UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

GABRIEL COLOMBO

COMPARAÇÃO DE DESEMPENHO DO ALGORITMO *DEEP Q-LEARNING* EM AMBIENTES SIMULADOS COM ESTADOS CONTÍNUOS

PATO BRANCO

2022

GABRIEL COLOMBO

COMPARAÇÃO DE DESEMPENHO DO ALGORITMO *DEEP Q-LEARNING* EM AMBIENTES SIMULADOS COM ESTADOS CONTÍNUOS

COMPARISON OF PERFORMANCE OF THE DEEP Q-LEARNING ALGORITHM IN SIMULATED ENVIRONMENTS WITH CONTINUOUS STATES

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Engenharia de Computação da Universidade Tecnológica Federal do Paraná.

Orientador: Prof. Dr. Dalcimar Casanova

PATO BRANCO

2022

GABRIEL COLOMBO

COMPARAÇÃO DE DESEMPENHO DO ALGORITMO *DEEP Q-LEARNING* EM AMBIENTES SIMULADOS COM ESTADOS CONTÍNUOS

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Engenharia de Computação da Universidade Tecnológica Federal do Paraná.

Data de aprovação: 24/junho/2022

Prof. Dr. Dalcimar Casanova
Doutorado em Física Computacional
Universidade Tecnológica Federal do Paraná (UTFPR) - Pato Branco

---

Prof. Dr. Fábio Favarim
Doutorado em Engenharia Elétrica
Universidade Tecnológica Federal do Paraná (UTFPR) - Pato Branco

---

Prof. Dr. Marco Antonio de Castro Barbosa
Doutorado em Informática
Universidade Tecnológica Federal do Paraná (UTFPR) - Pato Branco

PATO BRANCO
2022

AGRADECIMENTOS

Gostaria de agradecer aos meus familiares por me apoiarem desde o início nesse ciclo desafiador que foi a UTFPR.

Agradeço também a todos os amigos que fiz durante esses anos e que de alguma forma me ajudaram a chegar até aqui.

A instituição UTFPR por ter me proporcionado a oportunidade de conhecer ótimas pessoas e excelentes profissionais, os quais foram essenciais na minha formação pessoal e profissional.

Por fim, agradeço ao professor Dalcimar Casanova pela orientação e pelos conhecimentos compartilhados.

Ao pensarmos no futuro do mundo, visamos o ponto em que ele estará se continuar a seguir o curso em que o vemos seguir hoje. Não percebemos que ele não se move em linha reta e que sua direção muda constantemente.
(WITTGENSTEIN, Ludwig).

RESUMO

A aprendizagem por reforço surgiu na década de 80 e compõe uma das três grandes áreas de aprendizagem de máquinas, sendo as outras duas a aprendizagem supervisionada e a não supervisionada. Os problemas de reforço possuem características peculiares, como a troca de informação que ocorre entre o agente e o ambiente em que ele está inserido, além disso, todos os problemas de aprendizagem por reforço são focados em objetivos e utilizam recompensas como estímulos para a aprendizagem. Outra particularidade da aprendizagem por reforço é que ela não precisa de informações prévias sobre o ambiente, pois é possível coletar os dados a partir das interações, utilizando técnicas de tentativa e erro. Apesar de ter surgido na década de 80, a aprendizagem por reforço voltou a ganhar popularidade recentemente com o avanço das redes neurais e o surgimento das redes neurais profundas, pois o fato de elas conseguirem encontrar aproximações de funções, tornou possível solucionar problemas com infinitos estados, que são mais semelhantes aos problemas existentes no mundo real. Uma grande ambição da aprendizagem por reforço é criar um algoritmo que possa ser generalizado e consiga se adaptar a diversos ambientes. Nesse sentido, esse trabalho tem o objetivo de avaliar o algoritmo *Deep Q-Learning* em 5 ambientes com estados contínuos e analisar tanto o seu desempenho quanto sua capacidade de adaptação para diferentes ambientes.

Palavras-chave: aprendizagem por reforço; *deep q-learning*; espaço de estados contínuo.

ABSTRACT

Reinforcement learning emerged in the 1980s and is one of three main areas of machine learning, the other two being supervised and unsupervised learning. Reinforcement problems have unique characteristics, such as the exchange of information between the agent and the environment in which it is inserted. In addition, all reinforcement learning problems are based on objectives and make use of rewards as stimulus for learning. Another particularity of reinforcement learning is that it does not need prior information about the environment, as it is possible to collect data from interactions, using trial and error techniques. Although it emerged in the 1980s, reinforcement learning has recently gained popularity with the advancement of neural networks and the emergence of deep neural networks, since the fact that they can find function approximations has made it possible to solve problems with infinite states, which are more similar to problems in the real world. A major ambition of reinforcement learning is to create an algorithm that can be generalized and adapted to various environments. In this sense, this work aims to evaluate the Deep Q-Learning algorithm on 5 continuous state environments and to analyze both its performance and its adaptation capacity for different environments.

Keywords: reinforcement learning; deep-q learning; continuous state space.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

A aprendizagem por reforço (AR) surgiu na década de 80 com a união de estudos sobre psicologia comportamental e controle ótimo, este método pode ser resumido como a construção de um algoritmo (ou um agente inteligente) que deve aprender diretamente por meio de interações com um ambiente, realizando ações que geram recompensas. O intuito do agente é maximizar a sua recompensa total enquanto busca alcançar um objetivo. O ambiente pode ser o mundo real, um jogo de computador, uma simulação ou até mesmo um jogo de tabuleiro como o xadrez. Assim como os humanos, o agente aprende com a consequência das suas ações, ao invés de precisar ser explicitamente ensinado. (SUTTON; BARTO, 2018)

A AR compõe uma das três grandes áreas de aprendizagem de máquinas, juntamente com a aprendizagem supervisionada e a aprendizagem não supervisionada. Na aprendizagem supervisionada não é necessário saber a relação entre os atributos de entrada, apenas o tipo de dado esperado na saída. Para o treinamento da aprendizagem supervisionada, são utilizados conjuntos de atributos com as suas respectivas saídas (ZHU, 2005). Após o treinamento, um algoritmo de aprendizagem supervisionada deve ser capaz de reconhecer, por exemplo, se determinada imagem é de um gato ou de um cachorro, ou ainda, se um cliente irá pagar ou não o seu empréstimo.

O segundo tipo de aprendizagem é a não supervisionada, em que o objetivo é encontrar estruturas escondidas nas informações de entrada, descobrindo correlações entre os dados (SUTTON; BARTO, 2018). A aprendizagem não supervisionada é utilizada para tarefas como agrupar países de acordo com o seu índice de desenvolvimento humano (RENDE; DONDURAN, 2013). Mais aplicações de aprendizagem não supervisionada podem ser encontradas em Kohonen (1990).

É importante ressaltar que as três formas de aprendizagem são necessárias e que cada uma possui o seu campo de atuação. Em problemas típicos de aprendizagem por reforço, pode parecer viável a utilização de aprendizagem supervisionada, no entanto, ao analisar os problemas mais a fundo, surgem as seguintes complicações:

- Na aprendizagem supervisionada, todas as possíveis combinações de estados e ações devem ser utilizadas como entrada para que o agente saiba qual ação tomar quando estiver em determinado estado. Em um jogo como o *Go*, por exemplo, o número total de estados é de $8 * 10^{100}$, ou ainda, se o agente for um carro autônomo e o ambiente for uma cidade, a quantidade total de estados seria incontável, tornando o treinamento computacionalmente inviável. Já na aprendizagem por reforço, existem métodos de treinar o agente partindo apenas da sua interação direta com o ambiente (tentativa e erro), sem a necessidade de fornecer informações prévias, ou seja, mesmo com uma quantidade elevada de estados, ainda é possível realizar o treinamento.

- Mesmo quando é possível treinar o algoritmo utilizando todas as combinações existentes como entrada, na aprendizagem supervisionada cada estado específico resulta sempre na mesma ação por parte do agente, diferente da aprendizagem por reforço, em que as ações são estocásticas, ou em outras palavras, para um mesmo estado, o agente pode escolher ações diferentes, possibilitando a descoberta de novas soluções para o problema, além de passar a ideia de que o agente pode executar ações criativas, o que é um dos princípios da aprendizagem por reforço.

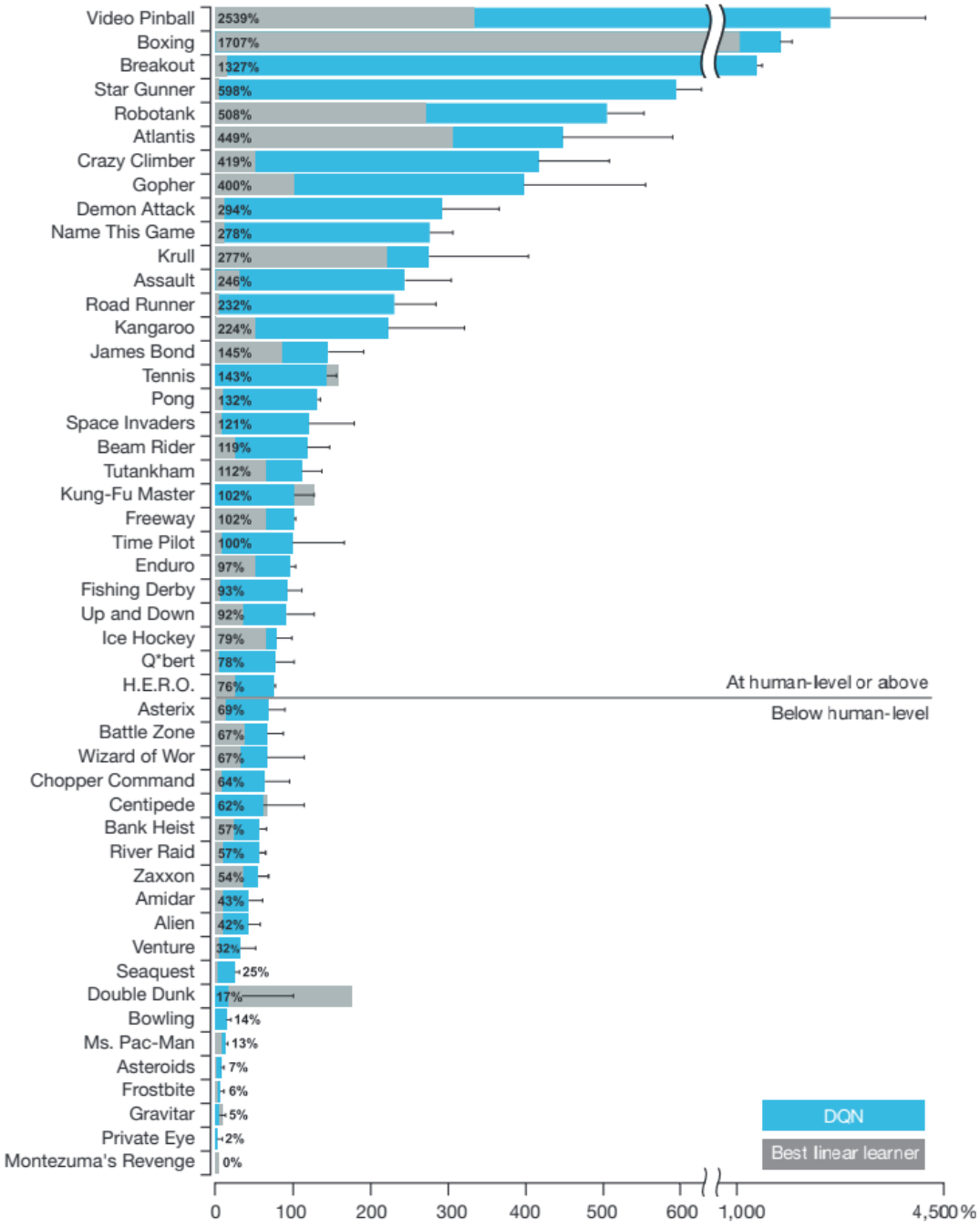
Apesar de ter surgido nos anos 80, a aprendizagem por reforço voltou a ganhar destaque recentemente, por conta dos avanços na área de aprendizagem profunda (*deep learning*), que possibilitaram a utilização de redes neurais profundas no processamento sensorial (KRIZHEVSKY; SUTSKEVER; HINTON, 2012), combinadas com o algoritmo *Q-Learning* dando origem ao *Deep Q-Learning* (LILLICRAP *et al.*, 2015). Esta combinação viabilizou a resolução de problemas com estados em tempo contínuo, ou seja, problemas com infinitos estados.

Uma das principais ambições da aprendizagem por reforço, é criar um algoritmo generalista, que seja capaz de aprender a se comportar de maneira correta em qualquer tipo de ambiente. Visando este objetivo, o criador do algoritmo *Deep Q-Learning* submeteu o seu agente a 49 jogos de *Atari 2600* sem alterar os parâmetros e a arquitetura da rede neural. Os resultados obtidos são demonstrados na Figura 1 (MNIH *et al.*, 2015).

É possível observar a comparação entre o algoritmo *Deep Q-Learning* e os melhores algoritmos de aprendizagem supervisionada criados especificamente para cada jogo. Na grande maioria dos jogos, o agente de AR obteve um resultado melhor do que os algoritmos treinados de maneira supervisionada. Outro fator interessante da Figura 1 é que em 29 dos 46 jogos, o agente obteve um desempenho equivalente ou superior ao desempenho médio de um ser humano.

De maneira semelhante ao estudo de Mnih *et al.* (2015), neste trabalho cinco ambientes foram submetidos ao algoritmo *Deep Q-Learing*. Após os treinamentos, foram realizadas comparações entre o desempenho do agente nos ambientes e os resultados obtidos em Mnih *et al.* (2015). Os ambientes propostos são os jogos de *Atari Breakout*, *Enduro*, *Freeway*, *Space Invaders* e *Beam Rider*. Foram feitas algumas modificações nos parâmetros de treinamento por limitações de *hardware*.

Figura 1 – Desempenho do algoritmo *Deep Q-Learning* em ambientes de *Atari 2600*



Fonte: Retirado de (MNIH *et al.*, 2015)..

1.1 OBJETIVO GERAL

Implementar o algoritmo *Deep Q-Learning* e verificar a sua capacidade de adaptação nos ambientes propostos.

1.2 OBJETIVOS ESPECÍFICOS

- Realizar o levantamento dos ambientes de aplicação e suas respectivas particularidades.
- Implementar o algoritmo *Deep Q-Learning*.
- Analisar o seu desempenho e capacidade de adaptação em problemas com estados contínuos, por meio das recompensas obtidas em cada um deles.

2 REFERENCIAL BIBLIOGRÁFICO

O Capítulo 2 está organizado de forma cronológica, de acordo com a evolução da aprendizagem por reforço, portanto, cada seção serve como embasamento para a seção seguinte.

A seção 2.1 apresenta brevemente as inspirações da aprendizagem por reforço e introduz os elementos presentes no processo de aprendizagem. A seção 2.2 mostra como formalizar os elementos apresentados na seção 2.1. A seção 2.3 utiliza o formalismo da seção 2.2 para mostrar o famoso algoritmo *Q-Learning*. A seção 2.4 expõe de maneira superficial o funcionamento de uma rede neural profunda. A seção 2.5 traz a união entre as redes neurais profundas e o algoritmo *Q-Learning*, que deu origem ao *Deep Q-Learning*. A seção 2.6 mostra algumas das principais plataformas de avaliação de aprendizagem por reforço disponíveis.

2.1 APRENDIZAGEM POR REFORÇO

Um dos principais objetivos da inteligência artificial é desenvolver agentes completamente autônomos, que interagem com determinado ambiente a fim de aprender a se comportar de maneira ótima. Criar sistemas inteligentes responsivos e capazes de efetivamente obter conhecimento é um desafio de longa data, abrangendo desde robôs, que conseguem sentir e reagir ao ambiente ao seu redor, até mesmo agentes baseados apenas em *software*, que conseguem interagir com linguagem natural e multimídia (ARULKUMARAN *et al.*, 2017).

A aprendizagem por reforço trata problemas enfrentados por um agente inteligente que deve aprender a se comportar adequadamente em um ambiente. O aprendiz coleta informações sobre o seu ambiente utilizando sensores e então decide qual ação tomar baseado em suas experiências anteriores e visando alcançar um objetivo (ex.: vencer uma partida de xadrez). Cada ação possui uma consequência, fazendo com que o agente fique mais próximo ou mais distante da sua meta, desse modo, são estimadas recompensas positivas ou negativas para cada ação do agente (SUTTON; BARTO, 2018).

As seções 2.1 e 2.2 têm o objetivo de dar a fundamentação teórica e matemática necessária sobre a área de aprendizagem por reforço e são baseadas principalmente no material de Sutton e Barto (2018).

2.1.1 INSPIRAÇÕES

A área de AR surgiu principalmente da união entre dois campos de estudos distintos. Um deles é a aprendizagem baseada em tentativa e erro, que possui origem na psicologia de aprendizagem animal e tenta mostrar como estímulos podem influenciar na tomada de decisão. Já a segunda área de estudos é a de controle ótimo e a utilização de programação dinâmica na sua resolução.

