Ministério da Educação
Universidade Federal de Ouro Preto
Escola de Minas
Departamento de Engenharia de Produção, Administração e Economia

Thales Alexandre Martins Lopes da Costa.

Implementação de métodos de aprendizagem por reforço para a resolução do problema CartPole

Ouro Preto
2022

Thales Alexandre Martins Lopes da Costa.

Implementação de métodos de aprendizagem por reforço para a resolução do problema CartPole

Monografia apresentada ao Curso de Engenharia de Produção da Universidade Federal de Ouro Preto como parte dos requisitos para a obtenção do Grau de Engenheiro de Produção.

Universidade Federal de Ouro Preto

Orientador: Prof. Me. Davi das Chagas Neves

Ouro Preto
2022

SISBIN - SISTEMA DE BIBLIOTECAS E INFORMAÇÃO

C837i Costa, Thales Alexandre Martins Lopes Da.
Implementação de métodos de aprendizagem por reforço para a resolução do problema CartPole. [manuscrito] / Thales Alexandre Martins Lopes Da Costa. - 2022.
36 f.: il.: color., gráf., tab..

Orientador: Prof. Me. Davi das Chagas Neves.
Monografia (Bacharelado). Universidade Federal de Ouro Preto. Escola de Minas. Graduação em Engenharia de Produção .

1. Inteligência artificial - CartPole. 2. Aprendizado do computador - Aprendizagem por reforço. 3. Aprendizado do computador. I. das Chagas Neves, Davi. II. Universidade Federal de Ouro Preto. III. Título.

CDU 658.5

Bibliotecário(a) Responsável: Maristela Sanches Lima Mesquita - CRB-1716

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE OURO PRETO
REITORIA
ESCOLA DE MINAS
DEPARTAMENTO DE ENGENHARIA DE PRODUCAO, ADMINISTRACAO E ECON

FOLHA DE APROVAÇÃO

Thales Alexandre Martins Lopes da Costa

Implementação de métodos de aprendizagem por reforço para a resolução do problema CartPole

Monografia apresentada ao Curso de Engenharia de Produção da Universidade Federal de Ouro Preto como requisito parcial para obtenção do título de Engenheiro de Produção

Aprovada em 26 de Outubro de 2022

Membros da banca

Me. Davi das Chagas Neves - Orientador - Universidade Federal de Ouro Preto
Dra. Irce Fernandes Gomes Guimarães - Universidade Federal de Ouro Preto
Dr. Jorge Luiz Brescia Murta - Universidade Federal de Ouro Preto

O professor Davi das Chagas Neves, orientador do trabalho, aprovou a versão final e autorizou seu depósito na Biblioteca Digital de Trabalhos de Conclusão de Curso da UFOP em 26/10/2022

Documento assinado eletronicamente por **Davi das Chagas Neves**, **PROFESSOR DE MAGISTERIO SUPERIOR**, em 08/11/2022, às 15:44, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site http://sei.ufop.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0 , informando o código verificador **0424066** e o código CRC **2865836B**.

R. Diogo de Vasconcelos, 122, - Bairro Pilar Ouro Preto/MG, CEP 35402-163
Telefone: 3135591540 - www.ufop.br

Agradecimentos

Aos meus pais por me apoiarem durante toda a minha graduação e que sempre se dedicaram para que eu e meu irmão tivéssemos uma vida mais fácil do que tiveram.

Ao meu orientador Prof. Me. Davi das Chagas Neves, por ter me dado suporte neste desafio e se dedicado para que este trabalho fosse bem sucedido.

E a todos que contribuíram de forma direta ou indiretamente para a minha graduação e realização deste trabalho.

"Digo que me encontro no conhecimento de uma única ciência: a do amor."
Sócrates

Resumo

Neste trabalho procura-se demonstrar o conceito de aprendizagem por reforço e como ele, aliado a algoritmos de *Machine Learning*, pode ser utilizado para resolver o problema do pêndulo invertido. De tal forma, para alcançar este objetivo, serão implementados quatro algoritmos: *Q-Learning*, Escalada, *REINFORCE* e *Deep Q Network*. Estes algoritmos serão desenvolvidos na linguagem *Python* e seus resultados demonstrados. Observou-se que todos os algoritmos foram capazes de solucionar o problema do pêndulo - alguns sendo mais eficientes que os outros. Os algoritmos se provaram extremamente úteis para a resolução de problemas complexos e que podem ser aproveitados em diversas áres de pesquisa.

Palavras-chave: aprendizagem por reforço, pêndulo invertido, controle de sistemas dinâmicos, aprendizado de máquinas

Abstract

This work aims to demonstrate the concept of reinforcement learning and how it, with the help of machine learning algorithms, can be used to solve the inverted pendulum problem. In this way, in order to reach this goal, four algorithms will be implemented: Q-Learning, Hill Climbing, REINFORCE and Deep Q Network. These algorithms will be developed in Python and its results demonstrated. It was observed that all of the algorithms were capable of solving the pendulum problem - with some being more efficient than others. These algorithms have proved to be extremely useful in solving complex problems that may be utilized within many fields of study.

Keywords: reinforcement learning, inverted pendulum, dynamic systems control, machine learning

Lista de abreviaturas e siglas

DQN	*Deep Q Network* (Rede Neural Q)
IA	Inteligência Artificial
PDM	Processos de Decisão de Markov

Lista de ilustrações

Figura 1 – Processo de interação do agente com o ambiente. 14
Figura 2 – Pêndulo simples 15
Figura 3 – Pêndulo invertido 17
Figura 4 – Ilustração do método de escalada. 23
Figura 5 – Redes neurais do tipo Q. 25
Figura 6 – Redes neurais do tipo Q utilizando ações e estados. 29
Figura 7 – Gráfico dos resultados gerado pelo programa. 32
Figura 8 – Gráfico das recompensas totais em cada episódio. 34
Figura 9 – Gráfico das recompensas totais em cada episódio. 35
Figura 10 – Gráfico das recompensas e média ao longo dos episódios. 36

Lista de tabelas

Tabela 1 – Critérios de finalização do ambiente 22
Tabela 2 – Resultados obtidos com o algoritmo *Q-learning* 31
Tabela 3 – Resultados obtidos com o algoritmo de escalada 33
Tabela 4 – Informações mostradas durante a execução do algoritmo 35

Sumário

	Lista de ilustrações	10
	Lista de tabelas	11
1	INTRODUÇÃO	13
2	REFERENCIAL TEÓRICO	15
2.1	Sistemas de Pêndulo	15
2.1.1	Pêndulo Simples	15
2.1.2	Pêndulo Invertido	17
2.1.3	A Caoticidade de Sistemas Dinâmicos	19
2.2	Aprendizagem por Reforço	19
2.2.1	Processos de Decisão de Markov	19
2.2.2	Biblioteca OpenGYM	20
2.2.3	Sistemas de Recompensa	21
2.2.4	O problema CartPole	21
2.3	Algoritmos da Aprendizagem por Reforço	22
2.3.1	Q-Learning	22
2.3.2	Escalada (HillClimbing)	23
2.3.3	REINFORCE	24
2.3.4	Deep Q Network (DQN)	25
3	IMPLEMENTAÇÃO DOS ALGORITMOS	26
3.1	Q-learning	26
3.2	Escalada (Hill Climbing)	27
3.3	REINFORCE	28
3.4	Deep Q Network (DQN)	29
4	RESULTADOS	31
4.1	Q-Learning	31
4.2	Escalada (Hill Climbing)	33
4.3	REINFORCE	35
4.4	DEEP Q NETWORK (DQN)	36
5	CONCLUSÕES E CONSIDERAÇÕES FINAIS	37
	REFERÊNCIAS	38

1 Introdução

Com o avanço da ciência e constante desenvolvimento de novas tecnologias, surgem novos problemas e desafios. Sistemas sofisticados de vôo, carros autônomos e sistemas robóticos de manufatura são alguns exemplos que possuem diversos problemas a serem resolvidos. Para Harmon e Harmon (1997), muitos destes problemas atualmente não possuem solução, não porque os computadores são lentos ou tem pouca memória, mas simplesmente porque é difícil determinar o que um programa deve fazer para atingir uma solução.

