UNIVERSIDADE FEDERAL DO AMAZONAS
FACULDADE DE TECNOLOGIA
CURSO DE GRADUAÇÃO EM ENGENHARIA MECÂNICA

ALEXANDRE MENDONÇA KRUL

APRENDIZAGEM POR REFORÇO COMO TÉCNICA DE CONTROLE PARA O PROBLEMA DO PÊNDULO INVERTIDO

Manaus
2021

ALEXANDRE MENDONÇA KRUL

APRENDIZAGEM POR REFORÇO COMO TÉCNICA DE CONTROLE PARA O PROBLEMA DO PÊNDULO INVERTIDO

Versão Original

Trabalho de Conclusão de Curso apresentado à Faculdade de Tecnologia da Universidade Federal do Amazonas como parte dos requisitos necessários para obtenção do título de Bacharel em Engenharia Mecânica.

Orientador: Prof. Dr. Danilo de Santana Chui

Manaus
2021

Ficha Catalográfica

Ficha catalográfica elaborada automaticamente de acordo com os dados fornecidos pelo(a) autor(a).

Mendonça, Alexandre Krul
M539a Aprendizagem por reforço como técnica de controle para o problema do pêndulo invertido / Alexandre Krul Mendonça . 2021
99 f.: il. color; 31 cm.

Orientador: Danilo de Santana Chui
TCC de Graduação (Engenharia Mecânica) - Universidade Federal do Amazonas.

1. controle de sistemas dinâmicos. 2. aprendizado de máquinas. 3. pêndulo invertido. 4. aprendizado por reforço. I. Chui, Danilo de Santana. II. Universidade Federal do Amazonas III. Título

Ministério da Educação
Universidade Federal do Amazonas
Coordenação do Curso de Engenharia Mecânica

DECISÃO DA BANCA EXAMINADORA DE TRABALHO DE CONCLUSÃO DE CURSO (TCC)

A **BANCA EXAMINADORA DE TCC,** em reunião realizada em 22 de janeiro de 2021, após a sessão de apresentação oral perante a Faculdade de Tecnologia/UFAM, conforme Resolução nº 02/2013 do Colegiado do Curso de Engenharia Mecânica, **APROVOU**, por unanimidade, o TCC do aluno **ALEXANDRE MENDONÇA KRUL**, com o título **"APRENDIZAGEM POR REFORÇO COMO TÉCNICA DE CONTROLE PARA O PROBLEMA DO PÊNDULO INVERTIDO"**.

Prof. Dr. Danilo de Santana Chui
Orientador (DEMEC/UFAM)

Prof. Dr. Gustavo Cunha da Silva Neto
Membro (DEMEC/UFAM)

Prof. Paulo Roberto Oliveira Martins
Membro (DEMEC/UFAM)

Em Manaus, 25 de janeiro de 2021.

Documento assinado eletronicamente por **Danilo de Santana Chui**, **Professor do Magistério Superior**, em 25/01/2021, às 21:33, conforme horário oficial de Manaus, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Gustavo Cunha da Silva Neto**, **Professor do Magistério Superior**, em 25/01/2021, às 21:39, conforme horário oficial de Manaus, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Paulo Roberto Oliveira Martins**, **Professor do Magistério Superior**, em 25/01/2021, às 22:41, conforme horário oficial de Manaus, 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 https://sei.ufam.edu.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **0428531** e o código CRC **F54CCA3E**.

Av. Octávio Hamilton Botelho Mourão - Bairro Coroado 1 Campus Universitário Senador Arthur Virgílio Filho, Setor Norte - Telefone: (92) 3305-1181
CEP 69080-900, Manaus/AM, cmecanica@ufam.edu.br

AGRADECIMENTOS

Aos meus pais, Genuir e Yara, por serem as pessoas mais importantes da minha vida e que sempre sofreram para que eu possa ter uma vida um pouco mais fácil do que suas vidas foram no passado.

Ao Movimento Empresa Júnior que me ensinou a ser uma pessoa melhor, me apresentou amigos que vou levar para a vida, fez eu ter uma nova perspectiva em relação à minha atuação profissional e que, acima de tudo, complementou minha formação de tal forma que eu me tornasse mais comprometido e capaz de transformar o Brasil. Agradeço à Coltech, Baré Júnior e Brasil Júnior por serem as pontes que me conectaram com esse movimento.

Ao meu orientador, Prof. Dr. Danilo de Santana Chui, por ter comprado o desafio junto a mim e se dedicado integralmente para que o trabalho fosse bem sucedido. Agradeço também pelas motivações e resiliência mesmo eu conciliando a execução deste trabalho final com meu emprego.

A todos os professores da universidade que me ensinaram grande parte das coisas que sei hoje. Em especial para Profa. Yashi, Prof. Nilton, Prof. Cláudio e Prof. Gustavo.

À Universidade Federal do Amazonas, por possibilitar a realização deste curso e onde eu pude me desenvolver como pessoa acima de tudo.

A todos que contribuíram direta ou indiretamente para realização deste trabalho.

“We can only see a short distance ahead, but we can see plenty there that needs to be done”

(Alan Turing)

RESUMO

KRUL, Alexandre. **Aprendizagem por reforço como técnica de controle para o problema do pêndulo invertido. 2021. 81f.** Trabalho de Conclusão de Curso (Engenharia Mecânica) - Universidade Federal do Amazonas, Manaus, 2021.

Neste trabalho procura-se utilizar algoritmos de *Machine Learning* para resolver o problema do pêndulo invertido com um grau de liberdade e comparar os resultados com a técnica de alocação de polos. Com esse objetivo, foram implementados três algoritmos de aprendizagem por reforço, *HillClimbing* com escala adaptativa de ruído, *REINFORCE* e *DeepQNetworks* em linguagem computacional *python* e seus resultados foram comparados entre si e com o método de controle em espaço de estados por alocação de polos. Foi possível observar que todos os métodos utilizados conseguiram atingir o objetivo de equilibrar o pêndulo. Os erros ITAE em relação à posição angular vertical para os métodos *HillClimbing*, *REINFORCE*, *DeepQNetworks* e Alocação de Polos foram de 410, 55, 50 e 52, respectivamente.

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

ABSTRACT

KRUL, Alexandre. **Reinforcement learning as a control technique applied for the inverted pendulum problem. 2021. 81f.** Undergraduate thesis (Mechanical Engineering) - Federal University of Amazonas, Manaus, 2021.

This work aims to utilize some Machine Learning algorithms to solve the inverted pendulum problem with one degree of freedom and compare the outcomes with the pole placement method. In this way, three reinforcement learning algorithms were implemented in python: HillClimbing with adaptive noise scaling, REINFORCE and DeepQNetworks and their results were compared with the state space pole placement method, also implemented in this work. The results showed that all the method were able to balance the pendulum. The ITAE errors with relation to the vertical angular position for the methods HillClimbing, REINFORCE, DeepQNetworks and Pole Placement were 410, 55, 50 and 52, respectively.

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

SUMÁRIO

1 INTRODUÇÃO

Mecanismos de controle de sistemas dinâmicos estão presentes em todo o cotidiano: um computador possui um controle de temperatura, aviões possuem controle de altitude e velocidade, e os robôs industriais possuem diversos controles para poder produzir nossos equipamentos domésticos.

Um problema clássico e comum de ser encontrado em livros de controle de sistemas dinâmicos é o problema do Pêndulo Invertido, que é introduzido com o objetivo de demonstrar a eficácia de métodos de controle contínuos tradicionais. Segundo Furuta, Yamakita e Kobayashi (1991), o problema se refere a um pêndulo invertido preso a um carro com limite de movimento em uma ou duas direções. Assim, o controle do pêndulo se dá pela movimentação nos eixos do carro.

Nise (2007) explica que os problemas de controle de sistemas dinâmicos, como o pêndulo invertido, são resolvidos com algoritmos providos das regras e modelagem dos sistemas envolvidos. Assim, o modelo matemático do fenômeno estudado é obtido, geralmente na forma de equações diferenciais ordinárias ou funções de transferência, e, ao inserir entradas no sistema, a resposta é calculada como solução da EDO ou saída da função de transferência. A partir disso, algumas técnicas são utilizadas para modificar o processamento da resposta do sistema para que um determinado objetivo seja alcançado. Como exemplo de aplicação dessas técnicas, Bellinaso e Paglione (2014) mostram o uso do regulador linear quadrático e Silva (2013) implementa a técnica de alocação de polos para o problema do pêndulo invertido.

Com o avanço das pesquisas em inteligência artificial, principalmente com o crescimento do potencial de processamento dos computadores e dos algoritmos de aprendizagem de máquina e *deep learning*, problemas de controle tradicionais, como o do pêndulo invertido, ganharam um conjunto a mais de métodos de resolução, de acordo com Russel, Norvig et al. (2013).

Segundo Alpaydin (2020), existem diversas abordagens diferentes de aprendizagem de máquinas, como *clustering*, aprendizado supervisionado, aprendizado não supervisionado entre outros. Porém, Wiering e Otterlo (2012) apresentam a abordagem de aprendizado por reforço que vem ganhando muita representatividade segundo os próprios autores e que consiste na inserção de um agente em um ambiente de modo que suas ações que possam gerar o maior número de uma determinada recompensa ao longo do tempo. A maior diferença dessa abordagem é que entradas e saídas sobre o sistema não são dadas ao agente, assim ele precisa desenvolver aprendizagem conforme sua própria evolução de tomadas de decisões, que é definida como sua política. O avanço tecnológico dessa abordagem traz ganhos enormes para o desenvolvimento de robôs, veículos autônomos, assistentes pessoais, entre outros.

Com o objetivo de disseminar o conhecimento e avançar as práticas de aprendizagem por reforço, inúmeros projetos de código aberto foram criados pela comunidade. Um desses casos é comentado por Brockman et al. (2016), que apresentam uma organização sem fins lucrativos denominada *OpenAI*, que criou uma biblioteca com ambientes de desenvolvimento denominada *Gym*. Essa biblioteca serve como um conjunto de ambientes para pesquisadores implementarem seus algoritmos de *machine learning* em modelos dinâmicos já criados e validados. Um dos modelos disponíveis é o pêndulo invertido com deslocamento do carro com um grau de liberdade, denominada *CartPole-v1*.

O presente trabalho tem como objetivo geral comparar o desempenho de alguns métodos de aprendizagem por reforço com um método clássico aplicado ao problema do pêndulo invertido com um grau de liberdade. Para isso, tem-se os seguintes objetivos específicos:

(i) Avaliar o problema do pêndulo invertido em um grau de liberdade;

(ii) Implementar computacionalmente o modelo disponível da biblioteca *Gym CartPole-v1*;

(iii) Sintetizar um controlador do sistema usando 3 algoritmos diferentes de aprendizagem por reforço;

(iv) Comparar os resultados e desempenhos obtidos pelos três métodos utilizados;

(v) Comparar os resultados com a técnica de alocação de polos;

(vi) Concluir sobre as respostas de cada abordagem;

(vii) Avaliar a utilização dos algoritmos de *machine learning* como potenciais usos para problemas de controle clássico.

Para a realização do trabalho e alcance dos objetivos propostos, a primeira etapa será de estudo dos fundamentos teóricos necessários. Essa etapa buscará compreender o problema do pêndulo invertido e identificar algoritmos de *machine learning* de aprendizagem por reforço que possam ser utilizados para sintetizar um controlador nas etapas seguintes.

Em segundo lugar, os algoritmos serão desenvolvidos em linguagem computacional *Python* e serão implementados usando a modelagem dinâmica do pêndulo invertido disponível no pacote *CartPolev1* da biblioteca aberta *Gym*.

Em sequência, os resultados de cada algoritmo serão computados, analisados e comparados entre si, observando as respostas da posição angular e posição linear em relação ao tempo, tempo de processamento, quantidade de episódios até obter controle do pêndulo, entre outros fatores.

Por fim, as respostas da posição angular e posição linear serão comparadas com os resultados de uma abordagem clássica de controle para o mesmo caso e será realizada uma conclusão sobre os resultados obtidos.

O trabalho está organizado da seguinte maneira: após a presente Introdução, o Capítulo 2 traz a revisão da literatura. O Capítulo 3 apresenta a implementação dos algoritmos de aprendizagem por reforço, o Capítulo 4 a implementação de um controlador usando a técnica de controle clássica. No Capítulo 5 são apresentados e discutidos os resultados da pesquisa. Por fim, no Capítulo 6, as conclusões são trazidas juntamente com as perspectivas e sugestões para trabalhos futuros.

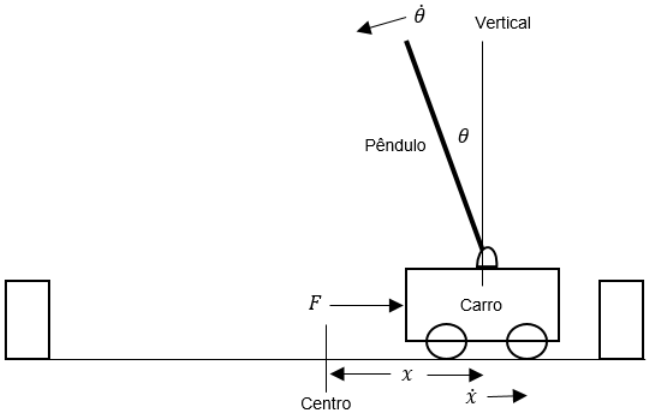
2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo serão introduzidos os principais conceitos e fundamentos que serão utilizados ao longo do trabalho, com uma breve revisão da literatura. Nas primeiras seções, o problema do pêndulo invertido e as técnicas de controle clássicas serão abordadas e, em sequência, a literatura referente ao aprendizado de máquinas e suas técnicas e algoritmos disponíveis de aprendizagem por reforço serão explorados.

2.1 Pêndulo Invertido

Como podemos ver em inúmeros estudos, como os de Dutech (2005), Anderson (1989), Barto, Sutton e Anderson (1983) e Michie e Chambers (1968) o problema do pêndulo invertido é comumente usado como base de comparação para o controle clássico e também para os algoritmos de aprendizado por reforço há muitos anos. Nagendra et al. (2017) descrevem o problema como um pêndulo pivotado em um carro, que possui um grau de liberdade ao longo do eixo horizontal. O objetivo do problema é equilibrar o pêndulo na posição vertical para cima usando as forças bidirecionais que são realizadas no carro através de uma força. Na figura 1 pode-se ver uma representação esquemática do problema:

Figura 1: Representação do problema do pêndulo invertido.



Adaptado de Nagendra et al. (2017) e traduzido pelo autor

Nagendra et al. (2017) em seu artigo também descrevem matematicamente a dinâmica envolvida no problema do pêndulo invertido. O estado do sistema em qualquer momento de tempo é definido pelo conjunto de variáveis de estado, que usualmente são definidas para esse problema como: a posição angular θ, a velocidade $\dot{\theta}$ do pêndulo, a posição linear x e velocidade $\dot{x}$ do carro.

O sistema não-linear analisado de acordo com as leis de Newton é descrito pelas seguintes equações:

$$\ddot{\theta} = \frac{(M+m)g\sin\theta - \cos\theta[F + ml\dot{\theta}^2\sin\theta]}{(4/3)(M+m)l - ml\cos^2\theta} \tag{2.1}$$

$$\ddot{x} = \frac{F + ml(\dot{\theta}^2\sin\theta - \ddot{\theta}\cos\theta)}{(M+m)} \tag{2.2}$$

onde M é a massa do carro e m a massa do pêndulo, g a aceleração da gravidade, F a força aplicada ao carro e l a distância entre o centro de massa do pêndulo para o pivô.

O problema aplicado aos algoritmos de aprendizado por reforço é desafiador pois um agente, no caso o carro, precisa selecionar inúmeras ações em um espaço discreto limitado para conseguir equilibrar o pêndulo na posição vertical.

2.2 Alocação de polos aplicada ao controle do pêndulo invertido

2.2.1 Método de alocação de polos em espaço de estados

Como é possível ver em Young e Meiry (1965) e Chaoui e Yadav (2016), o problema do pêndulo invertido, por ter suas características não-lineares, simplicidade para modelagem e um sistema de atuação relativamente simples, é utilizado entre os engenheiros de controle para testar e comparar suas técnicas desde 1950 até hoje. Uma das técnicas utilizadas é a alocação de polos aplicada ao controle de um modelo em espaço de estados. De acordo com Shehu et al. (2015), essa estratégia foca no ajuste intuitivo dos parâmetros de controle através do conhecimento do comportamento do sistema com relação à localização de seus polos e zeros no plano complexo.

Dado o sistema SISO em espaço de estados abaixo:

$$\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}u \tag{2.3}$$

$$y = \boldsymbol{C}\boldsymbol{x} + Du \tag{2.4}$$

onde $\boldsymbol{x}$ é o vetor de estados, y é o sinal de saída, u o sinal de controle, $\boldsymbol{A}$ uma matriz constante $n \times n$, $\boldsymbol{B}$ uma matriz constante $n \times 1$, $\boldsymbol{C}$ uma matriz constante $1 \times n$ e D uma constante.

O método de alocação de polos, segundo Ogata e Yang (2002), se baseia na premissa de que, dependendo de certas condições comentadas pelo autor, ao escolher uma matriz de ganho de realimentação de estado apropriada é possível forçar o sistema a ter polos de malha fechada nas posições desejadas, ou seja, que garanta um comportamento de segunda ordem com uma frequência natural e fator de amortecimento desejados. Assim, é definido que o sinal de controle é dado por:

$$u = -\boldsymbol{K}\boldsymbol{x} \tag{2.5}$$

onde $\boldsymbol{K}$ é a matriz de ganho de realimentação que força os autovalores de $\boldsymbol{A} - \boldsymbol{B}\boldsymbol{K}$ a

serem os valores desejados.

Assim, o projeto de um controlador utilizando a técnica de alocação de polos se baseia em, após a modelagem do sistema dinâmico, selecionar a resposta desejada do sistema, como tempo de pico, sobressinal máximo, tempo de acomodação entre outros, que darão o fator de amortecimento e frequência natural necessários e a partir desses fatores determinar a localização dos polos em malha fechada. Após a determinação dos polos desejados, é encontrada a matriz de ganho $\boldsymbol{K}$ e é adicionada como um compensador no sistema em malha fechada.

2.2.2 Controlabilidade

De acordo com Ogata e Yang (2002), para um sistema ser dito controlável no instante t deve existir uma entrada $u(t)$ para $0 \leq t \leq T$ com $T > 0$ e finito, capaz de transferir o sistema de qualquer estado inicial $x(0) = 0$ para qualquer outro estado x no intervalo de tempo T. A matriz de controlabilidade T pode ser definida como:

$$\boldsymbol{T} = \begin{bmatrix} B & AB & A^2B & \cdots & A^{n-1}B \end{bmatrix} \tag{2.6}$$

A partir de T, o sistema definido pelas matrizes (A,B) é controlável se $posto(T) = n$. O posto de uma matriz representa o número de colunas ou linhas linearmente independentes da matriz.

2.3 *Machine Learning*

De acordo com Goodfellow, Bengio e Courville (2016), o aprendizado de máquinas trata de técnicas de resolução de problemas que utilizam estatística computacional ao invés da lógica matemática. Na abordagem lógica, os problemas são resolvidos com algoritmos providos das regras e modelagem dos sistemas envolvidos, enquanto que na abordagem estatística baseada em dados os algoritmos criam um modelo preditivo a partir de dados que são inseridos no sistema.

Na abordagem clássica, obtém-se o modelo matemático do sistema objeto de estudo e, ao inserir entradas, a resposta do sistema é obtida. Na abordagem do aprendizado de máquinas, uma amostra de dados de entrada e saída do sistema em questão são conhecidos. Dentro desse contexto, existem várias abordagens de

machine learning, como: aprendizado supervisionado, aprendizado não supervisionado, aprendizado semissupervisionado e aprendizado por reforço, conforme apresentado por Alpaydin (2020).

Alpaydin (2020) também mostra as diferenças de cada abordagem. No aprendizado supervisionado, os dados de entrada e saída do sistema são fornecidos com suas classificações e o modelo é desenvolvido a partir desses dados, por isso o nome "supervisionado", pois entende-se que existe um "professor" que fornece esses dados e treina o algoritmo. Já no aprendizado não supervisionado, os dados são fornecidos ao sistema porém suas classificações não são disponíveis, deixando a cargo do algoritmo perceber padrões e desenvolver o modelo. A abordagem semissupervisionada é uma técnica híbrida em que parte dos dados são fornecidos com classificações, porém o treinamento do sistema é incompleto, deixando a cargo do algoritmo finalizar a classificação.

Segundo Li (2017), *Machine Learning* tem como suas bases a teoria de probabilidade e estatística e otimização, e esse conhecimento serve de base para *big data*, ciência de dados, modelagem preditiva de dados, mineração de dados entre outros problemas. Além disso, como bem mencionado pelo autor, é um campo de estudo bastante importante para os avanços tecnológicos necessários para visão computacional, processamento de linguagem natural e robótica por exemplo, já que é visto pela comunidade científica o uso dessa abordagem como principal meio de resolução desses problemas.

Um algoritmo de *machine learning*, segundo Goodfellow, Bengio e Courville (2016), é composto por: um conjunto de dados, uma função de perda ou de custo, um procedimento de otimização e um modelo. O conjunto de dados pode ser utilizado para treinamento do modelo, validação pós-treinamento e para testes posteriores. A função de perda ou de custo mensura o desempenho do modelo de acordo com a sua acurácia. Por sua vez, o procedimento de otimização é utilizado durante a etapa de treinamento do modelo. Durante essa etapa, um erro de treinamento é computado, e o algoritmo de *machine learning* deve reduzir a diferença entre esse erro e o erro obtido com novos dados de entrada.

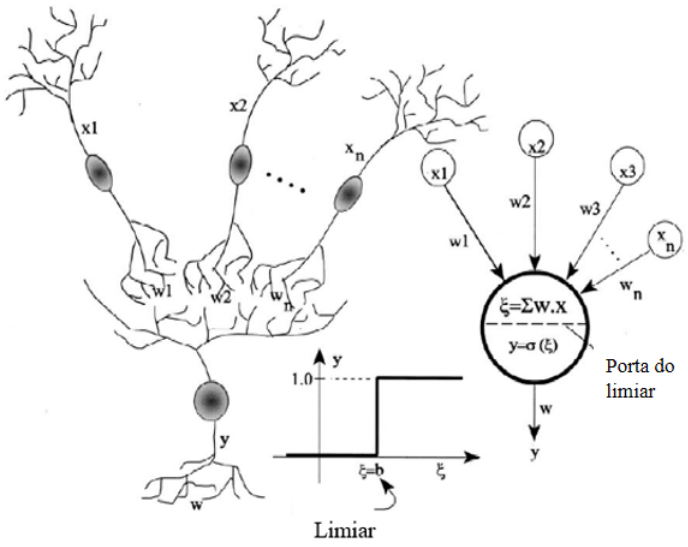
2.4 *Deep Learning*

Goodfellow, Bengio e Courville (2016) mostram em seu livro que muitos dos algoritmos de *machine learning* novos são inspirados em modelos biológicos de aprendizado. Dentro desses, Basheer e Hajmeer (2000) fazem uma revisão da literatura sobre as redes neurais artificiais, que são estruturas inspiradas no modelo biológico do cérebro humano e que possuem uma extensa aceitação em várias disciplinas por modelar problemas complexos do mundo real. Com isso, Goodfellow, Bengio e Courville (2016) declaram que o *deep learning* é um campo de estudo do *machine learning* que utiliza redes neurais artificiais como elemento central da resolução de seus problemas.

As redes neurais artificiais (RNAs) surgem como uma analogia à estrutura básica do cérebro humano, o neurônio. Essa estrutura segundo Schalkoff (1997) recebe sinais em elementos denominados dentritos e então esse sinal é transportado pelo axônio até ser passado para outro neurônio através da sinapse, que funciona como um impulso em forma de sinal elétrico. Segundo o mesmo autor, dependendo da intensidade do sinal sináptico, o próximo neurônio pode ser ativado, continuando o processamento do sinal, ou inibido, permanecendo desativado.

Basheer e Hajmeer (2000) mostram que a analogia entre a RNA e o neurônio se dá por no primeiro existir n neurônios com vários sinais de intensidade x e força sináptica w alimentando um neurônio com limiar de ativação b, como é possível ver na figura 2:

Figura 2: Analogia da Rede Neural Artificial com neurônio biológico.

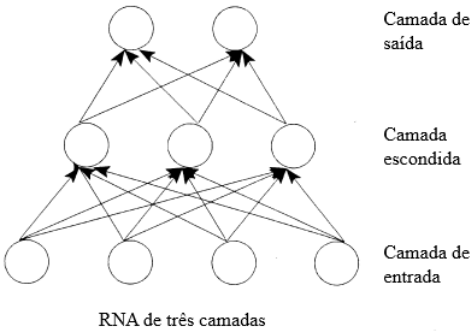


Adaptado Basheer e Hajmeer (2000) e traduzido pelo autor

O funcionamento da estrutura, segundo Hecht-Nielsen (1989), funciona com o recebimento de um sinal de entrada como estímulo do meio, realiza a combinação entre os sinais x e a força sináptica, também conhecida como peso w, e esse resultado é passado por uma função de ativação, como uma função degrau. Caso o resultado da ativação seja maior que o limiar b, o sinal é processado e ativado, caso contrário, não. Esse caso mais simples é chamado de *perceptron*

No entanto, como comentado por Hecht-Nielsen (1989), o *perceptron* não possui capacidade para resolver problemas não-lineares. Assim, são adicionadas camadas adicionais na estrutura da RNA para que problemas desse tipo possam ser resolvidos. Nesse caso, segundo o mesmo autor, camadas escondidas são adicionadas entre a entrada e a saída, e em cada camada, as saídas anteriores são somadas e multiplicadas por um peso da atual camada e, após isso, é aplicada a função de ativação para normalizar o valor obtido dentro de um intervalo fechado e esse resultado é passado adiante até a unidade de saída.