2.1.1.1 PSICOLOGIA

A primeira descrição sucinta de tentativa e erro como uma forma de aprendizagem presente no reino animal foi feita por *Edward Thorndike*:

> Das várias reações possíveis à uma mesma situação, aquelas que são acompanhadas por satisfação ao animal, mantendo todo o restante inalterado, o deixarão mais conectado com a situação, de modo que, quando ela se repetir, ele será mais propenso a repetir a ação; aquelas que são acompanhadas por desconforto ao animal, mantendo todo o restante inalterado, farão com que sua conexão com a situação fique enfraquecida, de modo que, quando ela se repetir, ele será menos propenso a repetir a ação. Quanto maior a satisfação ou desconforto, maior o fortalecimento ou enfraquecimento do vínculo (THORNDIKE, 1911).

De forma mais sucinta, quando um estímulo bom ocorre a um animal, a situação tem maior chance de se repetir, e em contrapartida, quando um estímulo ruim ocorre, as chances de a situação se repetir diminuem. *Thorndike* chamou isso de "Lei do Efeito", pois descreve o efeito da repetição (reforço) na tendência das ações tomadas pelos animais.

A implementação de tentativa e erro na computação, teve início junto com as primeiras ideias de inteligência artificial. Em 1948, *Alan Turing* projetou um modelo chamado "Sistema de Prazer e Dor", que funciona de acordo com a Lei de Efeito, nesse sistema, quando a máquina encontra uma configuração em que a ação a ser tomada não está predeterminada, os dados ausentes são escolhidos de forma aleatória e são então colocados provisoriamente na entrada da máquina. Se um estímulo de dor ocorrer, todas as entradas provisórias são canceladas, porém, caso ocorra um estímulo de prazer, as entradas se tornam permanentes (COPELAND, 2004).

2.1.1.2 CONTROLE ÓTIMO

O termo ”controle ótimo” começou a ser utilizado no final dos anos 50 e descreve o problema de projetar um controlador para minimizar ou maximizar uma medida de comportamento de um sistema dinâmico ao longo do tempo. Uma das abordagens para tratar esse tipo de problema foi feita por *Richard Bellman* na década de 50, utilizando o conceito de estados em sistemas dinâmicos e a equação chamada *Value Function*, atualmente conhecida como Equação de *Bellman*.

A classe de métodos que utilizam a Equação de *Bellman* para a resolução de problemas de controle ótimo, ficou conhecida como programação dinâmica (BELLMAN, 1966). Bellman introduziu também a versão discreta e estocástica de problemas de controle ótimo, conhecida como *Markov Decision Process* (MDP) (PUTERMAN, 2014). Ele foi a primeira pessoa a utilizar o famoso termo *”the curse of dimensionality”* (a maldição da dimensionalidade), explicando como os requisitos computacionais crescem de forma exponencial com o aumento da quantidade de estados.

A integração entre os métodos de programação dinâmica e a aprendizagem animal foi realizada principalmente por *Chris Watkins* em 1989 (WATKINS, 1989), que adotou o tratamento de aprendizagem por reforço utilizando o formalismo MDP. Desde então, vários estudos foram realizados utilizando essa combinação, dentre eles, destacam-se *Dimitri Bertsekas* e *John Tsitsiklis* (BERTSEKAS; TSITSIKLIS, 1996).

A adaptação do Processo de Decisão de *Markov* para problemas de inteligência artificial foi essencial para o surgimento e evolução da aprendizagem por reforço moderna e será retomada de forma detalhada na seção 2.2.

2.1.2 ELEMENTOS DA APRENDIZAGEM POR REFORÇO

Os problemas envolvendo aprendizagem por reforço geralmente possuem características semelhantes e podem ser formalizados utilizando conceitos de programação dinâmica, mais especificamente, MDP. A ideia básica é que existe um agente inteligente capaz de sentir o estado do ambiente em que está inserido e então realizar ações que afetem esse estado. Todos os agentes de aprendizagem por reforço possuem objetivos explícitos, conseguem sentir aspectos do seu ambiente e estão aptos a escolher ações que influenciem nesse ambiente. Esses são os aspectos básicos da aprendizagem por reforço (sensação, ação e objetivo).

Além do agente e do ambiente, outros elementos se destacam na aprendizagem por reforço, são eles: política, recompensa, função de valor e um modelo do ambiente.

2.1.2.1 POLÍTICA

A política define a forma como o agente se comporta em um determinado instante. De forma simples, a política é o mapeamento das possíveis ações a serem tomadas pelo agente após ele captar o estado em que se encontra. Isso corresponde ao que na psicologia é chamado de regras de estímulo-resposta, ou seja, para cada estado (estímulo), existe um conjunto de ações (respostas) que o agente pode realizar.

Em alguns casos, a política pode ser uma função simples ou uma tabela de consulta, enquanto que em outros casos, pode envolver um grande custo computacional, como por exemplo, algoritmos de busca. A política é o núcleo do agente inteligente, de forma que sozinha, é capaz de determinar o comportamento do aprendiz. Políticas podem ser tanto determinísticas quanto estocásticas, no caso de serem estocásticas, cada ação possui uma certa probabilidade de ser executada.

A otimização da política visa encontrar uma política ótima, ou seja, para todo estado, o agente saberá exatamente qual a melhor ação a ser tomada para alcançar seu objetivo. Encontrar a política ótima para um problema, significa que o processo de aprendizagem está concluído (LI, 2017).

2.1.2.2 SINAL DE RECOMPENSA

A recompensa define o objetivo de um problema de aprendizagem por reforço. Cada vez que o agente transita de um estado para outro, o ambiente envia um valor numérico como *feedback*, chamado de recompensa. A principal função do agente é maximizar a soma das recompensas recebidas durante todo o percurso. O sinal de recompensa define então quais são os eventos bons e ruins para o agente, analogamente, em um sistema biológico, as recompensas podem ser comparadas com estímulos de prazer e dor.

Esse retorno é também a principal base para realizar modificações na política, pois, se uma ação selecionada pela política resultar em uma recompensa baixa, a política pode ser atualizada para que o agente tome uma ação diferente quando se encontrar naquela mesma situação no futuro.

Uma política ótima é qualquer política que maximize a recompensa recebida pelo agente no ambiente, entretanto, uma das principais dificuldades existentes na aprendizagem por reforço é o fato de que o agente deve aprender apenas com a consequência das suas ações e encontrar uma política ótima através de tentativa e erro, visto que normalmente não existem informações sobre as transições de estados e a única resposta recebida pelo agente são as recompensas.

De forma resumida, toda interação com o ambiente gera informações, que são utilizadas para atualizar o conhecimento do aprendiz (ARULKUMARAN *et al.*, 2017). Na Figura 2 é possível observar os elementos da aprendizagem por reforço e como acontece a troca de informações na prática.

Figura 2 – Fluxograma da interação entre o agente e o ambiente

Fonte: Adaptado de (ARULKUMARAN *et al.*, 2017).

2.1.2.3 FUNÇÃO DE VALOR

A função de valor é uma previsão da soma das recompensas futuras, tornando possível estimar quão bom é estar em cada estado (LI, 2017). O valor de um estado pode ser definido

como o acúmulo das recompensas esperadas pelo agente, iniciando a estimativa nesse estado (ARULKUMARAN *et al.*, 2017). Enquanto um sinal de recompensa indica o que é bom apenas no momento atual, a função de valor especifica o que é bom a longo prazo, levando em consideração os estados que provavelmente serão alcançados e as recompensas presentes nesses estados. Por exemplo, um estado pode retornar uma recompensa baixa e mesmo assim possuir um valor alto, isso significa que os estados que o seguem possuem recompensas altas. O oposto também pode acontecer.

As recompensas podem ser vistas como elementos primários, enquanto que os valores, são previsões das recompensas e podem ser entendidos como elementos secundários, pois sem recompensas não haveriam valores e o único propósito em estimar valores é alcançar recompensas maiores. Mesmo assim, o valor é o principal elemento a ser examinado antes de se tomar uma decisão.

As ações do agente são escolhidas com base na maior estimativa de valor e não na maior recompensa, dado que essa estimativa resultará no maior montante de recompensas a longo prazo. Infelizmente, é muito mais difícil estimar o valor do que as recompensas, visto que as recompensas são recebidas diretamente do ambiente, já os valores precisam ser estimados e re-estimados a partir de várias observações feitas pelo agente durante todo o seu tempo de vida. Na verdade, o componente mais importante em praticamente todos os algoritmos de aprendizagem por reforço é o método utilizado para estimar valores de maneira eficiente.

2.1.2.4 MODELO DO AMBIENTE

Um modelo do ambiente é algo que tenta imitar o comportamento do ambiente, ou seja, permite fazer inferências sobre como o ambiente irá se comportar durante o treinamento. Por exemplo, dado um estado e uma ação, o modelo tentará prever qual será o próximo estado e a próxima recompensa (SUTTON; BARTO, 2018). A ideia chave por trás da utilização de um modelo, é conseguir antecipar a transição de estados e recompensas em um ambiente, mesmo sem nunca ter passado por esses estados, portanto, modelos são utilizados para o planejamentos das ações (ARULKUMARAN *et al.*, 2017). No xadrez, por exemplo, as próprias regras do jogo podem ser vistas como um modelo do ambiente, pelo fato de proverem informações prévias sobre o jogo para o agente.

Métodos que utilizam modelo do ambiente para a aprendizagem recebem o nome de *model-based* (baseados em modelo), enquanto que os métodos que não utilizam modelos, ou seja, aprendem apenas com tentativa e erro, são chamados de *model-free* (livres de modelo).

Treinamentos que utilizam aprendizagem baseada em modelo tendem a convergir para um resultado de forma mais rápida, porém, esse tipo de treinamento depende muito da qualidade do modelo utilizado e caso ele seja impreciso, a aprendizagem se torna instável, chegando em resultados não ótimos. Em problemas do mundo real, os modelos podem se tornar extre-

mamente complexos e imprevisíveis, o que torna a aprendizagem livre de modelo mais popular mesmo levando mais tempo para treinar.

2.2 PROCESSO DE DECISÃO DE *MARKOV*

O processo de decisão de *Markov* ou MDP (*Markov Decision Process*) é uma forma de modelar processos que possuem transição de estados probabilísticas, neste tipo de problema é possível observar em que estado o processo está e tomar uma decisão (executar uma ação) que irá influenciar no estado atual e nos estados futuros, influenciando consequentemente, nas recompensas. Esses processos devem obedecer a propriedade de *Markov*, que diz que o efeito de uma ação em um estado depende apenas da ação e do estado atual do sistema, ou seja, não depende dos estados anteriores e nem das ações tomadas para chegar ao estado atual (PELLEGRINI; WAINER, 2007).

Muitos agentes autônomos operam em um ambiente onde as ações possuem efeito estocástico. Os casos mais comuns são de robôs autônomos que recebem entradas por meio de uma matriz de sensores e então precisam decidir qual ação tomar baseados no seu estado, como pode ser observado em Atrash *et al.* (2009) e Riedmiller *et al.* (2009). Problemas com essa estrutura podem ser modelados utilizando o formalismo de MDP, o que faz com que esta seja uma das melhores maneiras de abordar problemas de aprendizagem por reforço (SHANI; PINEAU; KAPLOW, 2013).

2.2.1 FORMALISMO NA APRENDIZAGEM POR REFORÇO

Formalmente, problemas de AR podem ser descritos como um MDP, e são representados pela tupla (S, A, T, R, γ), que consiste em:

- Um conjunto S de todos os estados presentes no ambiente;
- Um conjunto A de todas as ações possíveis;
- Uma dinâmica de transição $T(s_{t+1}|s_t, a_t)$ que dá a probabilidade do sistema ir para o estado $s_{t+1} \in S$, estando no estado $s_t \in S$ e realizando uma ação $a_t \in A$. A notação $T(s'|s,a)$ também é comumente encontrada na literatura;
- Uma função de recompensa imediata $R(s_t,a_t,s_{t+1})$, que retorna a recompensa por tomar uma ação a estando no estado s em um tempo t;
- Um fator de desconto $\gamma \in [0,1]$, que influencia diretamente nas ações do agente.

Existe ainda uma equação de probabilidade mais geral, que estima tanto o próximo estado quanto a recompensa obtida por executar uma ação a no estado s.

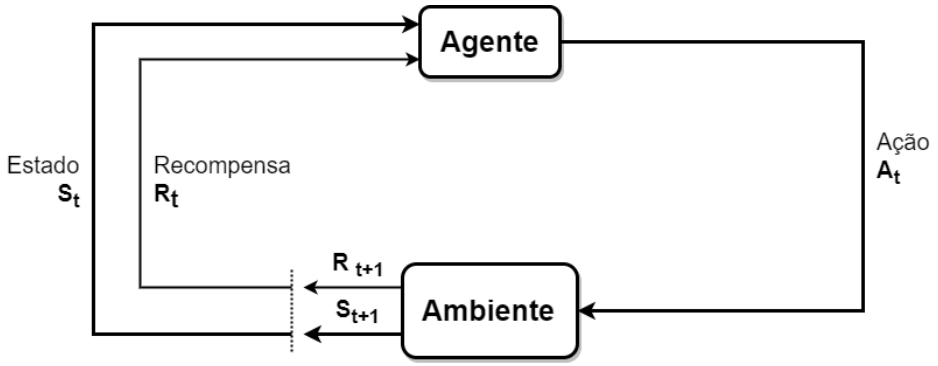
$$p(s',r|s,a) \tag{1}$$

Como é possível observar Equação 1, a probabilidade tanto da recompensa quanto do próximo estado, dependem apenas do estado anterior e da ação escolhida pelo agente naquele estado, isso implica que todas as informações relevantes sobre os estados passados, devem estar de alguma forma presentes em s_t, como determina a propriedade de *Markov* (ARULKUMARAN *et al.*, 2017).

Para facilitar a notação, o estado s_{t+1}, também pode ser escrito como s', e os estados e ações que ocorrem no tempo t, são escritos apenas como s e a ao invés de s_t e a_t, como mostra a Equação 1.

De forma mais direta, o agente e o ambiente interagem em uma sequência discreta de intervalos de tempo ($t = 1,2,3,...,n$), em cada intervalo t, o agente recebe uma informação sobre o estado do ambiente $s_t \in S$, e com base nisso, seleciona uma ação $a_t \in A$. No tempo seguinte $(t + 1)$, como consequência da ação escolhida, o agente recebe uma recompensa numérica $r_{t+1} \in R$ e muda para o estado s_{t+1}, como mostra a Figura 3.

Figura 3 – Interação Agente-Ambiente utilizando o formalismo de MDP



Fonte: Adaptado de (SUTTON; BARTO, 2018).

O *framework* do MDP é abstrato e flexível, podendo ser aplicado em muitos problemas, de várias maneiras diferentes. Por exemplo, os intervalos de tempo não precisam necessariamente ser intervalos de tempo real, eles podem se referir a estágios de tomada de decisão e ação. As ações podem ser tanto de baixo nível, como a tensão aplicada no movimento de um motor, quanto de alto nível, como escolher qual rota seguir para chegar no trabalho. Coisas como a delimitação entre o agente e o ambiente ou a forma como as recompensas são recebidas, podem ser projetadas de inúmeras formas, pois não seguem uma regra bem definida, e portanto, precisam ser analisadas ao formalizar um problema de aprendizagem por reforço.

2.2.2 LIMITES ENTRE O AGENTE E O AMBIENTE

O projeto e a modelagem de um problema de aprendizagem por reforço pode ser algo bastante trabalhoso e que demanda interpretação e testes por parte do projetista. Uma das

primeiras coisas que precisam ser bem definidas em um problema de AR, é a fronteira que define onde acaba o agente e começa o ambiente. Isso varia de projeto para projeto e não possui um algoritmo ou fórmula para determinar qual é a melhor escolha, mas existem algumas sugestões que facilitam a tomada de decisão no momento da modelagem.

Em geral, o limite entre o agente e o ambiente não é o mesmo que o limite físico do corpo de um robô ou de um animal com o local que ele está inserido. Frequentemente, o limite é muito mais próximo do agente, por exemplo, os próprios motores e ligações mecânicas de um robô podem ser considerados parte do ambiente e não parte do agente, de forma análoga, em uma pessoa ou animal, os músculos, esqueleto e órgãos, também devem ser considerados parte do ambiente, nesse caso, a mente seria o agente inteligente. A regra geral a ser seguida é que qualquer coisa que não pode ser modificada de forma arbitrária pelo agente, é considerada algo externo, e portanto, faz parte do ambiente.

A fronteira entre o agente e o ambiente pode ser localizada em diferentes pontos para diferentes propósitos. Em um robô complexo, podem existir vários agentes operando ao mesmo tempo, cada um com sua própria fronteira e seus próprios objetivos e ações. Na prática, a fronteira entre o agente e o ambiente é determinada quando é possível perceber estados, ações e recompensas particulares, e portanto, definir uma tarefa que seja importante para resolução do problema. Na aprendizagem por reforço, assim como nos outros tipos de aprendizagem, esse tipo de escolha é muito mais uma questão de arte do que de ciência (SUTTON; BARTO, 2018), mas pode influenciar tanto nas ações do agente quanto no desempenho final do algoritmo, como mostra Jiang (2019).