Nos últimos anos diversos avanços foram realizados na área de inteligência artificial (IA), onde métodos de aprendizado utilizando redes neurais, também conhecidos como *deep learning*, tem servido como ferramentas revolucionárias da resolução de problemas complexos. Pesquisas recentes demonstram que as técnicas de *deep learning* são extremamente úteis para resolver problemas com grandes volumes de informação e de forma rápida e eficaz.

Existem algumas abordagens para a aprendizagem de máquinas (machine learning), como o aprendizado surpevisionado e o não supervisionado. Glorennec (2000) define a aprendizagem supervisionada como um sistema ciente dos erros que comete a todo o tempo: para cada entrada *(input*), a saída *(output)* correspondente desejada é conhecida. Caso não hajam sinais de saída, o aprendizado será do tipo não supervisionado, resultando em distribuições estocásticas determinadas pelos dados de entrada.

A aprendizagem por reforço é uma área da aprendizagem de máquinas *(machine learning)* que consiste em um agente, as ações performadas por este agente, o ambiente no qual o agente está inserido e nas observações (estado atual e recompensa) retornadas por cada ação. O objetivo do agente na aprendizagem por reforço é maximizar as recompensas – os valores de recompensa retornados podem ser positivos ou negativos/nulos dependendo se a ação realizada gerou um estado que se aproxime, ou distancie, do estado objetivo.

A aprendizagem por reforço acontece através das experiências do agente e suas ações, sem que existam dados de input e output como é o caso do aprendizado supervisionado.

Figura 1 – Processo de interação do agente com o ambiente.

Fonte: Adaptada de MIT 6.S191 *Reinforcement Learning.*

Para disseminar e desenvolver os métodos e técnicas existentes de aprendizagem por reforço, inúmeros projetos de código aberto foram disponibilizados online para que fossem utilizados, testados e aperfeiçoados por qualquer pessoa e também servissem como uma ferramenta de aprendizado para o usuário. Um destes projetos é o *OpenAI*, uma instituição sem fins lucrativos com o objetivo garantir que a utilização da IA beneficie toda a humanidade.

O presente trabalho tem como objetivo comparar o desempenho de diferentes algoritmos de aprendizagem por reforço através do ambiente *CartPole-v0* aplicado ao problema do pêndulo invertido. Para a realização do trabalho e alcance dos objetivos, diferentes referências bibliográficas acerca do assunto serão analisadas a fim de estabelecer uma base teórica, compreendendo o problema em questão e identificar o melhor caminho para sua solução.

O método de resolução se dará através da implementação de diferentes algoritmos programados na linguagem *Python*, integrando a biblioteca *Gym* e o ambiente *CartPole-v0* onde parâmetros pré-estabelecidos vão definir este ambiente. Em seguida, os algoritmos escolhidos serão executados e os relatórios e gráficos gerados por eles analisados e comparados.

Este trabalho está está organizado em cinco capítulos. O Capítulo 2 traz a fundamentação teórica por trás do sistema de pêndulo e da aprendizagem por reforço. O Capítulo 3 apresentará a implementação de diferentes algoritmos de aprendizagem por reforço. No Capítulo 4 serão apresentados os resultados obtidos através da execução dos programas. Por fim, no Capítulo 5, apresenta-se as conclusões sobre o trabalho.

2 Referencial teórico

Neste capítulo serão introduzidos os principais conceitos e fundamentos utilizados ao longo do trabalho. Na primeira seção serão apresentados o problema dos pêndulos simples e duplo e também o conceito de caoticidade nestes sistemas.

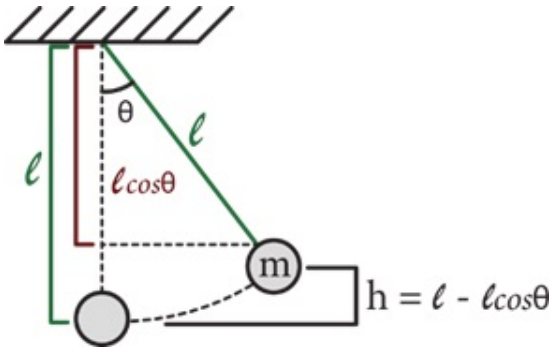
2.1 Sistemas de Pêndulo

2.1.1 Pêndulo Simples

O pêndulo simples ideal consiste em uma partícula suspensa por um fio inextensível e de massa desprezível, ilustrado na Figura 2 , onde quando deslocado de sua posição de equilíbrio, tende a oscilar ao longo de um plano vertical sob a ação da gravidade, sendo este movimento descrito como periódico e oscilatório.

Na ausência de forças de amortecimento – negligenciando o atrito no ponto pivô e a resistência do ar – um pêndulo simples ideal, quando deslocado do seu ponto de equilíbrio, oscila infinitamente com uma amplitude constante.

Figura 2 – Pêndulo simples



Fonte: autoria própria

De acordo com a segunda lei de Newton, que diz que a força é igual a massa vezes a aceleração ($F = m \cdot a$), colocada em sua forma angular ($\alpha = \frac{a}{l} \therefore a = \alpha \cdot l$) a equação de movimento do pêndulo simples é descrita da seguinte forma:

$$ml\frac{\partial^2\theta}{\partial t^2} = -mgsen\theta \tag{2.1}$$

onde θ é o deslocamento angular do pêndulo a partir do seu ponto de equilíbrio vertical e g a aceleração da gravidade. Assumindo que a amplitude de oscilação do pêndulo é pequena o suficiente e que $sen\theta \approx \theta$, a equação (2.1) pode ser reescrita da seguinte forma:

$$\frac{\partial^2\theta}{\partial t^2} + \frac{g}{l}\theta = 0 \tag{2.2}$$

A solução para a equação do movimento um oscilador harmônico simples é:

$$\theta = \theta_{max} \cdot sen\sqrt{\frac{g}{l}} \tag{2.3}$$

que é análoga à equação de movimento de um oscilador massa-mola

$$y = Asen\sqrt{\frac{k}{m}}t \tag{2.4}$$

De forma similar ao oscilador massa-mola, obtemos, respectivamente, a frequência angular, o período e a frequência do pêndulo:

$$\omega = \sqrt{\frac{g}{l}} \tag{2.5}$$

$$T = 2\pi\sqrt{\frac{l}{g}} \tag{2.6}$$

$$f = \frac{1}{2\pi}\sqrt{\frac{g}{l}} \tag{2.7}$$

Como não existem forças de amortecimento agindo sobre o sistema, a energia é conservada durante toda a trajetória, ou seja, a soma das forças cinética e potencial em qualquer tempo é constante. À medida que a uma energia aumenta, a outra diminui e assim se mantém este equilíbrio.

