

UNIVERSIDADE ESTADUAL DE CAMPINAS

FACULDADE DE ENGENHARIA MECÂNICA

Relatório Final
Trabalho de Conclusão de Curso

Estudo do controle de pêndulo inverso acoplado a carro comparando controle clássico (PID) e controlador neural (redes neurais)

Autor: Carlos Augusto Jardim Chiarelli
Orientador: Prof. Dr. Tiago Henrique Machado
Coorientador: Dr. Alexandre Mello Ferreira

Campinas, fevereiro de 2023

UNIVERSIDADE ESTADUAL DE CAMPINAS

FACULDADE DE ENGENHARIA MECÂNICA

Relatório Final
Trabalho de Conclusão de Curso

Estudo do controle de pêndulo inverso acoplado a carro comparando controle clássico (PID) e controlador neural (redes neurais)

Autor: Carlos Augusto Jardim Chiarelli
Orientador: Prof. Dr. Tiago Henrique Machado
Coorientador: Dr. Alexandre Mello Ferreira

Curso: Engenharia de Controle e Automação

Trabalho de Conclusão de Curso apresentado à Comissão de Graduação da Faculdade de Engenharia Mecânica como requisito para a obtenção do título de Engenheiro de Automação e Controle.

Campinas, 2023
SP - Brasil

Agradecimentos

Este trabalho não poderia ter sido iniciado e concluído sem a ajuda de diversas pessoas que cruzaram meu caminho em momentos importantes da minha vida.

Sra. Luzia das Graças de Souza Jardim, minha mãe, por sempre acreditar em mim e nunca desistir de me dar o melhor suporte possível para eu alcançar meus objetivos. Por isso e outros motivos ela é a pessoa que mais amo nessa vida.

Prof. Ms. Marcelo Evaristo Ferreira, meu professor de matemática no ensino médio na UNESP de Jaboticabal, foi quem me fez ser apaixonado por matemática além de sempre me incentivar a seguir caminhos desafiadores.

Dr. Alexandre Mello Ferreira que foi meu professor na UNICAMP em introdução a algoritmos, além de um grande amigo foi quem me apresentou o mundo da computação de uma forma bastante brilhante. Foi também uma das pessoas que revisou este trabalho.

Prof. Dr. Tiago Henrique Machado professor do curso de Engenharia de Controle e Automação por ter aceitado a orientar este estudo além de ser um professor com uma didática fascinante.

Por fim deixo um agradecimento a todos bons amigos que fiz nessa trajetória ao longo de 25 anos.

Índice

Resumo	1
Abstract	2
Lista de Figuras	3
Lista de Tabelas	5
Nomenclatura	6
Capítulo 1 Introdução	7
Capítulo 2 Fundamentação Teórica	11
2.1. Breve revisão bibliográfica	11
2.2. Pêndulo invertido	15
2.3. Controle do pêndulo invertido com alocação de polos	17
2.3.1 Alocação de polos com espaço de estados	17
2.3.2 Controlabilidade	18
2.4 Aprendizado de máquina	19
2.5 Aprendizado profundo	21
2.5.1 Funções de ativação	25
2.5.2 Métodos de aprendizado e otimização	27
2.6 Aprendizagem por reforço	31
2.6.1 Processos de Decisão de Markov	33
2.6.2 Algoritmo <i>Deep Q-Network</i>	36
Capítulo 3 Desenvolvimento dos controladores	39
3.1. Controlador PID	39

3.2.	Controlador <i>Deep Q-Network</i>	45
Capítulo 4	Resultados	50
Capítulo 5	Discussão	62
Capítulo 6	Conclusão	64
	Referências Bibliográficas	67

Resumo

CHIARELLI, Carlos Augusto Jardim, *Estudo do controle de pêndulo inverso acoplado a carro comparando controle clássico (PID) e controlador neural (redes neurais)*, Faculdade de Engenharia Mecânica, Universidade Estadual de Campinas, Trabalho de Conclusão de Curso, (2023), 72 pp.

Este trabalho utiliza um algoritmo de *Machine Learning* para equilibrar o pêndulo invertido com um grau de liberdade, comparando o resultado com a técnica de alocação de polos. O algoritmo de aprendizado por reforço *deep Q-Network* foi implementado em *python* e sua performance foi comparada com um controlador clássico. Os resultados mostraram que ambos os métodos conseguiram equilibrar o pêndulo, contudo apenas a alocação de polos conseguiu manter o carro centrado na origem. Os erros *RMSE* em relação à posição angular vertical para os métodos *deep Q-Network* e alocação de polos foram de 0,0052 e 0,0047, respectivamente. Para posição linear, os erros *RMSE* foram 0,0320 e 0,0068, respectivamente.

Palavras Chave: controle de sistemas mecânicos, aprendizado de máquina, pêndulo invertido, aprendizado por reforço.

Abstract

CHIARELLI, Carlos Augusto Jardim, *Study of inverse pendulum control coupled to a car comparing classic control (PID) and neural controller (neural networks)*, University of Mechanical Engineering, Campinas State University, Final paper, (2023), 72 pp.