2.2.3 OBJETIVOS E RECOMPENSAS

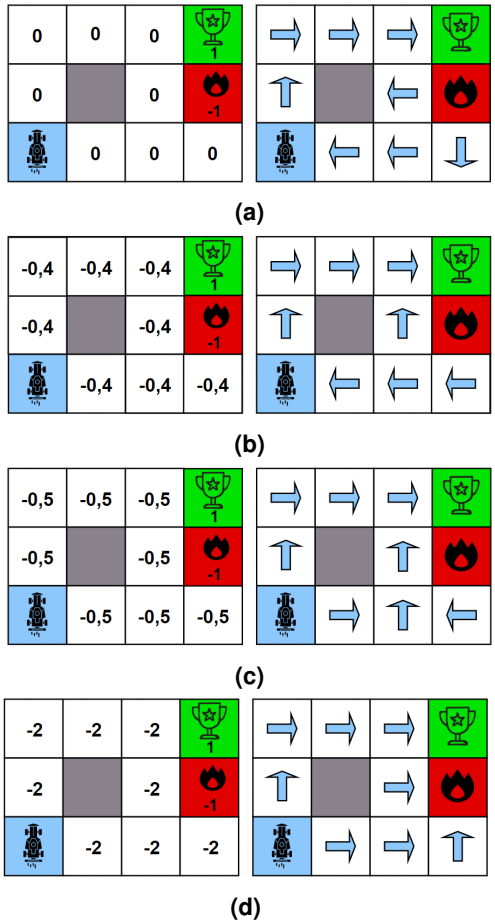
Na aprendizagem por reforço, o objetivo do agente é formalizado em termos de um sinal de recompensa, passado do ambiente para o agente. A recompensa é um valor numérico, com $R_t \in \mathbb{R}$, e de maneira informal, o objetivo do agente é maximizar o total de recompensas recebidas. Isso não significa maximizar as recompensas imediatas, mas sim as recompensas acumuladas durante todo o percurso. A utilização do sinal de recompensa para formalizar a ideia de objetivos, é uma das características mais marcantes da aprendizagem por reforço.

O sucesso de uma aplicação de aprendizagem por reforço depende fortemente de quão bem o sinal de recompensa enquadra o objetivo e quão bem ele avalia o progresso em alcançar aquela meta. Por esses motivos, modelar um sinal de recompensa é uma parte crítica de qualquer projeto de aprendizagem por reforço.

Como foi dito anteriormente, cada tipo de problema precisa de uma modelagem de recompensa que melhor se encaixe àquele objetivo. Por exemplo, para ensinar um robô a caminhar, pesquisadores forneceram recompensas proporcionais à distância que o robô se moveu para frente em cada intervalo de tempo, caso ele fosse para trás, a recompensa seria negativa como uma forma de punição. Para um robô que coleta lixo, a recompensa pode ser zero para a

maior parte do tempo e +1 para cada vez que ele realiza a coleta, pode existir também uma recompensa de -1 quando o robô bate em algo. Já em um jogo de xadrez, as únicas recompensas são +1 quando o agente vence o jogo e -1 quando perde, nos casos de empate, ou qualquer outro estado não terminal, a recompensa deve ser 0, isso se deve ao fato de que o agente sempre busca maximizar as suas recompensas, então caso ele receba uma recompensa para cada peça eliminada, o agente pode acabar priorizando capturar as peças do oponente do que vencer o jogo.

Figura 4 – Influência das recompensas no desempenho do jogo *Gridworld*



Fonte: Adaptado de(GRANATYR; PONTEVES; EREMENKO, 2019).

Na Figura 4, estão representados 4 treinamentos distintos do jogo *Gridworld*, nesse jogo o agente começa na posição (1,1), podendo se mover para cima, baixo, esquerda ou direita. Seu objetivo é chegar no troféu localizado na posição terminal (3,4), durante o percurso existe um obstáculo na posição (2,2) que o agente não consegue atravessar. A posição (2,4) também é terminal, porém, se trata de um estado de derrota e, portanto, o agente deve evitá-lo. As imagens da esquerda representam as recompensas do ambiente, enquanto que as imagens da direita mostram a política final após o treinamento.

A Figura 4a mostra o desempenho do agente quando existem recompensas apenas nos estados terminais: +1 para vitória, -1 para derrota e 0 para todos os outros estados. É possível

notar algumas falhas na solução encontrada após o treinamento, pois, se o agente estiver nos estados (1,4) ou (2,3), ele ficará preso e não conseguirá completar o jogo.

Na Figura 4b, foram adicionadas recompensas negativas de $-0{,}4$ para todos os estados não terminais, essa técnica se chama *living penalty* e ”pune” o agente para cada ação que ele tomar sem chegar no objetivo final. O intuito dessa estratégia é fazer com que o agente conclua o jogo o mais rápido possível. Nesse caso, o caminho encontrado pelo agente foi melhor que o anterior, pois agora ele consegue alcançar o seu objetivo a partir de qualquer estado.

A Figura 4c utiliza a mesma técnica que a Figura 4b, mas a punição foi ajustada para $-0{,}5$. É possível notar um aperfeiçoamento no caminho percorrido pelo agente, dado que a partir de qualquer estado não terminal, ele sempre irá seguir o menor caminho até o seu objetivo, portanto, esses são os valores ideais de recompensas e geram uma política ótima para o jogo.

Já na Figura 4d, a punição foi ajustada para -2 em cada estado não terminal. Ao analisar os caminhos escolhidos pelo agente, é possível notar que em alguns casos ele acaba indo para o estado terminal de derrota, isso se deve ao fato de a punição por se locomover ser maior do que a própria punição por perder o jogo que é de -1. Dessa forma, essa é uma punição exagerada e não se adéqua ao jogo em questão.

2.2.4 PROBLEMAS EPISÓDICOS E CONTÍNUOS

Como foi dito nas seções anteriores, o objetivo do agente é maximizar as recompensas acumuladas ao longo de todo o trajeto. Em geral, o que se quer maximizar é o retorno total esperado, denotado como G_t e definido como alguma função específica da sequência de recompensas. O caso mais simples da função de retorno é a soma das recompensas.

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + ... + R_T \tag{2}$$

Na Equação 2, T representa o tempo final. Este tipo de abordagem faz sentido apenas em problemas com uma quantidade limitada de intervalos de tempo, como jogos ou labirintos. Esse tipo de problema é chamado de episódico, um exemplo de problema episódico é o representado na Figura 4.

Cada episódio termina em um estado especial, chamado de estado terminal, logo em seguida o episódio é reiniciado em um dos possíveis estados iniciais. O próximo episódio começa independente do resultado do episódio anterior, ou seja, mesmo que o episódio anterior termine com uma vitória ou uma derrota.

Em problemas episódicos, algumas vezes é necessário distinguir o conjunto de estados não terminais S, do conjunto de todos os estados mais os estados terminais, denotados S^+. O tempo de término T, é uma variável aleatória, pois normalmente os episódios terminam com tempos diferentes.

Em outros casos, os problemas podem se estender por longos períodos de tempo, podendo até ser considerados infinitos. Tarefas desse tipo são classificadas como contínuas e precisam de uma abordagem especial, como pode ser visto em Lillicrap *et al.* (2015) e Duan *et al.* (2016). A Equação 2 não se aplica a este tipo de problema, pois nesse caso o último intervalo de tempo seria $T = \infty$ e o retorno que deve ser maximizado também pode ser infinito (SUTTON; BARTO, 2018).

Para tratar tarefas de controle contínuo, foi introduzido um novo elemento à equação de retorno, o fator de desconto, como mostra a Equação 3.

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{3}$$

A variável γ, presente na Equação 3, representa a taxa de desconto, com $0 \leq \gamma \leq 1$. Sempre que $\gamma < 1$, a soma infinita presente na Equação 3 converge para um valor real, o que permite o cálculo de G_t.

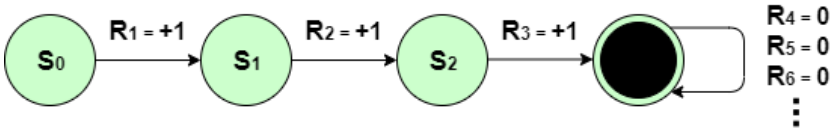
Na prática, quando $\gamma = 0$, o agente se preocupa apenas em maximizar as recompensas imediatas, não pensando nos estados futuros e acaba virando um algoritmo guloso, o que não é interessante, dado que um dos principais desafios da aprendizagem por reforço é o planejamento e maximização de recompensas a longo prazo. Em contrapartida, quanto mais γ se aproxima de 1, mais o agente leva em conta as recompensas futuras e planeja as suas ações a longo prazo, por esse motivo, geralmente o fator de desconto é próximo de 1, podendo até mesmo ser igual a 1 em problemas episódicos pequenos.

Com intuito de facilitar a implementação dos códigos de aprendizagem por reforço, a Equação 3 pode ser reescrita da seguinte maneira:

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

De maneira geral, problemas episódicos também podem ser tratados como contínuos, para isso basta adicionar uma nova transição no estado terminal, fazendo com que o estado retorne para ele mesmo e receba uma recompensa igual a zero, como mostra a Figura 5. Dessa forma, a Equação 3 passa a ser válida para ambos os tipos de tarefas, contínuas e episódicas. Essa modificação serve para simplificar tanto os cálculos, quanto a notação de problemas episódicos.

Figura 5 – Representação de um problema episódico como contínuo



Fonte: Adaptado de (SUTTON; BARTO, 2018).

2.2.5 POLÍTICA E FUNÇÃO DE VALOR

A grande maioria dos algoritmos de aprendizagem por reforço envolvem a estimativa de uma função de valor, para estimar quão bom é para o agente estar em cada estado em termo de recompensas esperadas. Porém, como foi visto anteriormente, a estimativa do valor da recompensa é feita com base em quais ações o agente irá tomar, e consequentemente, a função de valor é definida em relação à maneiras particulares que o agente pode agir, chamadas de políticas.

Representada pela letra grega π, uma política é o mapeamento das probabilidades do agente tomar cada uma das ações disponíveis. Se o agente está seguindo uma política π em um tempo t, então $\pi(a|s)$ corresponde a probabilidade do agente realizar a ação $a \in A(s)$, quando estiver no estado $s \in S$. Na aprendizagem por reforço, a política é atualizada a partir das experiências do agente e cada algoritmo possui o seu método de atualizá-la.

A função de valor de um estado s seguindo uma política π, denotado como $v_\pi(s)$ e é o retorno esperado começando no estado s e agindo de acordo com π. Formalmente, $v_\pi(s)$ é descrita pela Equação 5.

$$v_\pi(s) = E_\pi\left[G_t|S_t = s\right] = E_\pi\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1}\middle|S_t = s\right], \text{para todo } s \in S \tag{5}$$

O elemento E_π presente na Equação 5 representa o valor esperado de alguma variável aleatória, dado que o agente segue a política π. É importante ressaltar que o valor de estado terminal, quando ele existir, é sempre igual a zero. De forma análoga a Equação 5, é possível definir um valor para a escolha de uma ação a no estado s seguindo uma política π, essa equação é conhecida como função ação-valor e denotada como $q_\pi(s,a)$.

$$q_\pi(s,a) = E_\pi\left[G_t|S_t = s, A_t = a\right] = E_\pi\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1}\middle|S_t = s, A_t = a\right] \tag{6}$$

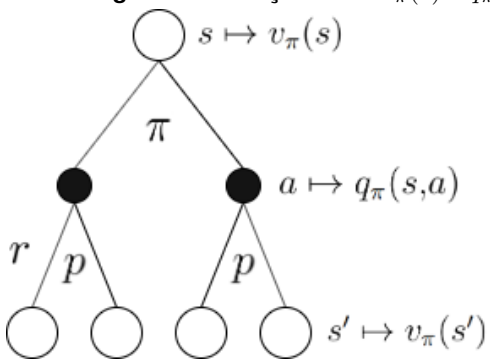
Uma propriedade fundamental de ambas as equações de valor utilizadas na aprendizagem por reforço é que elas podem ser escritas como funções recursivas, seguindo a mesma ideia da Equação 4. Para qualquer política π e qualquer estado s, esta propriedade se mantém

para o valor de s e de seus possíveis sucessores. Unindo as equações 4 e 5, é possível escrever $v_\pi(s)$ de maneira recursiva:

$$\begin{aligned} v_\pi &= E_\pi[R_{t+1} + \gamma G_{t+1} | S_t = s] \\ v_\pi &= E_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s] \end{aligned} \tag{7}$$

A Equação 7 é conhecida como Equação de *Bellman* para v_π, ela expressa a soma da qualidade das possíveis ações $q(s,a)$ ponderadas pela probabilidade (determinada pela política π) de tomar uma ação a estando no estado s. Como normalmente os problemas são estocásticos, após a escolha de uma ação a, existe uma probabilidade p de ir para o próximo estado s' e receber uma recompensa r, essa transição está representada na Figura 6.

Figura 6 – Diagrama da relação entre $v_\pi(s)$ e $q_\pi(s,a)$



Fonte: Adaptado de (SUTTON; BARTO, 2018).

Resolver um problema de aprendizagem por reforço significa encontrar uma política que alcance uma grande recompensa durante o percurso. Uma política π é dita melhor que outra política π', se e somente se $v_\pi(s) \geq v_{\pi'}(s)$ para todo $s \in S$. Existe sempre pelo menos uma política que é melhor ou igual a todas as outras, chamada de política ótima. As políticas ótimas são denotadas com $\pi*$, elas compartilham a mesma função de valor, chamada de função de valor ótima e denotada com $v*$, de maneira análoga, existe uma função ação-valor ótima denotada $q_{\pi*}(s,a)$.

Os algoritmos de aprendizagem por reforço que realizam a otimização da função ação-valor Q, podem ser classificados de duas maneiras:

- ***On-policy***: A política comportamental estudada até aqui (π) é avaliada e otimizada, e em seguida essa mesma política é utilizada para tomar decisões.

- ***Off-policy***: Utiliza duas políticas para a equação Q, a primeira é a política comportamental, denotada com $\mu(a|s)$ que é responsável por escolher as ações em todos os estados. A segunda política $\pi(s|a)$ é utilizada para realizar os cálculos de Q. A política μ é então utilizada para avaliar e otimizar a política π.

O maior interesse na aprendizagem por reforço é encontrar $q * (s,a)$. Obter a função de ação ótima implica que o agente sabe exatamente a qualidade das ações em qualquer estado, e portanto, pode decidir a partir da qualidade, qual ação tomar. Por esse motivo, foram criadas várias técnicas para otimizar a função $q(s,a)$, dentre elas está o algoritmo abordado neste trabalho, conhecido como *Q-Learning*.

2.3 *Q-LEARNING*

O algoritmo *Q-Learning* desenvolvido por Watkins (WATKINS, 1989) para encontrar a função ação-valor $Q*$ é uma das principais contribuições na história da aprendizagem por reforço e continua sendo utilizado como base para grande parte dos algoritmos atuais. Se trata de um método classificado como *off-policy*, pois a sua convergência para a função ação-valor ótima $Q*$ não depende da política inicial, dado que a função Q se aproxima diretamente da função $Q*$ por meio de atualizações nos pares ação-estado a medida que eles são visitados (JÚNIOR, 2009).

Como explicado anteriormente, algoritmos *off-policy* possuem duas políticas associadas a eles, neste sentido, o algoritmo *Q-Learning* é um caso especial de algoritmos *off-policy* em que a política objetivo $\pi(a|s)$ é gulosa em relação à função $Q(s,a)$, portanto, a estratégia do algoritmo é tomar ações que resultem no maior valor de Q.

A equação de atualização do algoritmo *Q-Learning* baseia-se na função ação-valor Q, e é denotada por:

$$\text{Novo } Q(s_t,a_t) = Q(s_t,a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1},a') - Q(s_t,a_t) \right] \tag{8}$$

- Novo $Q(s_t,a_t)$: Novo valor de Q para os pares de estado e ação (s_t,a_t);
- $Q(s,a)$: Valores atuais de $Q(s_t,a_t)$;
- α: Taxa de aprendizagem, responsável por definir o quanto a ação tomada irá influenciar no novo valor de Q;
- r_{t+1}: Recompensa por tomar a ação naquele estado;
- γ: Fator de desconto;
- $\max_a Q(s_{t+1},a')$: Recompensa máxima esperada, seguindo a política π gulosa.

O algoritmo *Q-Learning* foi o primeiro método de aprendizagem por reforço a possuir fortes provas de convergência. Segundo Watkins e Dayan (1992) e Watkins (1989), se cada par estado-ação for visitado infinitas vezes, a função $Q(s,a)$ irá convergir para $Q*$ com 100% de probabilidade, usando um α suficientemente pequeno (JÚNIOR, 2009).