Figura 3: RNA de três camadas.



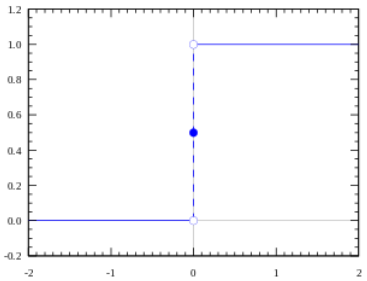
Fonte: (BASHEER; HAJMEER, 2000). Traduzido pelo Autor

2.4.1 Funções de ativação

Como apresentado na seção anterior, após a soma dos produtos da entrada de cada camada da rede neural com seus pesos, é aplicada uma função de ativação. Essa função de ativação é uma função matemática que, segundo Hecht-Nielsen (1989), determina se o neurônio será ativado ou não e, assim, permite computar resultados não lineares.

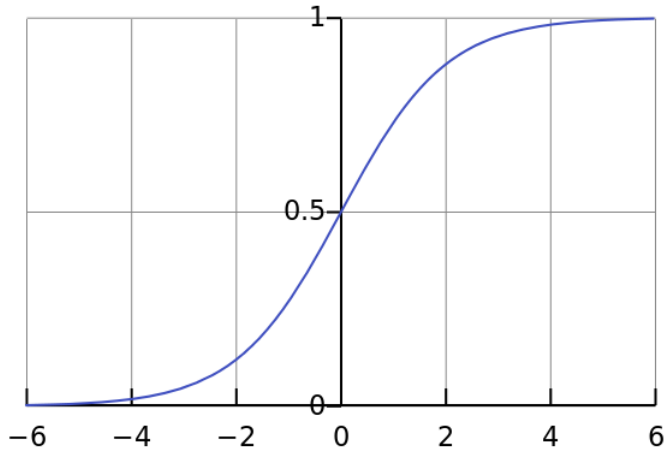
A função de ativação mais simples é a função degrau unitário, utilizada no *perceptron* de acordo com Hecht-Nielsen (1989), onde caso o valor comparado exceda um limiar a saída da função é 1, caso contrário, 0:

Figura 4: Função de ativação degrau unitário.



Outra função de ativação comumente utilizada, segundo Ramachandran, Zoph e Le (2017), é a função sigmóide $A(x) = \frac{1}{1+e^{-x}}$, que possui uma classificação binária mais suave:

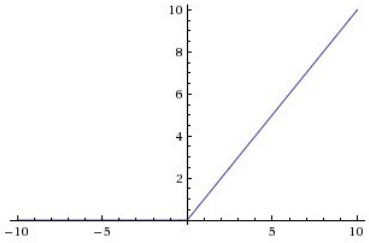
Figura 5: Função de ativação sigmóide.



Uma variação da função sigmóide é a função *softmax* utilizada para problemas multiclasses, segundo Basheer e Hajmeer (2000). Essa função possui a forma $A(x) = \frac{e^{x_i}}{\sum_{i=1}^{n} e^{x_i}}$

Além disso, Ramachandran, Zoph e Le (2017) apresentam também a função de ativação *reLU*, também conhecida como retificador linear unitário. Essa função é definida como $A(x) = max(0, x)$, e apresenta a seguinte forma:

Figura 6: Função de ativação reLU.



2.4.2 Métodos de otimização

Como uma característica inerente de um sistema inteligente é o aprendizado, essa estrutura precisa ter essa capacidade. Basheer e Hajmeer (2000) mostram que no *deep learning* o aprendizado é obtido através de algoritmos de otimização que processam os dados de entrada já conhecidos, comparam com a saída também já conhecida e ajustam os pesos w e o limiar b da RNA de forma interativa minimi-

zando a função custo J definida, geralmente o erro quadrático médio:

$$J = \frac{1}{N}\sum_{i=1}^{N}\left(\hat{y}_i - y_i\right)^2 \tag{2.7}$$

Onde $\hat{y}_i$ é a saída estimada através da rede neural e y_i o valor conhecido através dos dados fornecidos.

Denomina-se esse processo como treinamento da RNA. Nessa seção serão apresentados alguns métodos utilizados para esse processo.

2.4.2.1 Método Gradiente

Segundo Ruder (2016), o Método Gradiente, também conhecido como Gradientes Descendentes, é um dos mais utilizados para otimização em diversos campos de estudo diferentes. Segundo o autor, o método utiliza a derivada de primeira ordem da função custo para calcular em qual sentido os pesos da rede neural devem ser atualizados para que J atinja seu mínimo, de acordo com a equação abaixo:

$$w_{t+1} = w_t - \alpha \nabla_w J(w) \tag{2.8}$$

onde α é um escalar denominado coeficiente de aprendizado e que determina o tamanho da atualização dos pesos. $\nabla_w J(w)$ é o gradiente da função custo em relação aos pesos.

Sendo assim, após inicialização dos pesos w da rede neural, todos os dados de entrada são processados pela rede neural e o conjunto de saídas são comparados com os valores conhecidos através da função erro escolhida. Com isso, a equação 2.8 é calculada e os pesos da rede neural atualizados.

Robbins e Monro (1951) apresentam uma variação do método gradiente chamado método de aproximação estocástico, e Lydia e Francis (2019) mostra a adaptação desse método conhecido como Gradientes Descendentes Estocástico aplicado às redes neurais. Nesse caso, ao invés de atualizar os parâmetros das redes neurais após todo o processamento da base de dados completa, a atualização é feita a cada

informação de índice i obtida na base:

$$w_{t+1} = w_{t+1} - \alpha \nabla_w J(w, x_i, y_i) \tag{2.9}$$

2.4.2.2 *Adagrad*

O método *Adagrad* é proposto ajustando o coeficiente de aprendizado na equação (2.9) para cada parâmetro w_t e cada instante de tempo de cálculo t, e para o conjunto de dados de índice i, da seguinte forma:

$$w_{t+1,i} = w_{t,i} - \frac{\alpha}{\sqrt{G_{t,i} + \epsilon}} g_{t,i} \tag{2.10}$$

onde $g_{t,i} = \nabla_{w_t} J(w_{t,i})$, ϵ um valor na ordem de 10^{-8} para evitar divisão por zero, e $G_{t,i}$ é a soma dos gradientes previamente calculados até o instante de tempo t, calculado como:

$$G_{n,t,i} = \sum_{j=1}^{t} g_j g_j^T \tag{2.11}$$

2.4.3 *Adam*

O método *Adam* também conhecido por Estimação Adaptativa do Momento, apresentado por Kingma e Ba (2014), utiliza de momentos de primeira e segunda ordem para atualizar o coeficiente de aprendizado e evitar atualizações rápidas que possam ultrapassar o mínimo da função. Os momentos são definidos como:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_{n,t,i} \tag{2.12}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_{n,t,i}^2 \tag{2.13}$$

onde m_t é o momento de primeira ordem que representa a média e v_t o momento de segunda ordem que representa o desvio padrão do gradiente g_t, respectivamente. β_1 e β_2 são hiperparâmetros com valores padronizados 0,9 e 0,999 respectivamente, segundo Kingma e Ba (2014).

Após isso, os pesos w_t são atualizados através de:

$$w_{n,t+1} = w_{n,t} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \tag{2.14}$$

Onde $\hat{m}_t$ e $\hat{v}_t$ são correções realizadas nos momentos, definidos como:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1} \tag{2.15}$$

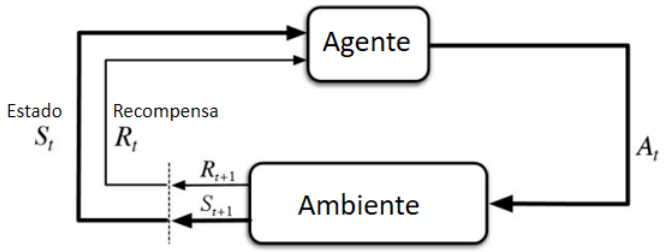
$$\hat{v}_t = \frac{v_t}{1 - \beta_2} \tag{2.16}$$

2.5 Aprendizagem por reforço

A abordagem de aprendizagem por reforço, tema central do presente trabalho, é uma das mais recentes e com maior potencial de crescimento em pesquisas, segundo Botvinick et al. (2019). Trata-se do caso em que o programa interage com um ambiente em busca de um objetivo ou recompensa, e as regras do ambiente não são fornecidas ao programa, deixando a cargo do algoritmo descobrir por si só o melhor modelo de atuação. Botvinick et al. (2019) mostram que os avanços recentes dentro da aprendizagem por reforço mostraram desempenhos positivos em tarefas como jogar *videogames*, poquer, jogos de tabuleiro e até xadrez melhores do que seres humanos na maior parte das vezes.

Nessa abordagem, segundo Duan et al. (2016), tem-se um agente que opera em um ambiente com regras físicas próprias, e sua interação com o ambiente representa suas ações A_t. Cada ação desenvolvida pelo agente redefine o seu estado S_t no ambiente e promove recompensas R_t ao agente. O objetivo do algoritmo é otimizar as ações do agente ao longo do tempo t de modo que ele obtenha o maior número de recompensas possível em um determinado tempo de interação com o ambiente. A forma com que o agente lida com o ambiente é chamada de política.

Figura 7: Representação do problema de aprendizagem por reforço.



Fonte: Autor.

Uma forma intuitiva de entender como funciona o tema é apresentado por Kaelbling, Littman e Moore (1996) e pode ser visto no exemplo abaixo:

Ambiente: Você está no estado 65 do espaço discreto. Você possui 4 ações possíveis.

Agente: Baseado na minha política, vou fazer a ação 2.

Ambiente: Você recebeu uma recompensa de 7 unidades e agora está no estado 15. Você pode fazer 2 ações nesse estado.

Agente: Baseado na minha política, vou fazer a ação 1.

Ambiente: Você recebeu uma recompensa de -4 unidades e agora está no estado 65. Você pode fazer 4 ações nesse estado.

Uma maneira mais formal de entender o problema de aprendizagem por reforço é apresentado por Sutton e Barto (2020). O agente e o meio interagem entre si em uma sequência discretizada de tempo, $t = 0, 1, 2, 3...T$. A cada instante de tempo t, o agente possui uma representação sua em relação ao meio chamada de estado, $S_t \in \mathscr{S}$ onde $\mathscr{S}$ é o espaço de todos os estados possíveis, e baseado nisso o agente seleciona uma ação $A_t \in \mathscr{A}(S_t)$, onde $\mathscr{A}(S_t)$ são todas as ações disponíveis no estado S_t. Um intervalo de tempo imediatamente depois, como consequência de sua ação, o agente recebe uma recompensa numérica $R_{t+1} \in \mathscr{R} \subset \mathbb{R}$, onde $\mathscr{R}$ é o conjunto de todas as recompensas possíveis para aquele determinado estado, e se encontra em um novo estado S_{t+1}. Para cada espaço de tempo t, existem probabilidades associadas a cada ação que pode ser escolhida para determinado estado S_t.. Esse mapeamento é chamado de política e é denotado por π_t, onde $\pi_t(a|s)$ é a probabilidade de $A_t = a$ dado que $S_t = s$. Assim, os métodos de aprendizado por reforço mostram como um agente muda sua política como resultado de sua própria

experiência, e, o objetivo é obter o máximo de recompensa no longo prazo.

De acordo com Kaelbling, Littman e Moore (1996) um desafio encontrado no aprendizado por reforço é o de *exploration* e *exploitation*[1]. Como o agente percorre o ambiente em busca de obter o maior número de recompensas possíveis no longo prazo, ele deve adotar a estratégia correta entre explorar todo o estado em busca de novas informações para melhorar sua política (*exploration*) ou permanecer utilizando sua política atual com o objetivo de ganhar recompensas com as informações já obtidas (*exploitation*). Para esse caso, Botvinick et al. (2019) mostram que os algoritmos de aprendizagem por reforço devem encontrar o balanço ideal entre os dois caminhos iniciando o período de treinamentos explorando mais o ambiente (*exploration*) e conforme a performance for aumentando permanecer na atual política por mais tempo (*exploitation*).

Sutton e Barto (2020) mostram em seu livro diversos exemplos práticos da aplicação do aprendizado por reforço:

- Um jogador de xadrez faz um movimento. Essa escolha é feita envolvendo tanto planejamento, inferindo os movimentos futuros após essa escolha, mas também com julgamentos intuitivos e imediatos;
- Um controlador dos parâmetros de uma refinaria de petróleo em tempo real. Esse controlador tenta maximizar as recompensas, ou seja, a produtividade, redução de custos, qualidade sem seguir *set points* originalmente programados pelos engenheiros;
- Robôs domésticos de limpeza que devem decidir se entram em um quarto novo para encontrar mais sujeira para limpar ou voltar para seu posto para recarregar sua bateria. Essa escolha é feita baseada no atual nível de sua bateria e em quão rápido foi no passado encontrar a tomada para recarregar sua energia

Como demonstrado por Nagendra et al. (2017), existem diversos algoritmos que podem ser aplicados ao problema do pêndulo invertido utilizando aprendizado por reforço. Esse trabalho focará em três desses algoritmos: *Hill Climbing*, *REINFORCE*

[1] Tanto exploration quanto exploitation são traduzidos em português pela mesma palavra: exploração. A primeira com o sentido de descoberta, e a outra com o sentido de tirar vantagem. Escolheu-se manter os termos em inglês justamente por indicar a conotação de cada tipo de escolha do agente de forma mais precisa.

e *Deep Q Networks*. Porém, antes é preciso entender com mais profundidade o que esses métodos buscam resolver.

2.6 Recompensas e Retorno

Como introduzido na seção anterior, quando é falado do objetivo de um problema de aprendizado por reforço, fala-se na obtenção do maior número de recompensas esperadas no longo prazo. Sutton e Barto (2020) apresentam uma definição formal para o objetivo, que consiste em maximizar o valor esperado da soma cumulativa do sinal escalar denominado por recompensa $R \in \mathscr{R}$. Se a sequência de recompensas obtidas pelo agente depois de cada espaço de tempo t é representada por $R_{t+1}, R_{t+2}, R_{t+3}...$, então temos o retorno G_t esperado como:

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + ... + R_T, \tag{2.17}$$

onde T é o instante de tempo final. Sutton e Barto (2020) mostram que essa abordagem faz sentido quando o problema possui um tempo delimitado de execução, denominado episódio, como uma rodada de um jogo ou iterações que se repetem em um problema mecânico. Nesse caso, cada episódio termina em um estado denominado estado terminal, e após a conclusão do episódio o problema é reiniciado para o estado inicial. Esses problemas são chamados por problemas episódicos e nesses casos diferenciamos o espaço de estados $\mathscr{S}$ do espaço de estados terminais $\mathscr{S}^+$.

Porém, como apresentado também pelos autores Sutton e Barto (2020) e também no livro de Goodfellow, Bengio e Courville (2016), em vários casos a interação agente-meio não possui uma quebra no tempo, sendo contínua. Esses problemas são chamados de problemas contínuos. Nesse caso, a definição acima para G_t não é a melhor pois como $T = \infty$ o retorno tende ao infinito facilmente. Assim, devemos implementar um desconto na soma dos retornos, e a definição passa a ser:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}, \tag{2.18}$$

onde γ é um parâmentro $0 \leq \gamma \leq 1$ denominado taxa de desconto. Essa taxa determina o valor presente das recompensas futuras. Se $\gamma = 0$ o agente é considerado

míope, pois só busca maximizar resultados imediatos.

Para obter uma única representação dos problemas e evitar descrever sempre quando um problema é episódico ou contínuo, Sutton e Barto (2020) utilizam a notação i para representar o episódio e o tempo t inicia em zero para cada novo episódio, ficando assim com $A_{t,i}, S_{t,i}, R_{t,i}$, etc. Assim, tratando-se de um problema contínuo o subscrito i sempre será o mesmo e t crescerá infinitamente. E, do mesmo modo, é definido G_t como:

$$G_t = \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1}, \tag{2.19}$$

Assim, inclui-se a possibilidade de $T = \infty$ ou $\gamma = 1$ (porém não ambos). No atual trabalho será utilizada essa notação para expressar o problema resolvido.

2.7 A Propriedade de Markov

Um processo markoviano, de acordo com Sutton e Barto (2020), é todo fenômeno no qual o estado futuro do sistema depende apenas do estado atual. Para um problema de aprendizagem por reforço é extremamente importante que cada estado S_t possua informações suficientes para que o agente possa tomar sua próxima decisão e tomar uma ação A_t, sem precisar olhar seu histórico de estados passados. Ainda em seu livro, Sutton e Barto (2020) definem formalmente a propriedade de Markov aplicada ao problema de aprendizagem por reforço considerando um número discreto de estados e recompensas. Assim, tem-se que para um meio que obtém uma resposta no instante de tempo $t+1$ a partir de uma ação tomada no tempo t, a dinâmica pode ser representada por:

$$Pr\{R_{t+1} = r, S_{t+1} = s^{'} | S_0, A_0, R_1, ..., S_{t-1}, A_{t-1}, R_t, S_t, A_t\}, \tag{2.20}$$

Para todo $r, s^{'}$ e todos os possíveis eventos no passado: $S_0, A_0, R_1, ..., S_{t-1}, A_{t-1}, R_t, S_t, A_t$ é dito que o problema possui a propriedade de Markov se a resposta no tempo $t+1$ só depende do estado e a ação realizada no espaço de tempo t. Nesse caso, a representação da dinâmica é definida somente

como:

$$p(s^{'}, r|s, a) = Pr\{R_{t+1} = r, S_{t+1} = s^{'}|S_t, A_t\}, \tag{2.21}$$

para todo $r, s^{'}, S_t$ e A_t. Assim, segundo Goodfellow, Bengio e Courville (2016) com essa propriedade pode-se estimar o próximo estado e recompensa esperada através do estado e ação atual.

2.8 Processo de Decisão de Markov

De acordo com Sutton e Barto (2020) um problema de aprendizado por reforço que satisfaz a propriedade de Markov é chamado de Processo de Decisão de Markov, ou simplesmente PDM. Nesse processo, dado um estado s e ação a, a probabilidade de alcançar-se cada novo estado $s^{'}$ e recompensa r é dada por:

$$p(s^{'}, r|s, a) = Pr\{R_{t+1} = r, S_{t+1} = s^{'}|S_t = s, A_t = a\}, \tag{2.22}$$

e, através dessa definição, qualquer variável pode ser definida para esse processo de decisão de Markov. Como apresentado pelos mesmos autores, a recompensa para um par de estado-ação é definida por:

$$r(s, a) = \mathbb{E}[R_{t+1}|S_t = s, A_t = a] = \sum_{r \in \mathscr{R}} r \sum_{s^{'} \in \mathscr{S}} p(s^{'}, r|s, a), \tag{2.23}$$

a transição de estados por:

$$p(s^{'}|s, a) = Pr\{S_{t+1} = s^{'}|S_t = s, A_t = a] = \sum_{r \in \mathscr{R}} p(s^{'}, r|s, a), \tag{2.24}$$

e a recompensa esperada para a tripla estado-ação-novo estado:

$$r(s, a, s^{'}) = \mathbb{E}[R_{t+1}|S_t = s, A_t = a, S_{t+1} = s^{'}] = \frac{\sum_{r \in \mathscr{R}} r\, p(s^{'}, r|s, a)}{p(s^{'}|s, a)}, \tag{2.25}$$

2.9 Funções de valor

Segundo Kaelbling, Littman e Moore (1996), os problemas de aprendizado por reforço envolvem a estimativa de funções de valor, que são, de acordo com os autores, funções de estado ou estado-ação que estimam a expectativa de ganhos de recompensa futura dado que o agente está em um dado estado, ou a expectativa de ganhos de recompensas futuras dado que o agente desempenhe uma ação específica em um estado. Como as recompensas futuras dependem das ações futuras desempenhadas pelo agente, é preciso considerar a política atual do agente, que indica a probabilidade de ações futuras dado os estados futuros.

Assim, como formalizado por Sutton e Barto (2020), a função valor de um agente em um estado $s \in \mathcal{S}$ em uma política π é denotado por $v_\pi(s)$ e representa o valor esperado de retorno iniciando no estado s e seguindo a política π, pode ser representada por:

$$v_\pi(s) = \mathbb{E}_\pi \left[G_t | S_t = s] = \mathbb{E}_\pi [\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s \right], \tag{2.26}$$

onde $\mathbb{E}_\pi$ representa o valor esperado de uma variável aleatória dado que o agente siga a política π, e t qualquer instante de tempo. A função v_π é a função estado-valor para a política π. De maneira similar, é importante definir a função valor para um estado s dado que o agente desempenhou uma ação a, representada por $q_\pi(s,a)$. De acordo com Sutton e Barto (2020):

$$q_\pi(s,a) = \mathbb{E}_\pi \left[G_t | S_t = s, A_t = a] = \mathbb{E} [\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a \right], \tag{2.27}$$

e chama-se a equação acima por função ação-valor para a política π.

Resolver um problema de aprendizado por reforço significa encontrar uma política que maximize as recompensas no longo prazo, isto é, uma política ótima. Sutton e Barto (2020) definem uma política ótima como a política π cujo $v_\pi(s)$ é maior que qualquer $v_{\pi'}(s)$ para todo espaço de estados $s \subset \mathcal{S}$. Ou seja, a política ótima, representada por π_*, acumula maior retorno no longo prazo do que qualquer outra

política existente para todos os espaços de estados. Dessa forma, temos que:

$$v_*(s) = max_\pi v_\pi(s), \tag{2.28}$$

para todo $s \in \mathscr{S}$.

Ao mesmo tempo, é interessante definir a função ótima ação-valor, como:

$$q_*(s,a) = max_\pi q_\pi(s,a), \tag{2.29}$$

para todo $s \in \mathscr{S}$ e $a \in \mathscr{A}$. Sutton e Barto (2020) escrevem em seu livro q_* em termos de v_* como:

$$q_*(s,a) = \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}|S_t = s, A_t = a)]. \tag{2.30}$$

2.10 Algoritmos de aprendizado por reforço

Conforme apresentado por Nagendra et al. (2017), é muito difícil obter conhecimento de todo o PDM do problema do pêndulo invertido, assim como em vários outros problemas clássicos de controle. Isso se deve ao fato de que as probabilidades de transição de estado não podem ser determinadas antes do controle ser otimizado, já que o agente aprende através de ações aleatórias, sem conhecimento das ações corretas até que a política esteja otimizada. Assim, ainda de acordo com o autor, os algoritmos de aprendizado por reforço que não necessitam do conhecimento da dinâmica $p(s^{'}, r|s, a)$ do sistema são ideais.

2.10.1 Algoritmo *Hill Climbing*

De acordo com Sutton e Barto (2020) e com o que foi apresentado nas seções anteriores, existem métodos de aprendizado por reforço que utilizam como diretriz principal encontrar a política ideal para um agente. Essa abordagem funciona da seguinte maneira:

1. Inicialize uma política π com pesos aleatórios w, $w_{melhor} = w$;

2. Use a política π_w para coletar as recompensas R_1, R_2,... ,R_T no meio;

3. Calcule o retorno descontado G_t de acordo com a equação 2.19;

4. Compare e atualize G_{melhor} e w_{melhor} se melhores retornos e pesos foram encontrados;

5. Atualize os pesos da política usando alguma regra de atualização;

6. Repita os passos 2-5 por um número n de episódios;

Ainda segundo Sutton e Barto (2020), a grande questão é como o passo 5 é realizado, diferenciando assim todos os métodos baseados em políticas. Um dos métodos divulgados é o *hillclimbing* com escala de ruído adaptativa, que segundo Ackley (1987) é uma técnica heurística de otimização utilizada no campo da inteligência artificial. Esse algoritmo gera uma política candidata através de uma pequena perturbação σ dos atuais melhores pesos e utiliza essa nova política para geração do episódio. O retorno é comparado e uma nova política é gerada caso obtenha melhores resultados. O termo escala de ruído adaptativa refere-se à variação do tamanho da perturbação σ dependendo dos resultados obtidos no episódio anterior. O algoritmo completo é realizado da seguinte forma:

1. Inicialize uma política π com pesos aleatórios w, $w_{melhor} = w$;

2. Use a política π_w para coletar as recompensas R_1, R_2,... ,R_T no meio;

3. Calcule o retorno descontado G_t de acordo com a equação 2.19;

4. Compare e atualize G_{melhor} se $G_t > G_{melhor}$;

5. Se $G_t > G_{melhor}$, faça $\sigma = max(\sigma_{min}, \sigma/2)$ onde σ_{min} é predefinido

6. Se $G_t < G_{melhor}$, faça $\sigma = min(\sigma_{max}, \sigma * 2)$ onde σ_{max} é predefinido

7. $w = w + \sigma$

8. Repita os passos 2-7 por um número n de episódios;

Ackley (1987) apresenta que o grande problema dessa técnica é que não necessariamente ela encontra o máximo global da função, podendo convergir para um máximo local na maior parte das vezes.

2.10.2 Algoritmo REINFORCE

Seguindo a mesma linha dos métodos baseados em políticas, Sutton et al. (2000) apresenta um algoritmo denominado *REINFORCE*, conhecido também como Gradientes de Política Monte Carlo. O objetivo desse método é encontrar uma política π que maximize a equação 2.25. Esse método utiliza o valor esperado dos retornos cumulativos como função objetivo, de acordo com a equação abaixo:

$$J(\tau) = \mathbb{E}[G_t|\pi_\tau] \tag{2.31}$$

E, após a diferenciação em relação à τ, segundo Sutton et al. (2000), é obtida a função objetivo a ser utilizada no algoritmo:

$$\nabla J(\tau) = \sum_{t=0}^{T-1} \nabla_\tau log\pi_\tau(a_t|s_t)G_t \tag{2.32}$$

Com isso, ainda de acordo com Sutton et al. (2000), o parâmetro τ da política é atualizado através de:

$$\tau_{t+1} = \tau_t + \sigma\alpha\nabla_{\tau_t} J(\tau_t) \tag{2.33}$$

onde α é o coeficiente de aprendizado já apresentado ao longo do trabalho e σ representa o quão distante o retorno obtido em um determinado instante de tempo está do valor do estado:

$$\sigma = G_t - v(s_t) \tag{2.34}$$

Segundo Sutton et al. (2000), a partir de τ a atualização da política a ser estimada é feita através da distribuição *soft-max*:

$$\pi(a|s,\tau) = \frac{e^{\tau^T \boldsymbol{x}(s,a)}}{\sum_b e^{\tau^T \boldsymbol{x}(s,a)}} \tag{2.35}$$

onde $\boldsymbol{x}(s,a)$ é chamado de vetor de *features*, que contém todo espaço de estados e ações.