This work aims to use a Machine Learning algorithm to balance the inverted pendulum with a degree of freedom, comparing the result with the pole allocation technique. The deep Q-Network reinforcement learning algorithm was implemented in python and its performance was compared with the classical controller. The results showed that both methods were able to balance the pendulum; however only the pole allocation managed to keep the car centered at the origin. The RMSE errors regarding the vertical angular position for the deep Q-Network and pole allocation methods were 0.0052 and 0.0047, respectively. For linear position, the RMSE errors were 0.0320 and 0.0068 respectively.

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

Lista de Figuras

Figura 1	Esquema do pêndulo invertido acoplado ao carro.	10
Figura 2	Rede neural de base radial.	12
Figura 3	Sistema carro com pêndulo invertido.	16
Figura 4	Analogia entre neurônio artificial e neurônio biológico.	22
Figura 5	Unidade lógica de limiar: um neurônio artificial que calcula uma ponderada de suas entradas e aplica uma função de grau.	23
Figura 6	RNA com uma camada escondida.	24
Figura 7	Funções de ativação e suas derivadas.	26
Figura 8	Funções de ativação sigmóide, <i>Leaky ReLu</i> e <i>ELU</i> .	27
Figura 9	Gradiente descendente.	29
Figura 10	Armadilhas do gradiente descendente.	30
Figura 11	AdaGrad versus Gradiente Descendente.	30
Figura 12	Representação do problema de aprendizagem por reforço.	32
Figura 13	Exemplo de uma cadeia de Markov.	34
Figura 14	Exemplo de uma matriz de Q-valores onde o algoritmo <i>Q-learning</i> mapeia os pares estado-ação.	37
Figura 15	Pêndulo invertido acoplado ao carro.	40
Figura 16	Posição de equilíbrio do pêndulo.	41
Figura 17	Arquitetura de treinamento.	46
Figura 18	Ambiente <i>CartPole</i> .	47
Figura 19	Treinamento utilizando o método do Otimizador Adam para taxa de aprendizado de 0,004.	50

Figura 20	Força de controle aplicada no carro pelo algoritmo.	51
Figura 21	Respostas do sistema após as forças aplicadas pelo controlador.	52
Figura 22	Esforços de controle (newtons) gerado pelo controlador com as respectivas matrizes de ganho K.	54
Figura 23	Esforço de controle com controlador livre do saturador.	55
Figura 24	Resposta da posição angular θ com controlador livre do saturador.	55
Figura 25	Resposta da posição linear x com controlador livre do saturador.	56
Figura 26	Resposta da posição linear x com controlador sob efeito do saturador.	57
Figura 27	Resposta da posição angular θ com controlador sob efeito do saturador.	57
Figura 28	Resposta da velocidade linear com controlador sob efeito do saturador.	58
Figura 29	Resposta da velocidade angular com controlador sob efeito do saturador.	58
Figura 30	Comparação da resposta da posição angular θ entre os algoritmos.	59
Figura 31	Comparação da resposta da posição linear x entre os algoritmos.	60

Lista de Tabelas

Tabela 1	Vantagens e desvantagens de diferentes métodos de ML para controle do pêndulo invertido.	15
Tabela 2	Comparação dos otimizadores.	31
Tabela 3	Valores das variáveis físicas do sistema mecânico.	40
Tabela 4	Valores dos polos do sistema.	42
Tabela 5	Polos e seus respectivos requisitos.	44
Tabela 6	Matrizes de ganho com variação dos parâmetros do projeto.	53
Tabela 7	Erros em relação à $\theta = 0$ e $x = 0$.	61

Nomenclatura

Abreviações

ML	<i>Machine Learning</i>
DL	<i>Deep Learning</i>
RNA	Rede Neural Artificial
PID	Proporcional Integral Derivativo
EDO	Equação Diferencial Ordinária
SCI	Sistema de Controle Inteligente
SCIE	Sistema de Controle Inteligente Evolucionário
RL	<i>Reinforcement Learning</i>
MLP	<i>Multilayer Perceptron</i>
PDM	Processo de Decisão de Markov
DQN	<i>Deep Q-Network</i>

Capítulo 1

Introdução

Mecanismos para controlar sistemas eletromecânicos fazem parte do nosso dia a dia: um computador possui um controle de temperatura, aviões apresentam controle de altitude e velocidade, e os robôs industriais contém diversos controles para poder produzir diferentes tipos de equipamentos.

O panorama atual ocorre devido a uma longa jornada onde engenheiros de controle estudaram diferentes métodos para controlar sistemas dinâmicos com naturezas distintas. Assim foram consolidadas técnicas em que se aplicam transformações para simplificar sistemas não-lineares e instáveis (DRUMMOND; OLIVEIRA; BAUCHSPIESS, 1999; GUPTA; SINHA, 1999).

Essas técnicas são comumente utilizadas no controle clássico (controlador proporcional integral derivativo, PID) baseado em modelos linearizados dos sistemas físicos. Essa linearização representa perda de informações que muitas vezes são relevantes para o funcionamento de uma planta com alto nível de exigência e para um controle mais preciso e robusto (DRUMMOND; OLIVEIRA; BAUCHSPIESS, 1999).