A escolha do par ação-estado a ser atualizado é feita a partir da política de comportamento $\mu(a|s)$, geralmente esta política é do tipo ϵ-Gulosa, ou seja, em cada intervalo de tempo o agente tem uma chance $\epsilon \in [0,1]$ de escolher uma ação aleatória. Se o valor de ϵ for maior do que um valor $p \in [0,1]$ gerado aleatoriamente, o agente executa uma ação aleatória. Caso contrário, a ação é escolhida de maneira gulosa com relação à $Q(s,a)$ (OPPERMANN, 2018):

$$\mu(a|s) = \begin{cases} a \text{ aleatória}, & \text{se } \epsilon \geq p \\ \max_a Q(s,a), & \text{se } \epsilon < p \end{cases} \tag{9}$$

A escolha do valor de ϵ para a política comportamental, traz um dilema importante para a aprendizagem por reforço conhecido como *exploration vs. exploitation*, que será abordado a seguir.

2.3.1 *EXPLORATION VS. EXPLOITATION*

Uma das maiores dificuldades na aprendizagem por reforço é encontrar um equilíbrio entre os dois tipos de movimentos:

- ***Exploitation*:** Escolhe a melhor ação com base nas informações atuais;
- ***Exploration*:** Busca mais informações, explora novos caminhos.

Este dilema consiste em decidir quando o agente deve escolher ações que não são ótimas, com o intuito de explorar o seu ambiente e encontrar novos caminhos, possivelmente aperfeiçoando a sua política, ou quando o agente deve escolher apenas as ações ótimas definidas por μ, a fim de obter progresso na tarefa (ARULKUMARAN *et al.*, 2017). Caso o agente esteja programado para realizar apenas *exploitation*, a sua política nunca será atualizada, e portanto, existe uma grande chance de ele não aprender a solução ótima para o problema, pois geralmente ela é encontrada a partir da combinação entre *exploration* e *exploitation*.

Uma solução comumente utilizada para esse dilema por conta da sua simplicidade e consistência é a ϵ-Gulosa Decadente, representada na Equação 10, em que n equivale ao número de iterações. Nesta abordagem os movimentos do agente começam completamente aleatórios, com objetivo de explorar o ambiente e coletar novas informações, porém, com o passar do tempo a política gulosa começa a escolher a maior parte das ações, garantindo uma boa recompensa final.

$$\epsilon = \frac{1}{\sqrt{n+1}} \tag{10}$$

A Equação 10 é uma das formas de se escrever a função ϵ-Gulosa Decadente, mas não é a única, outra forma bastante utilizada é iniciando ϵ com 1 e reduzindo de forma constante (ex: 0,01) a cada n iterações, até que chegue ao valor 0 ou até que o episódio termine. Outras

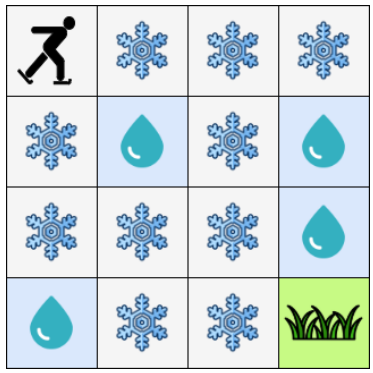
estratégias de *exploration* e *exploitation* podem ser encontradas em Kaelbling, Littman e Moore (1996).

2.3.2 TABELA-Q

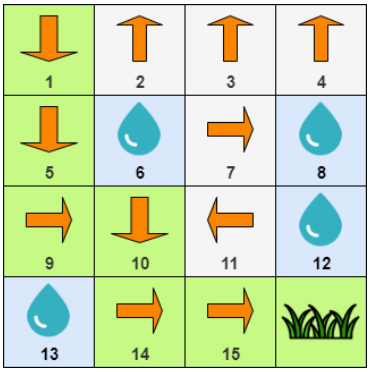
O algoritmo *Q-Learning* funciona estimando um valor de qualidade para todos os pares estado-ação de um ambiente, como mostra a Equação 10, na prática isso significa que deve-se criar uma matriz de qualidade, conhecida como Tabela-Q com dimensão SxA que mapeia o valor de todas as combinações entre estados e ações.

O exemplo abaixo é de um jogo simples disponibilizado pela OpenAI (OPENAI, 2019) e será utilizado para demonstrar o funcionamento da Tabela-Q.

Figura 7 – Jogo Lago Congelado



(a)



(b)

Fonte: Adaptado de (OPENAI, 2019).

A Figura 7a mostra o jogo Lago Congelado, nesse jogo o esquiador deve encontrar um caminho seguro para atravessar lago e chegar na terra firme. Os blocos com gelo representam os caminhos que podem ser percorridos, enquanto que os blocos com água são buracos no gelo e devem ser evitados. Caso o esquiador caia em um buraco o episódio termina e ele volta para o início.

No treinamento com o algoritmo *Q-Learning*, a política inicial foi com todas as estimativas iguais a zero e os seguintes parâmetros da Equação 10 foram ajustados: taxa de aprendizagem $\alpha = 0{,}1$, fator de desconto $\gamma = 0{,}99$. Para a troca entre *exploration* e *exploitation* foi utilizada a

estratégia ϵ-Gulosa Decadente representada na Equação 9. A recompensa é igual a 1 no estado terminal de vitória e 0 em todos os outros estados. Ao todo foram realizados 10.000 episódios de treino.

Pelo fato de a política inicial ser igual a zero para todas as combinações entre estados e ações, e pela utilização da técnica de *exploration vs. exploitation* da Equação 10, a primeira iteração do algoritmo resulta em uma ação totalmente aleatória. Supondo que a ação aleatória seja mover o agente para a direita, a Equação 8, ficaria da seguinte maneira:

$$\begin{aligned} &\text{Novo } Q(s_t,a_t) = Q(s_t,a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1},a') - Q(s_t,a_t) \right] \\ &\text{Novo } Q(s_1,a_{direita}) = 0 + 0{,}1 \left[0 + 0{,}99 * 0 - 0 \right] \\ &\text{Novo } Q(s_1,a_{direita}) = 0 \end{aligned} \tag{11}$$

O resultado obtido para a primeira iteração do algoritmo se deve à forma como as recompensas estão distribuídas no ambiente. O fato de que a estratégia *living penalty*, vista anteriormente não foi utilizada e que a única recompensa diferente de zero está presente no estado de vitória, faz com que o agente comece a aprender apenas depois de atingir este estado pelo menos uma vez, e então comece a atualizar a qualidade de todos os estados anteriores. Após diversas iterações, a política do algoritmo começará a convergir para o resultado esperado.

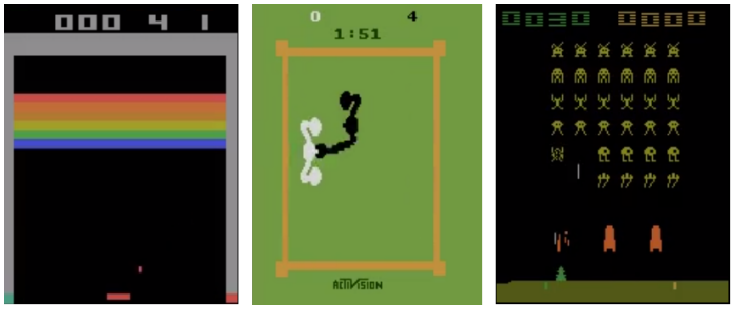
A saída do algoritmo após 10.000 iterações, é a qualidade estimada de todas as ações, representada na Tabela 1 em que as linhas representam os estados e as colunas representam as ações. Esta tabela define a política de comportamento do agente após o treinamento, pois a ação com maior estimativa de qualidade é a que será executada. Os valores 0 na tabela significam que aquele é um estado terminal ou que determinada ação nunca foi testada naquele estado. A Figura 7b é a representação da tabela na prática.

Tabela 1 – Tabela-Q para o jogo Lago Congelado.

	Esquerda	**Baixo**	**Direita**	**Cima**
Estado 01	0,405	0,577	0,504	0,473
Estado 02	0,079	0,165	0,160	0,450
Estado 03	0,375	0,183	0,189	0,389
Estado 04	0,015	0,000	0,000	0,021
Estado 05	0,427	0,594	0,438	0,456
Estado 06	0,000	0,000	0,000	0,000
Estado 07	0,181	0,137	0,316	0,093
Estado 08	0,000	0,000	0,000	0,000
Estado 09	0,435	0,322	0,460	0,314
Estado 10	0,336	0,753	0,422	0,506
Estado 11	0,657	0,481	0,321	0,358
Estado 12	0,000	0,000	0,000	0,000
Estado 13	0,000	0,000	0,000	0,000
Estado 14	0,531	0,639	0,861	0,531
Estado 15	0,687	0,940	0,765	0,715
Estado 16	0,000	0,000	0,000	0,000

O fato de que todas as combinações entre ação e estado precisam ser mapeadas é uma grande limitação para o algoritmo *Q-Learning*, pois em um problema com sensores no mundo real ou até mesmo em jogos um pouco mais complexos, os estados são contínuos e a discretização utilizando a Tabela-Q requer uma quantidade inviável de memória e processamento. Para se ter ideia, em jogos como os da Figura 8, cada possível combinação de *pixels* seria tratado como um estado diferente pelo algoritmo *Q-Learning*, resultando no problema conhecido como "A Maldição da Dimensionalidade".

Figura 8 – Jogos de Atari 2600



Fonte: Adaptado de (OPENAI, 2019).

Mesmo com as limitações citadas, o algoritmo *Q-Learning* obteve sucesso em diversas tarefas, como pode ser visto em Smart e Kaelbling (2002), Crites e Barto (1996) e Júnior (2009). Porém, a aprendizagem por reforço utilizando métodos tabulares pode ser utilizada apenas em problemas simples, o que resultou em uma significativa perda de popularidade e redução de pesquisas na área, dado que a maior parte dos problemas envolvem uma quantidade incontável de dados e estados.

A AR tornou-se popular de novo somente após os recentes progressos na área de aprendizagem profunda, que faz a utilização de redes neurais artificiais e da sua poderosa capacidade em encontrar aproximações de funções, além do conceito de aprendizagem de representação, que proporciona formas de substituir os antigos métodos tabulares (ARULKUMARAN *et al.*, 2017).

2.4 REDES NEURAIS ARTIFICIAS

Para entender como a utilização de aprendizagem profunda possibilitou o renascimento da aprendizagem por reforço, é necessário antes dar uma breve introdução sobre redes neurais artificias.

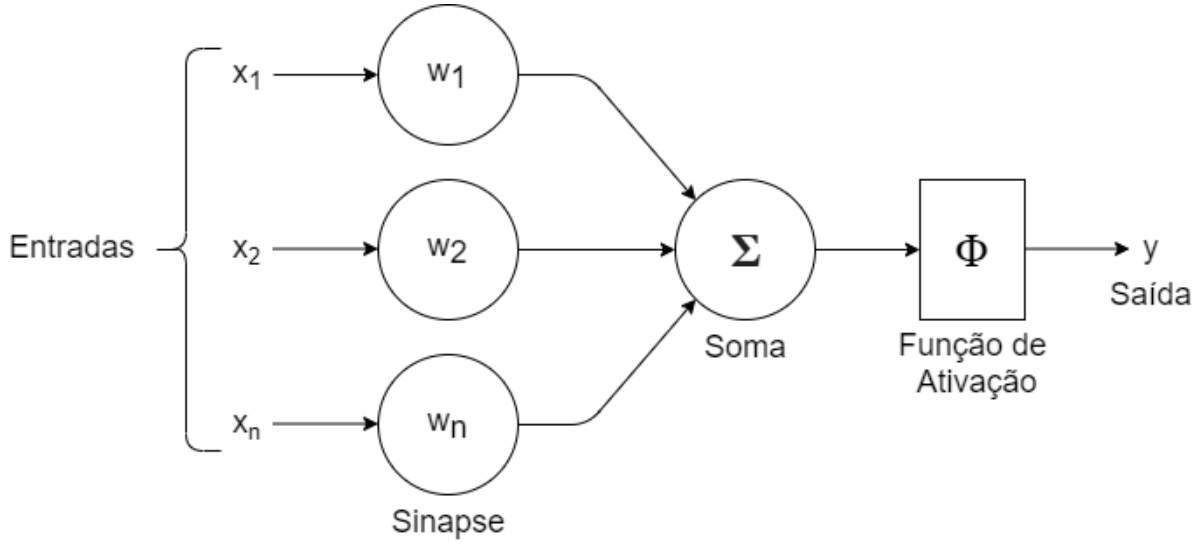
2.4.1 NEURÔNIO ARTIFICIAL

A partir da estrutura e funcionamento do neurônio biológico, pesquisadores tentaram reproduzir o sistema computacionalmente. O modelo que mais se destacou foi proposto por

McCulloch e Pitts (1943), que implementa de maneira simplificada os componentes e o funcionamento de um neurônio biológico (ACADEMY, 2019).

Neste modelo, os impulsos elétricos provenientes de outros neurônios são representados por sinais de entrada (x), o grau de excitação proporcionado por cada estímulo é representado por meio de pesos sinápticos (w). A junção das entradas é representada por um somatório de x_i ponderado pelos respectivos pesos, posteriormente, o resultado das somas passa por uma função de ativação que definirá a saída do neurônio (y). Assim como no modelo biológico, o estímulo pode ser excitatório ou inibitório, ou seja, positivo ou negativo (ACADEMY, 2019). A Figura 9 demonstra esse modelo.

Figura 9 – Modelo de um neurônio artificial



Fonte: Adaptado de (ACADEMY, 2019).

No modelo proposto por McCulloch e Pitts (1943), a função de ativação era sempre binária, o que resultava em uma saída binária do neurônio, porém, com o tempo foram criadas outras funções de ativação que variam de acordo com a aplicação do neurônio. Duas das funções mais utilizadas atualmente são a *Sigmoid* e a *ReLU*, representadas pelas equações 12 e 13, respectivamente.

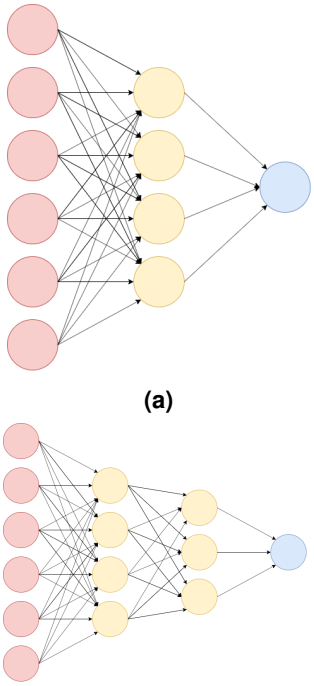
$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{12}$$

$$R(x) = \max\{0,x\} \tag{13}$$

2.4.2 REDES NEURAIS ARTIFICIAIS PROFUNDAS

Uma rede neural artificial é formada pela união de vários neurônios artificiais dispostos em diferentes camadas, como mostra a Figura 10a:

Figura 10 – Rede Neural Artificial Simples e Rede Neural Artificial Profunda



(a)

(b)

Fonte: Adaptado de (NIELSEN, 2015).

- Os neurônios representados em vermelho formam a **camada de entrada** da rede neural, esta camada recebe as informações do mundo externo à rede e apenas repassa os valores para a camada oculta.
- A **camada oculta**, que está representada em laranja, é responsável pelos cálculos e por transferir os dados da camada de entrada para a camada de saída. Ela recebe o nome de camada oculta pelo fato de não ter contato direto com o mundo externo.
- A camada em azul é chamada de **camada de saída**, ela é responsável por calcular a última função de ativação e transmitir as informações da rede neural para o mundo externo.

As redes neurais artificiais que possuem duas ou mais camadas ocultas, recebem o nome de Redes Neurais Artificiais Profundas, como mostra a Figura 10b. Esse tipo de rede neural deu origem à sub-área de aprendizagem de máquinas conhecida como Aprendizagem Profunda ou *Deep Learning* e teve grande impacto em várias áreas da inteligência artificial, possibilitando a resolução de tarefas como reconhecimento de objetos, reconhecimento de fala, e tradução de linguagens (LECUN; BENGIO; HINTON, 2015).

A forma como os neurônios são organizados na arquitetura de uma rede neural profunda definem o seu funcionamento e aplicações, em VEEN (2016) é possível verificar várias dessas arquiteturas.

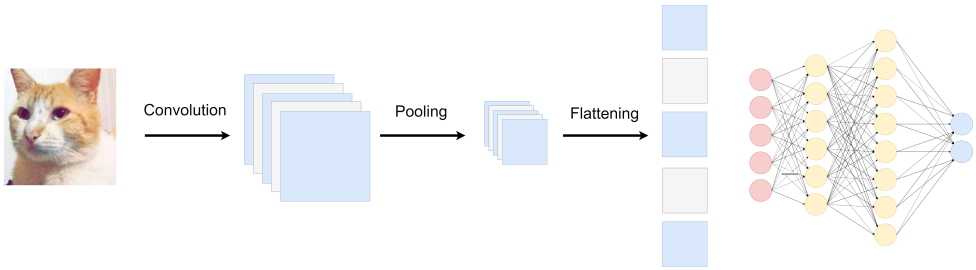
A propriedade mais importante das redes neurais profundas é que elas conseguem encontrar de maneira automática representações compactas de dados com muitas dimensões.