Desse modo, os passos ilustrados para o algoritmo *REINFORCE* podem ser re-

sumidos abaixo:

1. Inicialize com uma política arbitrária $\pi(a|s,\tau)$;

2. Execute o modelo utilizando a política atual;

3. Armazene as distribuições de probabilidades logarítmicas e as recompensas em cada instante de tempo;

4. Calcule G_t de acordo com a equação (2.19)

5. Estime $v(s_t)$ e compute σ;

6. Compute o gradiente da política e atualize o parâmetro da política através das equações (2.32) e (2.33);

7. Repita os passos de 2-6

2.10.3 Algoritmo Deep Q-Network

Em sua pesquisa, Mnih et al. (2015) apresentam o algoritmo denominado *Deep Q-Network* resolvendo jogos de *Atari* para mostrar o potencial de solução dessa técnica, que combina um algoritmo clássico de aprendizado por reforço *Q-Learning* com Redes Neurais Artificias. A ideia central deste método é encontrar a função q_* apresentada na equação (2.30), pois com ela é possível determinar o máximo retorno G_t que pode ser obtido a partir de um estado s, tomando uma ação a e seguindo a política ótima a partir de então.

A ideia central do algoritmo *Q-Learning* é utilizar a equação 2.30 de maneira iterativa, pois, segundo Mnih et al. (2013), a medida que $i \rightarrow \infty$:

$$q_{i+1}(s,a) \leftarrow \mathbb{E}[R_{t+1} + \gamma q_i(S_{t+1}|S_t = s, A_t = a)]. \tag{2.36}$$

temos que $q_i \rightarrow q_*$.

Porém, conforme apresentado por Mnih et al. (2015), para a maior parte dos problemas reais é impraticável representar a função q em uma tabela com os valores de todas as combinações de s e a. A partir disso, os autores utilizam uma aproximação de função através de uma rede neural com parametros ψ para estimar

os valores q. Isso pode ser feito minimizando o erro da equação:

$$L_i(\psi_i) = \mathbb{E}_{s,a,r,s'\ \rho}[(y_i - q(s,a,w_i))^2] \tag{2.37}$$

onde $y_i = r + \gamma max_{a'} q(s^{'}, a^{'}, \psi_{i-1})$ e é denominado como a diferença temporal e $y_i - q$ o erro temporal. ρ representa a distribuição das transições $s, a, r, s^{'}$ obtida pelo meio e w_i é um parâmetro de atualização das estimativas dos valores de q.

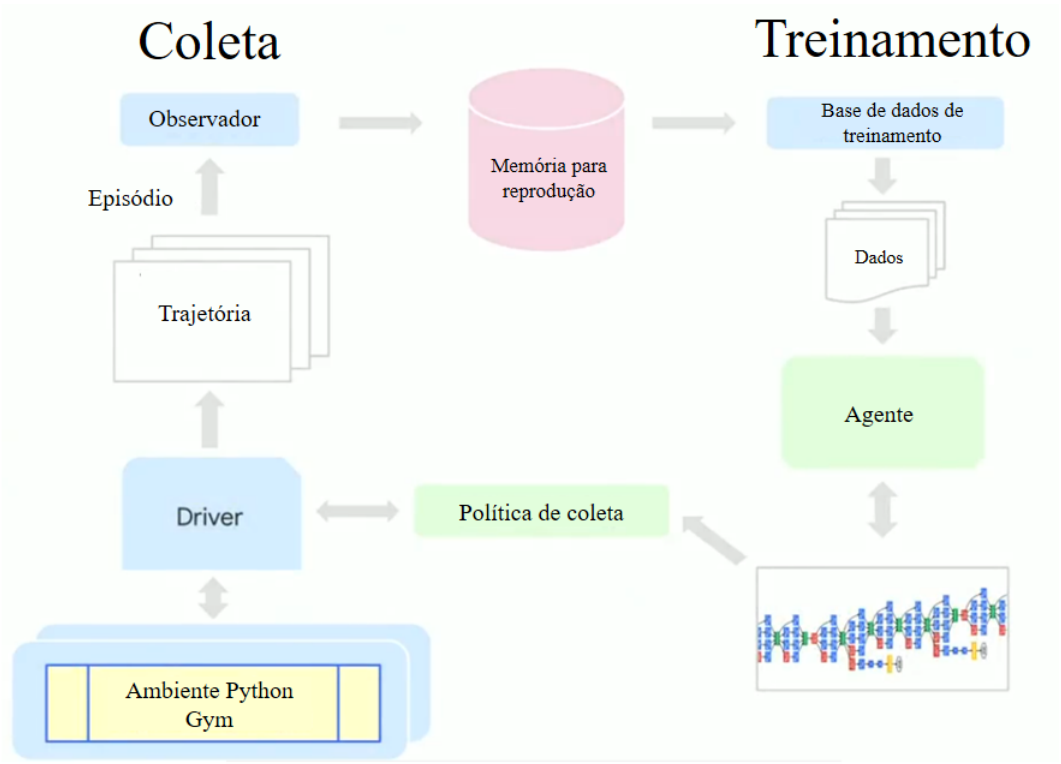
Para evitar computar todos os valores do erro *Deep Q-Network*, Mnih et al. (2013) o minimizam usando o método de gradiente descendente estocástico.

2.11 *TensorFlow Agents*

Segundo Abadi et al. (2016), alguns ambientes foram criados nos últimos anos com o objetivo de facilitar a implementação de modelos de aprendizado de máquinas. Um deles é o *TensorFlow*, uma aplicação criada pelo *Google* que possui diversos algoritmos disponíveis para a criação dos modelos. Um dos ambientes desenvolvidos é o *TensorFlow Agents*, que facilita o projeto, implementação e testes de algoritmos baseados em aprendizagem por reforço.

Na figura 8 é possível ver o método de funcionamento do *TensorFlow Agents*:

Figura 8: Diagrama de funcionamento do *TensorFlow Agents*.



Adaptado de Géron (2019) e traduzido pelo autor

Segundo Hafner, Davidson e Vanhoucke (2017), o processo de treinamento de um modelo acontece em duas etapas: a primeira é a coleção de episódios, onde o agente *driver* atua no ambiente por alguns episódios utilizando sua política atual, chamada de política de coleta e todas as trajetórias são salvas em um memória de computador, atuando como um observador das trajetórias do agente. A segunda etapa consiste no treinamento, onde as trajetórias são utilizadas para gerar um conjunto de dados de treinamento como entrada das redes neurais do modelo.

Com base nisso, é possível criar redes neurais de maneira mais fácil e criar agentes que utilizam os algoritmos citados nas seções anteriores para realizar o treinamento de acordo com o procedimento exemplificado acima.

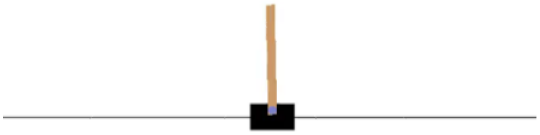
3 IMPLEMENTAÇÃO DOS ALGORITMOS DE APRENDIZAGEM POR REFORÇO

Esse capítulo abordará os passos utilizados para implementação dos três algoritmos de aprendizagem por reforço introduzidos na seção anterior. Inicialmente, será apresentado o modelo dinâmico utilizado através da biblioteca *Gym* e a modelagem do problema de aprendizagem por reforço. Após isso, a implementação dos três algoritmos será descrita.

3.1 Ambiente computacional *Gym Cartpole-v1*

Com o objetivo de estimular as pesquisas em aprendizado por reforço a nível mundial, foi criado um pacote denominado *Open AI Gym* na linguagem *python*, conforme apresentado por Brockman et al. (2016). Essa biblioteca, ainda de acordo com os autores, combina vários ambientes diferentes para que a comunidade científica possa utilizar como *benchmarking* de algoritmos de aprendizado por reforço. O problema do pêndulo invertido é um dos ambientes disponíveis para estudo no pacote, recebendo o nome de *Cartpole-v1*. Esse pacote será utilizado ao longo da implementação dos algoritmos de aprendizado por reforço neste trabalho por possuir a física do problema já implementada.

Figura 9: Ambiente *Cartpole-v1* do *OpenAI Gym*



No ambiente *Cartpole-v1*, o modelo não-linear do problema utilizado é o

mesmo representado pelas equações 2.1 e 2.2. Os seguintes parâmetros do modelo serão utilizados no presente trabalho:

Tabela 1: Valores dos parâmetros utilizados no trabalho

Parâmetro	Valor utilizado
g	$9.8 m/s^2$
M	$1 kg$
m	$0.1 kg$
l	$0.5 m$
F	$10 N$

Além disso, o ambiente computacional utiliza um intervalo de 200 instantes de tempo com intervalos de $0,02 s$ como discretização.

3.2 Formulação do problema de aprendizagem por reforço

De acordo com o que foi apresentado na seção anterior, um problema de aprendizagem por reforço consiste em um agente que possui um estado específico em um meio, e realiza um conjunto de ações em busca de uma recompensa. No problema do pêndulo invertido deste trabalho, o objetivo do agente é equilibrar o pêndulo na posição vertical. Para esse caso, serão consideradas as premissas adotadas no ambiente *Gym Cart-pole v1* que consistem em (i) manter a posição angular do pêndulo dentro do intervalo $\pm 12^o$ a partir da posição vertical e (ii) manter a posição linear dentro dos limites de $\pm 2.4 m$ a partir do centro do eixo. Conforme apresentado nas equações 2.1 e 2.2, as variáveis que representam o estado do agente são $[\theta, \dot{\theta}, x, \dot{x}]$.

O problema do pêndulo invertido também pode ser descrito como um processo de decisão de Markov. No caso, consiste em:

i. S, um conjunto finito de variáveis representando o estado do agente, $[\theta, \dot{\theta}, x, \dot{x}]$;

ii. Um conjunto finito de ações A: mover o carro para a esquerda ou para a direita aplicando a força $-F, F$;

iii. P, uma matriz de transição de estados apresentada na equação 2.24, já que como o meio é determinístico, uma ação a a partir de um estado s sempre resultará em um novo estado $s^{'}$;

iv. R, uma função de recompensas, onde é dada uma recompensa de +1 para cada instante de tempo que o agente mantém o pêndulo nas condições de $\pm 12^o$ e $\pm 2.4m$ e 0 caso não mantenha;

v. Um coeficiente de aprendizado γ no intervalo $[0,1]$, cujo valor dependerá do algoritmo utilizado

O problema será considerado episódico, com cada episódio com tamanho $T = 200$ instantes de tempo. Assim, 200 instantes de tempo serão computados a cada episódio e o agente tomará 200 ações, podendo obter recompensa máxima no episódio de 200. Caso o número de recompensas esteja acima de 195, será considerado sucesso durante a atuação no episódio. Caso a posição angular ou a posição linear esteja fora dos limites definidos, o episódio terminará no mesmo instante.

Assim, o problema pode ser enunciado matematicamente da seguinte forma: seja um pêndulo invertido num carro, conforme o ambiente *Gym Cart-pole v1*, e a recompensa num instante t é definida por:

$$R(t) = \begin{cases} +1 & \text{se} -12^\circ \leq \theta \leq 12^\circ \text{e} -2,4m \leq x \leq 2,4m, \\ 0 & \text{se qualquer outro caso} \end{cases} \tag{3.1}$$

O problema é

$$G_T = \sum_{t=1}^{T} R(t) \rightarrow \max, \tag{3.2}$$

com as restrições

$$T = 200 \tag{3.3}$$

$$u(t) = \{-F; F\} \tag{3.4}$$

3.3 Treinamento e validação

Durante a etapa de treinamento, os três algoritmos apresentados no capítulo anterior serão utilizados: *HillClimbing*, *REINFORCE* e *Deep-Q Network*. Como condição inicial para equilíbrio, as quatro variáveis de estado $\theta, \dot{\theta}, x, \dot{x}$ serão inicializadas a cada episódio com um valor aleatório dentro do intervalo $[-0.05, 0.05]$. Essa inicialização com valores aleatórios tem como objetivo evitar possível vício em uma solução, como comentado por Sutton e Barto (2020). O treinamento de cada algoritmo será feito com um número máximo episódios em cada caso, porém, caso o

valor total de recompensas médias dos últimos 10 episódios for maior que 200.0 será finalizado o programa.

Após o treinamento, será avaliado o desempenho de cada algoritmo comparando o valor total de recompensas obtidas a cada episódio, com o objetivo de avaliar o número de episódios necessários para conclusão do treinamento. O valor inicial de θ será de $0,02042\,rad$ com o objetivo de avaliar a resposta do sistema para cada um dos três algoritmos de acordo com a perturbação inicial. Por limitações da biblioteca utilizada, as outras variáveis de estado serão inicializadas com uma magnitude dentro de um intervalo de $[-0,05; 0,05]$.

3.4 *HillClimbing*

Como o método trabalha com a estimativa de uma política que mapeia os estados e as ações que devem ser seguidas pelo agente, serão utilizados pesos **w** que serão multiplicados pelo vetor estados $\mathbf{s} = [x, \dot{x}, \theta, \dot{\theta}]^T$ e após isso será aplicada uma regressão logística para direcionar a ação $\mathbf{a} = -10, 10$ que o agente deve tomar.

A matriz de pesos **w** terá a seguinte forma:

$$\mathbf{w} = \begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \end{bmatrix} \tag{3.5}$$

Dessa forma, com o produto entre a matriz pesos **w** e o vetor de estados **s**, aplica-se a regressão logística como função de ativação (função *softmax*). Assim, a saída do sistema é um vetor $\mathbf{y} = [a_1 \qquad a_2]^T$ da seguinte forma::

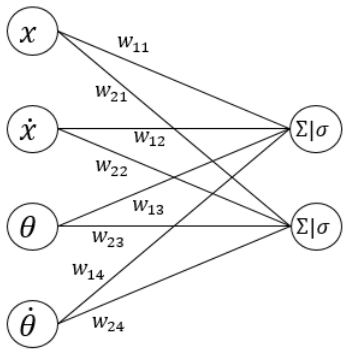
$$\mathbf{y}^T = \begin{bmatrix} \dfrac{e^{w_{11}x + w_{12}\dot{x} + w_{13}\theta + w_{14}\dot{\theta}}}{e^{w_{11}x + w_{12}\dot{x} + w_{13}\theta + w_{14}\dot{\theta}} + e^{w_{21}x + w_{22}\dot{x} + w_{23}\theta + w_{24}\dot{\theta}}} \\ \dfrac{e^{w_{21}x + w_{22}\dot{x} + w_{23}\theta + w_{24}\dot{\theta}}}{e^{w_{11}x + w_{12}\dot{x} + w_{13}\theta + w_{14}\dot{\theta}} + e^{w_{21}x + w_{22}\dot{x} + w_{23}\theta + w_{24}\dot{\theta}}} \end{bmatrix} \tag{3.6}$$

com as estimativas da melhor ação a ser tomada, sendo a_1 a aplicação da força $+10N$ e a_2, $-10N$. Dessa forma, a política tomará a maior probabilidade como ação à ser realizada pelo agente. Assim, pode-se descrever a política como:

$$\pi(s) = \max(\mathbf{y}) \tag{3.7}$$

De modo geral, o procedimento acima pode ser simplificado pela rede neural abaixo:

Figura 10: Rede Neural para estimativa da política π



O algoritmo *HillClimbing* será utilizado como procedimento de otimização para maximizar o retorno G_T através dos ajustes da matriz **w** conforme apresentado no capítulo anterior. Os seguintes parâmetros serão utilizados:

Tabela 2: Valores dos parâmetros utilizados no algoritmo *HillClimbing*

Parâmetro	Valor utilizado
Número máximo de episódios	1000
γ	1
σ	0,01

onde o número máximo de episódios é selecionado baseado em treinamentos observados em Nagendra et al. (2017), γ definido como 1 pois trata-se de um problema episódico e escala de ruído $0,01$ como convenção conforme apresentado por Sutton e Barto (2020).

3.5 *REINFORCE*

Com o objetivo de maximizar a equação 2.32, será construído um agente utilizando a *API TensorFlow Agents*, que é uma *Aplication Programming Interface (API)* que fornece uma biblioteca de métodos de aprendizagem por reforço no ambiente *TensorFlow* do *Google*.

Para estimativa da função de valor $v(s_t, \boldsymbol{w})$ para cálculo de 3.9 será utilizada uma Rede Neural Artificial com nós totalmente conectados, onde $\boldsymbol{w}$ é a matriz de pesos da rede neural, que possui os parâmetros abaixo:

Tabela 3: Rede neural artificial utilizada para o algoritmo *REINFORCE*

Camada	Descrição
Camada de entrada	4 elementos do estado do sistema
Camada escondida	100 unidades com função de ativação *softmax*
Camada de saída	2 unidades com estimativa de $v(s, \boldsymbol{w})$

onde, a cada etapa de treinamento, dois episódios são executados e são coletadas as trajetórias contendo o estado anterior, a ação tomada pelo agente pela política atual e o novo estado:

$$\boldsymbol{\tau_t} = [\boldsymbol{s_t}, a_t, \boldsymbol{s_{t+1}}] \tag{3.8}$$

E, essas trajetórias são salvas no *buffer* e utilizadas como entrada para a rede neural atualizar seus pesos.

Para a otimização dos pesos da Rede Neural, os seguintes métodos de otimização foram utilizados: Gradientes Descendentes, Otimizador *Adam* e Otimizador *Adagrad*. Para os três métodos de otimização, várias sessões de treinamentos serão feitas alterando o coeficiente de aprendizado α para avaliar o comportamento de cada método e eficácia do treinamento.

O treinamento através da *API TensorFlow Agents* será realizado com os seguintes parâmetros:

Tabela 4: Valores dos parâmetros utilizados no algoritmo *REINFORCE*

Parâmetro	Valor utilizado
Número máximo de iterações	250
Quantidade de episódios executados por iteração	2

Após atualização, é calculado:

$$\sigma = G_t - v(\boldsymbol{S_t}, \boldsymbol{w}) \tag{3.9}$$

E com σ é feita a atualização do parâmetro da política conforme equação 2.33

3.6 *Deep-Q Network*

Com o objetivo de minimizar o erro quadrático médio apresentado na equação 2.37, o valor das ações $q(s,a,w_i)$ será estimado através de uma rede neural com matriz pesos w_i construída através da *API TensorFlow Agents*.

A Rede Neural Artificial será criada seguindo o modelo de rede neural totalmente conectada com os parâmetros abaixo:

Tabela 5: Rede neural artificial utilizada para estimativa de q

Camada	Descrição
Camada de entrada	4 elementos do estado do sistema
Camada escondida	100 unidades com função de ativação *reLU*
Camada de saída	2 unidades com estimativa de $q(s,a,\boldsymbol{w})$

E para a otimização dos pesos w_i da Rede Neural, os seguintes métodos de otimização foram utilizados: Gradientes Descendentes, Otimizador *Adam* e Otimizador *Adagrad*. Para os três métodos, várias sessões de treinamentos serão feitas alterando o coeficiente de aprendizado α para avaliar o comportamento de cada método e eficácia do treinamento.

O treinamento através da *API TensorFlow Agents* será realizado com os seguintes parâmetros:

Tabela 6: Valores dos parâmetros utilizados no algoritmo *DeepQ Network*

Parâmetro	Valor utilizado
Número máximo de iterações	20000
Quantidade de episódios executados por iteração	2

O número maior de iterações escolhido para esse método se dá pela necessidade de um maior número de episódios para a estimativa da função ação-valor observada no trabalho de Duan et al. (2016).

Assim, a cada etapa de treinamento, um episódio é executado e são coletadas as trajetórias contendo o estado anterior, a ação tomada pelo agente pela política atual e o novo estado:

$$\boldsymbol{\tau_t} = [\boldsymbol{s_t}, a_t, \boldsymbol{s_{t+1}}] \tag{3.10}$$

E, essas trajetórias são salvas no *buffer* e utilizadas como entrada para a rede neural artificial poder estimar os valores das ações q.

Com as atualizações dos valores da matriz q, o erro da equação 2.37 é calculado e a atualização é feita utilizando o método de gradiente descendente estocástico.

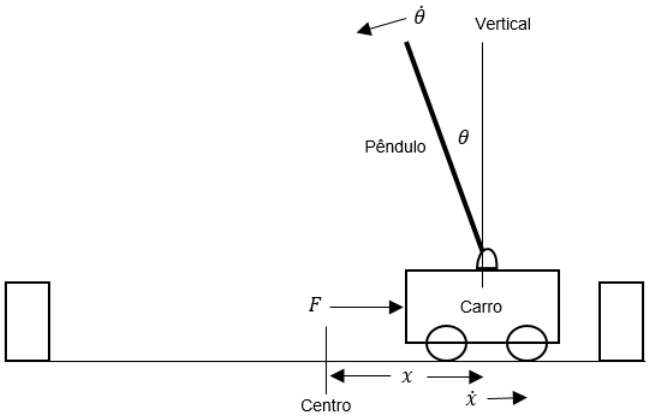
4 MODELAGEM E CONTROLE POR ALOCAÇÃO DE POLOS

Esse capítulo abordará os passos utilizados para o controle do pêndulo invertido utilizando a técnica de alocação de polos. Inicialmente, será apresentada a modelagem em espaços de estados do sistema dinâmico e em sequência a abordagem de controle utilizada.

4.1 Modelagem em espaços de estados

Como introduzido na seção de revisão da literatura, o modelo dinâmico não-linear para o pêndulo invertido é:

Figura 11: Representação do problema do pêndulo invertido.



Fonte: (NAGENDRA et al., 2017) Adaptado e traduzido pelo autor

$$\ddot{\theta} = \frac{(M+m)g\sin\theta - \cos\theta[F + ml\dot{\theta}^2\sin\theta]}{(4/3)(M+m)l - ml\cos^2\theta} \tag{4.1}$$

$$\ddot{x} = \frac{F + ml(\dot{\theta}^2 \sin\theta - \ddot{\theta}\cos\theta)}{(M+m)} \tag{4.2}$$

Linearizando o sistema fazendo a consideração de pequenos ângulos, tem-se que $\sin\theta = \theta$, $\cos\theta = 1$, $\dot{\theta}^2 = 0$ e $\dot{\theta}\theta = 0$. Assim, são obtidas as equações linearizadas:

$$\ddot{\theta} = \frac{3(M+m)g\theta - 3F}{(4M+m)l} \tag{4.3}$$

$$\ddot{x} = \frac{-3mg\theta + 4F}{4M+m} \tag{4.4}$$

E, a partir das equações acima, é possível definir o modelo em espaço de estados abaixo:

$$\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}u \tag{4.5}$$

$$y = \boldsymbol{C}\boldsymbol{x} + Du \tag{4.6}$$

onde:

$$\boldsymbol{x} = \begin{bmatrix} x & \dot{x} & \theta & \dot{\theta} \end{bmatrix}^T \tag{4.7}$$

$$\dot{\boldsymbol{x}} = \begin{bmatrix} \dot{x} & \ddot{x} & \dot{\theta} & \ddot{\theta} \end{bmatrix}^T \tag{4.8}$$

$$\boldsymbol{A} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & \dfrac{-3mgl}{4m+m} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & \dfrac{3(M+m)g}{(4M+m)l} & 0 \end{bmatrix} \tag{4.9}$$

$$\boldsymbol{B} = \begin{bmatrix} 0 \\ \dfrac{4}{4M+m} \\ 0 \\ \dfrac{-3}{(4M+m)l} \end{bmatrix} \tag{4.10}$$

$$\boldsymbol{C} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{4.11}$$

e $\boldsymbol{D} = 0$ e $u = F$ é a entrada do sistema.

Considerando os mesmos valores de parâmetros utilizados na resolução com aprendizagem por reforço:

Tabela 7: Valores dos parâmetros utilizados no trabalho

Parâmetro	Valor utilizado
g	$9.8m/s^2$
M	$1kg$
m	$0.1kg$
l	$0.5m$

Obtêm-se:

$$\boldsymbol{A} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & -0,7171 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 15,7751 & 0 \end{bmatrix} \tag{4.12}$$

$$\boldsymbol{B} = \begin{bmatrix} 0 \\ 0,9751 \\ 0 \\ -1,4634 \end{bmatrix} \tag{4.13}$$

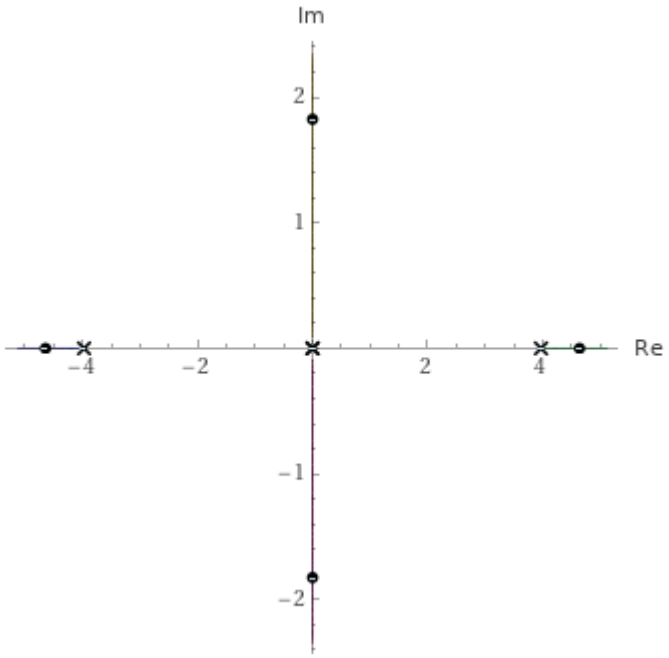
4.1.1 Análise do Sistema

A partir do modelo em espaço de estados, é possível perceber através das raízes da equação abaixo a localização dos polos do sistema:

$$det(sI - A) = 0 \tag{4.14}$$

onde I representa a matriz de identidade. Através da equação 4.14, obtêm-se dois polos complexos conjugados com parte real nula e dois polos reais, um em $-3,971$ e outro no plano direito em $3,971$, o que implica em um sistema instável. A imagem abaixo representa o lugar geométrico das raízes:

Figura 12: Lugar geométrico das raízes para o sistema modelado.



