2020. v. 33, p. 4235–4246. Disponível em: <https://proceedings.neurips.cc/paper/2020/file/2c6a0bae0f071cbbf0bb3d5b11d90a82-Paper.pdf>. Citado 2 vezes nas páginas 56 e 80.

WANG, Z. et al. Dueling network architectures for deep reinforcement learning. In: BALCAN, M. F.; WEINBERGER, K. Q. (Ed.). *Proceedings of The 33rd International Conference on Machine Learning*. New York, New York, USA: PMLR, 2016. (Proceedings of Machine Learning Research, v. 48), p. 1995–2003. Disponível em: <http://proceedings.mlr.press/v48/wangf16.html>. Citado 3 vezes nas páginas 8, 48 e 49.

WATKINS, C. J. C. H. *Learning from Delayed Rewards*. Tese (Doutorado) — King's College, Cambridge, UK, May 1989. Disponível em: <http://www.cs.rhul.ac.uk/~chrisw/new_thesis.pdf>. Citado 3 vezes nas páginas 28, 33 e 69.

YANG, D. et al. Fully parameterized quantile function for distributional reinforcement learning. In: WALLACH, H. et al. (Ed.). *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2019. v. 32. Disponível em: <https://proceedings.neurips.cc/paper/2019/file/f471223d1a1614b58a7dc45c9d01df19-Paper.pdf>. Citado 11 vezes nas páginas 8, 9, 10, 19, 58, 59, 60, 66, 75, 77 e 78.

Young, K.; Tian, T. Minatar: An atari-inspired testbed for thorough and reproducible reinforcement learning experiments. *arXiv preprint arXiv:1903.03176*, 2019. Disponível em: <https://arxiv.org/abs/1903.03176>. Citado 6 vezes nas páginas 9, 61, 62, 72, 73 e 74.

ZEILER, M. D. ADADELTA: an adaptive learning rate method. *CoRR*, abs/1212.5701, 2012. Disponível em: <http://arxiv.org/abs/1212.5701>. Citado na página 44.

Apêndices

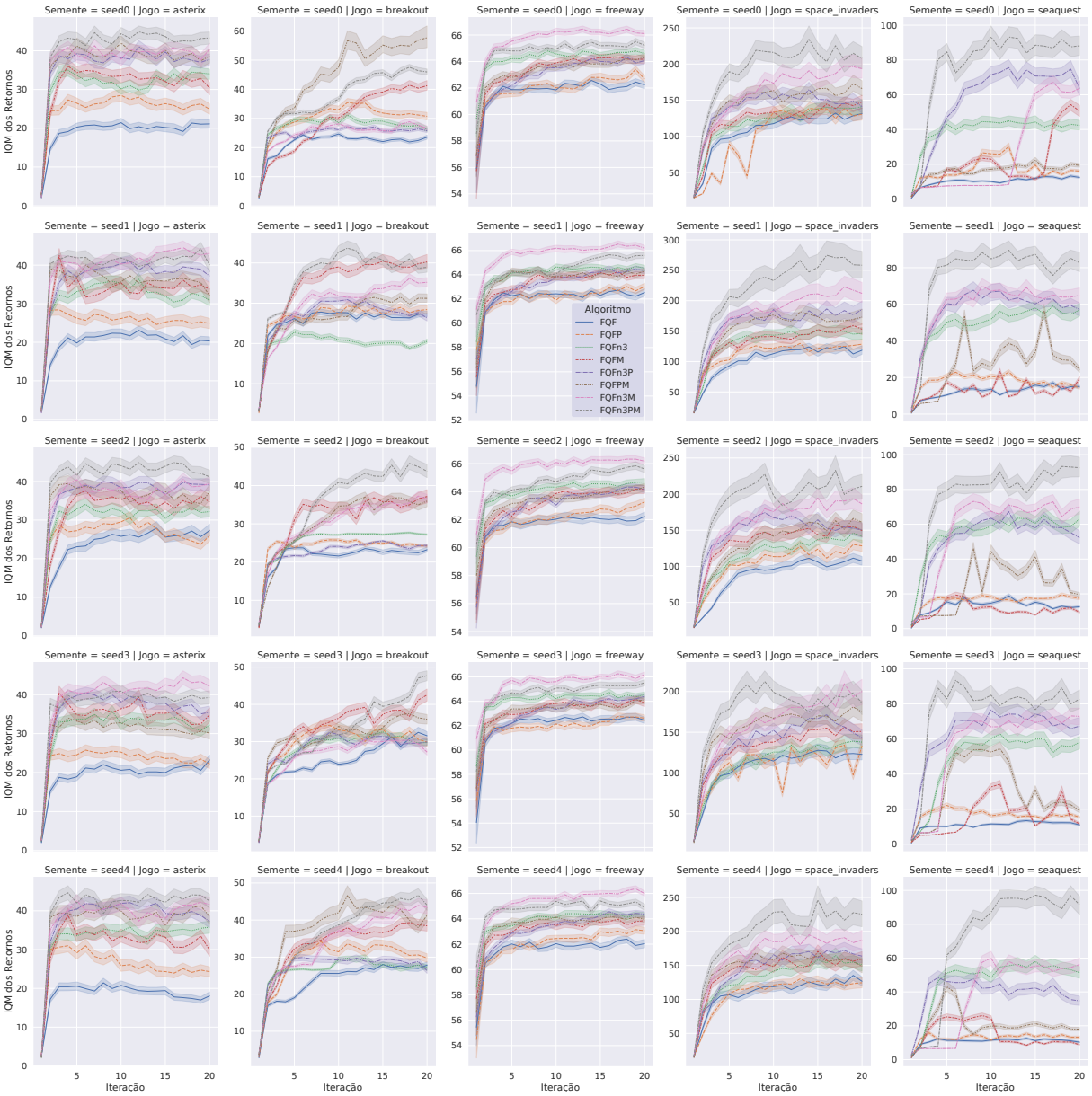


Figura 29 – IQM dos Retornos exibido para cada semente.