Essa área de estudo é chamada de aprendizagem de representação e tem como objetivo principal encontrar diferentes representações de dados com intuito de tornar mais fácil a extração de informações úteis. Uma boa representação de dados também pode ser considerada como aquela que é útil quando utilizada como entrada de um sistema de treinamento (BENGIO; COURVILLE; VINCENT, 2013).

2.4.3 REDES NEURAIS CONVOLUCIONAIS

As redes neurais convolucionais (*Convolutional Neural Networks* - CNN) são um tipo de rede neural especializada no processamento de dados com estrutura de grade, como por exemplo imagens, que são grades em 2-D de *pixels*. Essa arquitetura de redes neurais têm sido extremamente bem-sucedida em aplicações práticas e se tornou famosa por reduzir significativamente a dimensão dos seus dados de entrada e ainda manter as características importantes para o treinamento (GOODFELLOW; BENGIO; COURVILLE, 2016).

Figura 11 – Estrutura básica de uma Rede Neural Convolucional



Fonte: Adaptado de (GOODFELLOW; BENGIO; COURVILLE, 2016).

Na Figura 11 é possível notar algumas camadas peculiares das redes neurais convolucionais:

- **Camada Convolucional *(Convolution)***: Submete o dado de entrada (nesse caso a imagem de um gato) à vários filtros, conhecidos como *kernels*, cada um desses filtros possui uma função específica, como por exemplo, detecção de bordas ou alteração do brilho. Após filtrar os dados, é gerado um mapa de características da imagem (GOODFELLOW; BENGIO; COURVILLE, 2016).

- **Camada de *pooling***: A camada de *pooling* geralmente é utilizada após uma camada convolucional e tem como objetivo reduzir a dimensão da entrada para diminuir o custo computacional, além de ajudar a fazer com que a representação fique invariante à pequenas translações. Geralmente uma CNN possui várias camadas de convolução e de *pooling* (GOODFELLOW; BENGIO; COURVILLE, 2016).

- **Camada de *flattening***: Organiza os dados de forma que possam ser utilizados como entrada para a rede neural artificial da próxima camada.

- **Rede neural totalmente conectada**: Nesse tipo de rede neural, todos os neurônios de uma camada são conectados com todos os neurônios da camada subsequente. Ela é responsável por classificar os dados das camadas anteriores utilizando a função de ativação *Softmax* na camada de saída. No caso da imagem, a classificação deve indicar se a figura da entrada corresponde a um gato ou a um cachorro.

Na aprendizagem por reforço, as redes neurais convolucionais são utilizadas para extrair informações sobre o estado do ambiente a partir de capturas de imagens, e então representar essa informação de forma que seja possível utilizar como entrada para o algoritmo *Q-Learning*.

2.5 *DEEP Q-LEARNING*

A utilização de redes neurais convolucionais combinadas com o algoritmo *Q-Learning* foi proposta por Mnih *et al.* (2015), com intuito de criar um algoritmo genérico que possa ser aplicado na resolução de problemas mais complexos e mais próximos dos problemas do mundo real (MNIH *et al.*, 2015).

Nos problemas propostos em Mnih *et al.* (2015), a entrada do algoritmo é feita diretamente com imagens brutas em uma rede neural chamada *Deep Q-Network* ou DQN. Cada estado é composto por m imagens concatenadas ao longo do tempo. Elas são pré-processadas em escalas de cinza e então passam por diversas camadas convolucionais a fim de extrair informações espaço-temporais a respeito do estado atual. O conjunto das m imagens utilizadas como entrada para o algoritmo é denotado por ϕ.

A saída da última camada convolucional é então processada por várias camadas totalmente conectadas, responsáveis por codificar implicitamente o efeito das ações. A saída da rede neural é o valor estimado de *Q* para cada uma das possíveis ações em valores percentuais (ARULKUMARAN *et al.*, 2017). Na Figura 12a é possível observar uma possível entrada para o algoritmo, enquanto que na Figura 12b está representada a estrutura da rede neural.

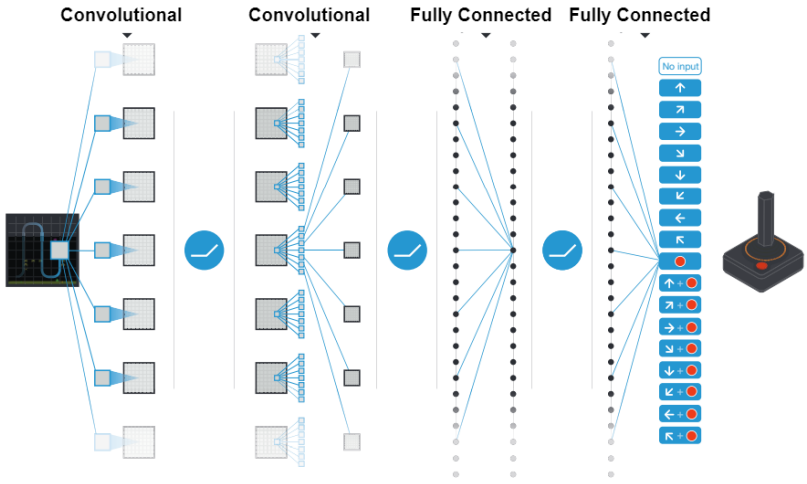
Além de permitir a resolução de problemas com uma quantidade infinita de estados, o algoritmo *Deep Q-Learning* diminuiu significativamente a ocorrência de dois problemas presentes no *Q-Learning*:

- Muitas vezes o *Q-Learning* superestima o valor das ações no cálculo de Q, gerando resultados irreais.
- Pequenas variações no valor de Q podem implicar em alterações drásticas na política resultante.

Para tratar esses problemas, foram utilizadas as estratégias *experience replay* e *target-network* no *Deep Q-Learning* (ARULKUMARAN *et al.*, 2017).

Figura 12 – Exemplo de entrada e da arquitetura da *Deep Q-Network*

(a)



(b)

Fonte: Adaptado de (MNIH *et al.*, 2015).

2.5.1 *EXPERIENCE REPLAY*

Experience replay é uma técnica inspirada na biologia do cérebro humano, ela já havia sido utilizada por Lin (1993). A ideia básica é reutilizar experiências passadas, fazendo com que o agente as vivencie repetidas vezes (LIN, 1993).

Durante o treinamento de uma rede neural, se um padrão de entradas não for utilizado por um certo período de tempo, a rede neural pode esquecer o que havia aprendido sobre aquele padrão e quando ele acontecer novamente, ela terá que reaprender tudo, portanto, o *experience replay* serve para lembrá-la de determinadas entradas, diminuindo o tempo de treinamento (LIN, 1993).

No algoritmo *Deep Q-Learning* a técnica de *experience replay* é utilizada salvando a experiência do agente em cada intervalo de tempo (MNIH *et al.*, 2015):

$$e_t = (s_t, a_t, r_t, s_{t+1}) \tag{14}$$

Essas experiências são salvas em uma base de dados $D_t = \{e_1,...,e_t\}$ reunida a partir de vários episódios e conhecida como *replay memory*. Durante a aprendizagem, o algoritmo *Q-Learning* é aplicado em algumas amostras das experiências salvas, escolhidas de maneira aleatória. Após realizar a *experience replay*, o agente executa uma ação de acordo com a política ϵ-Gulosa Decadente (MNIH *et al.*, 2013).

Dentre as vantagens proporcionadas pela utilização desse método, é possível destacar (MNIH *et al.*, 2015):

- Cada experiência é potencialmente utilizada em várias atualizações de pesos, aumentando a eficiência dos dados.
- No *Q-Learning* o treinamento é feito diretamente com amostras consecutivas, o que é ineficiente devido à alta correlação entre os dados. Escolher as amostras de maneira aleatória quebra essa correlação, reduzindo a variância das atualizações.
- A utilização de um valor médio entre várias amostras suaviza a aprendizagem, evitando mudanças bruscas na política final.

Na prática, o algoritmo armazena apenas as últimas N tuplas de experiência na *replay memory*, e então escolhe de maneira aleatória quais tuplas dentro de D serão utilizadas para realizar atualizações em Q (MNIH *et al.*, 2015).

2.5.2 *TARGET-NETWORK*

Outra grande diferença do *Deep Q-Learning* é a utilização de uma segunda rede neural para estimar o valor de $Q(s_{t+1},a_{t+1})$ durante a atualização do valor de $Q(s_t,a_t)$. De maneira direta, a cada C atualizações, a *Q-Network* é clonada para gerar a *Target-Network*, que é utilizada para o cálculo de Q objetivo durante as próximas C atualizações do Q atual. Isso torna o algoritmo mais estável quando comparado ao *Q-Learning* normal, em que a atualização de $Q(s_t,a_t)$ também atualiza $Q(s_{t+1},a_{t+1})$, podendo gerar oscilações na política (MNIH *et al.*, 2015).

Os cálculos de atualização de Q na iteração i utilizam a seguinte função de minimização de perda:

$$L_i(\theta_i) = E_{(s,a,r,s')\sim U(D)}\left[\left(r + \gamma \max_{a'} Q(s',a';\theta_i^-) - Q(s,a;\theta_i)\right)^2\right] \tag{15}$$

É possível notar que a Equação 15 é muito semelhante à Equação 8, as únicas diferenças são $E_{(s,a,r,s')\sim U(D)}$ que denotam as amostras sorteadas aleatoriamente a partir da *replay memory* e os parâmetros θ_i e θ_i^- que são os pesos da *Deep Q-Network* e da *Target-Network*, respectivamente.

Portanto, o *Deep Q-Learning* pode ser escrito como mostra o algoritmo 1.

Inicializar a *replay memory* D com capacidade N

Inicializar a *DQN* (Q) com pesos aleatórios θ

Inicializar a *Target-Network* $(\hat{Q})$ com pesos $\theta^{-} = \theta$

for *episódio = 1,*M **do**

  Inicializar a sequência $s_1 = \{x_1\}$ e a sequência pré-processada $\phi_1 = \phi(s_1)$

  for $t = 1,T$ **do**

    Com probabilidade ϵ, selecionar uma ação aleatória a_t

    Caso contrário, selecionar $a_t = \max_a Q(\phi(s_t),a;\theta)$

    Executar a ação a_t no simulador e adquirir a recompensa r_t e a imagem x_{t+1}

    Definir $s_{t+1} = s_t,a_t,x_{t+1}$ e pré-processar $\phi_{t+1} = \phi(s_{t+1})$

    Armazenar a transição $(\phi_t,a_t,r_t,\phi_{t+1})$ em D

    Sortear amostras de $(\phi_j,a_j,r_j,\phi_{j+1})$ aleatoriamente em D

    Definir $y_j = \begin{cases} r_j, & \text{se } j+1 \text{ for terminal} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1},a';\theta^{-}), & \text{caso contrário} \end{cases}$

    Fazer o cálculo de descida do gradiente em $(y_j - Q(\phi_j,a_j;\theta))^2$ com respeito aos parâmetros θ

    A cada C iterações definir $\hat{Q} = Q$

  end

end

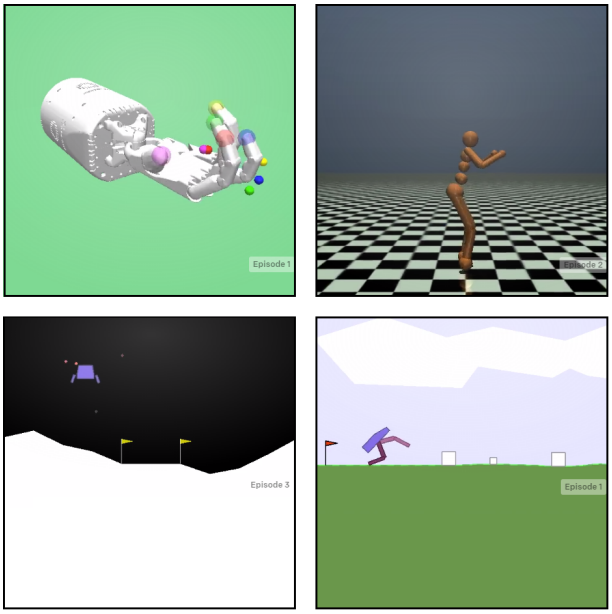
Algoritmo 1: *Deep Q-Learning*

2.6 PLATAFORMAS DE AVALIAÇÃO

A recente popularização da AR resultou em diversas pesquisas e inovações nos métodos de aprendizagem, porém, a falta de padronização nos ambientes utilizados em publicações, torna muito difícil a comparação de desempenho entre os algoritmos propostos. Para amenizar esse problema e proporcionar *benchmarks* de fácil acesso, como as que existem na aprendizagem supervisionada, foi criada a plataforma *Gym* (OPENAI, 2019), que oferece diversos ambientes com vários graus de dificuldade para teste e comparação de algoritmos focados em aprendizagem por reforço.

Dentre os ambientes presentes na plataforma *Gym*, os mais populares são os jogos clássicos de *Atari*, disponibilizados pela *Arcade Learning Environments* e os humanoides criados pela *MuJoCo*. Existem ainda ambientes que simulam braços robóticos, circuitos elétricos, carros autônomos, além dos problemas clássicos de controle, como o pêndulo invertido. Por conta da grande diversidade de problemas oferecidos, ela é atualmente a plataforma mais utilizada como referência no teste de algoritmos de AR. A Figura 13 mostra alguns desses ambientes.

Figura 13 – Ambientes disponíveis na plataforma *Gym*



Fonte: Retirado de (OPENAI, 2019).

Outra plataforma que está ganhando bastante popularidade é a *PyGame Learning Environment* ou PLE (PLE, 2016), que apesar de ainda não contar com muitos ambientes, apresenta alguns desafios interessantes e com interface semelhante aos ambientes da *Gym*.

3 MATERIAIS E MÉTODOS

Neste capítulo estão descritos os passos seguidos durante o pré-processamento e treinamento do algoritmo *Deep-Q Learning*, bem como as bibliotecas utilizadas para auxiliar nestes passos.

Os parâmetros do algoritmo *Deep Q-Learning* foram baseados em Mnih *et al.* (2015), por conta dos seus bons resultados obtidos na literatura.

Foram utilizados arquivos ROM (*Read Only Memory*) para emular os ambientes dos jogos da plataforma Atari 2600. Esses arquivos possuem as regras de um jogo específico, como ações, pontuação e objetivo. ROMs são arquivos binários que permitem a interação e treinamento do agente utilizando bibliotecas de aprendizagem por reforço, como a *Gym*.

Wrap-deepmind: permite configurar o ambiente de uma forma que facilite a utilização da rede neural convolucional, como por exemplo, realizar o pré-processamento das imagens que serão a entrada da CNN. (DHARIWAL *et al.*, 2017)

Keras Chollet *et al.* (2015) foi utilizado para criar a estrutura da rede neural;

Numpy Harris *et al.* (2020) e Matplotlib Hunter (2007) foram utilizados para trabalhar com operações matemáticas e gráficos;

Gym-wrapper: possui o método monitor, que permite criar vídeos do agente interagindo com o ambiente, facilitando na análise visual do treinamento. (BROCKMAN *et al.*, 2016a)

Google Colaboratory foi utilizado para o treinamento da rede neural por conta da disponibilidade de GPU.

A avaliação do agente se deu pela observação do quanto a taxa de recompensa aumenta conforme o número de episódios e *frames* aumentam. A recompensa do agente é calculada pela média simples das recompensas obtidas nos últimos 100 episódios.

3.1 CRIAÇÃO DO AMBIENTE E PRÉ-PROCESSAMENTO

Os ambientes são criados utilizando o método *make* da biblioteca *OpenAI Gym* Brockman *et al.* (2016b). Do ponto de vista de programação, os ambientes são classes que implementam os métodos de todas as interações entre o agente e o jogo.

Para retornar ao estado inicial de um ambiente, é possível chamar o método *reset*. O estado inicial de um ambiente *Atari* é composto por um tensor com dimensões $210x160x3$, ou seja, é uma imagem com 210 *pixels* de altura, 160 de largura e 3 canais de cores (RGB), para reduzir o custo computacional e, consequentemente, o tempo de treinamento, os estados são convertidos para imagens utilizando apenas o canal Y, responsável pela luminância, o que resulta em uma nova dimensão de $210x160x1$. Além disso, a altura e largura dos estados são reduzidas para 84 *pixels*. Todas essas transformações são feitas utilizando a biblioteca *wrap-deepmind*.

Outra utilidade importante da *wrap-deepmind* é o agrupamento de 4 *frames* seguidos, o que possibilita ao agente ter noções da dinâmica do ambiente, como velocidade e direção dos movimentos, algo que seria inviável utilizando apenas uma imagem isolada. Desse modo, cada estado de um ambiente passa a ser composto por 4 imagens e não apenas uma, resultando em uma dimensão de $84x84x4$, ou seja, a variável ϕ do algoritmo 1 possui um $m = 4$.

Cada ambiente possui um número específico de possíveis ações, que varia de acordo com as regras de cada jogo. No caso do jogo *Breakout*, as possíveis ações são: *Noop* (Não realizar ação), *Fire* (lançar a bola), *Left* (mover para a esquerda) e *Right* (mover para a direita).