Hoje se estuda controladores inteligentes, os quais conseguem atuar no sistema e controlando-o sem a necessidade de suas equações físicas terem sido linearizadas (GUPTA; SINHA, 1999). Antes dessa abordagem moderna era necessário um trabalho árduo para mapear as equações que definem o comportamento de cada sistema.

Nise (2007) diz que os problemas de controle clássico de sistemas dinâmicos, como um “pêndulo invertido”, são resolvidos com algoritmos providos das regras e modelagem dos respectivos sistemas. 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 (equação diferencial ordinária) ou saída da função de transferência (KRUL, 2021). Para o controle clássico funcionar é necessário conhecer previamente as equações físicas do sistema.

Pensando nesse problema, foi em 1950 que K. S. Fu apresentou o termo "Controle Inteligente" ou "Sistema de Controle Inteligente" (SCI). Esse tipo de

controlador possui a habilidade de sentir o seu ambiente, processar as informações para reduzir as incertezas nos parâmetros do processo, planejar, gerar e executar ações de controle (CAVALCANTI; ALSINA; FERNEDA, 1999).

Um SCI é capaz de distribuir as tarefas de decisão entre um conjunto de executores de tarefas, utilizando intensivamente os computadores disponíveis para inferir o estado atual do sistema e detectar mudanças no seu estado interno e na sua circunvizinhança. Ele se diferencia de sistemas convencionais por sua habilidade de tomar decisão e por sua capacidade de aprendizagem, que o permite inferir a dinâmica do processo de uma forma adaptativa e preditiva sem que as equações físicas (planta do sistema) sejam previamente conhecidas (CAVALCANTI; ALSINA; FERNEDA, 1999).

Essas são grandes vantagens de um controlador inteligente:

- consegue atuar em sistemas físicos com equações diferenciais desconhecidas;
- não necessita a linearização das equações físicas do sistema.

O controle de sistemas eletromecânicos pode ser feito de forma tradicional com um controlador linear do tipo PID ou com um SCI utilizando técnicas de aprendizado de máquina (*machine learning - ML*) (CAVALCANTI; ALSINA; FERNEDA, 1999).

Para o SCI, geralmente é utilizado um algoritmo de aprendizado profundo (*deep learning - DL*) devido à complexidade das equações do sistema (CARVALHO; JOTA, 1999). Em muitos casos, a Rede Neural Artificial (RNA) atua como o controlador do sistema, precisando interagir com ele para entender seu comportamento e ajustar os parâmetros apreendidos (CAVALCANTI; ALSINA; FERNEDA, 1999). Uma outra abordagem interessante para realizar o controle é aplicar algoritmos genéticos que utilizam um controlador adaptativo evolucionário (DRUMMOND; OLIVEIRA; BAUCHSPIESS, 1999; CAVALCANTI; ALSINA; FERNEDA, 1999).

As maiores desvantagens das técnicas de inteligência artificial para controle de sistemas são: a necessidade de ter disponível um conjunto de dados de treinamento (considerando o aprendizado supervisionado), o risco do algoritmo ficar preso em mínimos locais durante o aprendizado e as incertezas inerentes às técnicas (CARVALHO; JOTA, 1999). Ter um conjunto de dados significa dizer que

houve um controlador programado para extração dos dados, o que nem sempre é possível.

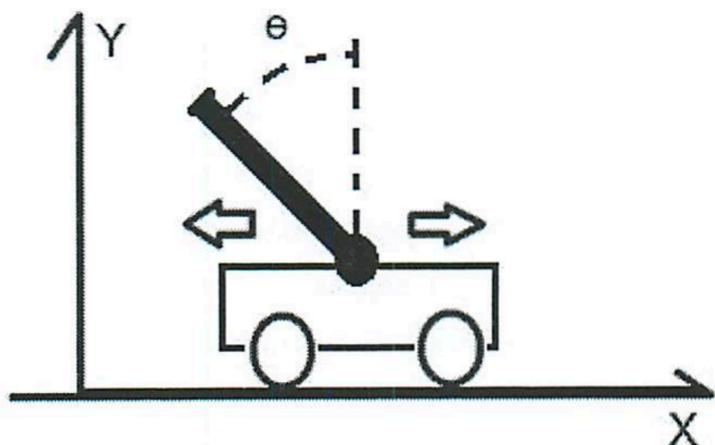
Os dados de treino representam exemplos de como a RNA, por exemplo, deve se comportar em diversas situações, isto é, são pares de entrada e saída do controlador. Para obter estes dados (exemplos), é necessário ter, a priori, uma estratégia de controle definida (CARVALHO; JOTA, 1999). Muitas vezes isto não é possível e nem desejável, já que o que se quer é exatamente obter a melhor estratégia de controle.

Durante o processo de aprendizado existe a minimização de uma função de custo que define a direção (gradiente) do aprendizado para se obter o menor erro da resposta possível. Contudo, durante o processo, pode acontecer do algoritmo ficar preso em um mínimo local da hipersuperfície de erro e encerrar o aprendizado dando a falsa impressão de um possível mínimo global (GORI; TESI, 1992).