Um outro método bastante usado, e mais simples, para o cálculo das equações de movimento de sistemas como o do pêndulo é o método de Lagrange. Nele os termos da energia cinética e potencial são utilizados para deduzir as equações de movimento.

O método de Lagrange consiste na resolução da equação diferencial (2.8):

$$\frac{\partial}{\partial t}(\frac{\partial L}{\partial \dot{x}}) - \frac{\partial L}{\partial x} = 0 \tag{2.8}$$

A energia cinética é dada por:

$$E = \frac{1}{2}I\omega^2, \tag{2.9}$$

onde I é o momento de inércia de um ponto de massa e ω é a velocidade angular do movimento.

A energia potencial é dada por:

$$PE = mgL = mg(l - lcos\theta) = mgl(1 - cos\theta), \tag{2.10}$$

onde h é o comprimento da haste menos a distância vertical do pivô até o ponto de massa, como indicado na Figura 2.

Obtendo a Lagrangeana:

$$L = E_c - E_p$$

$$L = \frac{1}{2}I\dot{\theta}^2 - mgl(1 - cos\theta)$$

$$L = \frac{1}{2} I\dot{\theta}^2 - mgl - mglcos\theta \tag{2.11}$$

Obtendo os termos da equação diferencial utilizando a equação Lagrangeana obtida acima:

$$\partial L = -mglsen\theta \tag{2.12}$$

$$\frac{\partial L}{\partial \dot{\theta}} = I\dot{\theta} = ml^2\dot{\theta}, I = ml^2 \text{ (Momento de inércia de um ponto de massa)} \tag{2.13}$$

$$\frac{\partial}{\partial t}(\frac{\partial L}{\partial \dot{\theta}}) = ml\ddot{\theta} \tag{2.14}$$

Resolvendo a equação diferencial:

$$\frac{\partial}{\partial t}(\frac{\partial L}{\partial \dot{\theta}}) - \frac{\partial L}{\partial \theta} = 0$$
$$ml^2\ddot{\theta} + (mglsen\theta) = 0$$
$$l\ddot{\theta} + gsen\theta = 0$$

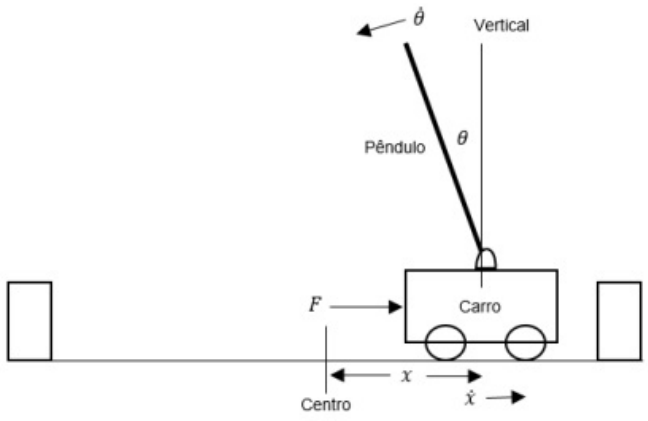
$$\ddot{\theta} = \frac{g}{l} sen\theta \text{ (Eq. de movimento)} \tag{2.15}$$

2.1.2 Pêndulo Invertido

De acordo com Hehn e D'Andrea (1997), o problema pêndulo invertido (Figura 3) é um problema clássico de controle, oferencendo um dos sistemas instáveis não-lineares mais intuitivos, facilmente descritos e realizáveis. Tem sido investigado por diversas décadas e é frequentemente utilizando para demonstrações de avanços teóricos, como a aprendizagem por reforço e redes neurais.

O modelo do pêndulo invertido é um sistema bastante conhecido e utilizado em diversas aplicações na área da engenharia, robótica e automação. O sistema pode ser visto em veículos, guindastes, braços robóticos, entre outros.

Figura 3 – Pêndulo invertido



Fonte: Nagendra et al. (2017)

Para entender e controlar um sistema complexo como o pêndulo invertido, é preciso que se conheça as variáveis e mecanismos que envolvem o sistema para que um modelo matématico que descreva seu movimento seja criado. Barto, Sutton e Anderson (1983) definem estas variáveis:

- x - posição do pêndulo;
- θ - ângulo do bastão com uma linha vertical imaginária;
- $\dot{x}$ - velocidade do carrinho;
- $\dot{\theta}$ - taxa de variação do ângulo;

A equação do movimento de rotação da haste do pêndulo em torno do centro de gravidade, considerando $sen\theta \approx \theta$, $cos\theta \approx 1$ é:

$$I\frac{\partial^2}{\partial t^2}\theta = F_V l\theta - F_H l \tag{2.16}$$

onde F_V é a força vertical e F_H a força horizontal, l é o comprimento do bastão e I seu momento de inércia.

F_H e F_V são dados por:

$$F_H = m\frac{\partial^2}{\partial t^2}(x + l\theta) \tag{2.17}$$

$$F_V = mg + m\frac{\partial^2}{\partial t^2}l \tag{2.18}$$

Substituindo $\frac{\partial^2\theta}{\partial t^2}$ por $\ddot{\theta}$ e $\frac{\partial^2 x}{\partial t^2}$ por $\ddot{x}$, tem-se que:

$$I\ddot{\theta} = F_V l\theta - F_H l$$

$$F_H = m(\ddot{x} + l\ddot{\theta}) \tag{2.19}$$

$$F_V = mg \tag{2.20}$$

$$M\ddot{x} = F - m(\ddot{x} + l\ddot{\theta}) \tag{2.21}$$

E da equação (2.21) obtém-se a equação de movimento em função da variável de posição:

$$\ddot{x} = \frac{F - ml\ddot{\theta}}{M + m} \tag{2.22}$$

Considerando $I = \frac{ml^2}{2}$, obtém-se a segunda equação de movimento em função do ângulo entre a haste e a linha imaginária:

$$\ddot{\theta} = \frac{g\theta - \frac{F}{M+m}}{l(\frac{4}{3} - \frac{m}{M+m})} \tag{2.23}$$

As equações (2.22) e (2.23) estão integradas no ambiente *CartPole-v0* e são utilizadas para o cálculo em tempo real do estado em que a gaste se encontra em um momento t.

2.1.3 A Caoticidade de Sistemas Dinâmicos

Rasband (1990) define os sistemas não-lineares como sistemas onde os valores medidos das propriedades de um sistema em um estado futuro depende, de maneira complicada, nos valores medidos em estados anteriores. A natureza imprevisível e irregular de sistemas deste tipo é também chamada de caos.

Um sistema caótico é um sistema determinístico, regido por leis físicas bem definidas, aperiódico e extremamente dependente das condições iniciais, sendo impossível prever seu comportamento a longo prazo sem elas. Sistemas meteorológicos, por exemplo, podem ser considerados caóticos, uma vez que uma pequena variação nas condições iniciais pode ter um enorme impacto em condições futuras por todo o planeta.

De acordo com Rasband (1990), dinâmica caótica se refere a uma evolução determinística com um resultado caótico, ou seja, de momento a momento o sistema evolui de maneira determinística – o estado atual do sistema depende do estado anterior de forma rigidamente determinada.

Para o caso dos pêndulos, o caos é observado apenas quanto o sistema envolve grandes quantidades de energia. Se o sistema é pequeno, onde é possível utilizar a aproximação de pequenos ângulos, este não é caótico, mas se a quantidade de energia nesse sistema for aumentada além de um certo ponto, a complexidade, ou o caos, do seu movimento se torna maior.