A classe ”ambiente” também possui o método *”step”*, que recebe uma ação como entrada e retorna a recompensa recebida para aquela ação, além de uma variável chamada *done*, que é responsável por indicar se o agente está em um estado terminal. No caso do *Breakout*, um estado terminal é quando a bola toca o chão, resultando na perda de uma vida. Quando a variável *done* recebe o valor *True*, significa que é o fim de um episódio.

3.2 CRIAÇÃO DA *DEEP Q-NETWORK*

A DQN recebe como entrada um estado, que no caso de ambientes *Atari*, é composto por uma sequência de 4 *frames* concatenados. Por esse motivo, a arquitetura de rede neural que mais se adéqua ao problema é a rede neural convolucional, famosa por receber tensores como entrada e aplicar filtros a fim de extrair características importantes desses tensores.

A entrada da rede neural é um conjunto de tensores com dimensão $(84x84x4)x32$, em que 4 representa os quatro *frames* e 32 representa amostras sorteadas aleatoriamente do *replay buffer*.

A estrutura da rede neural é composta pelas seguintes camadas:

- **Camada de entrada:** Tensores com dimensão $(84x84x4)x32$
- **Camada 1**: Camada convolucional com 32 filtros, *Kernel* $8x8$, passo 4 e função de ativação *ReLU*.
- **Camada 2**: Camada convolucional com 64 filtros, *Kernel* $4x4$, passo 2 e função de ativação *ReLU*.
- **Camada 3**: Camada convolucional com 64 filtros, *Kernel* $3x3$, passo 1 e função de ativação *ReLU*.
- **Camada 4**: Camada de *flattening*.
- **Camada 5**: Camada completamente conectada com função de ativação *ReLU*.
- **Camada de Saída**: Camada completamente conectada, possui dimensão equivalente ao número de ações do ambiente e função de ativação linear.

O valor da camada de saída corresponde ao *Q-Value* (estimativa da recompensa recebida ao tomar determinada ação estando em um determinado estado).

Com objetivo de melhorar a estabilidade do modelo, são criadas duas redes neurais com a mesma arquitetura. A primeira é a DQN, que é responsável por calcular o *Q-Value* de cada ação. A segunda rede é a *Target Network*, utilizada para predizer recompensas futuras.

Os pesos da *Target Network* são atualizados com os valores da DQN somente a cada 10.000 passos, desse modo, enquanto o erro entre o *Q-Value* e o *Target Q-Value* é calculado, a *Target Network* fica estável, melhorando a convergência do modelo.

3.3 HYPERPARÂMETROS E TREINAMENTO

Os seguintes parâmetros foram ajustados para o treinamento da DQN nos ambientes propostos:

- **Gamma** (γ) foi definido com valor igual a 0,99, o que significa que as recompensas futuras terão tanta influência durante o treinamento quanto as recompensas imediatas. Um valor alto para *gamma* é recomendado quando o ambiente possui muitos estados.
- **Epsilon** (ϵ) com valor inicial igual a 1, pois é importante que o agente explore o ambiente no início do treinamento.
- O **Epsilon Decay** foi dividido em três intervalos, com funções de desconto diferentes em cada um dos intervalos. Cada intervalo corresponde a 1.000.000 frames.

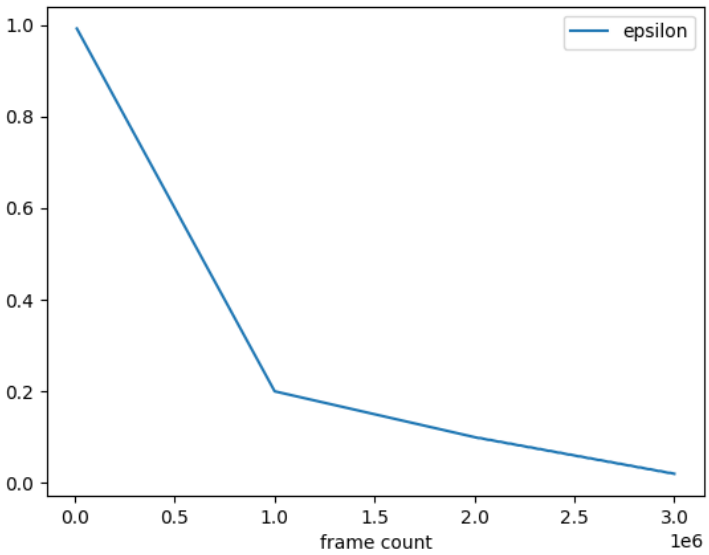
  O primeiro intervalo vai de $\epsilon = 1$ até $\epsilon = 0{,}2$, com decaimento de $0{,}8 * 10^{-6}$ para cada ação tomada.

  O segundo intervalo de decaimento vai de $\epsilon = 0{,}2$ até $\epsilon = 0{,}1$ com desconto de $0{,}1 * 10^{-6}$ por ação.

  O terceiro e último intervalo de decaimento é de $\epsilon = 0{,}1$ até $\epsilon = 0{,}02$ com desconto de $0{,}08 * 10^{-6}$ para cada ação tomada.

  Caso o treinamento ultrapasse $3.000.000$ *frames*, o valor de ϵ é fixado em $0{,}02$, para que mesmo em estágios avançados do treinamento, em que o agente está focado em explotar o ambiente, ele possa realizar algumas ações exploratórias. Os intervalos do decaimento de Epsilon são demonstrados na Figura 14.
- ***Epsilon Random Frames*** é uma variável criada para garantir que nos primeiros 50.000 frames, o agente tome ações aleatórias, explorando o ambiente e descobrindo novos estados.
- ***Max Memory Length*** é o tamanho máximo do *replay buffer*. No artigo original, é recomendada a utilização de $1.000.000$ amostras (MNIH *et al.*, 2015), porém, devido a limitações de *hardware*, foi utilizado um *replay buffer* com tamanho de 190.000.

Figura 14 – Função de decaimento de Epsilon



Fonte: Autoria própria..

- ***Batch Size*** é a quantidade de amostras que são utilizadas do *replay buffer* durante o treinamento. Foram utilizadas 32 amostras, que são escolhidas aleatoriamente toda vez que a DQN possui os seus pesos atualizados.
- ***Max Step per Episode*** é uma variável de segurança, que evita que o agente fique preso em um episódio por muito tempo. Foi definida como 10.000.
- ***Update After Actions*** indica quantas ações serão tomadas antes que a rede neural seja treinada novamente. Foi utilizado o valor 20.
- ***Update Target Network*** é responsável por definir quando a *Target Network* $(\hat{Q})$ será atualizada com os valores da DQN. Neste trabalho ela foi atualizada a cada 10.000 estados.
- **Learning rate** $\alpha = 0{,}00025$ e algoritmo de otimização Adam. Uma taxa de aprendizagem pequena aumenta as chances do modelo chegar em *Q-Values* satisfatórios, apesar de aumentar consideravelmente o tempo de treinamento necessário.

4 RESULTADOS E DISCUSSÕES

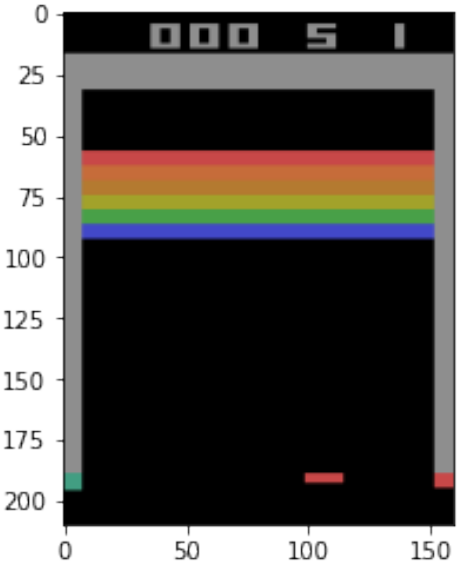
Neste capítulo estão listados os resultados do algoritmo *Deep Q-Learning* em cada um dos ambientes propostos, além de uma análise do desempenho e adaptação do agente aos ambientes.

4.1 AMBIENTE 1: *BREAKOUT*

Breakout é um jogo para *Arcade* desenvolvido pela *Atari*, e lançado em 13 de maio de 1976. Foi idealizado por Nolan Bushnell e Steve Bristow e influenciado pelo jogo de *arcade* de 1972 *pong*, também da *Atari*. O jogo foi convertido para os *consoles* e atualizado como *Super Breakout*. Além disso, *Breakout* foi a base e inspiração para livros, jogos, e o computador *Apple II*.

Em *Breakout* há uma camada de tijolos alinhada no topo da tela. A bola passa pela tela, rebatendo nas paredes laterais e superiores da tela. Quando um tijolo é atingido, a bola rebate de volta e o tijolo é destruído. O jogador perde uma vida quando a bola toca a parte inferior da tela. Para prevenir que isso aconteça, o jogador move uma palheta para rebater a bola para cima, mantendo-a no jogo. (CLASSICS, 2022)

Figura 15 – Imagem do ambiente *Breakout* antes do pré-processamento

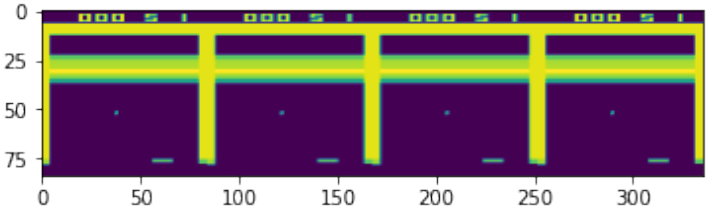


Fonte: Autoria própria.

A Figura 15 mostra como a imagem do ambiente é recebida a partir do arquivo *ROM*. É nesta imagem que os pré-processamentos precisam ser aplicados para que ela possa servir de entrada para a *Deep Q-Network*.

Seguindo os passos listados na seção 3.1, obtém-se a sequência de 4 *frames* redimensionados e concatenados, como mostra a Figura 16.

Figura 16 – Ambiente *Breakout* após pré-processamento

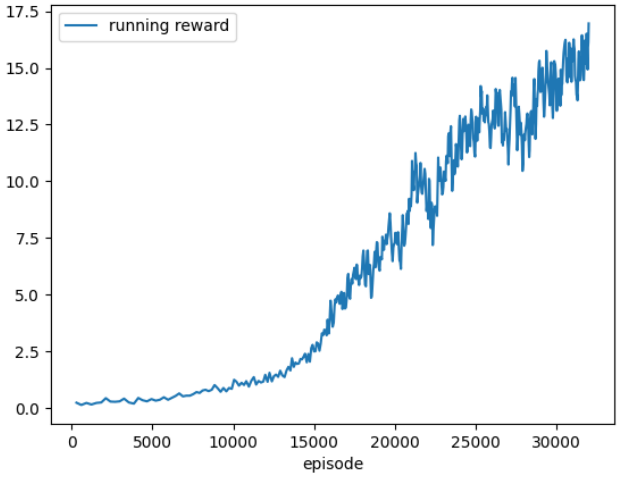


Fonte: Autoria própria.

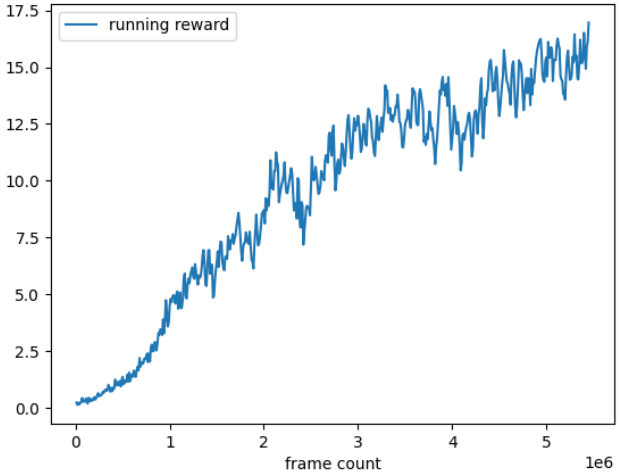
Com os *frames* de entrada pré-processados, a camada de entrada da *DQN* está pronta, agora basta definir a camada de saída para que a arquitetura esteja completa. No caso do *Breakout*, o agente pode realizar 4 possível ações: NOOP (não se mover), FIRE (lançar a bola), RIGHT (mover para direita) e LEFT (mover para a esquerda), logo, a camada de saída da rede neural terá tamanho 4.

Após criada a *Deep Q-Network*, é possível iniciar o treinamento utilizando os parâmetros propostos na seção 3.3.

Figura 17 – Recompensa média durante treinamento no ambiente *Breakout*.



(a)



(b)

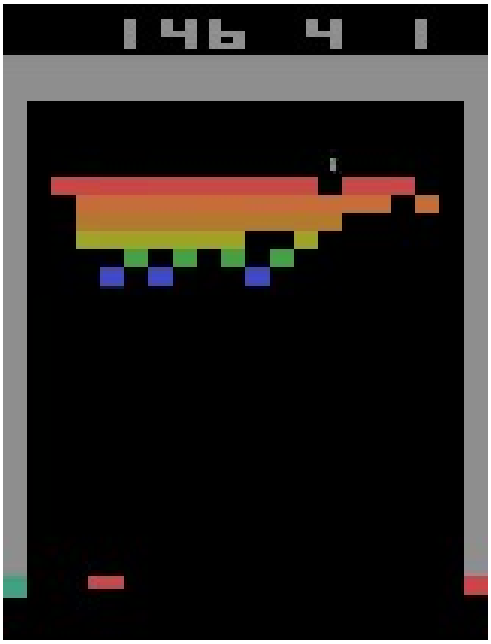
Fonte: Autoria própria.

Apesar de possuir poucas opções de movimento, a dinâmica do jogo *Breakout* é bastante punitiva, dado que os erros resultam na perda de uma vida, e consequentemente em um estado terminal. Outra característica do jogo *Breakout*, é que o agente tem contato com recompensas positivas desde o início do episódio, pois sempre que ele faz o primeiro lançamento da bola, ela se choca com pelo menos 1 tijolo, somando recompensas para o agente.

Essas particularidades do ambiente, fazem com que o agente comece a se adaptar às dinâmicas do jogo desde os primeiros episódios. Como é possível ver na Figura 17a, a curva de aprendizagem já começa a ganhar forma nos primeiros episódios, mesmo que lentamente.

É interessante ressaltar que em algumas das interações do agente com o ambiente após o treinamento, ele priorizou destruir os tijolos localizados nos cantos, para que fosse possível lançar a bola acima da barreira, e consequentemente, destruir vários tijolos com apenas um lan-

Figura 18 – *Frame* do agente durante uma interação com o ambiente *Breakout*.



Fonte: Autoria própria.

çamento, na Figura 18 é possível observar uma dessas interações. Em Mnih *et al.* (2015) essa estratégia é vista como um conjunto de ações que resultam em altos valores de recompensa.

4.2 AMBIENTE 2: *ENDURO*

Enduro é um jogo de jogo de corrida produzido pela *Activision* em 1983 para o *console Atari 2600*. Foi um dos últimos sucessos do *Atari* antes da crise na indústria eletrônica entre 1983 e 1984.

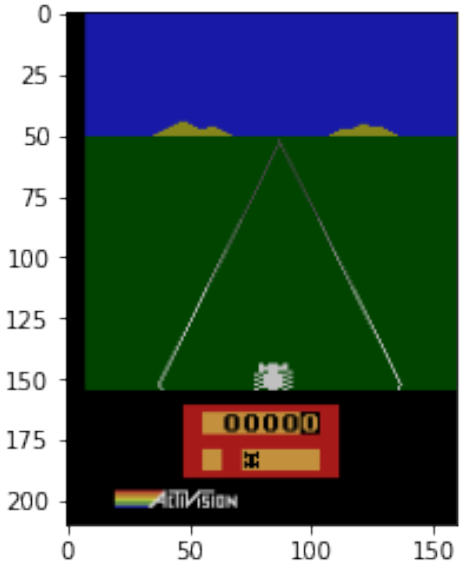
O jogo apresenta um ciclo diurno e noturno, criando um novo conceito nos jogos.

Em *Enduro* o jogador controla um veículo similar a um carro de Fórmula 1, e deve competir no *National Enduro*, uma corrida de resistência de longa distância, cujo percurso é aleatório. O objetivo da corrida é ultrapassar uma certa quantidade de carros a cada dia, para permitir ao jogador continuar correndo no dia seguinte. O jogador deve desviar de outros pilotos, ultrapassando 200 carros no primeiro dia, e 300 carros nos dias posteriores.

A medida que o jogador avança, a visibilidade também muda. À noite, o jogador só consegue ver as luzes traseiras dos carros. Conforme os dias passam, os carros se tornam mais difíceis de ultrapassar. O clima também influencia a jogabilidade, podendo resultar em trechos de gelo ou de neblina. (CLASSICS, 2022)

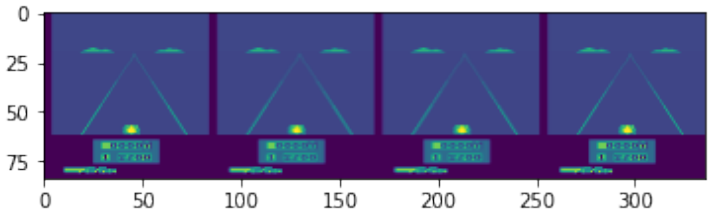
Na Figura 19 está representada a imagem do ambiente disponibilizada pela ROM, já a Figura 20 apresenta a entrada da DQN após o pré-processamento.