Dadas as diferentes formas de controle, o presente estudo visa realizar o controle do “pêndulo invertido” utilizando um controle clássico (controlador linear PID) e um controle inteligente (com RNA) sendo, ao final, o desempenho dos modelos comparados. A avaliação do desempenho utiliza as métricas de controle clássico como sobre-elevação e tempo de estabilização.

O pêndulo invertido sendo um sistema instável e não-linear se torna um problema de controle com ótimas características dinâmicas para verificar a eficácia e qualidade dos controladores (CARVALHO; JOTA, 1999). O sistema consiste em uma haste vertical (pêndulo) presa a um carro motorizado com um grau de liberdade que se movimenta sobre um trilho (ALVARENGA, 2013). A variável manipulada é a força que se exerce no carro. O objetivo do controle é manter o pêndulo equilibrado na posição vertical mesmo quando perturbações são aplicadas ao sistema.

Figura 1 - Esquema do pêndulo invertido acoplado ao carro.



Fonte: Cavalcanti, Alsina, Ferneda (1999).

A Figura 1 representa o sistema mecânico deste estudo. É possível notar o deslocamento do carro para frente e para trás (variável x) e o movimento angular do pêndulo (variável θ). A única interação possível com o sistema é aplicar uma força horizontal no centro do carro na direção positiva ou negativa de x . A direção y não é utilizada (CAVALCANTI; ALSINA; FERNEDA, 1999).

O objetivo deste estudo é comparar o desempenho do método clássico (PID) com uma técnica aprendizado de máquina (aprendizado por reforço) aplicado ao problema do pêndulo invertido com um grau de liberdade.

Capítulo 2

Fundamentação teórica

Neste capítulo serão abordados alguns estudos onde pesquisadores controlaram o pêndulo invertido utilizando diferentes técnicas de aprendizado de máquina. Em seguida, o sistema mecânico escolhido será apresentado junto com o detalhamento da técnica de aprendizado escolhida.

2.1 Breve revisão bibliográfica

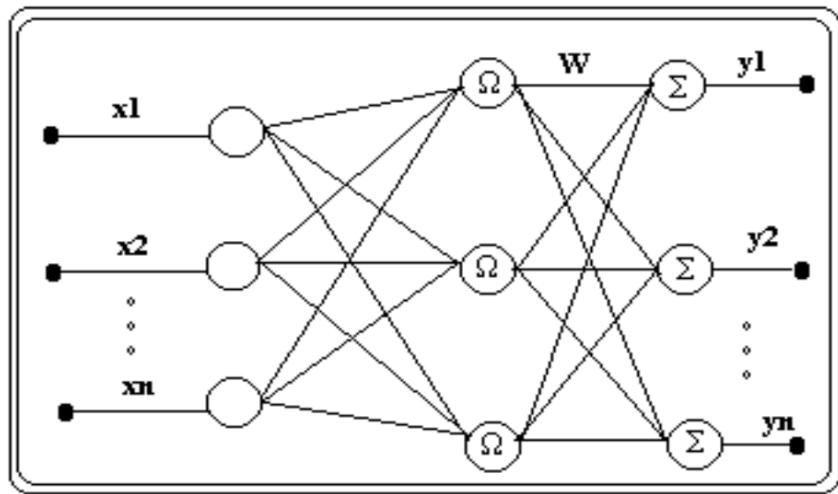
Trabalhos com controladores inteligentes tem se tornado cada vez mais comum devido a popularização de técnicas de DL com o fácil acesso a alto poder computacional.

Drummond, Oliveira e Bauchspieess (1999) afirmam que é possível controlar o pêndulo invertido utilizando uma rede neural de base radial. Trata-se de uma rede que utiliza como função de ativação uma função gaussiana. Ela é radialmente simétrica e as constantes média e desvio padrão são hiperparâmetros a serem definidos antes do treinamento.

Essa função de ativação foi escolhida pela autora por possuir treinamento mais rápido e uma melhor generalização quando comparado a funções mais simples como a degrau (*step*) permitindo fugir de mínimos locais (DRUMMOND; OLIVEIRA; BAUCHSPIESS, 1999).

O modelo preditivo é uma “*Perceptron*” multicamadas (*MLP - Multi Layer Perceptron*) de uma única camada escondida. A rede neural possui os neurônios totalmente conectados entre todas as camadas com apenas uma camada escondida com 247 neurônios e um único neurônio de saída.

Figura 2 - Rede neural de base radial.



Fonte: Drummond, Oliveira, Bauchspieß (1999).

A Figura 2 representa a arquitetura da rede neural de base radial. Cada círculo representa 1 nó (neurônio). Ela apresenta 3 camadas. A primeira possui as entradas do sistema (X_1, X_2, \dots, X_n). A segunda é a camada escondida e Ω a função de ativação gaussiana. A terceira é a camada de saída onde cada nó possui uma somatória simples gerando como saída um escalar (DRUMMOND; OLIVEIRA; BAUCHSPIESS, 1999).