2.2 Aprendizagem por Reforço

Na estrutura da aprendizagem por reforço, um agente interage com um ambiente desconhecido e tenta maximizar seus lucros ao longo do tempo, o executados. O caso mais entendido e estudado é quando o ambiente é um Processo de Decisão de Markov com o critério de recompensa descontada total esperada. (SZITA; SZEPESVÁRI, 2010)

2.2.1 Processos de Decisão de Markov

Processos de Decisão de Markov (PDM) são uma classe de processos estocásticos sequenciais de decisão onde as funções de custo e transição dependem apenas da ação e estado atuais. Este modelo tem sido aplicado em diversas áreas, especialmente em enfileiramentos e controles de inventário. Um processo de decisão sequencial é um modelo para sistemas dinâmicos sob o controle de um tomador de decisões. (PUTERMAN, 1990)

Para entender os PDM precisa-se conhecer a Propriedade de Markov: o futuro é independente do passado dado o presente e pode ser matematicamente escrito como:

$$P[S_{t+1}|S_t] = P[S_{t+1}|S_1, ..., S_t] \tag{2.24}$$

onde S_t é o estado atual do agente e S_{t+1} o estado seguinte. Se o lado direito da equação é igual ao lado esquerdo, então o sistema possui a Propriedade de Markov. A

probabilidade de transição para o estado sucessor em um é dada por:

$$P_{ss'} = P[S_{t+1} = s'|S_t = s] \tag{2.25}$$

Na aprendizagem por reforço, uma PDM é definida por quatro variáveis $S, a, P(...)$ e $R(...)$ onde S é um conjunto de estados, A um conjunto de ações, $P(S_{t+1}|S_t, a_t)$ é a função de transição de estado do ambiente e $R(S_t, a_t, S_{t+1})$ a função de recompensa do ambiente. (GRAESSER; KENG, 2020)

Outra importante informação é que os agentes não tem acesso às funções de transição e recompensa. A única maneira do agente conseguir informações sobre estas funções é através da experiência e dos valores retornados pela função.

2.2.2 Biblioteca OpenGYM

Dentro do *OpenAI* existe a biblioteca *Gym*. De acordo com Brockman et al. (2016), criadores da *Gym*, a biblioteca foca na configuração episódica da aprendizagem por reforço, onde as experiências do agente são divididas em episódios. Em cada episódio um estado inicial aleatório é gerado através de uma distribuição de probabilidade e as interações do agente ocorrem até que o programa retorne um sinal para encerrar o episódio, quando atinge seu estado objetivo.

A parte fundamental do *OpenAI Gym* é a classe *env*. A classe basicamente implementa um simulador que roda o ambiente no qual o usuário deseja treinar um agente. A biblioteca possui diversos ambientes, como o do pêndulo invertido utilizado neste trabalho, e também permite que o usuário crie seu próprio ambiente.

Na biblioteca existem funções da classe env que auxiliam o agente a interagir com o ambiente. São elas:

- *reset*: Esta função reseta o ambiente para seu estado inicial e retorna a observação correspondente a ele;
- *step*: Esta função recebe uma açãoi como input e a aplica ao ambiente, levando à transição para um novo estado;

A função *reset* retorna quatro informações chave:

- *observation*:A observação do estado do ambiente;
- *reward*: A recompense recebida pela ação executada pela função *step*;
- *done*: Retorna *true* ou *false*. Se o episódio está finalizado o valor retornado é *true*;
- *info*: Fornece informações adicionais que podem ser utilizadas para *debugging*;

2.2.3 Sistemas de Recompensa

Recompensas são valores númericos que o agente recebe através da execução de uma ação a em um estado S do ambiente. Estes valores podem ser positivos ou negativos/nulos e dependem das ações do agente.

Na aprendizagem por reforço o objetivo é maximizar as recompensas totais recebidas (a soma de todas as recompensas imediatas recebida em cada estado) ao longo de *timesteps* – a duração dos episódios não é feita por tempo, e sim pela quantidade de passos executados – e pode ser escrita da seguinte forma:

$$G_t = r_{t+1} + r_{t+2} + ... + r_t \tag{2.26}$$

onde r_{t+1} é a recompensa recebida pelo agente no $timestept[0]$. De forma similar, r_{t+2} é a recompensa recebida no *timestep* $t[1]$ e r_t a recompensa recebita no último *timestep*.

Esta soma de recompensas funciona apenas para tarefas episódicas, pois se a tarefa for contínua a soma das recompensas tenderia ao infinito. Para isso existe o fator de desconto, γ, que o grau de importância que é dado à recompensa imediata e às futuras. O valor deste fator de desconto varia de 0 a 1. Caso este valor seja 0, significa que mais importância é dada à recompensa imediata e caso seja 1, mais importância é dada às recompensas futuras.

A equação para as recompensas descontadas é:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{+\infty} \gamma^k R_{t+k+1} \tag{2.27}$$

2.2.4 O problema CartPole

Um dos ambientes mais conhecidos da $OpenAIGym$ é o $CartPole-v0$. Este ambiente corresponde ao problema sugerido por Barto, Sutton e Anderson (1983) que envolve o equilíbrio de um bastão preso por uma de suas extremidades, a um carrinho limitado a se mover apenas para a direita ou esquerda ao longo de um trilho sem atrito. O objetivo é equilibrar o bastão aplicando forças na direção esquerda ou direita de forma sucessiva.

Existem duas possíveis ações que o a gente pode realizar: mover o carrinho para esquerda e movê-lo para direita. Estas duas ações tem números correspondentes a elas, sendo 0 o movimento para esquerda de 1 para direita.

O espaço de observação tem seus valores máximos definidos de acordo com a Tabela 1:

Tabela 1 – Critérios de finalização do ambiente

NUM	OBSERVAÇÃO	MÍNIMO	MÁXIMO
0	Posição do carrinho	-4,8	4,8
1	Velocidade do carrinho	$-\infty$	∞
2	Ângulo do bastão	-0,418 rad (-24°)	0,418 rad (24°)
3	Velocidade angular do bastão	$-\infty$	∞

Fonte: Notas do algoritmo *CartPole*

É importante ressaltar enquanto a tabela denota os possíveis valores para o espaço de observação de cada elemento, ela não reflete os valores permitidos para o espaço de estado em um episódio que ainda não foi finalizado.

A posição do carrinho no eixo X pode assumir valores entre -4,8 e 4,8, porém o episódio é terminado se este valor exceder -2,4 e 2.4. O mesmo acontece para o valor do ângulo do bastão, que pode assumir entre -24° e 24°, mas o episódio é terminado caso o valor do ângulo for menor que -12° ou maior que 12°.

O agente é recompensado um valor de "+1" para cada passo enquanto um ou mais os critérios de finalizam não forem atingidos.