Para verificar a controlabilidade do sistema, a matriz de controlabilidade foi calculada:

$$\boldsymbol{T} = \begin{bmatrix} 0 & 0,975 & 0 & 1,049 \\ 0,9751 & 0 & 1,0494 & 0 \\ 0 & -1,463 & 0 & -23,085 \\ -1,463 & 0 & -23,085 & 0 \end{bmatrix} \tag{4.15}$$

E realizando o posto da matriz T obtem-se o valor 4, igual ao número de estados

do sistema. Dessa forma, de acordo com Ogata e Yang (2002), é garantido que o sistema linear é controlável.

4.1.2 Sistema de controle

Uma vez que o sistema seja controlável, é possível utilizar a técnica de alocação de polos para criar um controlador, utilizando $u = -Kx$. No entanto, para efeitos de comparação posteriores com os resultados obtidos com a modelagem com aprendizado por reforço, será adicionado uma limitação no atuador do sistema de modo que a entrada seja $u = -F$ ou $u = F$. Sendo assim, caso $u = -Kx$ for menor que zero, a atuação será de $u = -10N$, caso contrário, $u = 10N$.

Como existe uma limitação na atuação do sistema, os requisitos adotados serão determinados de tal forma que o sistema consiga obter o menor sobressinal MS e tempo de acomodação T_s possíveis sem que haja saturação na entrada do sistema, ou seja, a entrada $u = -Kx$ deverá estar dentro do intervalo de $-10N$ e $+10N$, justamente para obter a melhor comparação com os sistemas resolvidos por aprendizagem por reforço. Os seguintes parâmetros serão testados:

Tabela 8: Variação de valores de tempo de pico e tempo de acomodação

Sobressinal	Tempo de acomodação
$MS = 5\%$	$T_s = 1,0s$
$MS = 8\%$	$T_s = 1,5s$
$MS = 9\%$	$T_s = 1,5s$
$MS = 10\%$	$T_s = 2,0s$
$MS = 10\%$	$T_s = 2,2s$

Com isso é possível encontrar as coordenadas dos polos dominantes através das equações:

$$MS = e^{\left(\frac{-\zeta\pi}{\sqrt{1-\zeta^2}}\right)} \tag{4.16}$$

$$T_s = \frac{4}{\zeta\omega_n} \tag{4.17}$$

$$\omega_d = \omega_n\sqrt{1-\zeta^2} \tag{4.18}$$

A partir disso, a localização dos polos dominantes são definidos como $\zeta\omega_n \pm j\omega_d$. De agordo com Ogata e Yang (2002) os outros dois polos precisam ser colocados em outro um ponto à esquerda bem distante dos polos dominantes. Sendo assim, são obtidos os seguintes valores:

Tabela 9: Polos necessários para cada requisito

Sobressinal	Tempo de acomodação	Polos dominantes	Localização dos outros polos
$MS = 5\%$	$T_s = 1,0s$	$-4 \pm j4,194$	-40 e -45
$MS = 8\%$	$T_s = 1,5s$	$-2,667 \pm j2,797$	-25 e -30
$MS = 9\%$	$T_s = 1,5s$	$-2,667 \pm j3,480$	-25 e -30
$MS = 10\%$	$T_s = 2,0s$	$-2 \pm j2,729$	-20 e -25
$MS = 10\%$	$T_s = 2,2s$	$-1,818 \pm j2,481$	-18 e -20

Após a definição da alocação de polos a matriz de ganho K é identificada. Com as matrizes de ganho para cada condição apresentada na tabela acima, o sistema é testado da mesma forma que será realizada com os algoritmos de aprendizagem por reforço.

Para observação da resposta do sistema, será realizada a simulação do modelo com discretização do tempo com intervalos de $0,02s$ e em 200 instantes de tempo, igual à simulação dos modelos de aprendizagem por reforço, com condições iniciais:

$$x = [0 \qquad 0 \qquad -0.02042 \qquad 0]^T \tag{4.19}$$

5 RESULTADOS

Nesse capítulo os resultados obtidos através das abordagens apresentadas no capítulo anterior serão apresentados. Inicialmente, será detalhada a etapa de treinamento dos três algoritmos, e, após isso, o comportamento do sistema em cada algoritmo será apresentado. Em sequência, serão apresentados os resultados obtidos através da resolução do problema através do método de alocação de polos. Por fim, será realizada uma comparação entre os resultados de cada método proposto.

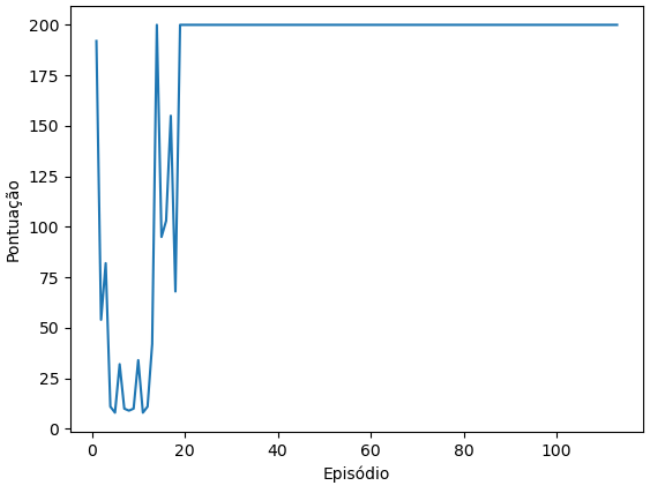
5.1 Treinamento com aprendizagem por reforço

Os algoritmos apresentados nas seções anteriores foram implementados usando a linguagem de programação *python*. Todos os códigos foram implementados no *Google Colab*, um serviço do *Google* que hospeda um *Jupyter Notebook* sem necessidade de configuração inicial e que pode ser acessada em Google... (). A a etapa de treinamento foi computada utilizando a *Tensor Processor Unit* (TPU) na nuvem para execução. O funcionamento dessa arquitetura em nuvem é apresentado por Shariar e Hasan (2020).

5.1.1 Hillclimbing

Na figura abaixo é possível observar a evolução das recompensas acumuladas durante um total de 100 episódios de treinamento. É possível notar que após 20 episódios o retorno acumulado G_t obtido foi de 200, valor máximo possível para o caso, mostrando que o algoritmo convergiu em um número pequeno de iterações.

Figura 13: Pontuação em relação ao número de episódio



Fonte: o autor

Após etapa de treinamento, a matriz peso w da política $\pi(s_t, w)$ obtida foi:

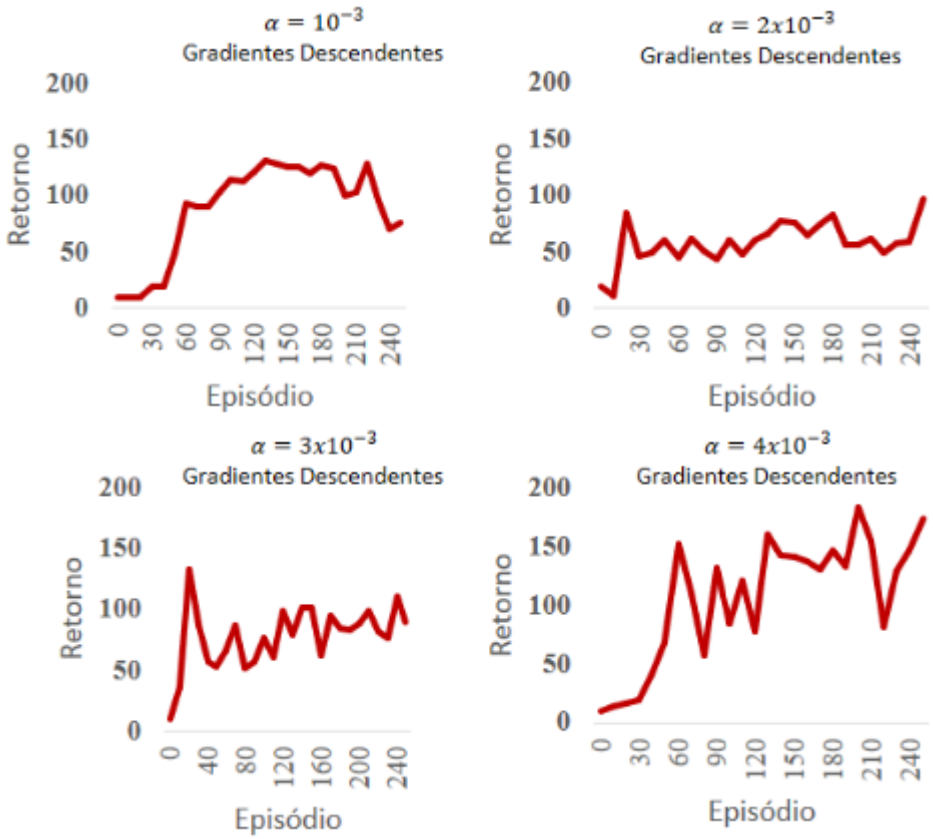
$$\mathbf{w} = \begin{bmatrix} 3,6701 & 3,0202 & 4,2271 & 2,8429 \\ 3,9922 & 4,7857 & 5,6427 & 4,5405 \end{bmatrix} \tag{5.1}$$

5.1.2 *REINFORCE*

Como apresentado na seção anterior, foram utilizados três métodos de otimização da rede neural artificial para estimativa da função valor $v(s_t, w)$: Método do Gradiente, Adam e Adagrad. Para cada um deles, foi realizada uma varição no coeficiente de aprendizado entre os intervalos de 0,001 a 0,010 em um total de 250 passos.

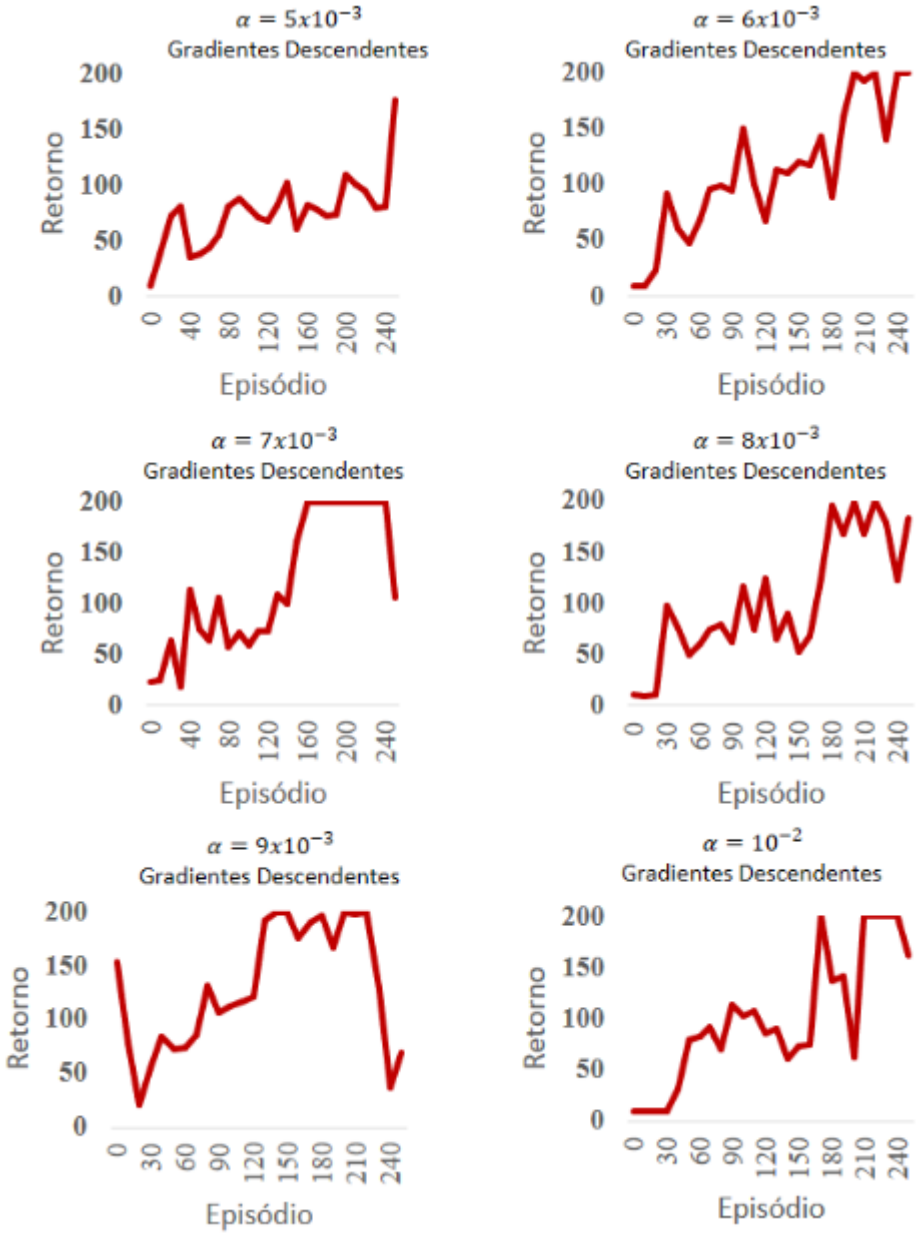
Os resultados obtidos na etapa de treinamento do algoritmo *REINFORCE* utilizando o método do gradiente como função de otimização da rede neural podem ser vistos na figura abaixo:

Figura 14: Treinamento utilizando método do gradiente com diferentes valores de α



Fonte: o autor

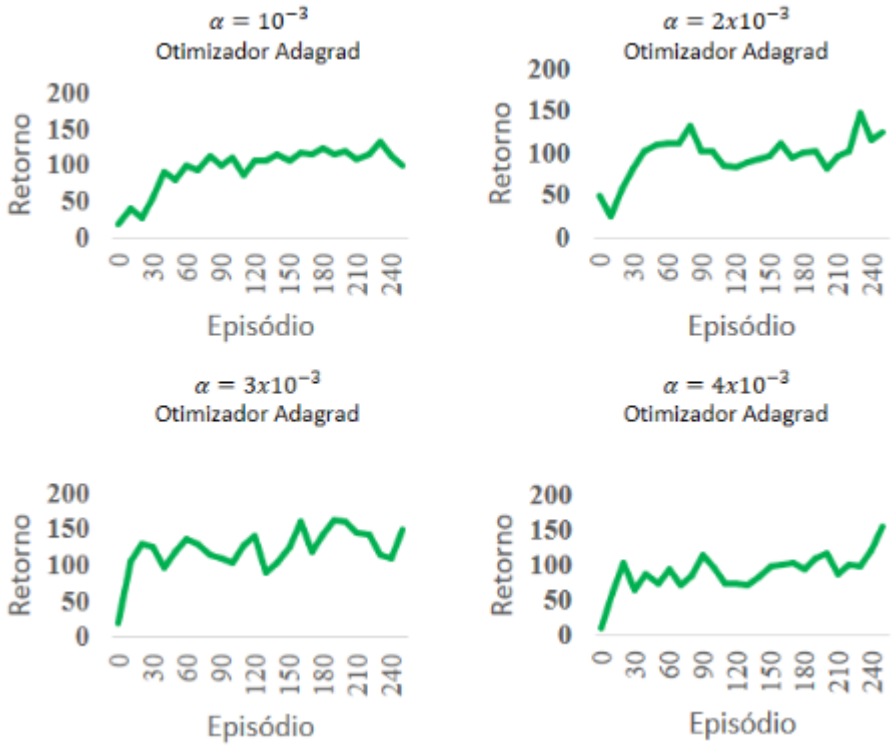
Figura 15: Treinamento utilizando método do gradiente com diferentes valores de α



Fonte: o autor

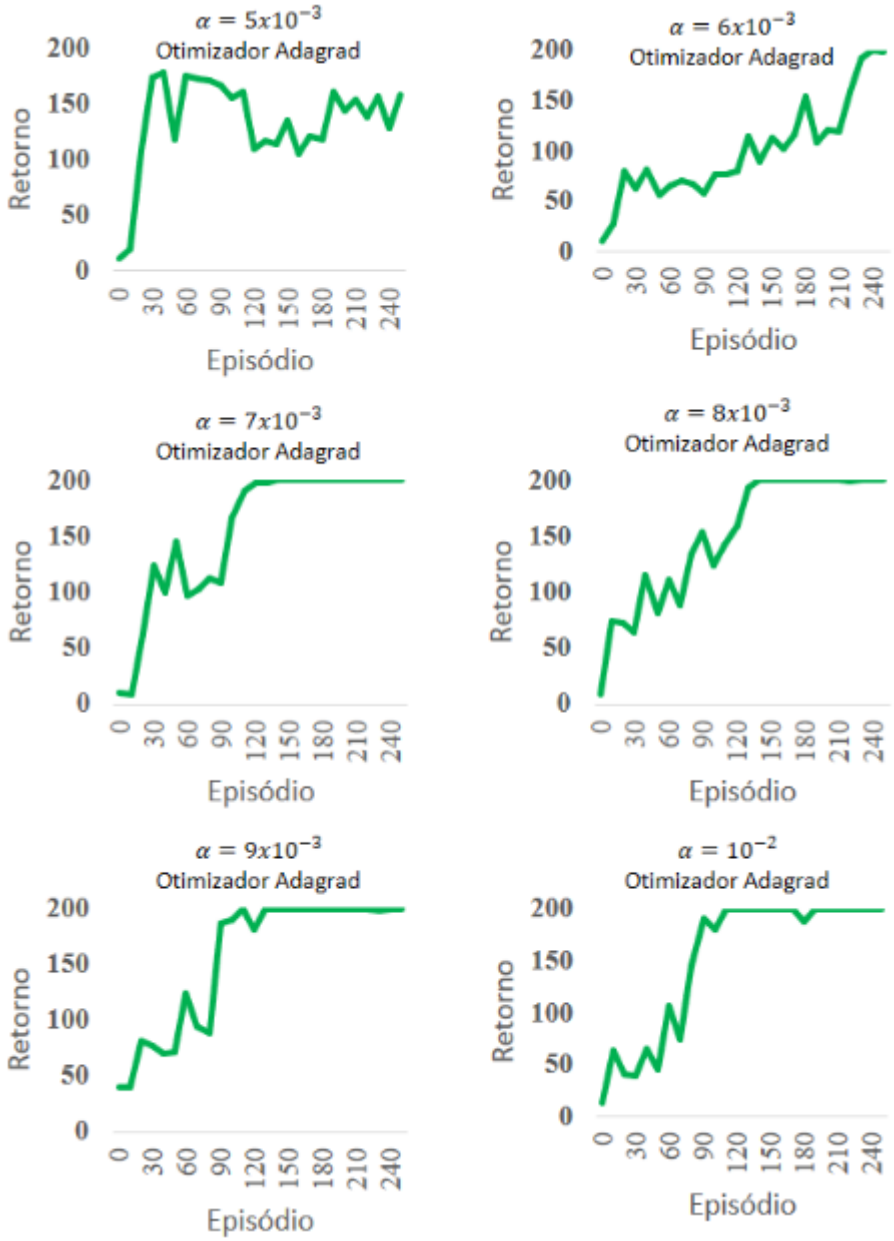
Já o treinamento realizado utilizando o Otimizador Adagrad:

Figura 16: Treinamento utilizando método dos Otimizador Adagrad para diferentes valores de α



Fonte: o autor

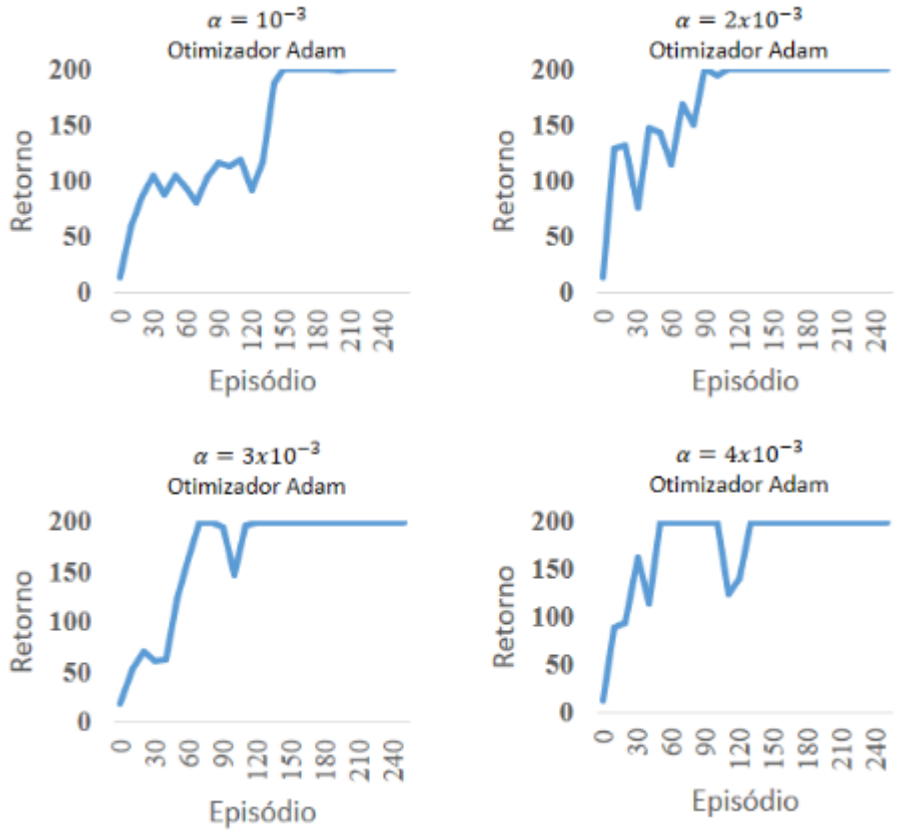
Figura 17: Treinamento utilizando método dos Otimizador Adagrad para diferentes valores de α



Fonte: o autor

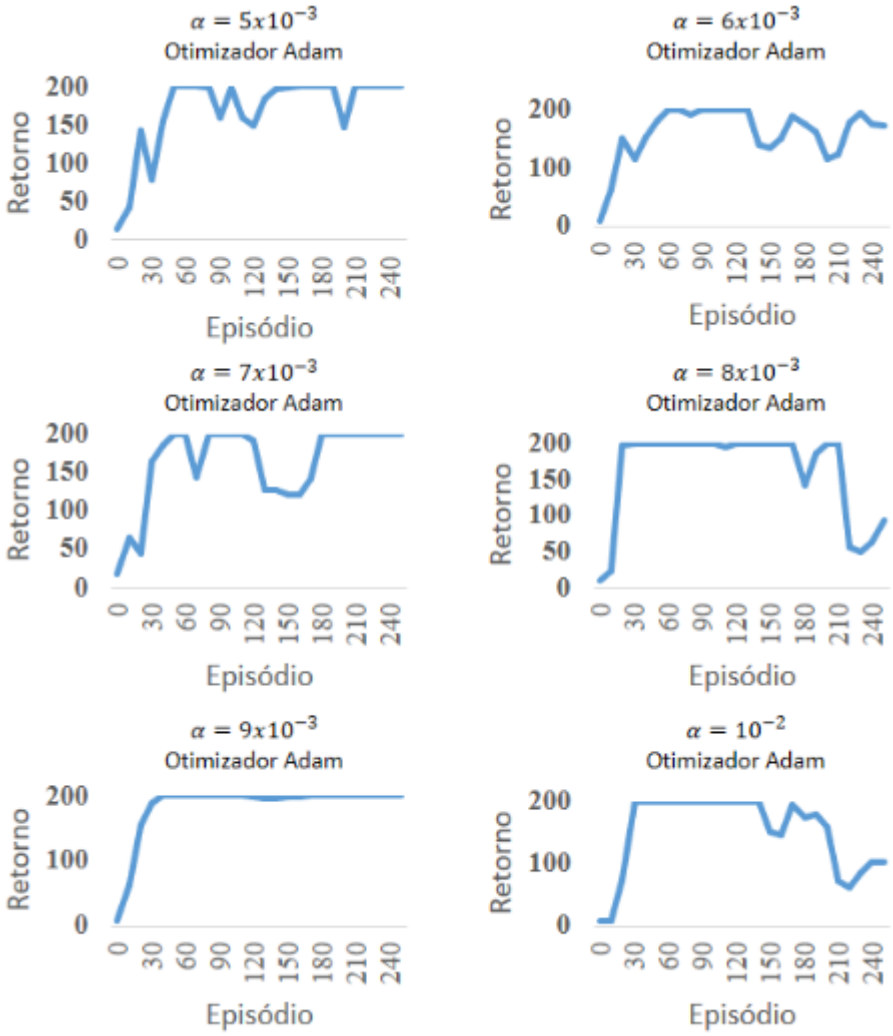
E, com o Otimizador Adam:

Figura 18: Treinamento utilizando método dos Otimizador Adam para diferentes valores de α



Fonte: o autor

Figura 19: Treinamento utilizando método dos Otimizador Adam para diferentes valores de α

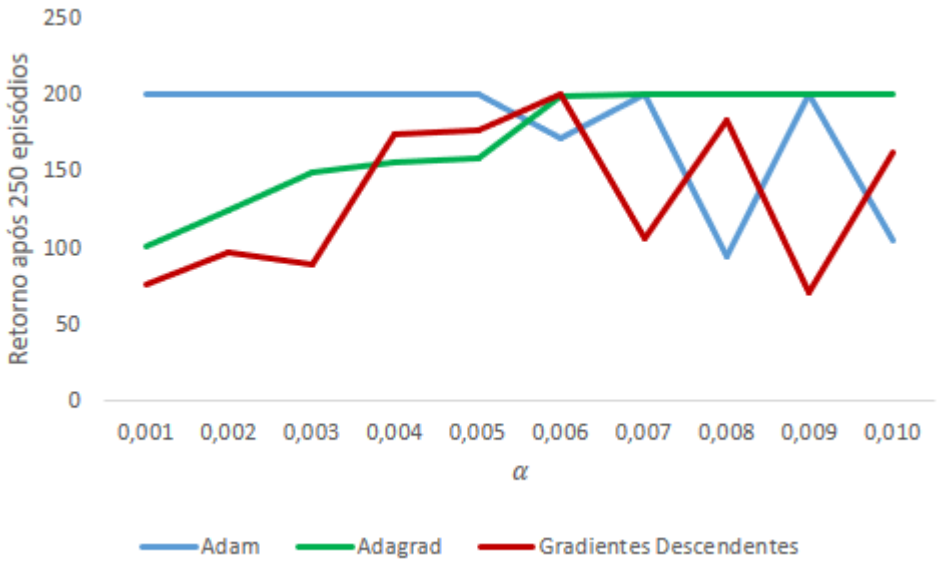


Fonte: o autor

É possível observar através dos resultados do treinamento que o método Adam obteve um melhor desempenho quando comparado com os outros métodos, já que para a maioria dos coeficientes de aprendizado utilizados foi possível atingir o retorno máximo com número de episódios menor que os outros métodos. O coeficiente de aprendizado com melhor desempenho para o método Adam foi de $9x10^{-3}$.