Como comentado, a função de ativação é uma função de base radial (função gaussiana) e pode ser visualizada na Equação 1.

$$f(x) = \exp\left[-(x - K_e)^2 / 2\sigma^2\right] \quad (1)$$

As constantes K_e e σ se referem ao valor médio e espalhamento da gaussiana, respectivamente.

A rede neural foi utilizada diretamente com o controlador, ou seja, recebe as quatro entradas do sistema que são posição linear, posição angular, velocidade linear e velocidade angular.

Para ajuste do controlador neural foi feito um treinamento não supervisionado e um supervisionado. O primeiro com intuito de determinar as constantes K_e e σ para que elas não fossem arbitrárias aleatoriamente. O segundo é realizado conhecendo a resposta desejada (*target*), mensurando os erros e atualizando os pesos da rede

após cada iteração (época) com os dados. Os dados de treinamento foram um conjunto de entrada-saída com 400 amostras.

Drummond, Oliveira e Bauchspieß (1999) obtiveram bons resultados onde o controlador neural se comportou muito bem com um resultado robusto comparado ao controlador PID clássico tanto em simulação quanto em bancada. Além disso, validou o método de aprendizado não supervisionado para definir os parâmetros estáticos da função de base radial de cada neurônio.

Um ponto que vale ressaltar é o fato da obrigatoriedade de um conjunto de treino onde, além disso, foi necessário utilizar o modelo linear do sistema para comparar com o erro do controlador inteligente. Esses dois itens nem sempre são triviais para um sistema real que a planta é desconhecida.

Géron (2021) mostra que existem quatro tipos de aprendizado de máquina: supervisionado, não supervisionado, semissupervisionado e aprendizado por reforço. Neste trabalho será utilizado o método aprendizado por reforço.

Aprendizado supervisionado envolve dados rotulados ou anotados onde é conhecida a variável de saída. Para o não supervisionado a variável de saída (*target*) é desconhecida e o modelo tenta descobrir relações entre os dados por si só. Semissupervisionado para parte dos dados se conhece a variável que se deseja prever enquanto a outra parte é desconhecida. O RL é uma categoria diferente das anteriores, já que existe um agente que interage com o ambiente ganhando recompensas ou sendo penalizado de acordo com cada ação tomada.

Uma coisa em comum entre todos os métodos mencionados é que existe uma métrica de avaliação de desempenho para saber se o modelo está se saindo bem ou mal. Geralmente se quer fazer uma predição ou estimar uma variável que minimize uma função de custo que mostrará o caminho para o modelo atingir o melhor desempenho possível. Neste trabalho, o melhor resultado é controlar o pêndulo na vertical mantendo o carro na posição de origem.

No aprendizado baseado em modelo de ML acontece a interação com o conjunto de dados (treinamento) ou ambiente com intuito de minimizar a função de custo. Essa minimização ocorre com o ajuste dos parâmetros de aprendizado que armazenam informações que serão salvas após o algoritmo encontrar um mínimo local ou mínimo global no espaço de busca definido pelo erro.

Um exemplo de parâmetros de aprendizado são os coeficientes linear e angular de uma regressão linear (equação 2) representados pela letra grega θ .

$$y = \theta_0 + \theta_1 x \quad (2)$$

Krul (2021) procurou uma abordagem diferente para a solução do pêndulo invertido. Ao invés de utilizar aprendizado supervisionado e não supervisionado, utilizou três tipos diferentes de aprendizado por reforço (*reinforcement learning* - RL): *HillClimbing*, *REINFORCE* e *Deep Q-Network*. Os dois últimos mais recentes são compostos por redes neurais, enquanto o primeiro faz uso apenas de uma política a ser seguida.

RL é um 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 (KRUL, 2021). O ambiente consiste nas posições e velocidades lineares e angulares enquanto o agente pode atuar uma força (ação) que pode levá-lo a outro estado, sendo considerado um Processo de Decisão de Markov (KRUL, 2021).

Os três diferentes algoritmos de RL se mostraram robustos e com resultados satisfatórios comparados a um controlador clássico PID.

Krul (2021) provou que não é necessário um conjunto de treino ou um controlador pré-programado para gerar um controlador inteligente que possa aprender interagindo com o sistema sem conhecer suas equações físicas.

Uma outra abordagem interessante é utilizar inteligência artificial clássica para definir parâmetros de controladores clássicos, tal como definir os ganhos proporcional, integral e derivativo do controlador. Foi o que fez Alvarenga (2013) levantando a planta do pêndulo invertido experimentalmente e controlando-a com três tipos de técnicas distintas: *Fuzzy*, PID e PID gerado por algoritmos genéticos.

Nessa abordagem, o autor demonstra que é possível utilizar um algoritmo evolutivo para gerar controladores do tipo PID chegando a resultados satisfatórios. Apesar da ideia de determinação dos ganhos ser interessante, a eficácia do controlador não é boa o suficiente sendo conhecido o modelo matemático preexistente do problema em questão (ALVARENGA, 2013).