Figura 19 – Imagem do ambiente *Enduro* antes do pré-processamento



Fonte: Autoria própria.

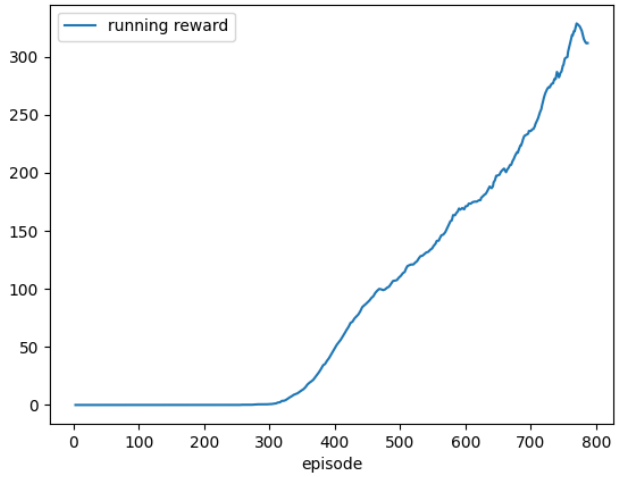
Figura 20 – Ambiente *Enduro* após pré-processamento



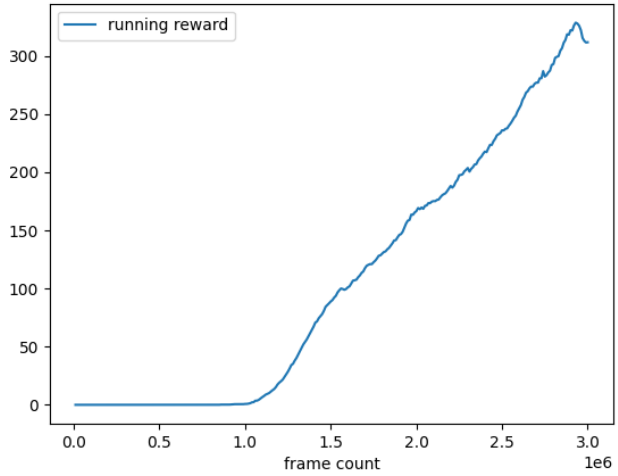
Fonte: Autoria própria.

O jogo *Enduro* possui 9 possíveis ações: NOOP (não se mover), FIRE (acelerar), RIGHT (mover para a direita), LEFT (mover para a esquerda), DOWN (mover para baixo), DOWNRIGHT (mover na diagonal para baixo e para a direita), DOWNLEFT (mover na diagonal para baixo e para a esquerda), RIGHTFIRE (mover para a direita enquanto acelera), LEFTFIRE (mover para a esquerda enquanto acelera). Seguindo a arquitetura proposta na seção 3.2, a rede neural terá uma camada de saída contendo 9 neurônios.

Figura 21 – Recompensa média durante treinamento no ambiente *Enduro*.



(a)



(b)

Fonte: Autoria própria.

No jogo *Enduro*, diferentemente do *Breakout*, o agente tem a possibilidade de realizar uma quantidade muito maior de ações, o que atrasa o seu aprendizado.

Outro fator do jogo *Enduro*, é que o agente só se depara com recompensas ao ultrapassar outros carros, logo, o agente só começa a efetivamente aprender, após um longo período de ações sem receber recompensas. Na Figura 21a é possível notar que o agente passa mais de 300 episódios sem receber recompensa alguma, porém, após receber as primeiras recompensas, a aprendizagem cresce rapidamente. Isso se deve ao fato de que algumas ações como ”Acelerar” são muito mais vantajosas do que as outras, e que provavelmente, o agente abusa dessas ações para maximizar as recompensas.

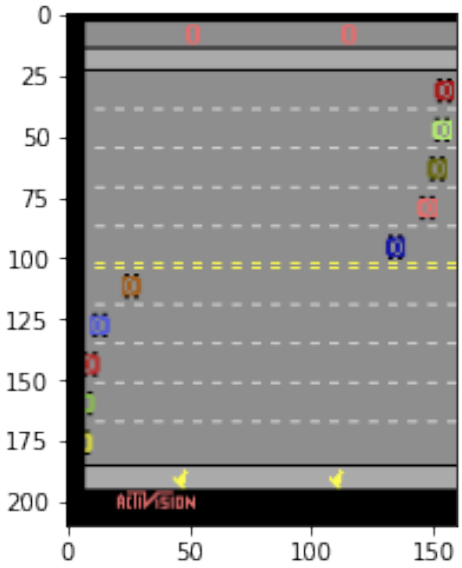
4.3 AMBIENTE 3: *FREEWAY*

Em *Freeway* , o jogador controlava uma galinha que deveria atravessar uma rodovia cheia de automóveis e de caminhões, com mais de seis pistas. Um jogo simples lançado para o *Atari* 2600 em 1981.

O jogo teve o nome de *Bloody Human Freeway* na sua fase de protótipo, nesta versão quando o jogador era atingido, seu *sprite* se tornava uma grande poça vermelha, mas como *video-game* era considerado um nicho juvenil, o jogo foi adaptado para esconder a violência.

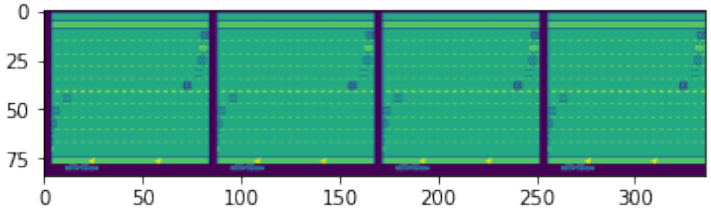
De acordo com David Crane, ele teve a ideia para a *Freeway* ao observar um homem tentando cruzar a *Lake Shore Drive* durante o tráfego da hora do *rush*, enquanto participava do *Consumer Electronic Show de Chicago*. (CLASSICS, 2022)

Figura 22 – Imagem do ambiente *Freeway* antes do pré-processamento



Fonte: Autoria própria.

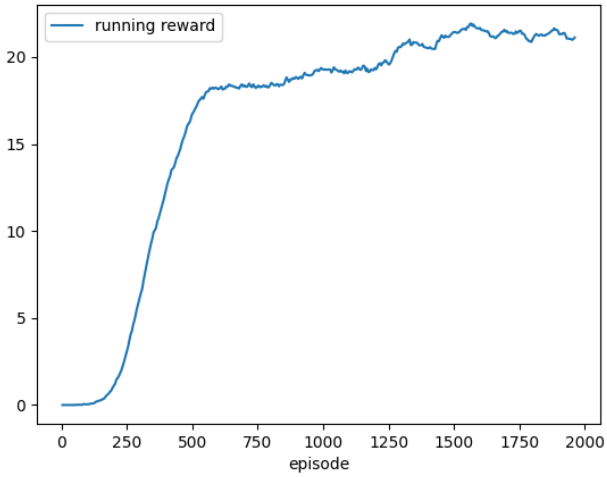
Figura 23 – Ambiente *Freeway* após pré-processamento



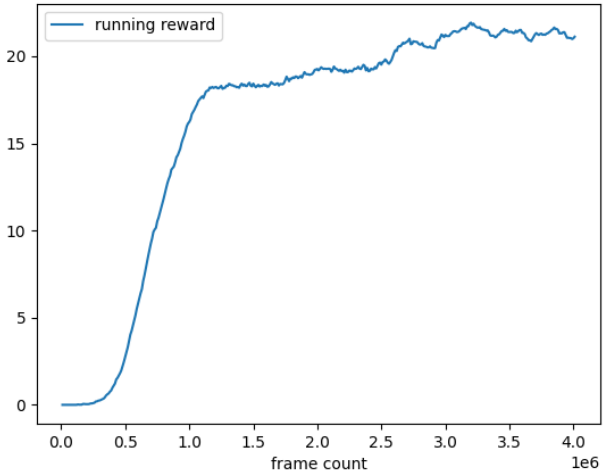
Fonte: Autoria própria.

O jogo *Freeway* possui apenas 3 possíveis ações: NOOP (não se mover), UP (mover para cima) e DOWN (mover para baixo), logo, a rede neural terá uma camada de saída de tamanho 3.

Figura 24 – Recompensa média durante treinamento no ambiente *Freeway*.



(a)



(b)

Fonte: Autoria própria.

Por se tratar de um jogo extremamente simples e com poucas ações, é possível notar que o agente se adapta rapidamente ao ambiente de *Freeway*, porém, também é possível observar que em poucos episódios, a aprendizagem fica estagnada. Isso provavelmente se deve ao fato de o agente priorizar a ação ”UP” em relação às outras ações. Dificilmente o agente opta por recuar, mesmo que isso acabe resultando em colisão com um veículo.

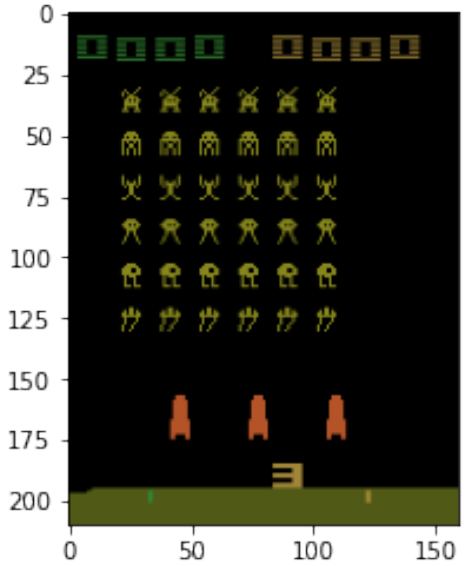
4.4 AMBIENTE 4: *SPACE INVADERS*

Space Invaders é um jogo de videogame de *arcade* desenhado por Tomohiro Nishikado e lançado em 1978. Foi originalmente construído pela *Taito Corporation* e um tempo depois foi

licenciado para produção nos Estados Unidos pela *Midway Games*. *Space Invaders* foi um dos primeiros jogos de tiro com gráfico bidimensional.

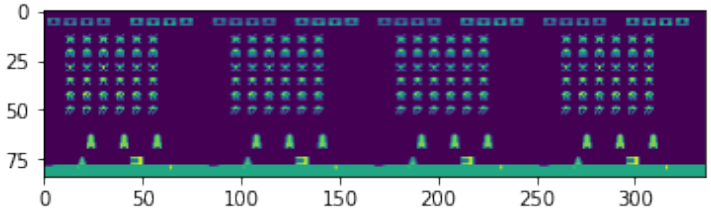
O objetivo é destruir ondas de naves com uma espaçonave humana para ganhar o maior número de pontos possível. Para construir o jogo, Nishikado se inspirou na mídia popular, como A Guerra dos Mundos e *Star Wars*. Apesar de seus controles simples comparados com os jogos de hoje, este jogo ajudou a expandir a indústria de video game para uma indústria mundial. (CLASSICS, 2022)

Figura 25 – Imagem do ambiente *Space Invaders* antes do pré-processamento



Fonte: Autoria própria.

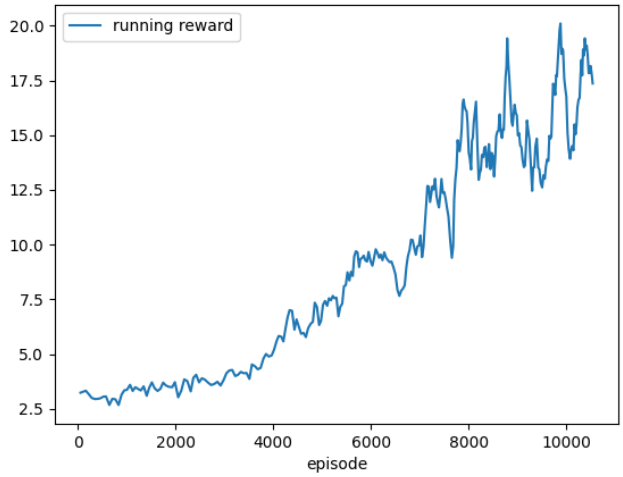
Figura 26 – Ambiente *Space Invaders* após pré-processamento



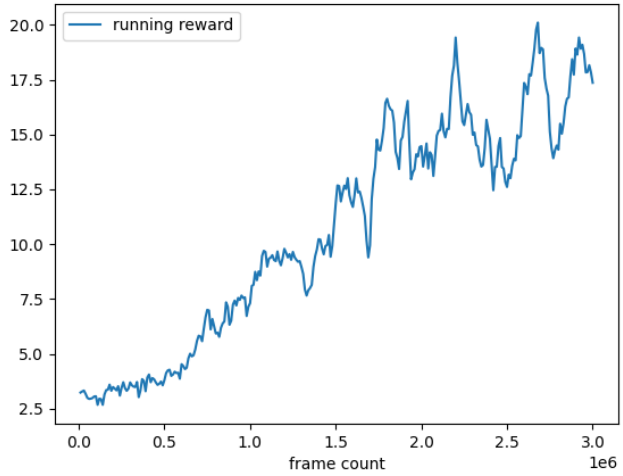
Fonte: Autoria própria.

O jogo *Space Invaders* possui 6 possíveis ações: NOOP (não se mover), FIRE (atirar), RIGHT (mover para a direita), LEFT (mover para a esquerda), RIGHTFIRE (mover para a direita e atirar) e LEFTFIRE (mover para a esquerda e atirar), o que resulta em uma rede neural com 6 neurônios na camada de saída.

Figura 27 – Recompensa média durante treinamento no ambiente *Space Invaders*.



(a)



(b)

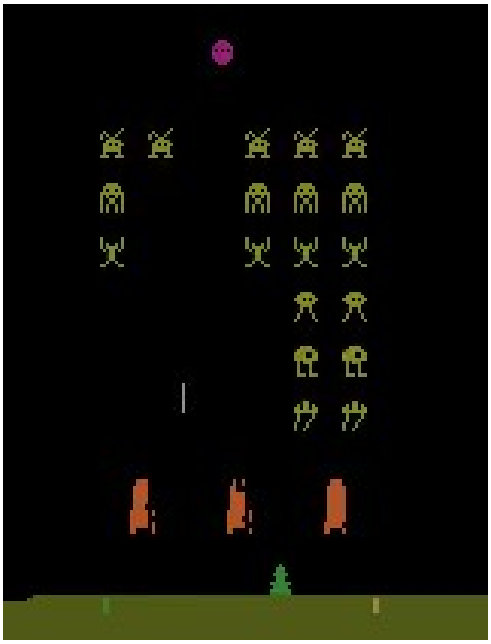
Fonte: Autoria própria.

A adaptação ao jogo *Space Invaders* não foi tão satisfatória com os hyperparâmetros propostos, principalmente por conta de uma dinâmica do ambiente, em que a ”Nave do comandante *Alien*” atravessa a parte superior do ambiente e precisa ser destruída o mais rápido possível, pois, enquanto ela está presente no mapa, as recompensas deixam de ser recebidas. Em nenhuma das interações do agente com o ambiente após o treinamento, ele priorizou destruir a nave. Na Figura 28 é mostrada uma dessas aparições.

4.5 AMBIENTE 5: *BEAM RIDER*

Beamrider é um jogo de nave e tiro espacial lançado pela *Activision* para o console *Atari 2600*, se passa acima da atmosfera da Terra, onde um grande escudo alienígena chamado de

Figura 28 – *Frame* do agente durante uma interação com o ambiente *Space Invaders*.

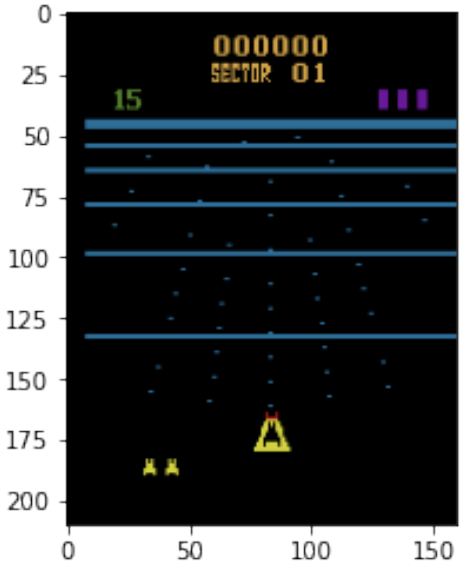


Fonte: Autoria própria.

Escudo Restritor rodeia a Terra. O objetivo do jogador é passar pelos 99 setores enfrentando as naves alienígenas enquanto pilota a nave *Beamrider* que é equipada com tiros infinitos de curto alcance e um suprimento limitado de torpedos. O jogador recebe três destes no início de cada setor.