Na figura 20 pode-se ver a comparação entre o retorno acumulado após 250 episódios usando entre os três métodos em função do coeficiente de aprendizado:

Figura 20: Comparação entre os três métodos de otimização



Fonte: o autor

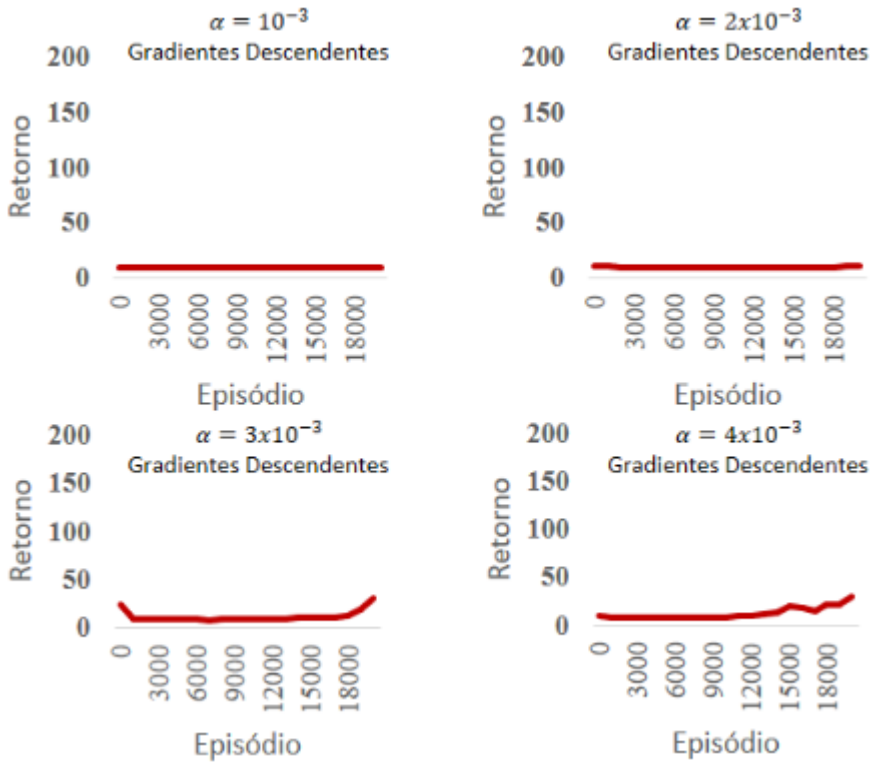
É possível observar que com apenas o coeficiente de aprendizado de 0,006 foi possível obter o resultado máximo com o método do gradiente. Além disso, o método *Adagrad* obteve melhores resultados com valores maiores do coeficiente de aprendizado, enquanto que o comportamento inverso foi observado para o método *Adam*.

5.1.3 *DeepQ Networks*

Para estimativa da matriz valor ação foi implementada uma rede neural artificial, conforme apresentado na seção anterior. Para treinamento da rede neural também foram utilizados os três métodos de otimização citados na seção anterior: Método do Gradiente, Adagrad e Adam. O coeficiente de aprendizado foi variado entre os intervalos de 0,001 a 0,010 em um total de 20.000 episódios.

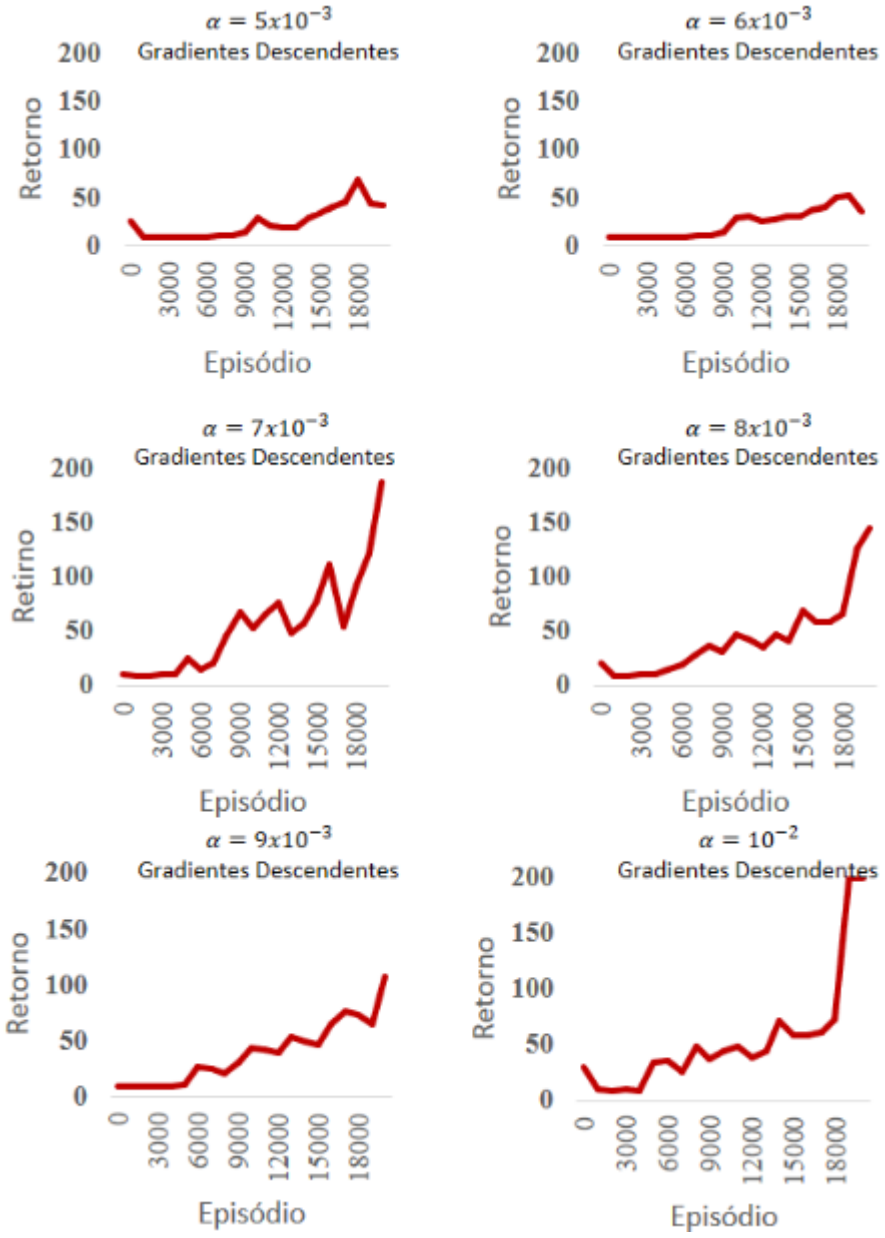
Os resultados obtidos na etapa de treinamento do algoritmo *DeepQ Networks* utilizando o método do gradiente como função de otimização da rede neural podem ser vistos na figura abaixo:

Figura 21: Treinamento utilizando método do gradiente para diferentes valores de α



Fonte: o autor

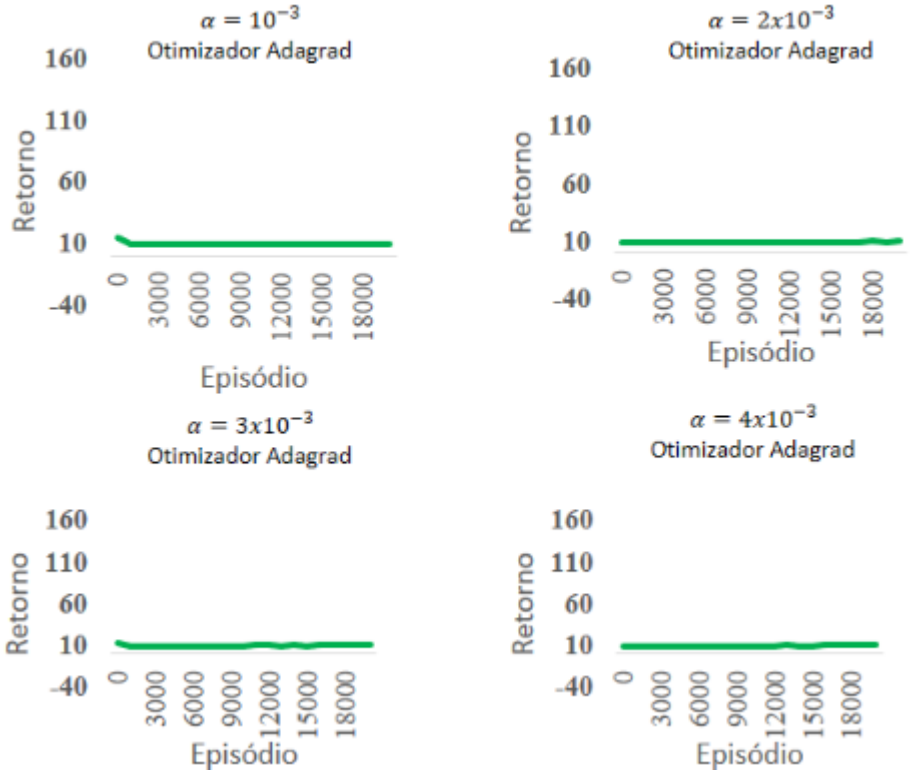
Figura 22: Treinamento utilizando método do gradiente para diferentes valores de α



Fonte: o autor

Já o treinamento realizado utilizando o Otimizador Adagrad:

Figura 23: Treinamento utilizando método dos Otimizador Adagrad para diferentes valores de α



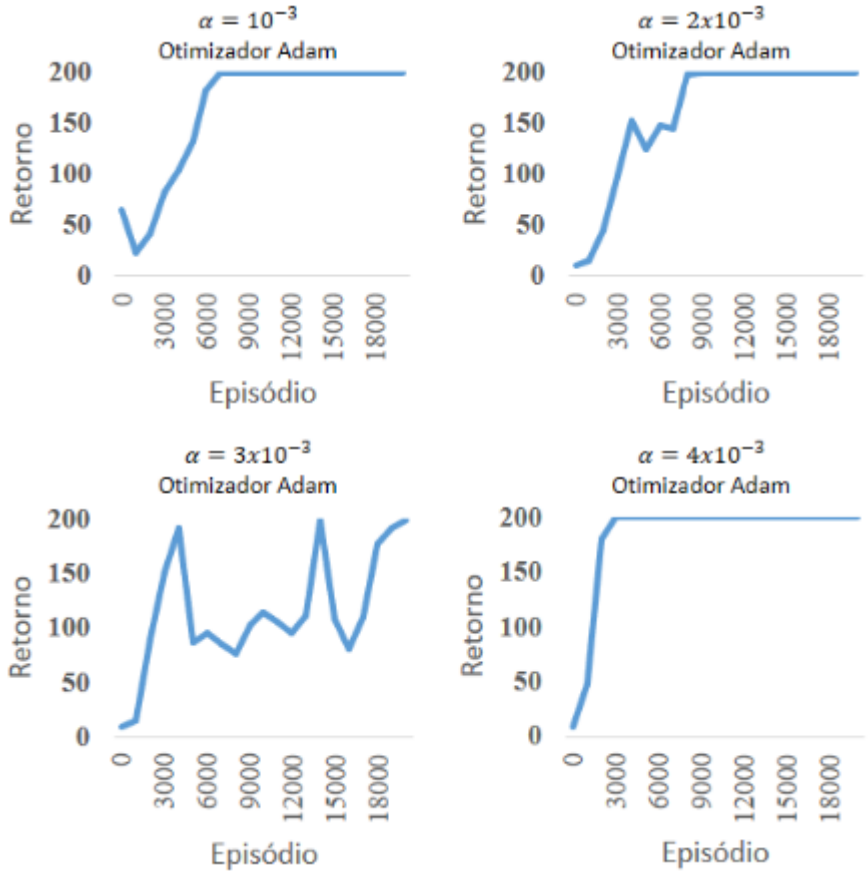
Fonte: o autor

Figura 24: Treinamento utilizando método dos Otimizador Adagrad para diferentes valores de α

Fonte: o autor

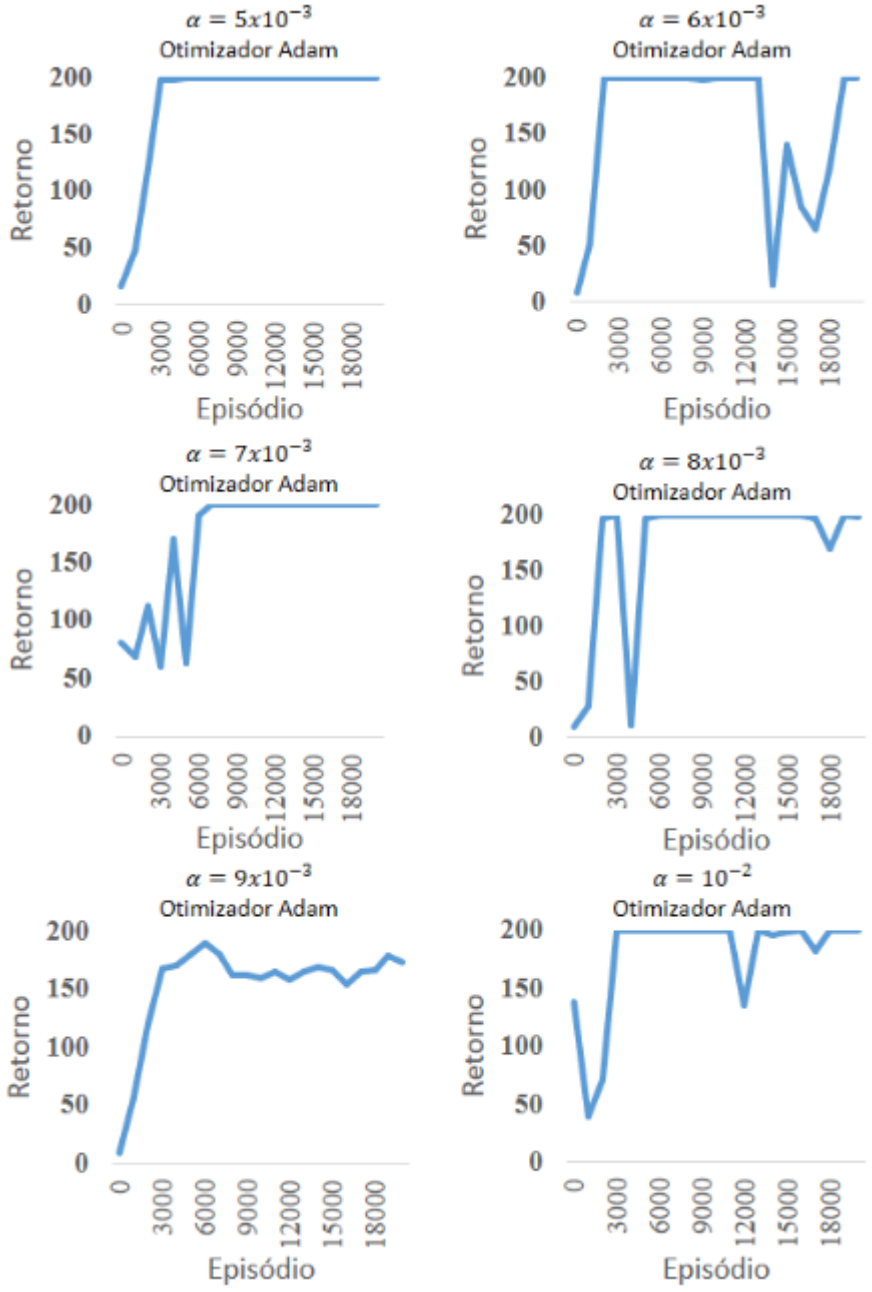
E, com o Otimizador Adam:

Figura 25: Treinamento utilizando método dos Otimizador Adam para diferentes valores de α



Fonte: o autor

Figura 26: Treinamento utilizando método dos Otimizador Adam para diferentes valores de α

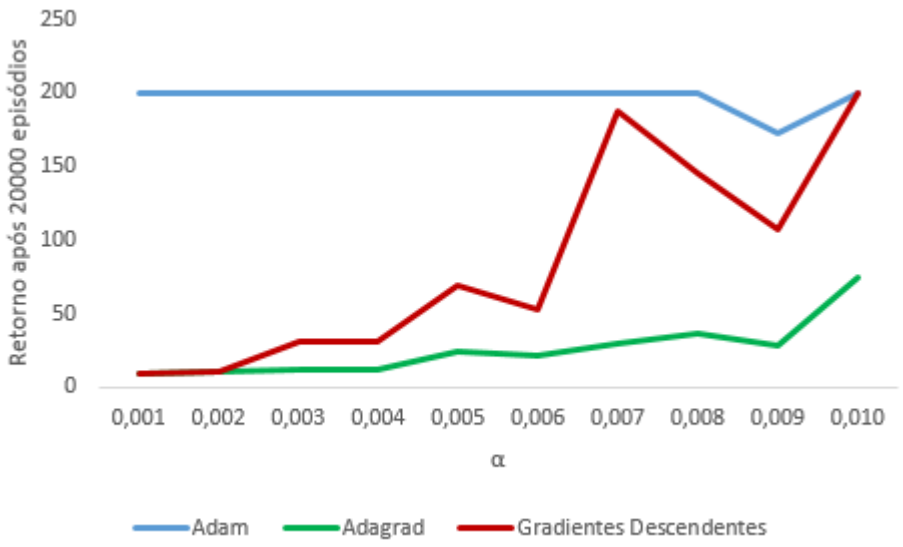


Fonte: o autor

Onde também é possível observar que o método Adam obteve um melhor desempenho em relação aos outros métodos. O coeficiente de aprendizado com melhor desempenho para o método Adam foi de $4x10^{-3}$. Além disso, o método *Adagrad* não conseguiu obter a recompensa máxima com nenhum dos valores de coeficientes de aprendizado utilizados.

Na figura 27 pode-se ver a comparação entre o retorno acumulado após 20.000 episódios usando entre os três métodos para otimização dos pesos da rede neural artificial em função do coeficiente de aprendizado:

Figura 27: Comparação entre os três métodos de otimização



Fonte: o autor

Através da comparação é possível ver que o método *Adam* consegue obter sucesso para 9 dos 10 valores de coeficiente de aprendizado testados. Já o método *Adagrad* não consegue obter recompensa máxima e o Método do Gradiente apenas com α de 0,010.

5.1.4 Comparação do treinamento dos três algoritmos

Todas as informações apresentadas acima podem ser resumidas na tabela 10:

Tabela 10: Resultados do treinamento

Algoritmo	α	**Otimizador da rede neural**	**Episódios para sucesso**	**Tempo de processamento até resolução**
HillClimbing	1	escala adaptativa de ruído	21	0,5s
REINFORCE	0,009	Otimizador Adam	50	88s
Deep-Q Network	0,004	Otimizador Adam	3000	57s

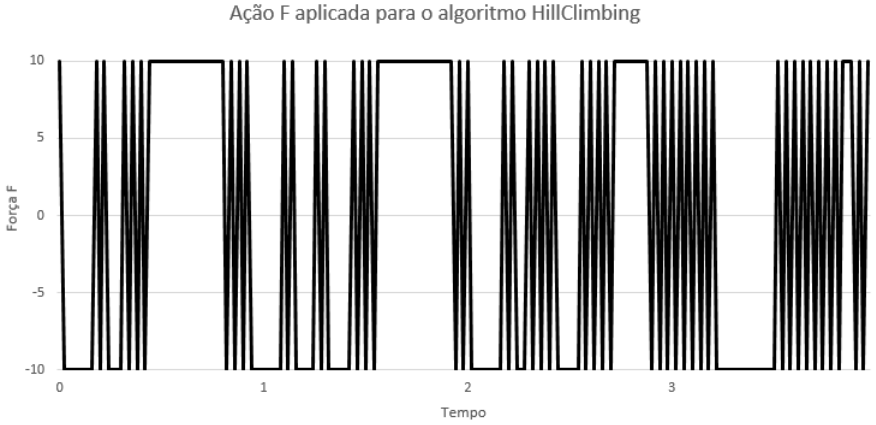
É possível observar que o treinamento do método *HillClimbing* precisou de 21 episódios em um tempo de processamento de 0,5s para obter sucesso. Já o algoritmo *Deep-Q Network* necessitou de mais episódios porém com um tempo de processamento inferior ao observado do método *REINFORCE*.

5.2 Avaliação

Após etapa de treinamento, os agentes de cada algoritmo foi colocado à prova para resolver um episódio de 200 instantes de tempo cada com condição inicial de θ_0 com um valor aleatório no intervalo de $[-0.05, 0.05]$.