Em suma, um episódio termina se pelo menos um dos seguintes casos ocorrer:

- O ângulo do bastão é menor que -12° ou maior que 12°;
- A posição do bastão excede -2,4 e 2.4, sendo 0 o ponto localizado na metade do trilho;
- O comprimento de um episódio atingir 500 *timesteps*;
- A soma das recompensas recebidas totaliza 200;

2.3 Algoritmos da Aprendizagem por Reforço

2.3.1 Q-Learning

O algoritmo *Q-Learning* é um método para problemas descontados, onde não existem modelos explícitos do sistema. De acordo com Bertsekas (2019), o método é relacionado à iteração de valores e pode ser usado diretamente no caso de múltiplas políticas. Em vez de aproximar a função custo de uma política específica, o algoritmo atualiza os parâmetros Q ($Q-factors$) associados a uma política ótima, evitando, dessa forma, os vários passos de avaliação de políticas durante a iteração. Watkins e Dayan (1992) descrevem a experiência do agente como uma sequência de distintos estágios, também chamado de episódios. No n-ésimo episódio, o agente:

- Observa o estado atual X_n;

- Seleciona e executa uma ação n_a;
- Observa o estado subsequente y_n;
- Recebe uma recompensa imediata r_n;
- Ajusta seus valores Q_{n-1} utilizando um fator de aprendizado α_n;

O agente aprende as políticas que determinam as melhores ações a serem tomadas com base em experiências anteriores e as recompensas recebidas. A cada episódio a $Q - table$ é atualizada com um novo valor Q, definida pela equação (2.28)

$$Q^{new}(S_t, a_t) = (1 - \alpha) \cdot Q(S_t, a_t) + \alpha \cdot (r_t + \gamma \cdot maxQ(S_{t+1}, a)) \tag{2.28}$$

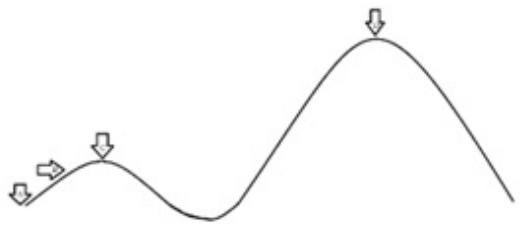
onde:

- Taxa de aprendizado (α): representa o quanto as novas informações substituem as antigas;
- Fator de desconto (γ): Determina a importância das recompensas futuras;
- Estimativa do valor futuro ótimo ($maxQ$): É o valor máximo obtido no estado S_{t+1} a partidar da ação a;

2.3.2 Escalada (HillClimbing)

O algoritmo de escalada é um método baseado em políticas muito utilizado em problemas de otimização matemática e busca a maximização de um dado valor até que seja encontrado um pico, ou solução ótima. O programa encerra uma vez que este pico é encontrado, quando não existe um valor vizinho maior ao valor atual. Existem três tipos de algoritmo de escalada: o simples, o de subida acentuada e o estocástico.

Figura 4 – Ilustração do método de escalada.



Fonte: autoria própria.

Em sua versão mais simples, o algoritmo de escalada apenas leva em conta o valor vizinho para sua operação. Se este o vizinho é melhor que o atual, então o programa o

define como o novo valor atual. O algoritmo resulta em soluções sub-ótimas e em algumas vezes a solução não é garantida.

A versão de subida acentuada é uma versão mais avançada que a simples. Nela o programa percorre todos os nós ao longo do caminho e seleciona aquele que está mais próximo ao estado objetivo.

Na escalada estocástica, o algoritmo seleciona de forma aleatória um dos melhores estados – aqueles que estão mais próximos do estado objetivo – e com base em um critério pré-definido ele decide se seleciona o nó atual ou um próximo nó.

2.3.3 REINFORCE

Williams (1992) estudou o problema de selecionar ações para maximizar a recompensa imediata. Ele identificou uma classe de regras de atualização que utilizam o método do gradiente descendente na recompensa esperada e mostrou como integrar estas regras através de retropropação. Está classe, que ficou conhecida como o algoritmo *REINFORCE*, realiza as atualizações para um parâmetro θ com base na equação 2.29:

$$\Delta W_{ij} = \alpha_{ij}(r - b_{ij}) \cdot \frac{\partial}{\partial W_{ij}} ln(g_j) \tag{2.29}$$

onde α_{ij} é um fator não-negativo, r é o reenforço atual, b_{ij} a base para o reenforço, g_j a função de densidade probabilística e o termo $\frac{\partial}{\partial W_{ij}} ln(g_j)$ é chamado de elegibilidade característica.

Tratando agora de algoritmos do tipo episódicos, o algoritmo *REINFORCE* tem função de recompensa é dada pela equação (2.30)

$$J(\theta) = \sum_{s \in S} d^{\pi}(s) V^{\pi}(s) = \sum_{s \in S} d^{\pi}(s) \cdot \sum_{a \in A} \pi_{\theta}(a|s) Q^{\pi}(s, a) \tag{2.30}$$

O cálculo do gradiente permite deslocar θ em direção ao gradiente $\nabla_{\theta} J(\theta)$, encontrando o valor de θ para a política $\pi\theta$ que oferece o maior retorno. Assim como em outros métodos, existe um fator de desconto γ que é utilizado na fórmula (2.31) para a recompensa futura G_t:

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2.31}$$

O *REINFORCE* depende do retorno esperado das ações de episódios inicialmente amostrados para atualizar o parâmetro θ da política a partir de um modelo parametrizado inicial. O algoritmo funciona porque a expectativa do gradiente de amostra é igual ao próprio gradiente:

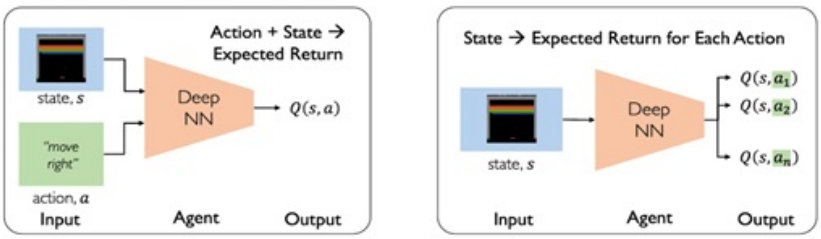
$$\begin{aligned} \nabla_{\theta} J(\theta) &= E_{\pi}[Q^{\pi}(s, a) \nabla_{\theta} ln \pi_{\theta}(a|s)] \\ &= E_{\pi}[G_t \nabla_{\theta} ln \pi_{\theta}(A_t|S_t)] \end{aligned} \tag{2.32}$$

2.3.4 Deep Q Network (DQN)

O algoritmo *Deep Q Network (DQN)* é uma combinação do algoritmo clássico de aprendizagem por reforço citado anteriormente, o *Q-Learning*, com a utilização de Redes Neurais Artificiais.

As redes neurais buscam o par estado (s_t) e ação (a_t) que proporcionam o maior valor de Q, ou seja, a maior probabilidade de recompensa (R_t). Deveras, uma rede densa pode simular o comportamento do agente descrito anteriormente, utilizando pares de estados e ações como sinais de entrada, determinando então a probabilidade Q como sinal de saída. Em outra alternativa, parametrizamos a ação e utilizamos apenas os estados como entrada, o que resulta em distintos *Q-values*, respectivos a cada ação, tornando o processamento e a análise da saída mais exequível.

Figura 5 – Redes neurais do tipo Q.



Fonte: retirada de *MIT 6.S191 Reinforcement Learning.*

Nas redes do tipo Q, o agente pode instituir uma política de aprendizagem $\pi(s)$ para inferir a ação a_t, respectiva ao estado s_t, que maximize a recompensa, ou o *Q-value*. Para a maior parte dos problemas, é impraticável respresentar a função q em uma tabela para cada combinação de estado e ação. Assim, as redes neurais são utilizadas para aproximar a função q utilizando parametros θ para estimar os valores q através da minimização da perda em cada etapa i:

$$L_i(\theta_i) = E_{s,a,r,s\sim\rho(.)} \cdot [(y_i - Q(s,a;\theta_i))^2], \text{ onde } y_i = r + \gamma \cdot max_{a'} \cdot Q(s',a';\theta_{i-1}) \quad (2.33)$$

Onde y_i é a diferença temporal e $y_i - Q$ o erro temporal. ρ representa a distribuição de comportamento, que é a distribuição ao longo das transições $\{s, a, r, s'\}$ recolhida a partir do ambiente.