Existem outras abordagens para construir um controlador inteligente, uma delas é a abordagem evolutiva. Cavalcanti, Alsina e Ferneda (1999) fazem o controle do pêndulo invertido utilizando um Sistema de Controle Inteligente Evolucionário

(SCIE) baseado em Controlador Neural, Lógica Nebulosa e Algoritmo Genético. No trabalho, os autores desenvolveram regras para o treinamento do algoritmo que, de acordo com a posição (quadrante) do pêndulo, definem se a força aplicada e o incremento dela devem ser positivos ou negativos. Além disso, utilizaram um motor para gerar o torque no sistema, esse dispositivo não será considerado no presente trabalho sendo a força de interesse aplicada diretamente no pêndulo. O bloco que contém o algoritmo genético dentro da arquitetura é responsável por definir os fatores de adaptação utilizados pelo controlador neural que se trata de uma rede neural multicamada totalmente conectada. Segundo Cavalcanti, Alsina e Ferneda, o SCIE é capaz de garantir o posicionamento do pêndulo invertido na maioria das vezes, mesmo falhando em algumas tentativas.

Carvalho e Jota (1999) mostram que também é possível controlar o pêndulo a partir de uma RNA treinada por uma técnica evolucionária. Um bom resultado conseguido por Carvalho e Jota foi a rede neural atingir um tempo de subida bastante baixo e uma baixa sobre-elevação (*overshoot*) para manter o pêndulo na posição desejada comparado a um controlador comercial. Um ponto é que a solução comercial se mostrou mais conservadora mantendo um esforço de controle e ângulo do pêndulo mais baixos.

Na Tabela 1 é possível ver as vantagens e desvantagens de cada trabalho.

Tabela 1 - Vantagens e desvantagens de diferentes métodos de ML para controle do pêndulo invertido.

Autor	Método	Vantagens	Desvantagens
Drummond	Aprendizado supervisionado e não supervisionado	Não é necessário definir os parâmetros do controlador manualmente	Necessidade de um conjunto de treino e modelo linear do sistema (<i>input/output</i>)
Krul	Aprendizado por reforço	Não é necessário conhecimento do sistema nem conjunto de treino	Necessário o ambiente de simulação (modelo real virtual)
Alvarenga	Algoritmo genético	Não é necessário definir os parâmetros do controlador manualmente	Necessidade do modelo linear do sistema (<i>input/output</i>) e baixo desempenho do controlador

Cavalcanti	Algoritmo genético	Não é necessário definir os parâmetros do controlador manualmente	Necessidade do modelo linear do sistema (<i>input/output</i>)
Carvalho	Algoritmo genético	Não é necessário definir os parâmetros do controlador manualmente	Necessidade do modelo linear do sistema (<i>input/output</i>)

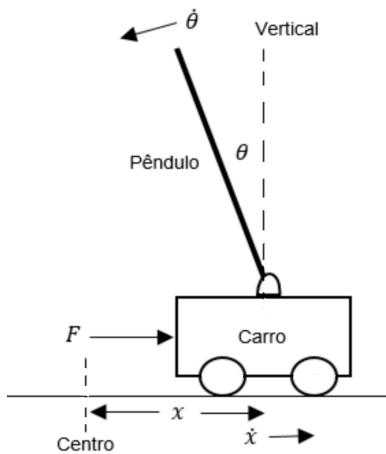
Fonte: Autor.

2.2 Pêndulo invertido

O carro com pêndulo invertido é um sistema mecânico bastante conhecido por ser naturalmente instável e não-linear. Isso significa que não é possível manter o pêndulo em uma posição desejada já que a gravidade tende a puxá-lo para baixo. As não-linearidades acontecem devido a duas equações diferenciais de segunda ordem, uma que caracteriza a dinâmica do carro (Equação 3) e a outra que caracteriza a do pêndulo (Equação 4). Essas equações podem ser desacopladas e resolvidas a partir de uma técnica chamada “espaço de estados”, que será utilizada no decorrer deste trabalho.

Nagendra et al. (2017) apresentam esse sistema como um pêndulo conectado a um carro que possui um grau de liberdade ao longo do eixo horizontal. O problema que se busca resolver é equilibrar o pêndulo na vertical aplicando no carro apenas forças bidirecionais na horizontal para frente e para trás, como é possível visualizar na Figura 3.

Figura 3 - Sistema carro com pêndulo invertido.



Fonte: Cavalcanti, Alsina, Ferneda (1999).

A seguir, a Equação 3 representa a aceleração linear do carro (KRUL, 2021).

$$\ddot{x} = \frac{F + ml(\dot{\theta}^2 \sin\theta - \ddot{\theta} \cos\theta)}{(M+m)} \quad (3)$$

A Equação 4 representa a aceleração angular do pêndulo (KRUL, 2021).

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

O problema do pêndulo invertido é um sistema determinístico, ou seja, possui um conjunto de entradas conhecido que resultará em um único conjunto de saídas. Assim, é possível que o sistema seja modelado analiticamente caso não tenha muitas variáveis envolvidas (OGATA; YANG, 2002).