O agente treinado com o algoritmo *HillClimbing* apresentou o seguinte comportamento de entrada:

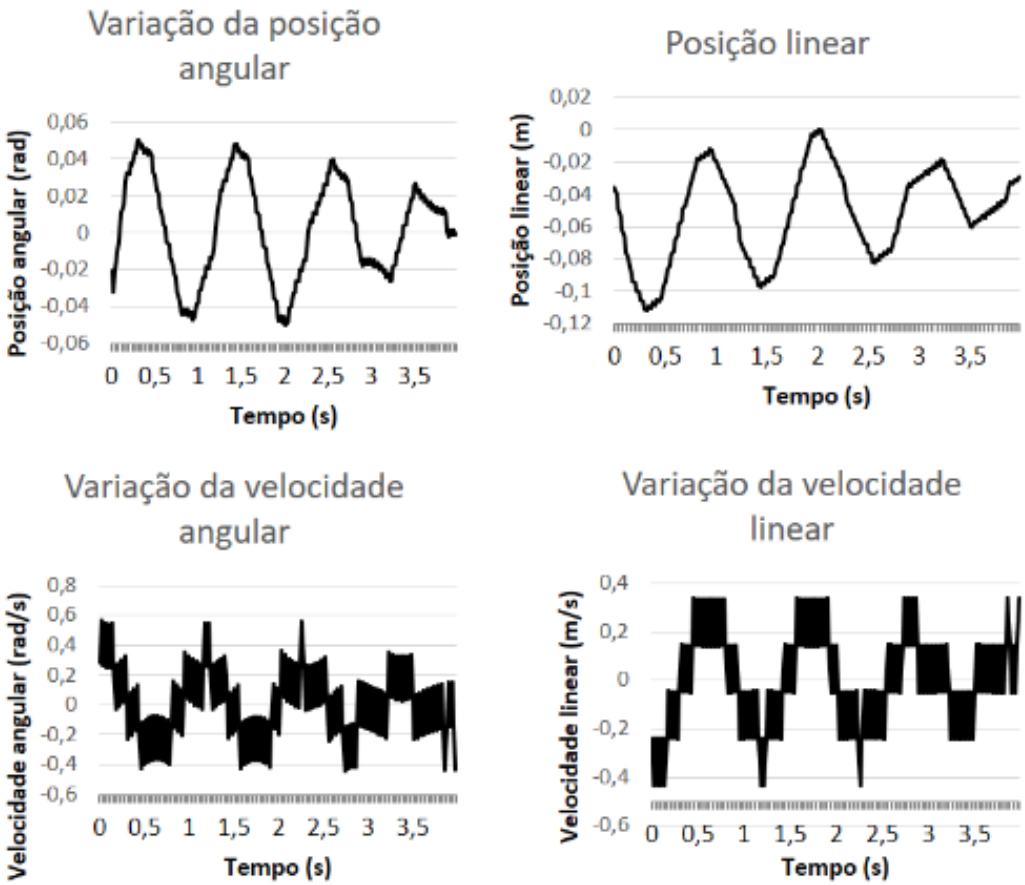
Figura 28: Entrada do agente treinado pelo método *HillClimbing*



Fonte: o autor

E possuiu a seguinte resposta:

Figura 29: Resposta do agente treinado pelo método *HillClimbing*



Fonte: o autor

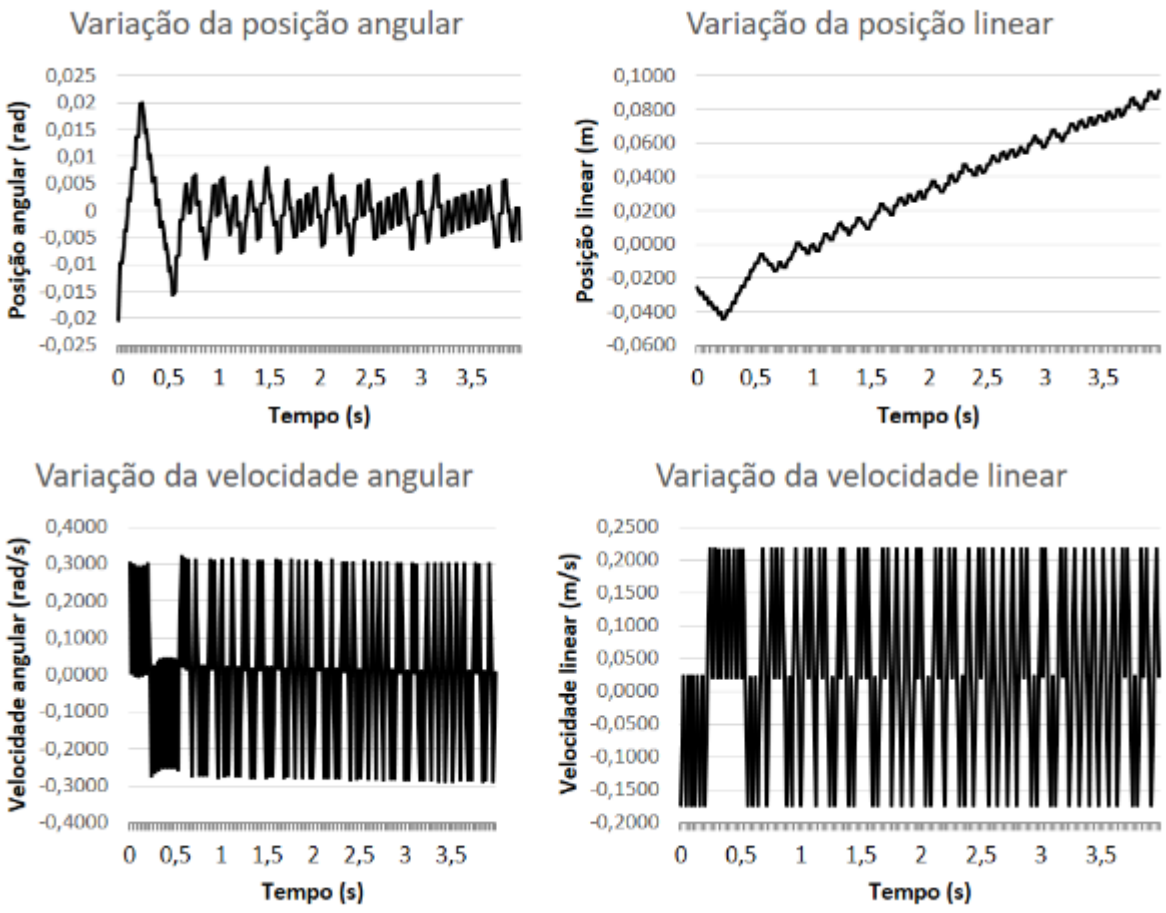
Já o agente treinado pelo método *REINFORCE* teve como comportamento de entrada:

Figura 30: Entrada do agente treinado pelo método *REINFORCE*

Fonte: o autor

E resposta:

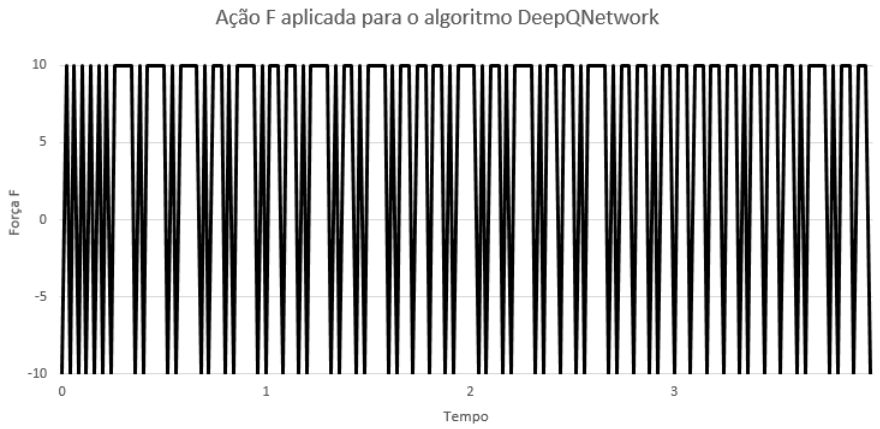
Figura 31: Resposta do agente treinado pelo método *REINFORCE*



Fonte: o autor

Por fim, o agente treinado pelo método *DeepQ Network*:

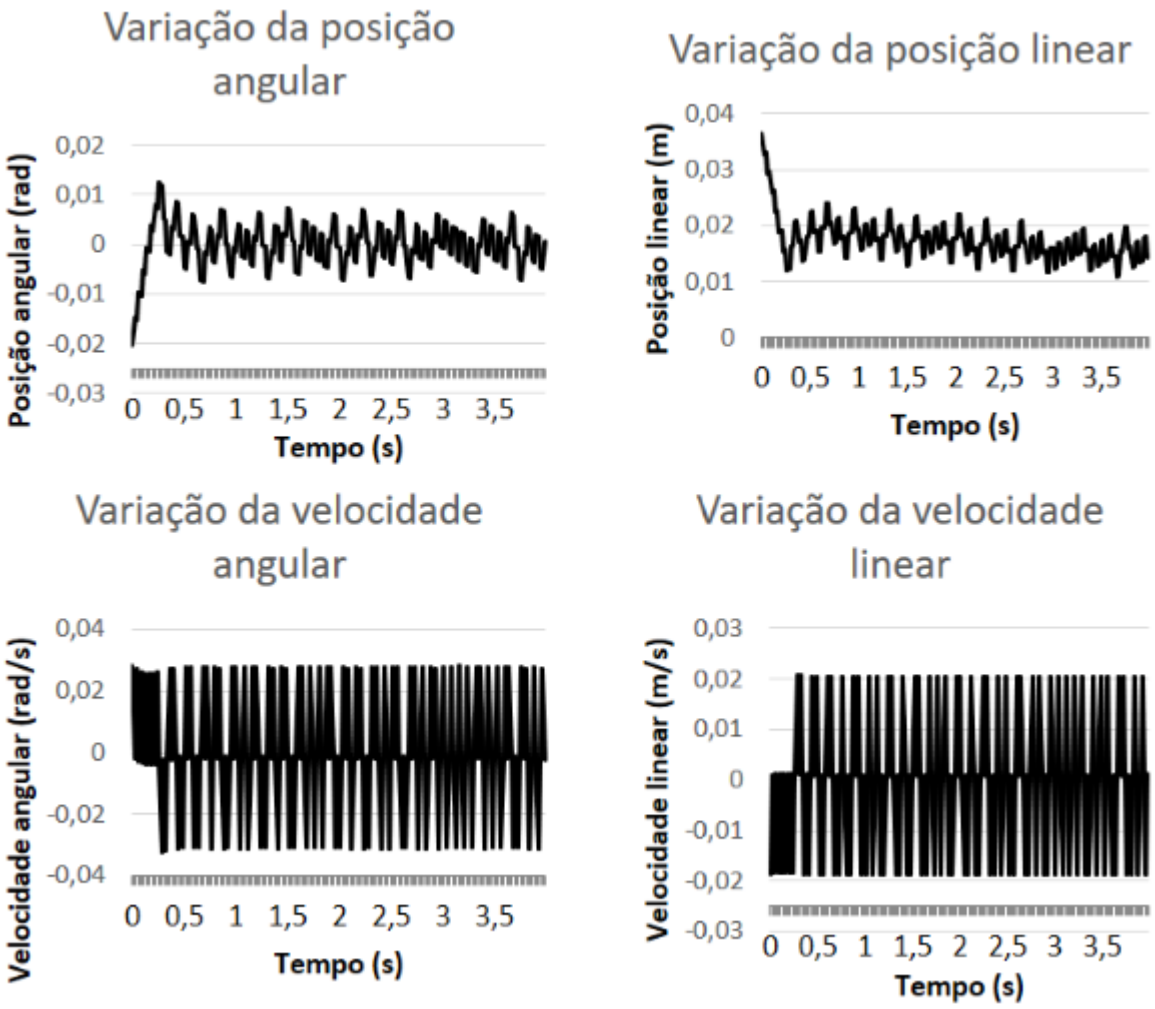
Figura 32: Entrada do agente treinado pelo método *DeepQNetwork*



Fonte: o autor

E com resposta:

Figura 33: Resposta do agente treinado pelo método *DeepQ Networks*



Fonte: o autor

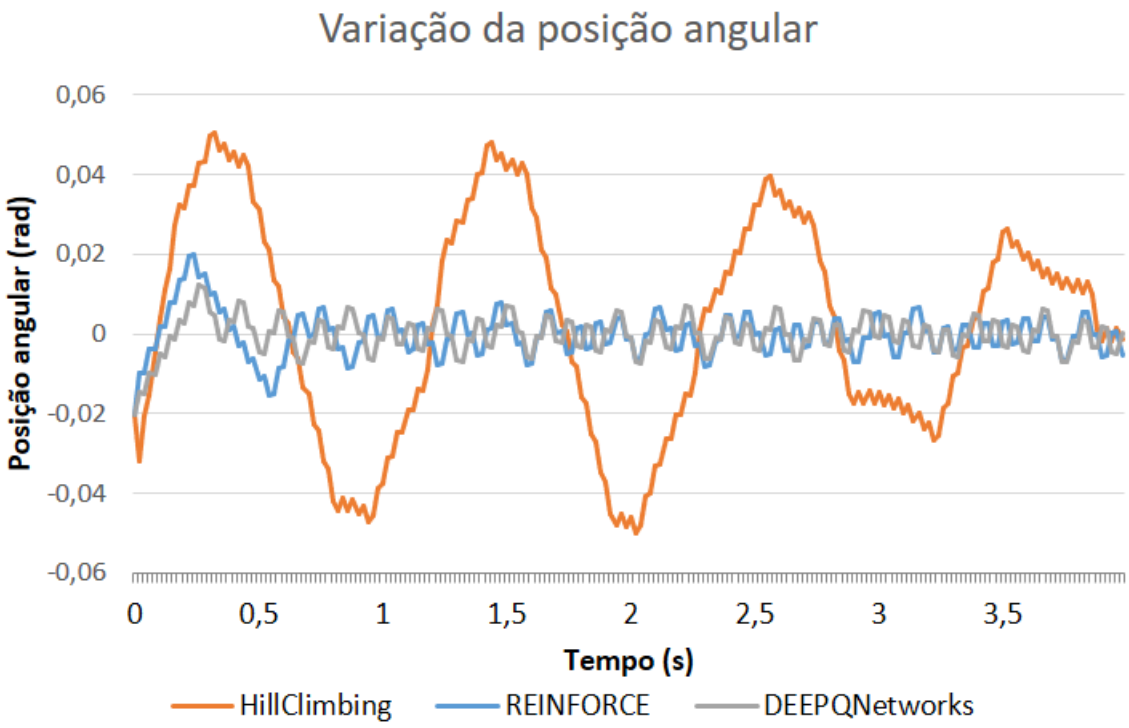
Nos resultados acima, é possível notar que o primeiro agente, treinado pelo método *HillClimbing* consegue equilibrar o pêndulo em torno da posição de $\theta = 0^o$ com uma amplitude máxima observada de $2,9^o$, com sua posição linear x ligeiramente deslocada para a esquerda, e velocidades angular e linear com variação dentro dos intervalos de $[-0.443, 0.564]\, rad/s$ e $[-0.435, 0.338]\, m/s$ respectivamente.

Já o agente treinado pelo método *REINFORCE* também consegue equilibrar o pêndulo em torno de $\theta = 0$ porém com uma amplitude máxima de $1,14^o$. Esse agente equilibrou o pêndulo deslocando sua posição linear x para a direita, e apresentou variações das velocidades angular e linear dentro dos intervalos de $[-0.288, 0.320]\, rad/s$ e $[-0.174, 0.217]\, m/s$ respectivamente.

O último agente, treinado pelo método *DeepQ Networs* equilibrou o pêndulo em torno de $\theta = 0$ com amplitude máxima de $0,70^o$ e nos primeiros instantes de tempo deslocou sua posição linear para a esquerda porém se manteve quase estável em torno de $x = 0,017m$. Suas velocidades angular e linear variaram dentro do intervalo de $[-0.032, 0.028]\, rad/s$ e $[-0.018, 0.020]\, m/s$ respectivamente.

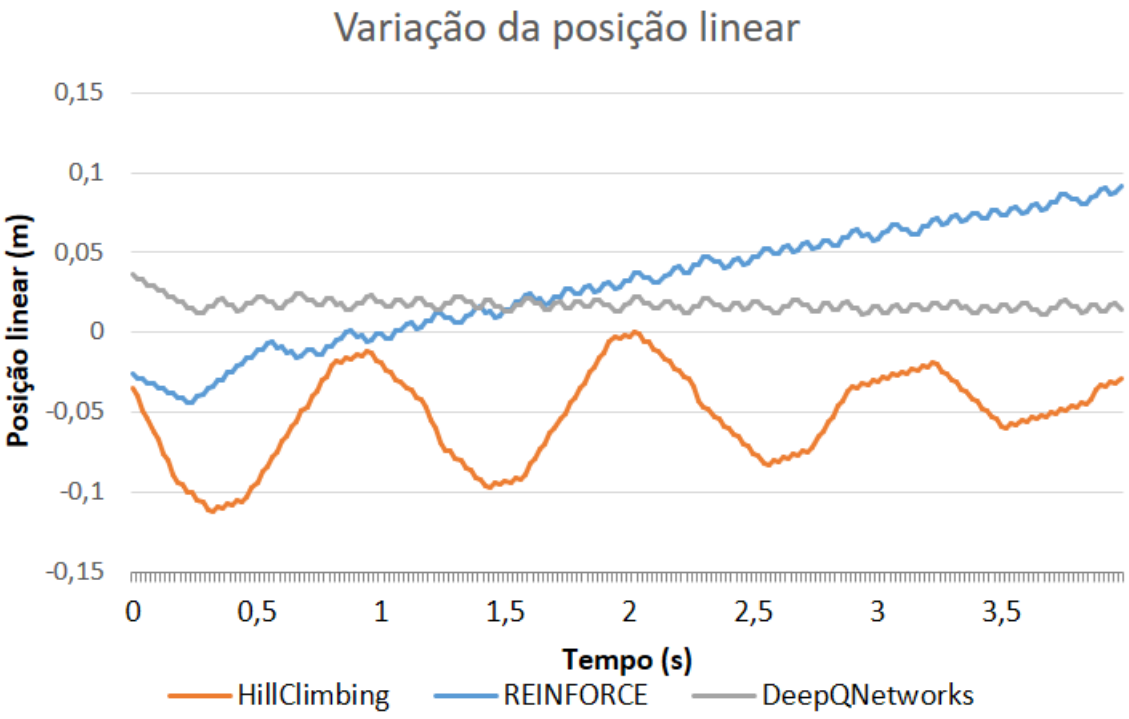
As imagens abaixo comparam os resultados observados das quatro variávies que representam o estado do agente para os três métodos:

Figura 34: Comparação da variação da posição angular entre os três algoritmos



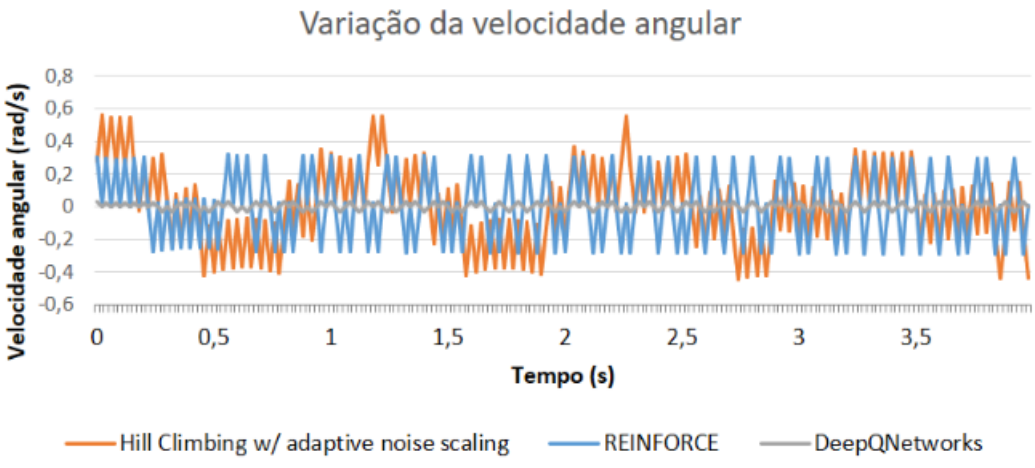
Fonte: o autor

Figura 35: Comparação da variação da posição linear entre os três algoritmos



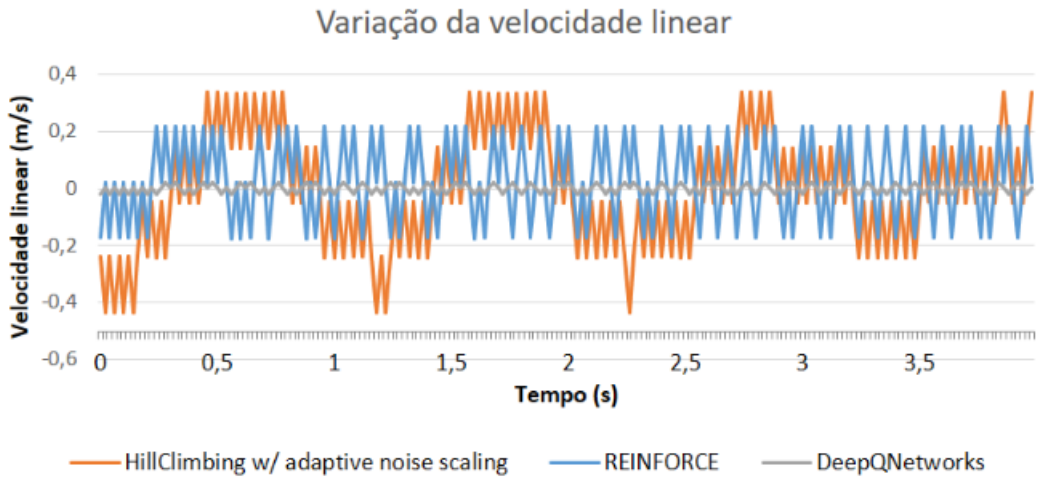
Fonte: o autor

Figura 36: Comparação da variação da velocidade angular entre os três algoritmos



Fonte: o autor

Figura 37: Comparação da variação da velocidade linear entre os três algoritmos



Fonte: o autor

Na tabela abaixo é possível ver a integral do erro absoluto $ITAE_\theta$ e $ITAE_x$ e a raizes dos erros quadráticos médios $RMSE_\theta$ e $RMSE_x$ calculados em torno de $\theta = 0$ e $x = 0$, respectivamente.

Tabela 11: Erros em relação à $\theta = 0$ e $x = 0$

Algoritmo	$ITAE_\theta$	$RMSE_\theta$	$ITAE_x$	$RMSE_x$
HillClimbing	410,52	$27,55 \times 10^{-3}$	928,06	$3,30 \times 10^{-2}$
REINFORCE	55,39	$5,57 \times 10^{-3}$	1023,80	$6,65 \times 10^{-2}$
Deep-Q Network	50,95	$4,63 \times 10^{-3}$	326,46	$1,95 \times 10^{-2}$

A partir das comparações acima, é possível perceber que o agente treinado pelo método *DeepQ Networks* conseguiu equilibrar sua posição angular com menores erros em relação$\theta = 0$. O agente *HillClimbing* obteve uma maior amplitude de sua posição angular e o agente *REINFORCE* resolveu o problema com tendência de deslocamento linear constante para a direita. Já o agente *HillClimbing* apresentou erros bem elevados quando comparados com os outros dois algoritmos.

5.3 Alocação de polos

A tabela 12 apresenta as matrizes de ganho que foram obtidas para cada parâmetro de projeto. Como é possível observar, os menores valores de MO e T_s exigem

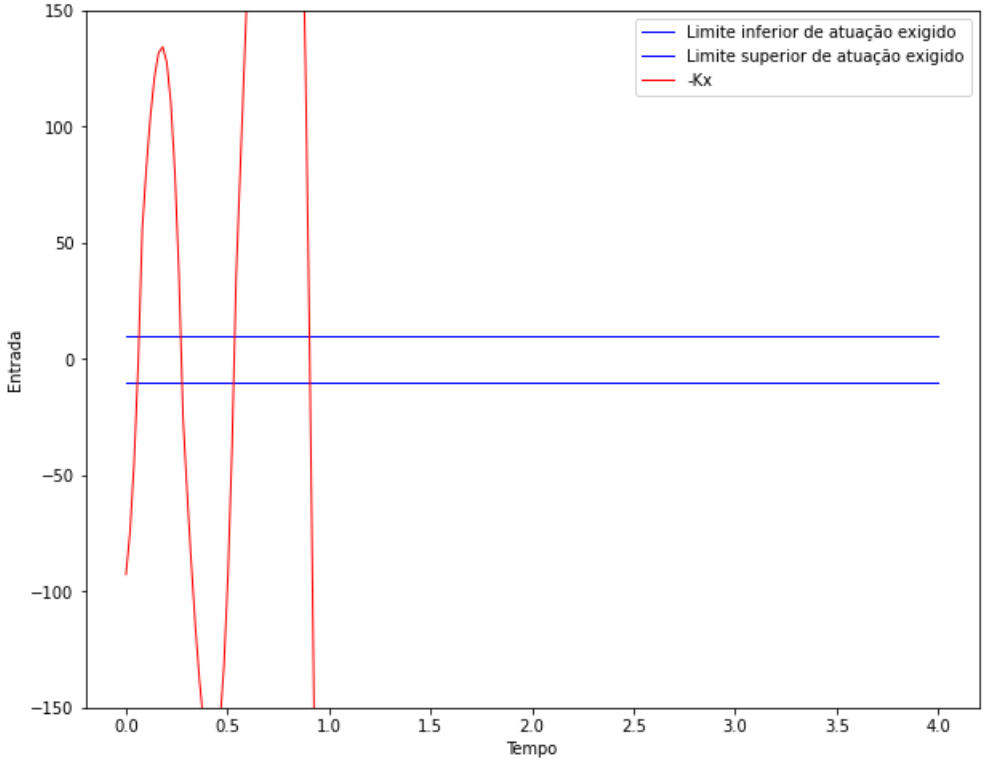
uma atuação maior no sistema, já que tem-se que $u = -Kx$, no entanto o sistema é limitado em operar em $-10N$ e $10N$ para comparação com os algoritmos de aprendizagem por reforço.