3 Implementação dos Algoritmos

Neste capítulo serão demonstradas as os passos tomados para a implementação de cada um dos algoritmos escolhidos para a simulação do ambiente *CartPole-v0* bem como suas funções.

3.1 Q-learning

No método *Q-Learning*, o objetivo é encontrar a função *Q-value* ótima de forma iterativa utilizando a Equação de Otimalidade de Bellman. Para isso, todos os valores Q são armazenados em uma tabela que é atualizada a cada passo através da seguinte iteração:

$$q^{new}(s,a) = (1-\alpha)\cdot q(s,a) + \alpha(R_{t+1} + \gamma \cdot max_{a'} q(s',a')) \tag{3.1}$$

onde α é a taxa de aprendizagem – um parâmetro muito importante que controla a convergência – e γ o fator de desconto.

Os passos tomados na implementação do algoritmo são:

1. Inicializar uma tabela aleatória de *Q-values* $Q(s,a)$;
2. Observar o atual estado, s;
3. Escolher uma ação, a, para o estado observado baseada nas políticas de seleção de ações;
4. Executar a ação e observar o retorno (recompensa), r, e o novo estado, s';
5. Atualizar o *Q-value* para o estado de acordo com a equação (3.1);
6. Repetir o processo até que um estado terminal seja alcançado;

3.2 Escalada (Hill Climbing)

Como demonstrado anteriormente, este algoritmo possui algumas variações. Nesta implementação, a variante escolhida para o algoritmo de escalada é a de subida acentuada.

A política do algoritmo faz o uso de uma rede neural para realizar o mapeamento dos estados e das probabilidades das ações. Os seguintes passos serão tomados para solucionar o problema:

1. Inicializar a política π com pesos aleatórios W, $\theta_{melhor} = \theta$;
2. Utilizar a política π_θ para acumular as recompensas $r_1, r_2, ..., r_t$;
3. Calcular o o retorno descontado, $G_{atual} = \sum_{i=t}^{T} \gamma^{i-t} r_i$
   onde o fator de desconto $\gamma \in [0, 1]$;
4. Comparar e atualizar G_{melhor} e W_{melhor} caso sejam encontrados
   melhores retornos e pesos;
5. Atualizar o peso da política utilizando uma regra de atualização;
6. Repetir os passos 2 a 5 até que o programa seja terminado;

3.3 REINFORCE

No algoritmo *REINFORCE* o agente coleta uma trajetória τ de episódio utilizando sua política atual e a utiliza para atualizar os parâmetros da política.

O pseudocódigo do algoritmo descrevendo em maiores detalhes o comportamento do método pode ser escrito da seguinte forma:

1. Inicializar o parâmetro θ da política com valores aleatórios;
2. Utilizar a política π_θ para coletar a trajetória
   $\tau = (s_0, a_0, r_1, s_1, a_1, r_2, s_2, ..., a_{H+1}, r_{H+1}, s_{H+1})$;
3. Estimar o retorno da trajetória $\tau : R(\tau) = G_0, G_1, ..., G_H)$,
   onde G_k é o retorno esperado para a transição de estados k;
4. Utilizar a trajetória τ para estimar o gradiente $\nabla_\theta U(\theta)$
5. Atualizar os pesos θ da política;
6. Realizar um *loop* dos passos 1 a 5 até que não convirja.

Também existem algumas limitações associadas ao *REINFORCE*:

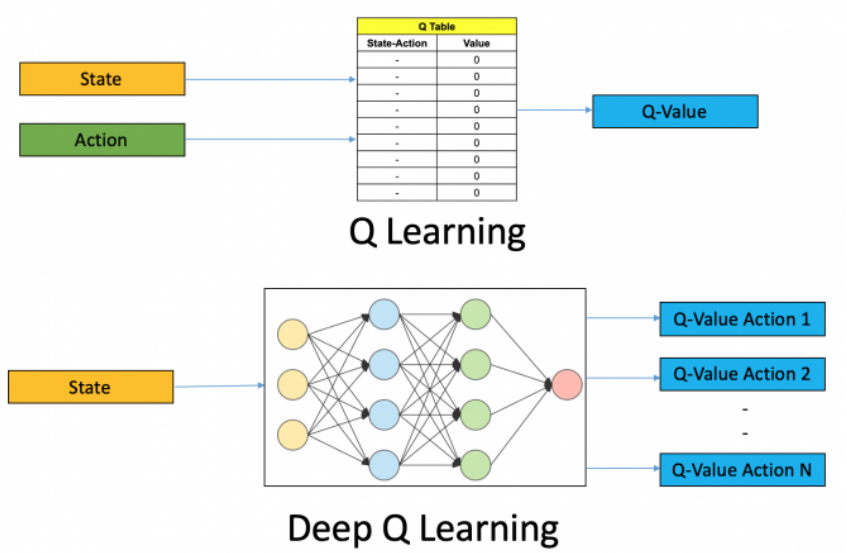
- O processo de atualização é ineficiente. A política é executada e atualizada uma só vez e a trajetória descartada;
- A estimativa através do gradiente possui muitos ruídos, levando à possibilidade de as trajetórias coletadas não serem representativas;
- Não existe uma atribuição de crédito clara. Uma trajetória pode conter diversas boas/más ações e se estas ações são reforçadas, depende apenas da saída final do programa;

3.4 Deep Q Network (DQN)

Uma *DQN* é uma função aproximadora para os *Q-values*. A base do método continua sendo o algoritmo *Q-Learning*, que funciona bem para estados finitos, mas em casos onde os espaços de estados e ações são contínuos, se torna impossível utilizar o *Q-Learning* sem o auxílio das redes neurais, uma vez que a quantidade de memória e tempo necessários para armazenar e atualizar a tabela aumenta consideravelmente.

A rede neural toma um estado s como entrada e calcula os *Q-values* para todas as possíveis ações naquele estado, escolhendo então a ação que gera o maior *Q-value*.

Figura 6 – Redes neurais do tipo Q utilizando ações e estados.



Fonte: (CHOUDHARY, 2019)

Os passos envolvidos no *DQN* são:

1. Processa e alimenta a rede neural com estados s que retornam os *Q-values* para todas as possíveis ações naquele estado;

2. Seleciona uma ação utilizando uma política *epsilon-greedy*. Com a probabilidade ϵ, uma ação aleatória é escolhida e com uma probabilidade $1-\epsilon$, é escolhida uma ação que possui o maior *Q-values*;

3. Executa a ação e transiciona para um novo estado para receber a recompensa, que é armazenada na memória;

4. Utiliza lotes aleatórios das transições armazenadas na memória e calcula a perda através da equação;

5. Realiza um gradiente com respeito aos parâmetros da rede a fim de minimizar a função de perda;

6. A cada N iterações, o peso atual da rede é copiado para o peso alvo da rede;

7. Os passos anteriores se repetem durante M episódios;

4 Resultados

Neste capítulo os resultados obtidos a partir da implementação dos algoritmos serão demonstrados e discutidos através dos relatórios e gráficos gerados pelos próprios algoritmos à medida que executam o aprendizado.