Segundo Nagendra et al. (2017) o estado do sistema é definido de acordo com quatro variáveis que podem ser visualizadas na Figura 3:

- θ - posição angular;
- $\dot{\theta}$ - velocidade angular;
- x - posição linear;
- \dot{x} - velocidade linear.

Todas as medidas angulares estão relacionadas ao pêndulo enquanto as lineares ligadas ao carro.

As demais variáveis que aparecem nas equações 3 e 4 são:

- M - massa do carro;

- m - massa do pêndulo;
- g - aceleração da gravidade;
- F - força aplicada no carro;
- I - distância entre o centro de massa do pêndulo para o eixo que conecta ao carro.

Podemos entender o problema como uma caixa preta que contém as equações diferenciais onde se aplica uma força (entrada) e se obtém quatro respostas (saídas).

Essa caixa preta não é obscura pelo fato de se conhecer as equações do sistema, contudo o algoritmo de ML não precisa saber delas, uma vez que ele interage com o sistema até que consiga entender como este funciona e, então, equilibrar o pêndulo na vertical.

2.3 Controle do pêndulo invertido com alocação de polos

2.3.1 Alocação de polos com espaço de estados

Uma das formas de controlar o pêndulo invertido é a alocação de polos em espaço de estados, esta é uma técnica de controle clássica e será usada no presente estudo.

Shehu et al. (2015) afirmam que este método consiste em um ajuste intuitivo dos parâmetros de controle conhecendo o comportamento do sistema no que diz respeito à disposição dos polos e zeros no plano complexo.

As Equações 5.1, 5.2, 5.3 e 5.4 representam um sistema SISO (única entrada e única saída) em espaço de estado.

$$x' = Ax + Bu \quad (5.1)$$

$$y = Cx + Du \quad (5.2)$$

$$x = [x, x', \theta, \dot{\theta}] \quad (5.3)$$

$$u = -Kx \quad (5.4)$$

Nas Equações 5.1, 5.2, 5.3 e 5.4 x representa o vetor de estados, na Equação 5.2 y é o sinal de saída, u (Equação 5.4) é a força aplicada no carro (sinal de controle). Nas equações 5.1 e 5.2, A é uma matriz constante $n \times n$ que varia de

acordo com as propriedades mecânicas do sistema, B uma matriz constante $n \times 1$, C uma matriz constante $1 \times n$ e D uma matriz constante.

Segundo Ogata e Yang (2002) ao definir corretamente a matriz A é possível fazer com que o sistema fique estável ajustando a posição dos polos em malha fechada e garantindo uma frequência natural e fator de amortecimento desejado para o sistema de segunda ordem.

A Equação 5.4 indica o esforço de controle após o ajuste dos polos em malha fechada. A matriz K é a matriz de ganho de realimentação que ajusta os autovalores $A - BK$ para os valores que tornam o sistema estável.

Como descrito por Krul (2021), 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, sobre-elevação máxima, 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 K e é adicionada como um compensador no sistema em malha fechada.

2.3.2 Controlabilidade

A controlabilidade de um sistema mecânico é determinada pela capacidade dele alcançar e manter uma condição de equilíbrio desejada com a aplicação de uma entrada controlável. Em outras palavras, é a capacidade do sistema de ser governado pelo sinal de controle.

Para analisar a controlabilidade, a equação de estado do sistema é primeiro formulada em uma representação matemática, que é usada para modelar a dinâmica do sistema. A partir daí, a teoria de controle linear é aplicada para avaliar a resposta do sistema à entrada controlável (OGATA; YANG, 2002).

A análise de controlabilidade envolve a avaliação dos autovalores e autovetores do sistema linear associado. Se todos os autovalores tiverem parte real positiva, o sistema não é controlável. Se, por outro lado, houver pelo menos um autovalor com parte real negativa, o sistema é controlável (OGATA; YANG, 2002).

Em resumo, a controlabilidade é uma medida importante para avaliar a eficiência e a efetividade do controle de um sistema mecânico.

Desta forma, antes de projetar o controlador através da alocação de polos é necessário verificar se o sistema é controlável.

Segundo Pereira (2022), dado o sistema de estado da equação 5.1, o sistema é considerado controlável caso a matriz $T^{n \times n}$ (Equação 6) possuir posto n .

$$T = [B \ AB \ A^2B \ \dots \ A^{n-1}B] \quad (6)$$

O posto de uma matriz corresponde ao número de linhas ou colunas linearmente independentes dela. Nesse caso particular, como se trata de uma matriz $n \times n$, posto igual a n é equivalente a determinante não nulo. (PEREIRA, 2022).

2.4 Aprendizado de máquina

Segundo Géron (2021), *Machine Learning* é a ciência (e a arte) da programação de computadores de modo que possam aprender com os dados.