Tabela 12: Matriz de ganho para cada parâmetro de projeto

Sobressinal	**Tempo de acomodação**	**Matriz de ganho**
$MO = 5\%$	$T_s = 1,0s$	$K_1 = \begin{bmatrix} -4216 & -1203 & -4539 & -865 \end{bmatrix}$
$MO = 8\%$	$T_s = 1,5s$	$K_2 = \begin{bmatrix} -947 & -348 & -1367 & -273 \end{bmatrix}$
$MO = 9\%$	$T_s = 1,5s$	$K_3 = \begin{bmatrix} -1005 & -352 & -1407 & -276 \end{bmatrix}$
$MO = 10\%$	$T_s = 2,0s$	$K_4 = \begin{bmatrix} -399 & -175 & -749 & -150 \end{bmatrix}$
$MO = 10\%$	$T_s = 2,2s$	$K_5 = \begin{bmatrix} -237 & -116 & -525 & -106 \end{bmatrix}$

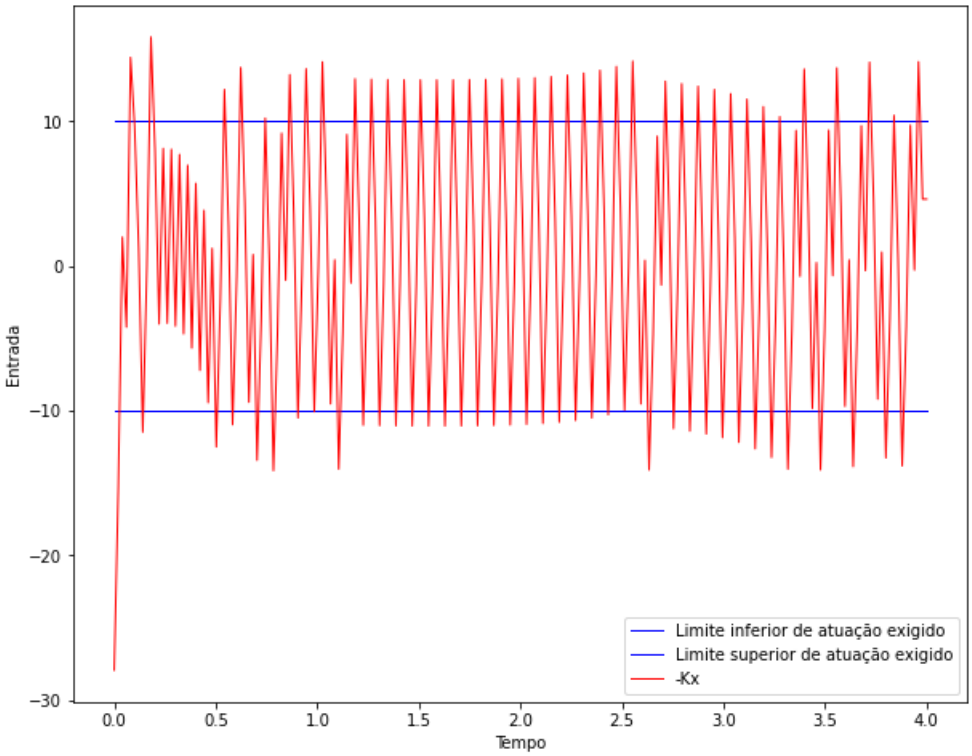
Com o objetivo de avaliar qual matriz K não necessita de saturação para as condições de comparação com parâmetros iniciais $x_0 = [0 \quad 0 \quad -0.02042 \quad 0]^T$, foi realizada uma simulação e a entrada $u = -Kx$ observada:

Figura 38: Entrada do sistema com K_1



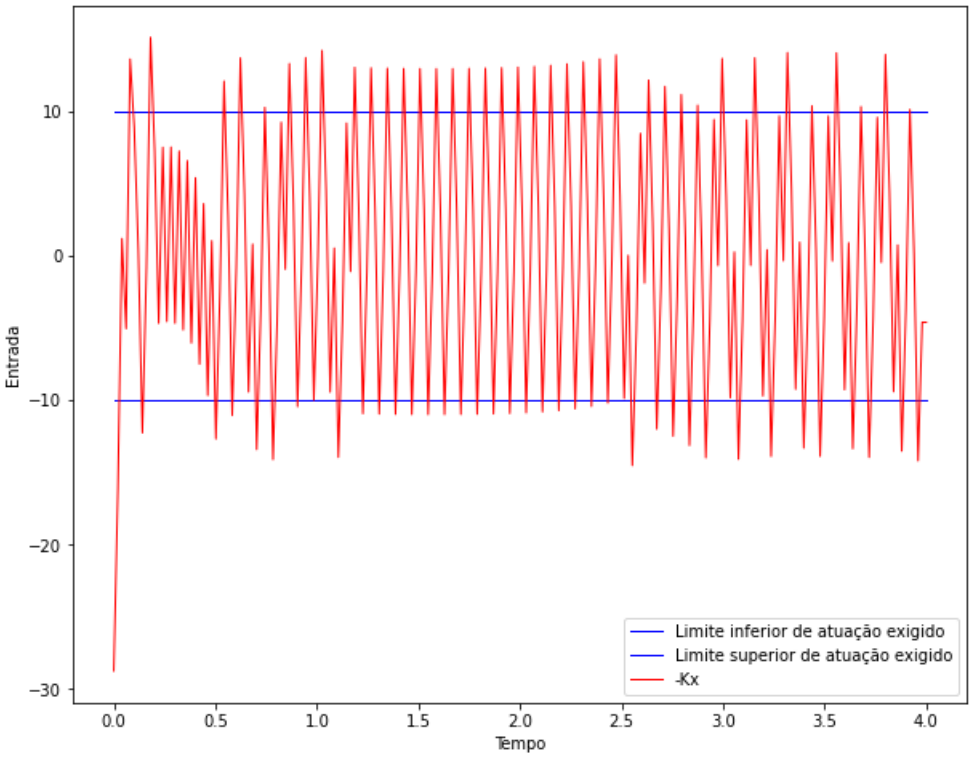
Fonte: o autor

Figura 39: Entrada do sistema com K_2



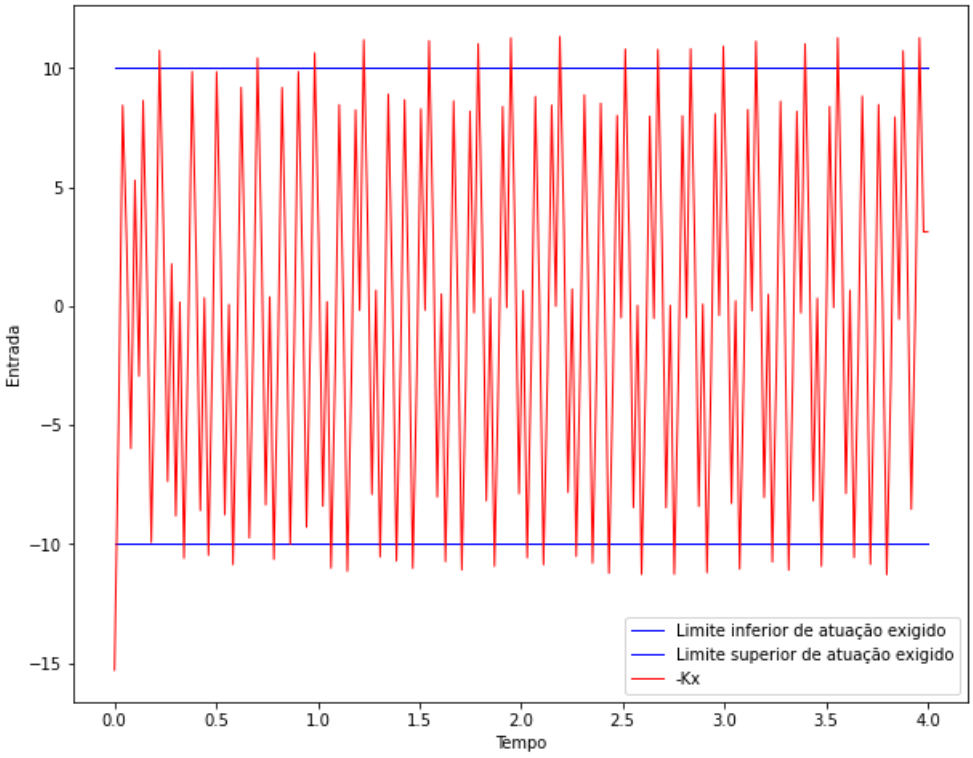
Fonte: o autor

Figura 40: Entrada do sistema com K_3



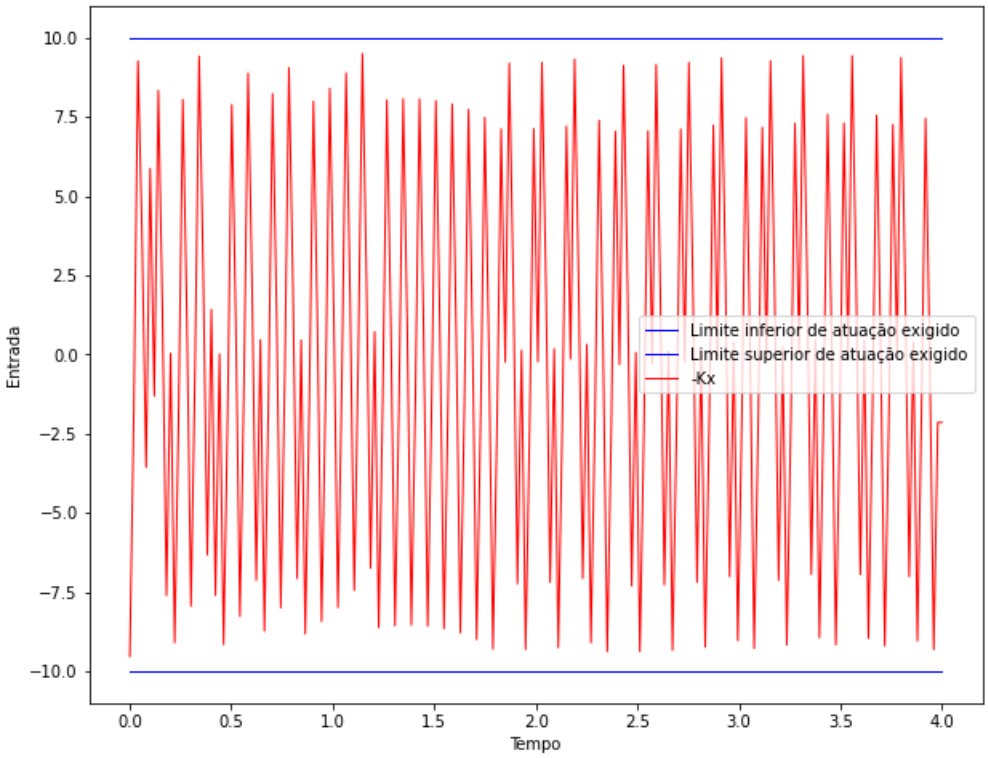
Fonte: o autor

Figura 41: Entrada do sistema com K_4



Fonte: o autor

Figura 42: Entrada do sistema com K_5

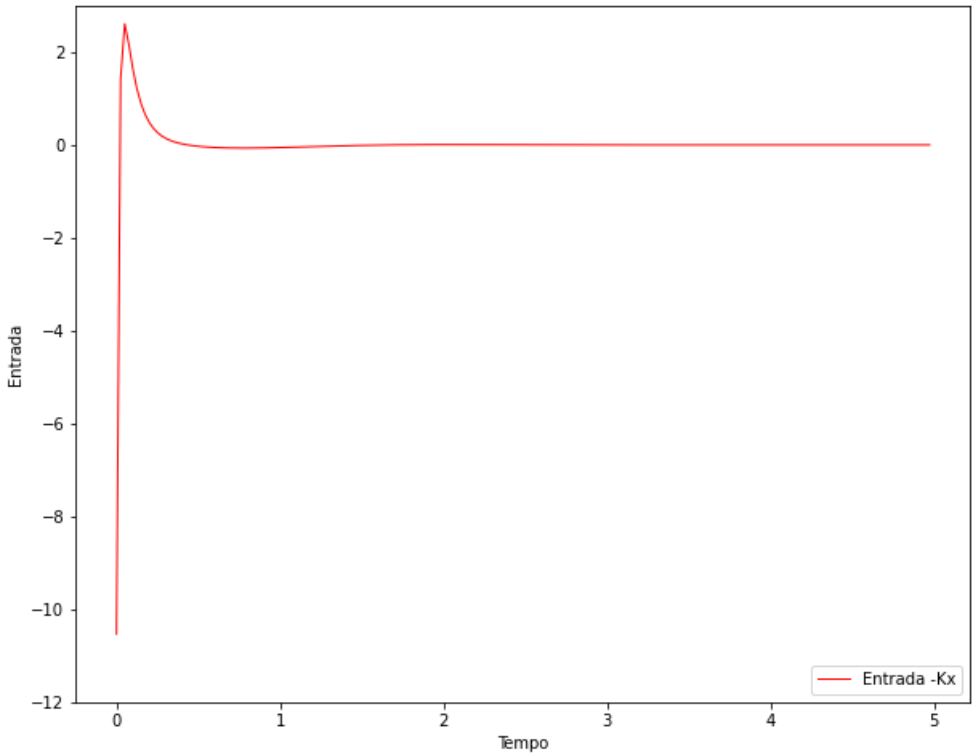


Fonte: o autor

É possível observar que para os quatro primeiros casos é exigida uma atuação além dos valores de $-10N$ e $10N$. Dessa forma, como o controlador exige uma atuação acima da capacidade de projeto e, para evitar saturação, esses controladores serão descartados.

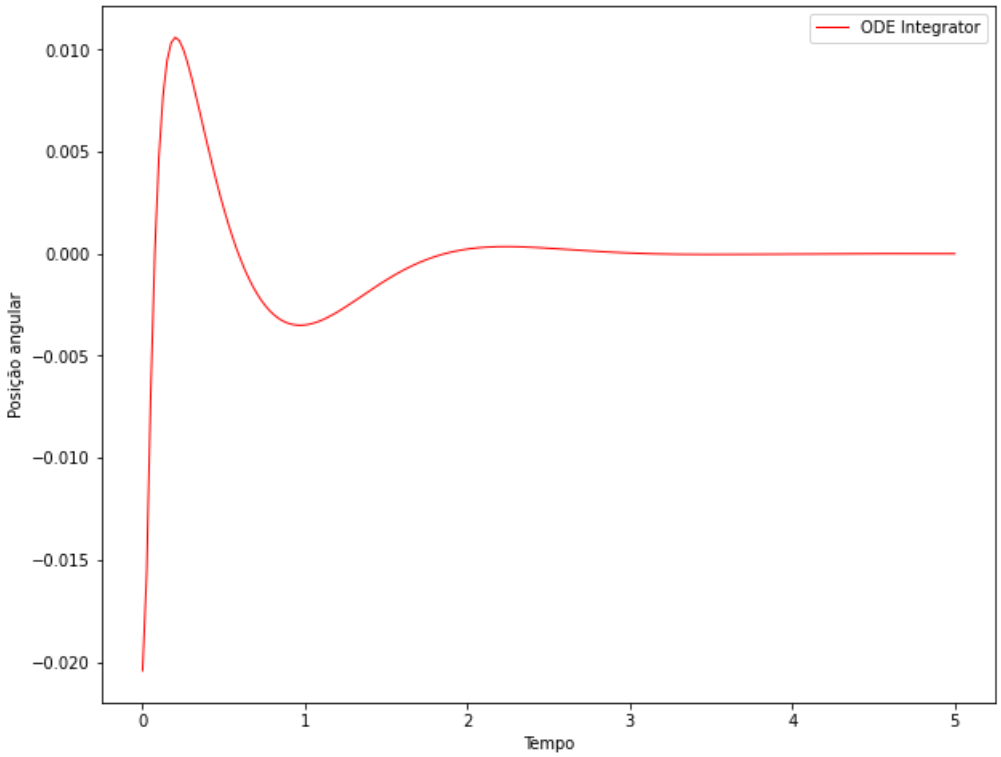
No entanto, o sistema que utiliza a matriz de ganho K_5 conseguiu atingir as especificações de entrada. Para uma condição inicial $x_0 = [0 \quad 0 \quad -0.02042 \quad 0]^T$ considerando a entrada contínua $u = -Kx$, o sistema obteve comportamento da entrada e respostas observadas nas imagens 43, 44 e 45.

Figura 43: Entrada do sistema com K_5



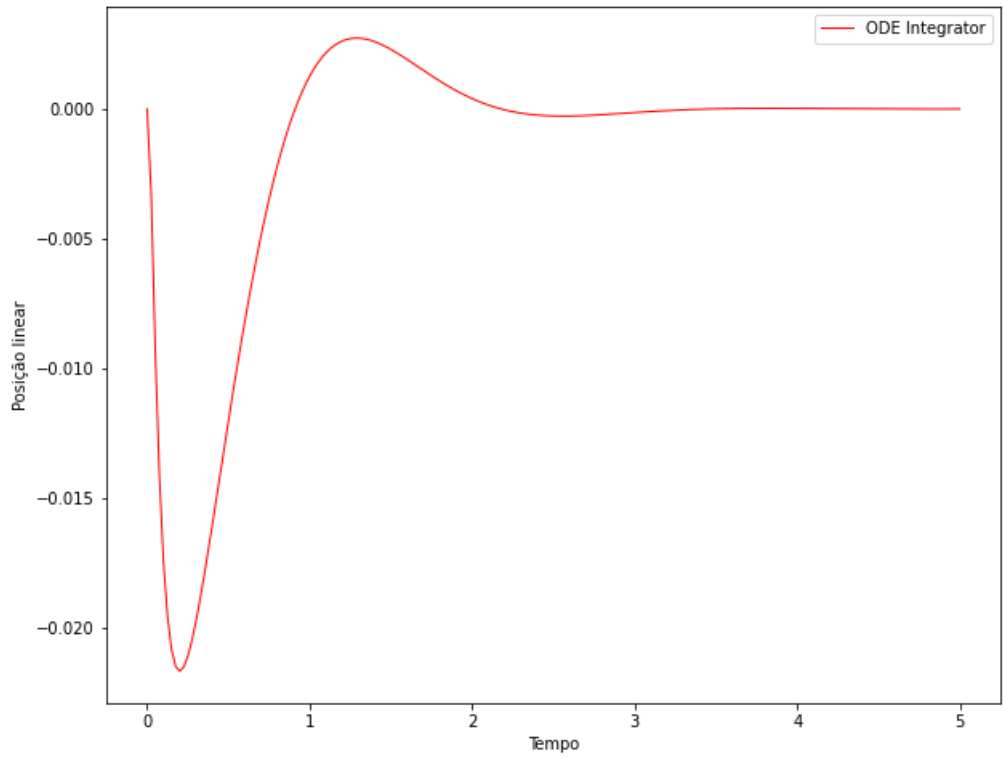
Fonte: o autor

Figura 44: Resposta da posição angular θ do sistema com K_5



Fonte: o autor

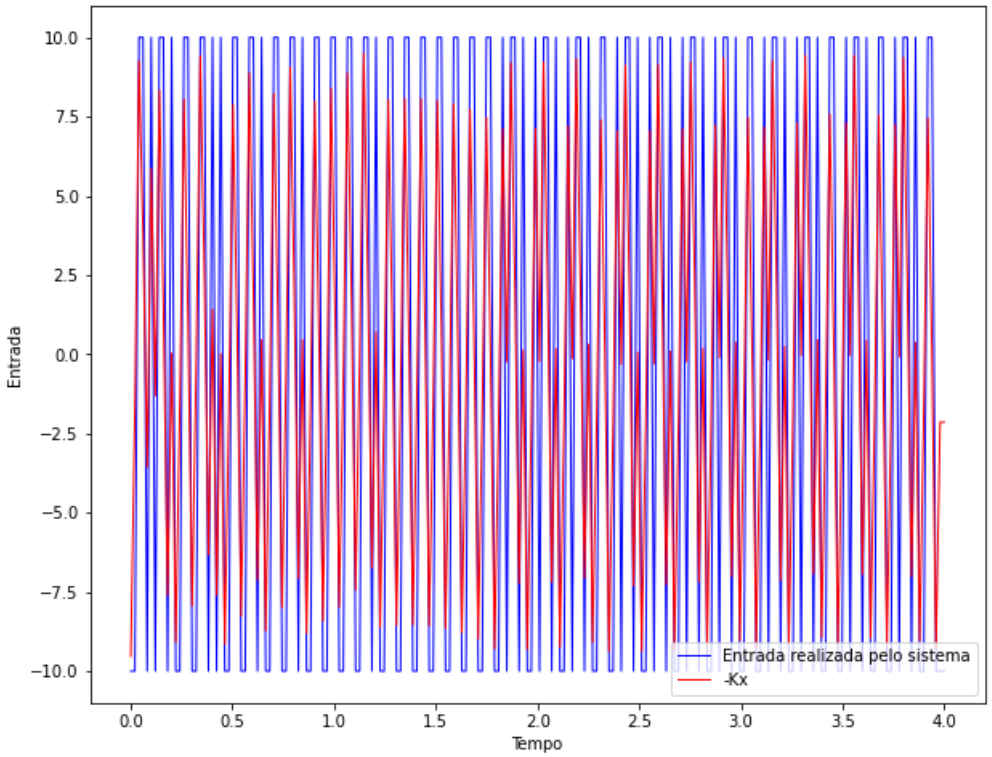
Figura 45: Resposta da posição linear x do sistema com K_5



Fonte: o autor

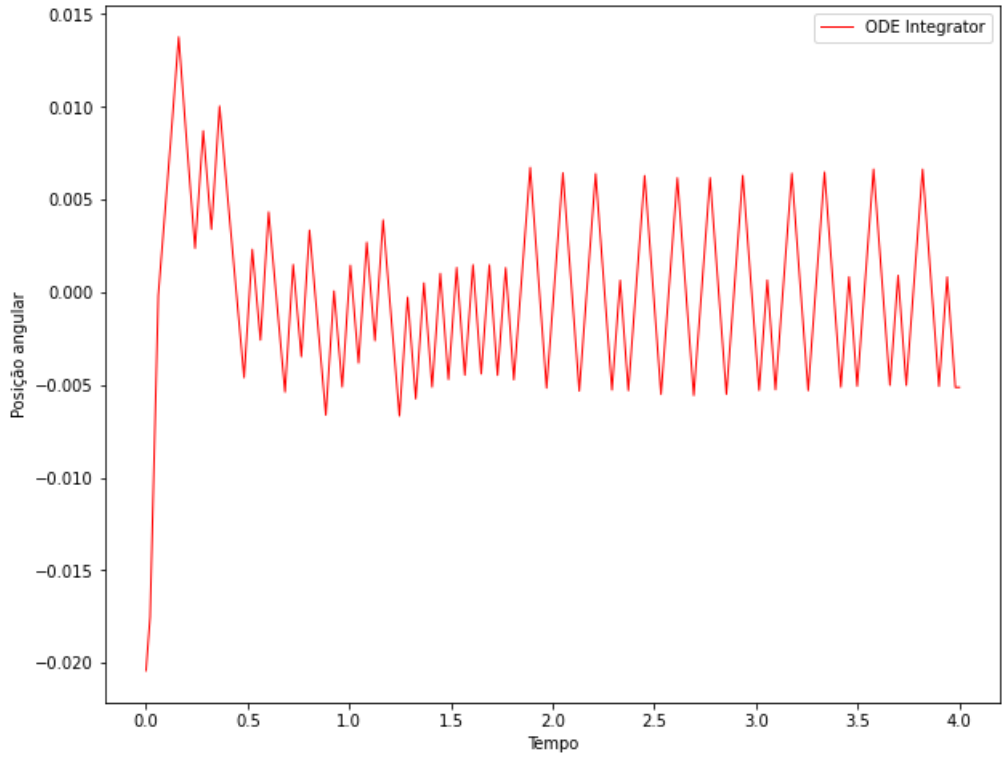
Onde é possível observar que o controlador garante o equilíbrio do pêndulo. Abaixo é possível ver a resposta das quatro variáveis de estado para o sistema com K_5, com a limitação na atuação em $-10N$ ou $+10N$ e condição inicial $x_0 = [0 \quad 0 \quad -0.02042 \quad 0]^T$:

Figura 46: Entrada do sistema com K_5 com limitação na atuação



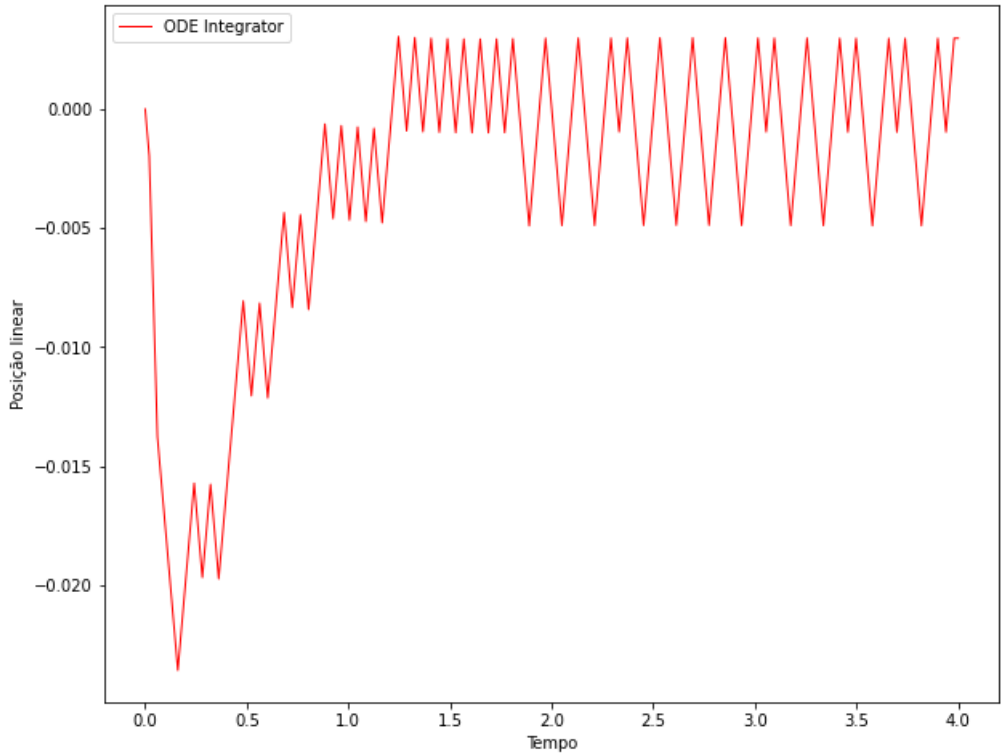
Fonte: o autor

Figura 47: Resposta da posição angular θ do sistema com K_5 com limitação na atuação



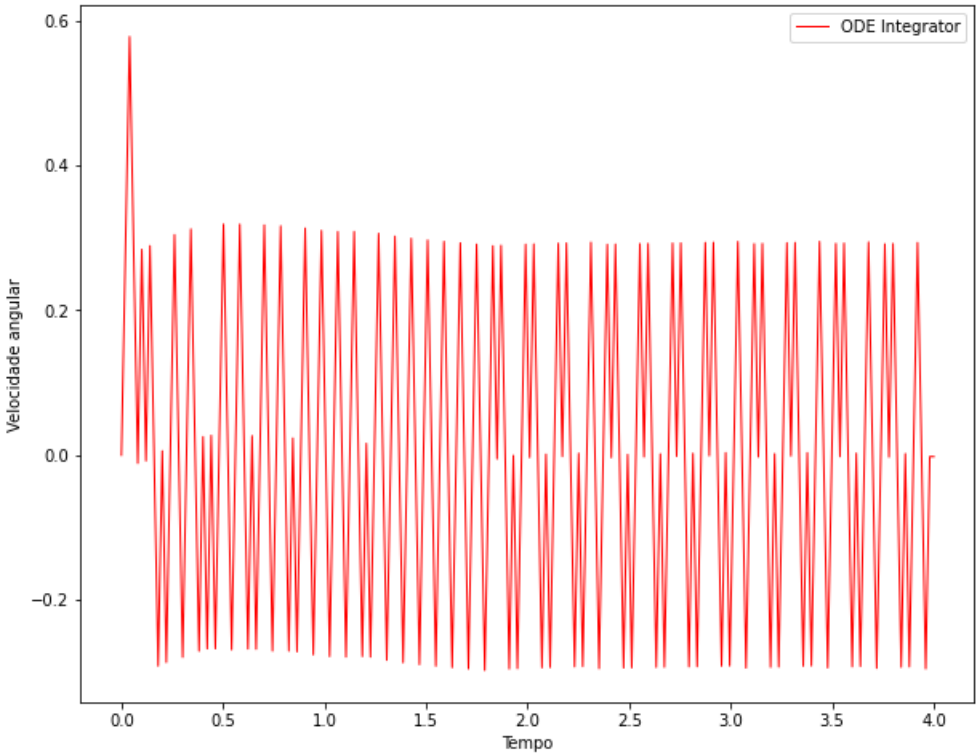
Fonte: o autor

Figura 48: Resposta da posição linear x do sistema com K_5 com limitação na atuação