4.1 Q-Learning

O algoritmo retorna as seguintes informações que auxiliam na visualização de como acontece o aprendizado no algoritmo *Q-Learning*:

Tabela 2 – Resultados obtidos com o algoritmo *Q-learning*

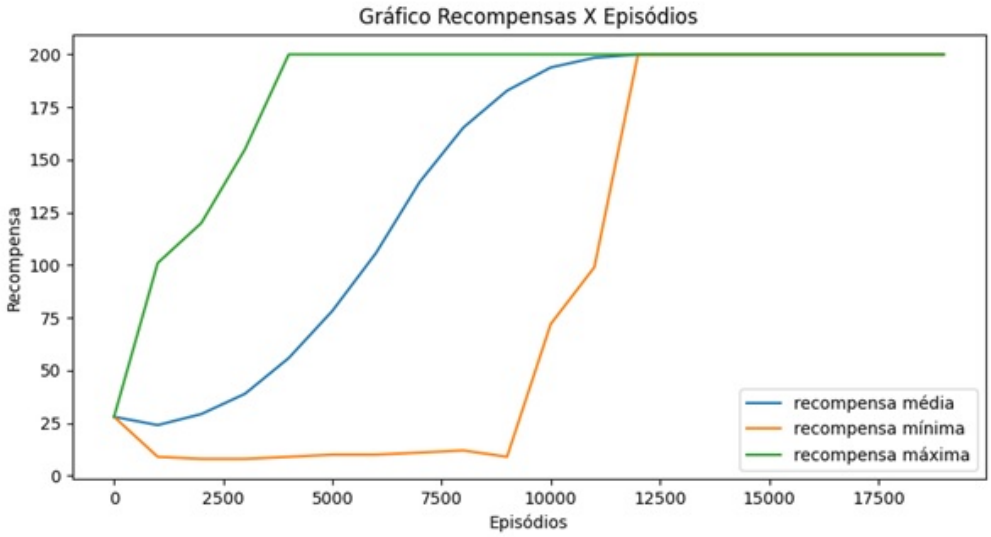
EPISÓDIO	RECOMPENSA MÍNIMA	RECOMPENSA MÁXIMA	MÉDIA
0	32	32	32,00
1000	8	123	24,63
2000	8	134	32,49
3000	8	166	39,84
4000	9	200	53,88
5000	9	200	78,44
6000	9	200	106,05
7000	12	200	138,86
8000	16	200	162,80
9000	16	200	182,88
10000	32	200	191,55
11000	13	200	199,94
12000	200	200	200,00
13000	20	200	200,00
14000	167	200	199,97
15000	0	200	200,00
16000	200	200	200,00
17000	20	200	200,00
18000	200	200	200,00
19000	20	200	200,00
20000	200	200	200,00

Fonte: autoria própria

Observa-se como a média total das recompensas aumentou gradativamente à medida que eram simulados os episódios, atingindo o objetivo pré-definido do ambiente *CartPole-v0* – o valor de 200 *timesteps* – a partir do episódio 12000, aproximadamente

O algoritmo também gera um gráfico (figura 7) relacionando as recompensas com o número de episódios e ilustra a curva de aprendizado:

Figura 7 – Gráfico dos resultados gerado pelo programa.



Fonte: autoria própria

4.2 Escalada (Hill Climbing)

As seguintes informações são mostradas durante a execução do algoritmo:

Tabela 3 – Resultados obtidos com o algoritmo de escalada

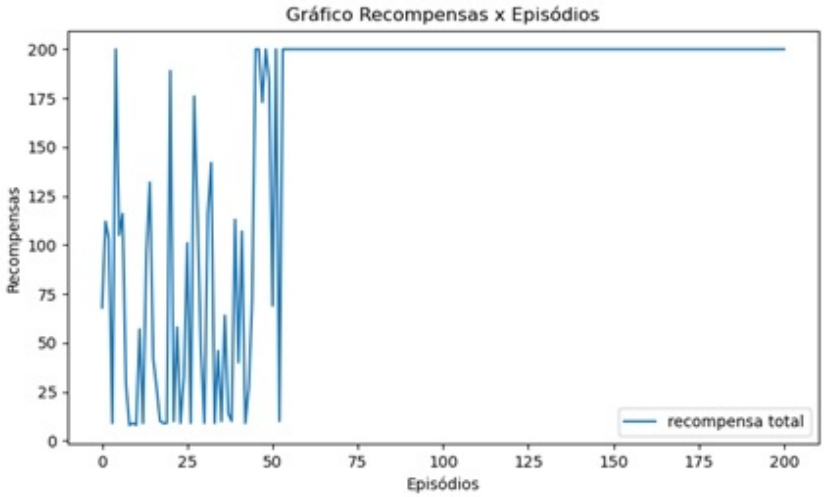
EPISÓDIO	RECOMPENSA TOTAL
0	68
10	8
20	189
30	9
40	40
50	69
60	200
70	200
80	200
90	200
100	200
110	200
120	200
130	200
140	200
150	200
160	200
170	200
180	200
190	200
200	200

Fonte: autoria própria

Neste caso o algoritmo atingiu seu objetivo de rapidamente, porém como os valores do estado inicial são aleatórios esta técnica pode levar durações diferentes para atingir o mesmo objetivo a cada instância. Dependendo da complexidade do ambiente o algoritmo pode nem sempre alcançar a solução ótima, o que não é o caso do ambiente *CartPole-v0* por ser um relativamente simples.

O gráfico a seguir (figura 8) ilustra o processo de aprendizagem até os valores das recompensas atingirem o valor máximo, se mantendo constante a partir do episódio 60, aproximadamente.

Figura 8 – Gráfico das recompensas totais em cada episódio.



Fonte: autoria própria

4.3 REINFORCE

Os resultados obtidos do treinamento com o algoritmo *REINFORCE* utilizando o método do gradiente foram:

Tabela 4 – Informações mostradas durante a execução do algoritmo

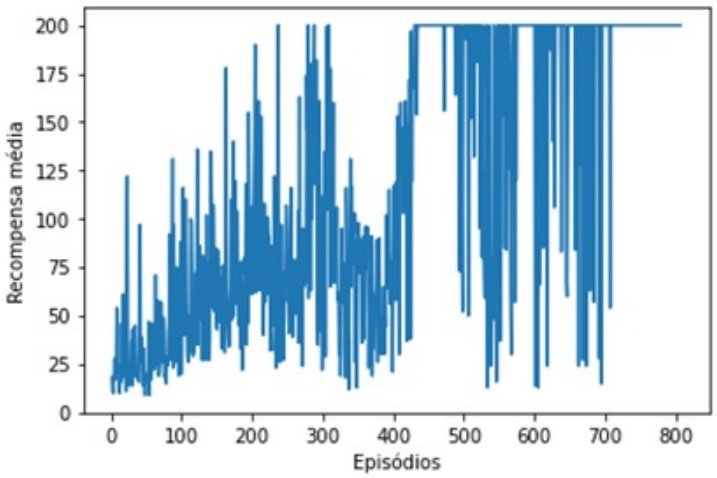
EPISÓDIO	RECOMPENSA MÉDIA
100	20,18
200	63,76
300	90,06
400	107,17
500	95,21
600	128,69
700	96,62
800	124,26
900	168,84
1000	128,07
1100	170,75
1200	192,95
1300	171,59
1400	200,00

Problema resolvido em 1394 episódios com recompensa média igual a 200.