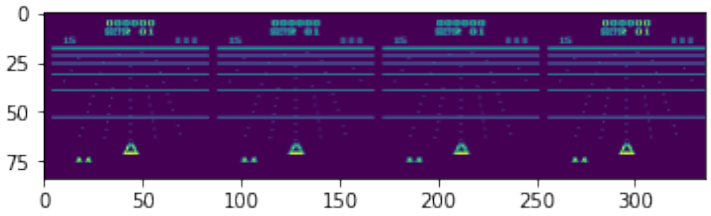
Beamrider possui ao todo 99 fases, para avançar para o próximo nível, o jogador deve destruir quinze naves inimigas que aparecem na cor branca. Uma ”Nave Sentinela” aparecerá então, que pode ser destruída usando um torpedo (se ainda restar algum), fazendo isso o jogador ganhará pontos de bônus. O inimigo tentará destruir a nave do jogador com tiros laser, bombas e vários tipos de naves, algumas só poderão ser destruídas com torpedos, mas o jogador pode simplesmente desviar destes ataques. Ocasionalmente, durante um setor, ”Rejuvenescedores Amarelos” (vidas extras) aparecem. Eles podem ser recolhidos para obter uma nave extra, mas se forem atingidos, se transformam em detritos e podem destruir a nave do jogador. (CLASSICS, 2022)

Figura 29 – Imagem do ambiente *Beam Rider* antes do pré-processamento



Fonte: Autoria própria.

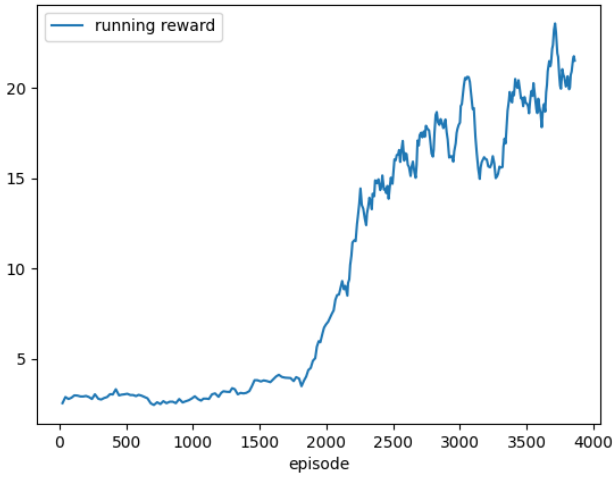
Figura 30 – Ambiente *Beam Rider* após pré-processamento



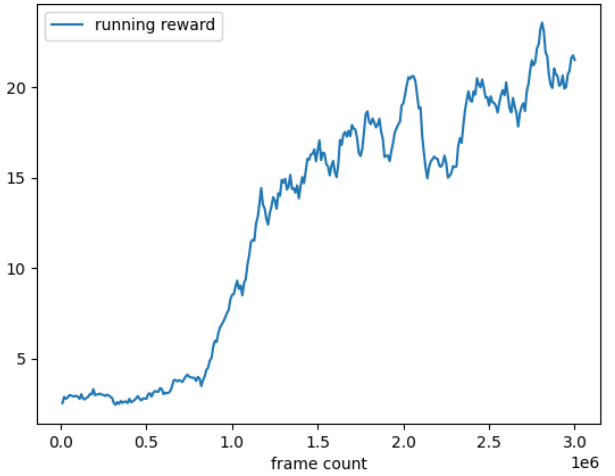
Fonte: Autoria própria.

No jogo *Beam Rider*, o agente pode realizar 9 ações: NOOP (não se mover), FIRE (atirar), UP (mover para cima), RIGHT (mover para direita), LEFT (mover para esquerda), UPRIGHT (mover na diagonal para cima e para a direita), UPLEFT (mover na diagonal para cima e para a esquerda), RIGHTFIRE (mover para direita e atirar), LEFTFIRE (mover para esquerda e atirar), desse modo, a *DQN* precisa de uma camada de saída com tamanho 9.

Figura 31 – Recompensa média durante treinamento no ambiente *Beam Rider*.



(a)

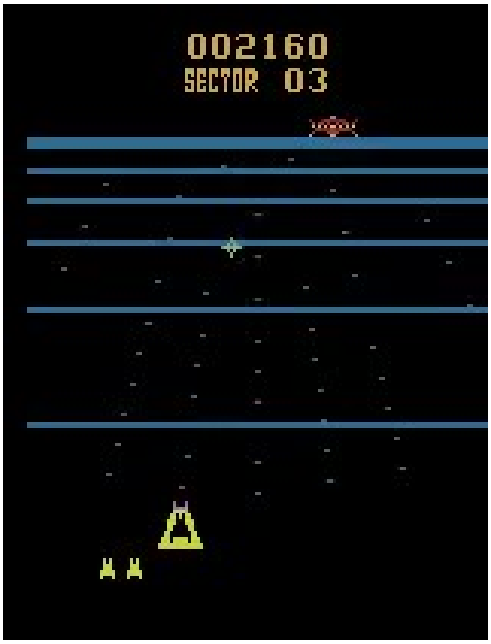


(b)

Fonte: Autoria própria.

Da mesma forma que nos outros ambientes com uma quantidade relativamente alta de ações, em *Beam Rider*, o agente demora vários episódios até começar a realmente aprender a dinâmica do ambiente e entender quais ações compensam mais. Na interação do agente com o ambiente após o treinamento, foi possível notar que ele aprendeu a desviar dos obstáculos e a destruir os inimigos, porém, o treinamento não foi suficiente para que o agente aprendesse a coletar as vidas que aparecem ao decorrer de cada fase e nem para que ele descobrisse como se destrói a Nave Sentinela, que aparece ao fim de cada fase e só pode ser destruída utilizando torpedos. Todas as vezes que o agente se deparou com a Nave Sentinela, ele já estava sem torpedos, como mostra a Figura 32.

Figura 32 – *Frame* do agente interagindo com o ambiente *Beam Rider*.



Fonte: Autoria própria.

4.6 DISCUSSÕES

Por limitações de tempo na plataforma *Google Colaboratory*, o agente foi treinado por um período entre 8 e 9 horas em todos os ambientes citados nas seções anteriores. Desse modo, é possível comparar as recompensas médias obtidas em cada um dos ambientes, levando em conta o número de ações disponíveis e a arquitetura de cada rede neural.

Tabela 2 – Comparação entre as redes neurais.

Ambiente	Nº de Ações	Parâmetros Treináveis	Recompensa Média
Breakout	4	1.686.180	17,0
Enduro	9	1.688.745	311,7
Freeway	3	1.685.667	21,1
Space Invaders	6	1.687.206	17,4
Beam Rider	9	1.688.745	21,5

Observando a Tabela 2, é possível notar que apesar de a quantidade de ações do ambiente ter um grande impacto nas primeiras épocas do treinamento, elas não são o principal fator para determinar a adaptação do agente ao ambiente, essa adaptação se deve muito mais à dificuldade do ambiente em si. Enquanto que o agente que interagiu com o ambiente *Enduro* possuía 9 ações e conseguiu atingir uma recompensa média de $311{,}7$, o agente do ambiente *Breakout* atingiu somente $17{,}0$ de recompensa média, mesmo possuindo apenas 4 ações.

Na Tabela 3 é possível comparar as recompensas de um testador de jogos humano, dos agentes treinados em Mnih *et al.* (2015) e neste trabalho, respectivamente. Com exceção do jogo *Enduro*, que ficou acima do jogador humano, os outros agentes obtiveram recompensas abaixo deste limiar, isso se deve, provavelmente às limitações de tempo e de *hardware* da plataforma *Google Colaboratory*, pois, se levarmos em consideração os gráficos de recompensas

Tabela 3 – Análise de desempenho de um testador de jogos profissional, da DQN de Mnih *et al.* (2015) e dos agentes treinados neste trabalho.

Ambiente	Humano	DQN (MNIH *et al.*, 2015)	DQN Treinada
Breakout	31,8	401,2	17,0
Enduro	309,6	301,8	311,7
Freeway	29,6	30,3	21,1
Space Invaders	1652,0	1976,0	17,4
Beam Rider	5775,0	6846,0	21,5

mostrados nas seções anteriores, todos os agentes, exceto o do ambiente *Freeway*, estavam com tendência de alta nos últimos episódios do treinamento, o que significa que se o treinamento se estendesse por mais episódios, o agente continuaria a aprender e a se aperfeiçoar nestes ambientes.

5 CONCLUSÃO

O agente se adaptou aos ambientes de acordo com a dinâmica de cada um deles. Foi possível notar as diferentes curvas de aprendizagem em cada ambiente, isso se deve aos diferentes espaços de ações e as variações na forma como a recompensa é adquirida em cada ambiente.

No ambiente *Breakout* foi interessante observar que o agente buscou quebrar os tijolos das extremidades e lançar a bola para cima, aumentando as recompensas obtidas. Tal estratégia já havia sido debatida em Mnih *et al.* (2015).

Nos ambientes *Enduro* e *Freeway* em que a velocidade é um fator crucial, o agente priorizou ações relacionadas a se movimentar para a frente. Em ambos os casos, essas ações nem sempre são as melhores, porém, elas sempre garantem recompensas, o que fez com que o agente às utilizasse mesmo quando outras ações eram visivelmente melhores para evitar colisões.

Da mesma forma, em ambientes de batalha (*Space Invaders* e *Beam Rider*), o agente priorizou ações relacionadas a atirar. Nesses casos, essas ações realmente são muito mais vantajosas, dado que a todo momento é necessário destruir obstáculos ou inimigos. Em ambos os ambientes, porém, foi possível notar que o agente não aprendeu como lidar com as situações que acontecem com frequência baixa, como destruir a Nave do Comandante *Alien* em *Space Invaders* e utilizar torpedos para destruir a Nave Sentinela em *Beam Rider*.

O comportamento do agente tanto nos ambientes de corrida quanto nos de batalha sugerem que ele aprendeu a dinâmica básica dos jogos, porém, o treinamento não foi suficiente para que ele aprendesse as especificidades de cada um, o que possibilitaria alcançar recompensas ainda maiores, superando jogadores humanos como foi demonstrado em Mnih *et al.* (2015). Isso se deve, principalmente a necessidade de treinamentos extremamente longos e a utilização de *hardwares* potentes, que possam proporcionar esses treinamentos.

REFERÊNCIAS

ACADEMY, D. S. **Deep Learning Book**. 2019. Disponível em: "http://deeplearningbook.com.br/".

ARULKUMARAN, K. *et al.* A brief survey of deep reinforcement learning. **arXiv preprint arXiv:1708.05866**, 2017.

ATRASH, A. *et al.* Development and validation of a robust speech interface for improved human-robot interaction. **International Journal of Social Robotics**, Springer, v. 1, n. 4, p. 345, 2009.

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

BENGIO, Y.; COURVILLE, A.; VINCENT, P. Representation learning: A review and new perspectives. **IEEE transactions on pattern analysis and machine intelligence**, IEEE, v. 35, n. 8, p. 1798–1828, 2013.

BERTSEKAS, D. P.; TSITSIKLIS, J. N. **Neuro-dynamic programming**. [*S.l.*]: Athena Scientific Belmont, MA, 1996. v. 5.

BROCKMAN, G. *et al.* **OpenAI Gym**. 2016.

BROCKMAN, G. *et al.* Openai gym. **arXiv preprint arXiv:1606.01540**, 2016.

CHOLLET, F. *et al.* **Keras**. 2015. https://keras.io.

CLASSICS, A. **Atari Classics**. 2022. https://atariclassics.com.br/atari2600. [Online; accessed 07-May-2022]. Disponível em: https://atariclassics.com.br/atari2600.

COPELAND, B. J. **The essential turing**. [*S.l.*]: Clarendon Press, 2004.

CRITES, R. H.; BARTO, A. G. Improving elevator performance using reinforcement learning. *In*: **Advances in neural information processing systems**. [*S.l.*: *s.n.*], 1996. p. 1017–1023.

DHARIWAL, P. *et al.* **OpenAI Baselines**. [*S.l.*]: GitHub, 2017. https://github.com/openai/baselines.

DUAN, Y. *et al.* Benchmarking deep reinforcement learning for continuous control. *In*: **International Conference on Machine Learning**. [*S.l.*: *s.n.*], 2016. p. 1329–1338.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep Learning**. [*S.l.*]: MIT Press, 2016. http://www.deeplearningbook.org.

GRANATYR, J.; PONTEVES, H.; EREMENKO, K. **Aprendizagem por Reforço com Deep Learning, PyTorch e Python**. 2019. Disponível em: https://www.udemy.com/course/aprendizagem-reforco-deep-learning-pytorch-python/.

HARRIS, C. R. *et al.* Array programming with NumPy. **Nature**, Springer Science and Business Media LLC, v. 585, n. 7825, p. 357–362, set. 2020. Disponível em: https://doi.org/10.1038/s41586-020-2649-2.

HUNTER, J. D. Matplotlib: A 2d graphics environment. **Computing in Science & Engineering**, IEEE COMPUTER SOC, v. 9, n. 3, p. 90–95, 2007.

JIANG, N. On value functions and the agent-environment boundary. **arXiv preprint arXiv:1905.13341**, 2019.

JÚNIOR, F. C. d. L. **Algoritmo Q-learning como estratégia de exploração e/ou explotação para metaheurísticas GRASP e algoritmo genético**. 7 2009. Tese (Doutorado) — Universidade Federal do Rio Grande do Norte, 7 2009. Disponível em: https://repositorio.ufrn.br/jspui/handle/123456789/15129.

KAELBLING, L. P.; LITTMAN, M. L.; MOORE, A. W. Reinforcement learning: A survey. **Journal of artificial intelligence research**, v. 4, p. 237–285, 1996.

KOHONEN, T. The self-organizing map. **Proceedings of the IEEE**, IEEE, v. 78, n. 9, p. 1464–1480, 1990.

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. Imagenet classification with deep convolutional neural networks. *In*: **Advances in neural information processing systems**. [*S.l.*: *s.n.*], 2012. p. 1097–1105.

LECUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. **nature**, Nature Publishing Group, v. 521, n. 7553, p. 436–444, 2015.

LI, Y. Deep reinforcement learning: An overview. **arXiv preprint arXiv:1701.07274**, 2017.

LILLICRAP, T. P. *et al.* Continuous control with deep reinforcement learning. **arXiv preprint arXiv:1509.02971**, 2015.

LIN, L.-J. **Reinforcement learning for robots using neural networks**. [*S.l.*], 1993.

MCCULLOCH, W. S.; PITTS, W. A logical calculus of the ideas immanent in nervous activity. **The bulletin of mathematical biophysics**, Springer, v. 5, n. 4, p. 115–133, 1943.

MNIH, V. *et al.* Playing atari with deep reinforcement learning. **arXiv preprint arXiv:1312.5602**, 2013.

MNIH, V. *et al.* Human-level control through deep reinforcement learning. **Nature**, Nature Publishing Group, v. 518, n. 7540, p. 529, 2015.

NIELSEN, M. A. **Neural networks and deep learning**. [*S.l.*: *s.n.*], 2015. v. 25.

OPENAI. **OpenAI Environments**. 2019. Disponível em: https://gym.openai.com/envs/.

OPPERMANN, A. **Self Learning AI-Agents Part II: Deep Q-Learning**. 2018. Disponível em: https://towardsdatascience.com/self-learning-ai-agents-part-ii-deep-q-learning-b5ac60c3f47.

PELLEGRINI, J.; WAINER, J. Processos de decisão de markov: um tutorial. **Revista de Informática Teórica e Aplicada**, v. 14, n. 2, p. 133–179, 2007.

PLE. **PygGame Learning Environment**. 2016. Disponível em: https://pygame-learning-environment.readthedocs.io/en/latest/index.html.

PUTERMAN, M. L. **Markov Decision Processes.: Discrete Stochastic Dynamic Programming**. [*S.l.*]: John Wiley & Sons, 2014.

RENDE, S.; DONDURAN, M. Neighborhoods in development: human development index and self-organizing maps. **Social indicators research**, Springer, v. 110, n. 2, p. 721–734, 2013.

RIEDMILLER, M. *et al.* Reinforcement learning for robot soccer. **Autonomous Robots**, Springer, v. 27, n. 1, p. 55–73, 2009.

SHANI, G.; PINEAU, J.; KAPLOW, R. A survey of point-based pomdp solvers. **Autonomous Agents and Multi-Agent Systems**, Springer, v. 27, n. 1, p. 1–51, 2013.

SMART, W. D.; KAELBLING, L. P. Effective reinforcement learning for mobile robots. *In*: IEEE. **Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No. 02CH37292)**. [*S.l.*], 2002. v. 4, p. 3404–3410.

SUTTON, R. S.; BARTO, A. G. **Reinforcement learning: An introduction**. [*S.l.*]: MIT press, 2018.

THORNDIKE, E. Provisional laws of acquired behavior or learning. **Animal Intelligence (New York: The Mc Millian Company)**, 1911.

VEEN, F. V. **The Neural Network Zoo**. 2016. Disponível em: https://www.asimovinstitute.org/neural-network-zoo/.

WATKINS, C. J.; DAYAN, P. Q-learning. **Machine learning**, Springer, v. 8, n. 3-4, p. 279–292, 1992.

WATKINS, C. J. C. H. **Learning from Delayed Rewards**. 5 1989. Tese (Doutorado) — King's College, Cambridge, UK, 5 1989. Disponível em: http://www.cs.rhul.ac.uk/~chrisw/new_thesis.pdf.

ZHU, X. J. **Semi-supervised learning literature survey**. [*S.l.*], 2005.