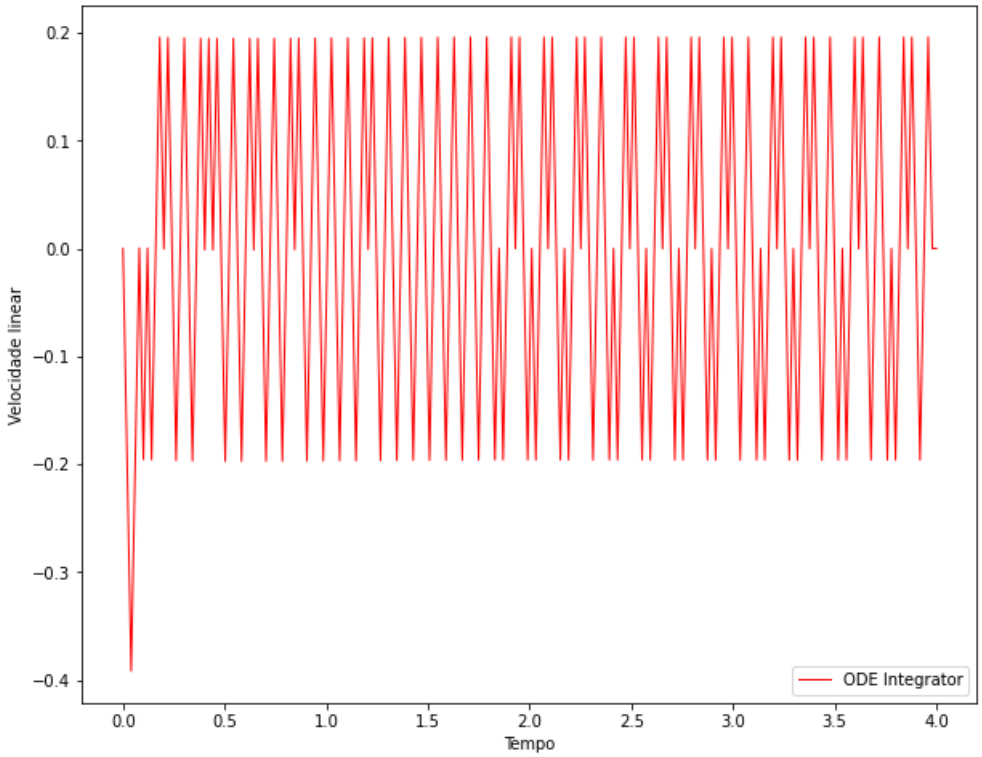
Fonte: o autor

Figura 49: Resposta da velocidade angular $\dot{\theta}$ do sistema com K_5 com limitação na atuação



Fonte: o autor

Figura 50: Resposta da velocidade linear $\dot{x}$ do sistema com K_5 com limitação na atuação



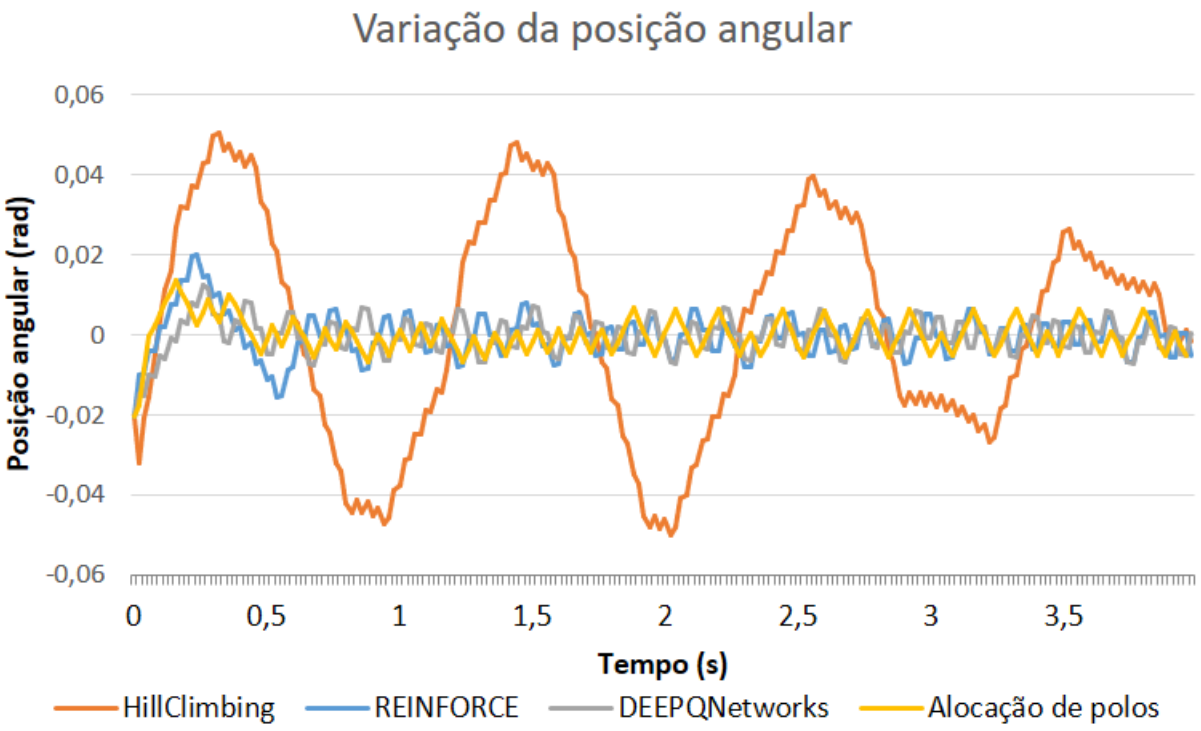
Fonte: o autor

Dessa forma, como o sistema controlado com K_5 possui os menores parâmetros de MO e T_s dentro dos valores que foram testados e cuja resposta tenha sido satisfatória dada a limitação do sistema, esse ganho será utilizado para avaliação e comparação com a resposta obtida pelos algoritmos de aprendizagem por reforço.

5.4 Comparação entre as abordagens

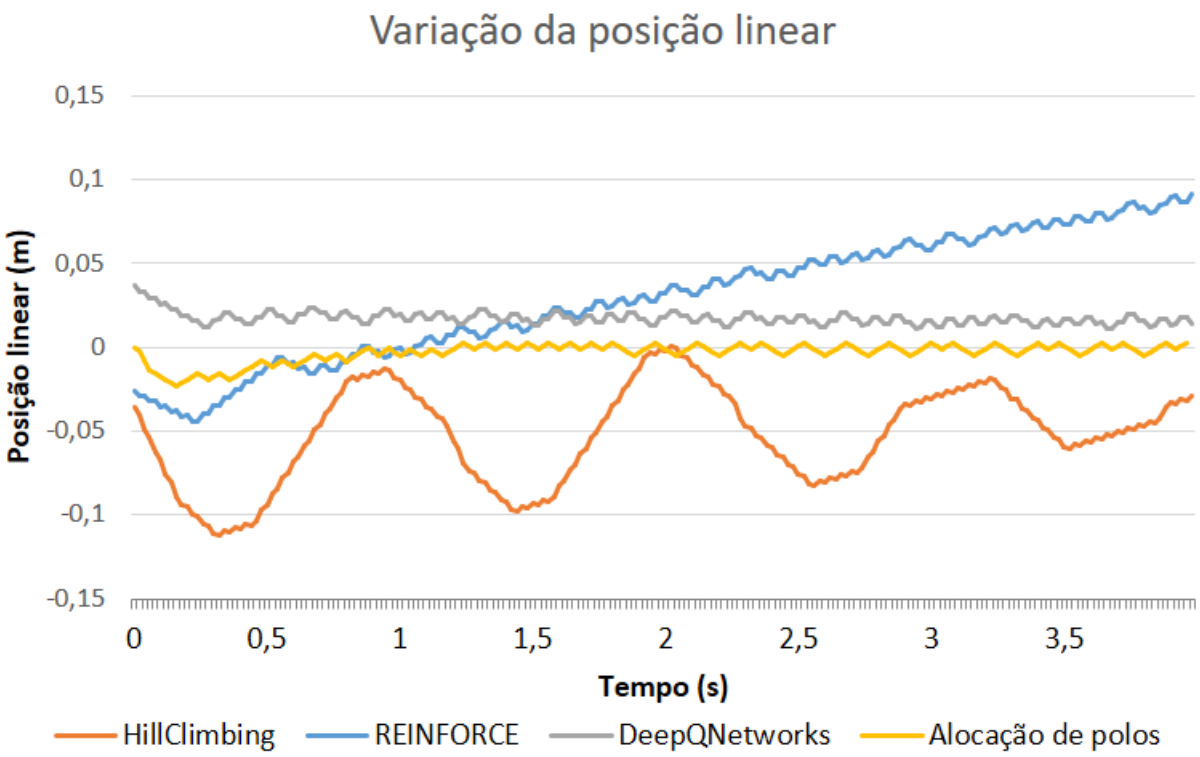
Nas figuras 51 e 52 é possível observar a resposta da posição angular e posição linear de todas as abordagens utilizadas neste trabalho.

Figura 51: Comparação da resposta da posição angular θ entre os algoritmos



Fonte: o autor

Figura 52: Comparação da resposta da posição linear x entre os algoritmos



Fonte: o autor

Nas figuras 51 e 52 é possível ver que a resposta da posição angular θ utilizando o sistema de controle com alocação de polos obteve uma similaridade com o resultado obtido com os algoritmos de *DeepQNetwork* e *REINFORCE*. Ao avaliar a resposta da posição linear x, mesmo com a limitação da condição inicial diferente, é possível ver o sistema controlado com alocação de polos e o algoritmo de *DeepQNetwork* obtiveram uma resposta centralizada em um valor, enquanto os outros dois algoritmos não.

Abaixo é possível observar a comparação entre os erros $ITAE$ e $RMSE$ entre os algoritmos de aprendizagem por reforço e o sistema utilizando alocação de polos com K_5 em relação à $\theta = 0$ e $x = 0$:

Tabela 13: Erros em relação à $\theta = 0$ e $x = 0$

Algoritmo	$ITAE_\theta$	$RMSE_\theta$	$ITAE_x$	$RMSE_x$
HillClimbing	410,52	$27,55 \times 10^{-3}$	928,06	$3,30 \times 10^{-2}$
REINFORCE	55,39	$5,57 \times 10^{-3}$	1023,80	$6,65 \times 10^{-2}$
Deep-Q Network	50,95	$4,63 \times 10^{-3}$	326,46	$1,95 \times 10^{-2}$
Alocação de polos com K_5	52,94	$7,16 \times 10^{-3}$	42,71	$0,73 \times 10^{-2}$

Na comparação acima é possível perceber que o método *HillClimbing* apresentou os maiores erros em relação à θ e x. Além disso, o sistema controlado com alocação de polos obteve os erros $RMSE_\theta$ e $ITAE_\theta$ similares quando comparados com os erros obtidos com os algoritmos *DeepQNetwork* e *REINFORCE*. No entanto, o sistema controlado com a abordagem clássica teve uma melhor resposta na posição linear para os dois erros calculados. É importante citar que a resposta do método por alocação de polos foi prejudicada pela limitação na atuação, porém os métodos de aprendizado por reforço de certa forma se ajustaram e se beneficiaram dessa limitação.

5.5 Discussão dos resultados

A partir dos resultados apresentados acima é possível observar que os algoritmos de aprendizagem por reforço apresentam um resultado interessante quando comparado com a abordagem tradicional utilizada no presente trabalho.

Em relação ao treinamento dos algoritmos por aprendizado por reforço, foi observado que o método *HillClimbing* teve um período de treinamento mais rápido em relação aos outros algoritmos, o que já era esperado devido à menor complexidade do método. Para o treinamento das redes neurais dos algoritmos *REINFORCE* e *DeepQNetworks* o método de otimização *Adam* apresentou melhores resultados para os dois algoritmos. Como introduzido na seção de fundamentação teórica, esse método de otimização é mais recente e parte de melhorias das outras duas técnicas, sendo assim, é natural esperar um melhor resultado no treinamento das redes neurais utilizando esse método. Além disso, o treinamento do agente *DeepQNetworks* precisou de um número muito maior de etapas de treinamento do que

o algoritmo *REINFORCE*, já que é um método que requer um maior conjunto de dados de exploração no ambiente para a estimativa da função ação-valor. No entanto, o tempo de execução do treinamento foi inferior ao tempo de treinamento do agente *REINFORCE*, mesmo tendo um maior número de episódios. Isso se deve, em sua maioria, pois a cada interação deste algoritmo há a atualização da política do agente a partir dos valores estimados da função valor, enquanto que no método *DeepQNetworks* a função ação-valor já é utilizada como norteadora das ações do agente, sem precisar utilizá-la para estimativa de outra função.

Ao comparar a resposta das variáveis de estado a partir da mesma posição angular inicial, o algoritmo *DeepQNetworks* teve erros em relação ao objetivo menores, com o algoritmo *HillClimbing* apresentando maiores amplitudes nas variações de posição angular e linear. Esses resultados eram esperados devido ao potencial de cada algoritmo já estudados em outros trabalhos, onde o *DeepQNetworks* é um método mais recente baseado em funções de ação-valor, que mapeiam diretamente para cada estado do sistema a ação que obtém o maior valor esperado de retornos futuros.

Durante a síntese do controlador por alocação de polos, foi percebido que, por conta da limitação da atuação do sistema, nem todos os valores de sobressinal e tempo de acomodação eram possíveis serem alcançados. Isso se deve ao fato de que para atingir o equilíbrio nos menores tempos possíveis a atuação no sistema deve ser alta, porém o atuador não consegue atuar de tal forma.

Comparando a resposta dos métodos de aprendizagem por reforço com o controlador por alocação de polos, foi encontrada uma similaridade nas respostas entre a abordagem clássica e os métodos *DeepQNetworks* e *REINFORCE*. Já o método *HillClimbing* obteve resultado com erros bem elevados quando comparados com os erros dos outros métodos.

6 CONSIDERAÇÕES FINAIS

O presente trabalho apresenta duas abordagens diferentes para a resolução do problema do pêndulo invertido com um grau de liberdade. A primeira abordagem, utiliza a vertente da aprendizagem por reforço do *machine learning* para o controle do pêndulo. Já a segunda, amplamente conhecida e estudada, faz o uso do controle em espaço de estados com alocação de polos.

A primeira abordagem foi implementada computacionalmente utilizando a linguagem de programação *python* dentro do ambiente *cartpole-v1* da *OpenAI Gym*, que serve como um *benchmarking* utilizado pela comunidade científica para comparar algoritmos de controle, principalmente aqueles utilizando aprendizagem por reforço. Dentro dessa abordagem, três métodos diferentes foram sintetizados: *Hill-Climbing* com ruído adaptativo (HCRA), algoritmo *REINFORCE* e *DeepQNetworks*, nos quais os ultimos dois foram implementados através da *API TensorFlow Agents*. Foi realizada, então, uma etapa de treinamento, na qual cada agente criado com cada algoritmo passou por diversos episódios dentro do ambiente com o objetivo de aprender a atuar sobre esse meio para equilibrar o pêndulo. Para isso, foi criado um sistema de recompensas, onde para cada instante de tempo que o agente mantivesse o pêndulo dentro de condições limitadas era dada uma recompensa para o sistema, e o objetivo geral do agente foi de coletar o maior número de recompensas no tempo determinado.

Durante o treinamento, foi observado que o algoritmo *HillClimbing* necessitou de menos episódios do que os outros métodos. Além disso, o método *DeepQ-Networks* convergiu no objetivo com um número de passos bem maior que o agente *REINFORCE*, porém, com um tempo computacional inferior. Outro detalhe observado nessa etapa foi que o método de otimização *Adam* foi mais eficiente para o treinamento das redes neurais dos dois algoritmos.

Em sequência, com o treinamento concluído, foram iniciados simulações do comportamento de cada agente através da posição angular inicial do pêndulo dentro de um ângulo de 1,15º. Os resultados do treinamento de cada um dos algoritmos foram satisfatórios, já que os três métodos apresentaram uma resposta de equilíbrio do pêndulo com erros satisfatórios em relação à posição de equilíbrio. Comparando cada um deles, o algoritmo HCRA, por ser mais simples e com estimativa da política com menos elementos na RNA, apresentou os maiores erros $ITAE$ e $RMSE$ em relação ao objetivo de manter o ângulo na posição vertical. Já os outros dois algoritmos mais recentes, *REINFORCE* e *DeepQNetworks*, apresentaram uma resposta com os erros bem menores, com o agente *DeepQNetworks* com a melhor resposta. Esses resultados já eram esperados dado que são três métodos desenvolvidos em diferentes momentos com o primeiro sendo mais antigo e o último, mais recente e com várias aplicações práticas apresentadas em problemas mais complexos.

Para a implementação da abordagem por alocação de polos, foi realizada a linearização e modelagem do sistema em espaços de estados dada as equações diferenciais ordinárias que representam esse problema já conhecidas e divulgadas na literatura. Com esse modelo, as matrizes de ganho necessárias para estabilizar o sistema foram encontradas com base em cinco valores de sobressinal e tempos de acomodação diferentes. Como existia uma limitação na atuação do sistema, nem todos os controladores conseguiram resolver o problema, já que para especificações de engenharia que exigiam uma resposta mais rápida, a atuação no sistema necessária foi acima da capacidade do atuador. Das cinco tentativas com parâmetros diferentes, dois conseguiram atingir o objetivo, e a solução que conseguiu controlar o sistema com menores sobressinal e tempo de acomodação foi selecionada para comparação.

Por fim, as respostas das duas abordagens foram comparadas. Foi observado que o a segunda, utilizando controle por alocação de polos, obteve uma resposta melhor do que o agente HCRA e um comportamento bem similar aos agentes *REINFORCE* e *DeepQNetworks*. Porém os erros em relação à posição linear central foram bem menores na resposta observado com o método de alocação de polos. Isso pode se dar pela modelagem de recompensas dos agentes de aprendizado por reforço, dado que os limites máximo e mínimo da posição linear para obter a recompensa eram grandes.

Dessa forma, conclui-se que os objetivos do trabalho foram atingidos, tendo

em vista que o principal objetivo era avaliar o problema dinâmico do pêndulo invertido com um grau de liberdade, propor a síntese de um controlador utilizando algorítmos diferentes de aprendizagem por reforço e comparar com um método de controle contínuo clássico. Com a comparação entre os métodos, observa-se que os algoritmos de aprendizagem por reforço apresentaram respostas bastante satisfatórias. Uma das vantagens do *machine learning* observadas é a não necessidade do conhecimento da física do problema, já que o agente aprende sozinho a atingir o objetivo definido. Para a engenharia mecânica, especialmente a engenharia de controle de sistemas dinâmicos, essa abordagem alia-se a um robusto leque de técnicas de controle para ampliar as opções que essa área possui na resolução de seus problemas.

Com o objetivo de dar continuidade à pesquisa, abordando aspectos não estudados no presente trabalho ou de melhorar as formulações apresentadas, faz-se a seguir algumas sugestões e considerações para trabalhos futuros:

a) Realização de simulações da resposta de cada método variando a posição angular inicial e avaliar a capacidade de atuação de cada método em cenários diferentes do qual foi testado no presente trabalho;

b) Aplicação dos algoritmos de aprendizado por reforço no ambiente *Real Gym* que simula condições reais, como arraste do pêndulo e atrito entre o pêndulo e o eixo de rotação e entre a roda do carro e o piso;

c) Variação das camadas das redes neurais dos agentes *REINFORCE* e *DeepQ-Networks* com o objetivo de obter um treinamento mais rápido;

d) Aplicar controle com atuação contínua entre -F e F;

e) Sintetizar um controlador utilizando a técnica LQR com o objetivo de limitar o sinal de controle através da seleção da matriz R;

REFERÊNCIAS

ABADI, M. et al. Tensorflow: A system for large-scale machine learning. In: **12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)**. [S.l.: s.n.], 2016. p. 265–283.

ACKLEY, D. H. Stochastic iterated genetic hillclimbing. Carnegie Mellon University, 1987.

ALPAYDIN, E. **Introduction to machine learning**. [S.l.]: MIT press, 2020.

ANDERSON, C. W. Learning to control an inverted pendulum using neural networks. **IEEE Control Systems Magazine**, IEEE, v. 9, n. 3, p. 31–37, 1989.

BARTO, A. G.; SUTTON, R. S.; ANDERSON, C. W. Neuronlike adaptive elements that can solve difficult learning control problems. **IEEE transactions on systems, manm and cybernetics**, IEEE, n. 5, p. 834–846, 1983.

BASHEER, I. A.; HAJMEER, M. Artificial neural networks: fundamentals, computing, design, and application. **Journal of microbiological methods**, Elsevier, v. 43, n. 1, p. 3–31, 2000.

BELLINASO, L. V.; PAGLIONE, P. ComparaÇÃo entre estratÉgias de controle por regulador quadrÁtico e controle por mapeamento exponencial aplicado a um sistema de pÊndulo invertido. 2014.

BOTVINICK, M. et al. Reinforcement learning, fast and slow. **Trends in cognitive sciences**, Elsevier, 2019.

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

CHAOUI, H.; YADAV, S. Adaptive motion and posture control of inverted pendulums with nonlinear friction compensation. In: IEEE. **2016 IEEE 25th International Symposium on Industrial Electronics (ISIE)**. [S.l.], 2016. p. 344–349.

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.

DUTECH, e. a. Reinforcement learning benchmarks and bake-offs ii. **Advances in Neural Information Processing Systems (NIPS) 17**, 2005.

FURUTA, K.; YAMAKITA, M.; KOBAYASHI, S. Swing up control of inverted pendulum. In: **IECON**. [S.l.: s.n.], 1991. v. 91, p. 2193–2198.

GÉRON, A. **Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems**. [S.l.]: O'Reilly Media, 2019.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep learning**. [S.l.]: MIT press, 2016.

GOOGLE Colab. <https://colab.research.google.com/notebooks/intro.ipynb>. Acessado: 2021-01-20.

HAFNER, D.; DAVIDSON, J.; VANHOUCKE, V. Tensorflow agents: Efficient batched reinforcement learning in tensorflow. **arXiv preprint arXiv:1709.02878**, 2017.

HECHT-NIELSEN, R. Neurocomputer applications. In: **Neural computers**. [S.l.]: Springer, 1989. p. 445–453.

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

KINGMA, D. P.; BA, J. Adam: A method for stochastic optimization. **arXiv preprint arXiv:1412.6980**, 2014.

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

LYDIA, A.; FRANCIS, S. Adagrad—an optimizer for stochastic gradient descent. **Int. J. Inf. Comput. Sci.**, v. 6, n. 5, 2019.

MICHIE, D.; CHAMBERS, R. A. Boxes: An experiment in adaptive controle. **Machine intelligence**, Citeseer, v. 2, n. 2, p. 137–152, 1968.

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

____. Human-level control through deep reinforcement learning. **nature**, Nature Publishing Group, v. 518, n. 7540, p. 529–533, 2015.

NAGENDRA, S. et al. Comparison of reinforcement learning algorithms applied to the cart-pole problem. In: IEEE. **2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)**. [S.l.], 2017. p. 26–32.

NISE, N. S. **CONTROL SYSTEMS ENGINEERING**. [S.l.]: John Wiley & Sons, 2007.

OGATA, K.; YANG, Y. **Modern control engineering**. [S.l.]: Prentice hall India, 2002. v. 4.

RAMACHANDRAN, P.; ZOPH, B.; LE, Q. V. Searching for activation functions. **arXiv preprint arXiv:1710.05941**, 2017.

ROBBINS, H.; MONRO, S. A stochastic approximation method. **The annals of mathematical statistics**, JSTOR, p. 400–407, 1951.

RUDER, S. An overview of gradient descent optimization algorithms. **arXiv preprint arXiv:1609.04747**, 2016.

RUSSEL, S.; NORVIG, P. et al. **Artificial intelligence: a modern approach**. [S.l.]: Pearson Education Limited, 2013.

SCHALKOFF, R. J. **Artificial neural networks**. [S.l.]: McGraw-Hill Higher Education, 1997.

SHARIAR, S.; HASAN, K. A. Gpu accelerated indexing for high order tensors in google colab. In: IEEE. **2020 IEEE Region 10 Symposium (TENSYMP)**. [S.l.], 2020. p. 686–689.

SHEHU, M. et al. Lqr, double-pid and pole placement stabilization and tracking control of single link inverted pendulum. In: IEEE. **2015 IEEE International Conference on Control System, Computing and Engineering (ICCSCE)**. [S.l.], 2015. p. 218–223.

SILVA, E. A. d. Construção, modelagem e controle de um pêndulo invertido com clp e software scada. UNESP, 2013.

SUTTON, R. S.; BARTO, A. G. **Reinforcement Learning: An Introduction 2nd edition**. [S.l.: s.n.], 2020.

SUTTON, R. S. et al. Policy gradient methods for reinforcement learning with function approximation. In: **Advances in neural information processing systems**. [S.l.: s.n.], 2000. p. 1057–1063.

WIERING, M.; OTTERLO, M. V. Reinforcement learning. **Adaptation, learning, and optimization**, Springer, v. 12, p. 3, 2012.

YOUNG, L.; MEIRY, J. Bang-bang aspects of manual control in high-order systems. **IEEE Transactions on Automatic Control**, IEEE, v. 10, n. 3, p. 336–341, 1965.