Fonte: autoria própria

O programa foi capaz de atingir uma recompensa média de 200, que é um critério de finalização e o valor objetivo do ambiente *CartPole-v0*, no episódio 1394.

Figura 9 – Gráfico das recompensas totais em cada episódio.



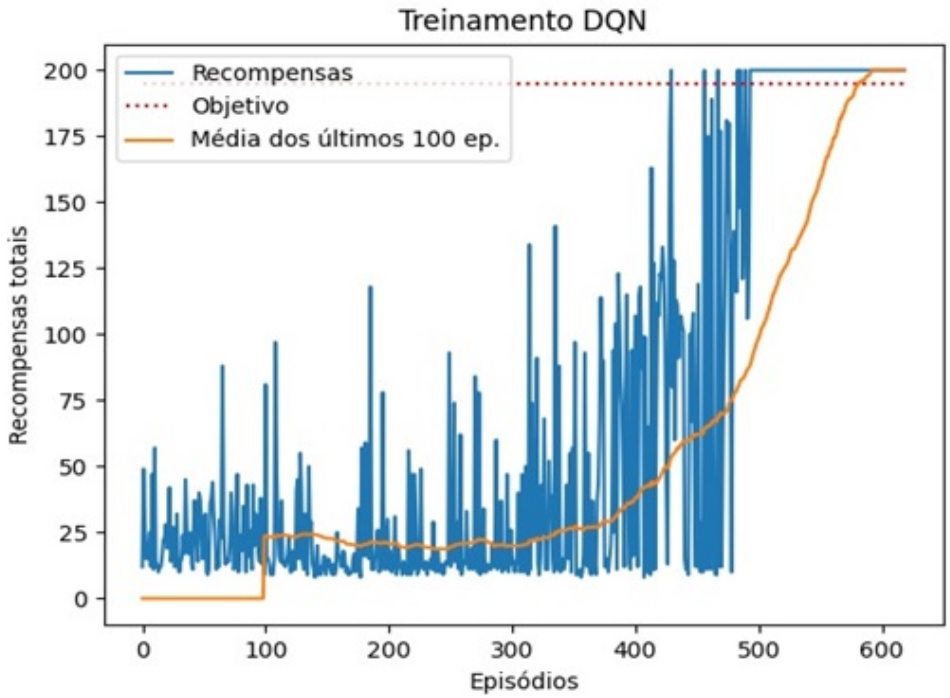
Fonte: autoria própria

Este método tem uma variância alta e pode performar de forma diferente dependendo dos valores dos pesos iniciais.

4.4 DEEP Q NETWORK (DQN)

Os resultados obtidos através do método *DQN* são indicados no gráfico abaixo e mostram que o algoritmo foi capaz de encontrar a solução para o problema em menos de 500 episódios, quando a média se manteve constante e foi igual a 200.

Figura 10 – Gráfico das recompensas e média ao longo dos episódios.



Fonte: autoria própria

Este método demanda uma capacidade de processamento maior por parte do *hardware* utilizado e um maior tempo para a simulação de cada episódio, porém é capaz de resolver problemas complexos de forma mais eficiente e confiável que os outros métodos.

5 Conclusões e considerações finais

O presente trabalho buscou introduzir o conceito de sistemas caóticos e da aprendizagem por reforço através de soluções computacionais que torna possível a resolução de problemas extremamente complexos. A partir do modelo do pêndulo invertido e com os algoritmos de aprendizagem por reforço, a solução para o problema *CartPole-v0* foi alcançada.

A abordagem para a solução do ambiente *CartPole-v0*, um dos diversos ambientes disponíveis na *OpenAI Gym*, foi feita com a utilização dos algoritmos *Q-Learning*, Escalada, *REINFORCE* e *Deep Q Network* desenvolvidos na linguagem de programação *Python*. Cada um dos algoritmos foram implementados com seus parâmetros bem definidos e executados através do compilador *Spyder*. As etapas de treinamento foram realizadas até que cada um dos algoritmos atingisse o objetivo final que é definido no ambiente *CartPole-v0*, um valor de recompensas totais igual a 200. Similarmente, caso fosse utilizado o ambiente *CartPole-v1*, este valor seria de 500 e ofereceria uma solução ainda mais confiável.

Com isso, o objetivo do trabalho de demonstrar o método de aprendizagem por reforço foi atingido. Alguns métodos como o de Escalada e o *DQN* foram capazes de resolver o problema em menos episódios, apesar do *DQN* exigir um maior poder de processamento e tempo. Com os algoritmos *Q-Learning* e *REINFORCE* o número de episódios foi elevado, porém o tempo de processamento foi menor. Todos os métodos estão sujeitos a variâncias e seus resultados podem variar, uma vez que utilizam pesos aleatórios para o estado inicial do ambiente.

Dessa forma, conclui-se que os objetivos deste trabalho foram cumpridos. Os conceitos de sistemas caóticos e aprendizagem por reforço foram introduzidos de forma sucinta e a implementação dos algoritmos realizadas sem muita dificuldade – a biblioteca *Open AI Gym* torna a implementação e simulação dos ambientes simples, cabendo ao usuário apenas a determinação dos parâmetros utilizados e do desenvolvimento do *loop* de treinamento.

Estas e outras ferramentas de machine learning são extremamente úteis, especialmente para área da engenharia, para a resolução de problemas que se realizados em ambientes reais, demandariam uma infinidade de tempo e recursos. Outra vantagem do método é a possibilidade de realizar estas simulações sem conhecer a fundo a física do problema, uma vez que ela está integrada no ambiente e o agente aprende por si só a alcançar o objetivo.

Referências

BARTO, A. G.; SUTTON, R. S.; ANDERSON, C. W. *Neuronlike adaptive elements that can solve difficult learning control problems. IEEE transactions on systems, man and cybernetics*. 5. ed. [S.l.]: IEEE, 1983.

BERTSEKAS, D. P. *Reinforcement Learning and Optimal Control*. 2019.

BROCKMAN, G. et al. *Open AI Gym*. 2016.

CHOUDHARY, A. *A Hands-On Introduction to Deep Q-Learning using OpenAI Gym in Python*. https://www.analyticsvidhya.com/blog/2019/04/introduction-deep-q-learning-python/: [s.n.], 2019.

GLORENNEC, P. Y. *Reinforcement Learning: an Overview*. 2000.

GRAESSER, L.; KENG, W. L. *Foundations of deep reinforcement learning: theory and practice in Python*. 2020.

HARMON, M. E.; HARMON, S. S. *Reinforcement Learning: A Tutorial*. 1997.

HEHN, M.; D'ANDREA, R. *A flying inverted pendulum*. [S.l.]: IEEE, 1997.

NAGENDRA, S. et al. *Comparison of reinforcement learning algorithms applied to the cart-pole problem. 2017 International Conference on Advances inComputing, Communications and Informatics (ICACCI)*. [S.l.]: IEEE, 2017.

PUTERMAN, M. L. *Chapter 8: Markov decision processes. Handbooks in Operations Research and Management Science*. 1990.

RASBAND, S. N. *Chaotic Dynamics of Nonlinear Systems*. 1990.

SZITA, I.; SZEPESVÁRI, C. *Model-based reinforcement learning with nearly tight exploration complexity bounds*. 2010.

WATKINS, C. J.; DAYAN, P. *Technical Note: Q-Learning. Machine Learning 8*. Londres: SpringerLink, 1992.

WILLIAMS, R. J. *Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine Learning 8*. Londres: SpringerLink, 1992.