Tom Mitchell (1997) elaborou uma definição de ML orientada a engenharia:

Alega-se que um programa de computador aprende pela experiência E em relação a algum tipo de tarefa T e alguma medida de desempenho P se o seu desempenho em T, conforme medido por P, melhora com a experiência E. (MITCHELL, 1997)

No presente estudo, o controlador inteligente é um programa de ML que pode aprender a controlar o pêndulo. Os exemplos utilizados pelo sistema para o aprendizado são chamados de conjunto de treinamento (para aprendizado supervisionado) ou ambiente de treinamento (para RL) onde o agente interage com ele. Nesse caso, a tarefa T é a força que deve ser aplicada no carro, a experiência E é a resposta do sistema (Equação 5.3) e a medida P pode ser definida como a diferença (erro) do pêndulo da posição na vertical e a distância do carro na origem.

O intuito do algoritmo é construir regras ou políticas que ao interagir com o sistema tenham o máximo desempenho a partir da métrica P definida. O erro entre o valor correto que se deseja atingir (pêndulo na vertical e carro na posição de origem) é definido por uma função de custo na qual se deseja minimizar.

Segundo Géron (2021) um projeto de ML consiste em:

- estudar os dados;
- selecionar o modelo;
- treinar o modelo nos dados de treinamento ou interagir no ambiente;
- aplicar o modelo para fazer previsões em novos casos ou no ambiente avaliado.

Em resumo, Krul (2021) expressa bem os aspectos genéricos de um modelo de ML:

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. (KRUL, 2021)

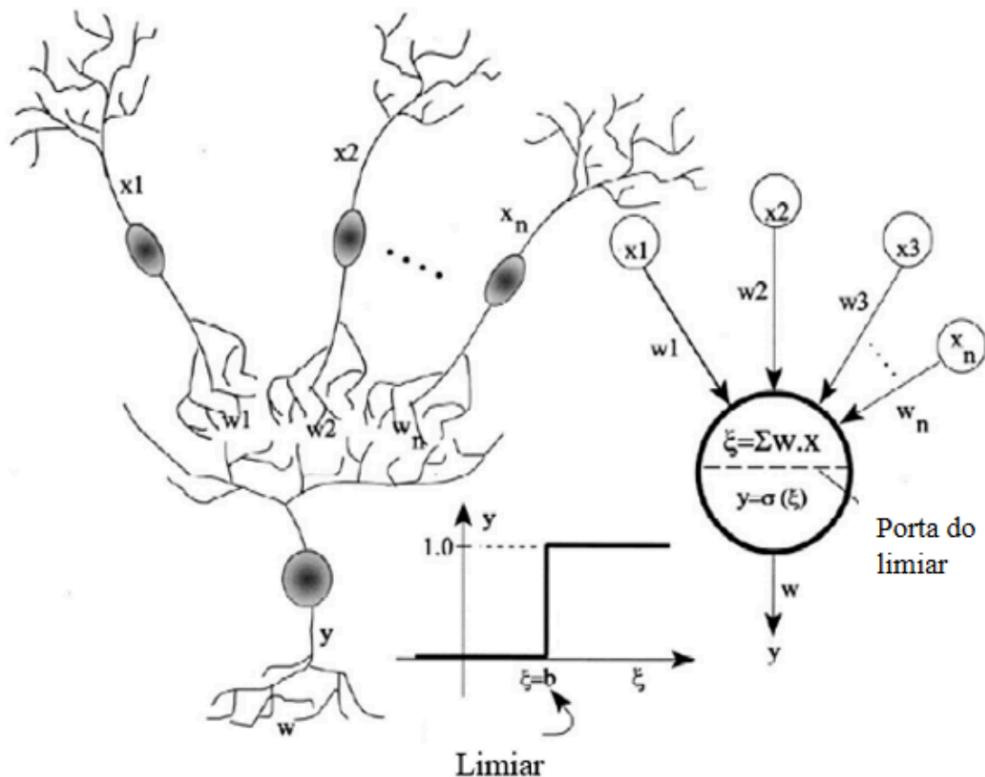
2.5 Aprendizado profundo

Parte das invenções concebidas pelos humanos foram inspiradas em elementos da natureza. Os aviões se basearam nos pássaros e as plantas do tipo bardana influenciaram a criação do velcro, além de outros exemplos.

Logo surgiram as RNAs que foram baseadas nas redes neurais biológicas (RNBs). E, assim, surge uma nova área de estudo dentro de ML conhecida como DL, que envolve modelos preditivos compostos por RNAs.

Krul (2021) cita uma analogia entre uma RNA e uma RNB. A RNA (a direita da Figura 4) possui n neurônios com vários sinais de intensidade x e força sináptica w alimentando um neurônio com limiar de ativação b , essa é arquitetura que permite a comparação de uma RNB (a esquerda da Figura 4).

Figura 4 - Analogia entre neurônio artificial e neurônio biológico.



Fonte: Carvalho, Jota (1999).

Krul (2021) ainda conclui que 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 (*weight*), 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*.

Géron (2021) diz que a *perceptron* é uma das arquiteturas mais simples de RNA, inventada em 1957 por Frank Rosenblatt. Ela toma como base um neurônio artificial ligeiramente diferente chamado unidade lógica de limiar (LTU) que faz a soma ponderada das entradas multiplicada pelos parâmetros w e aplica uma função de ativação que passa a informação para o próximo neurônio caso o limiar seja atingido.

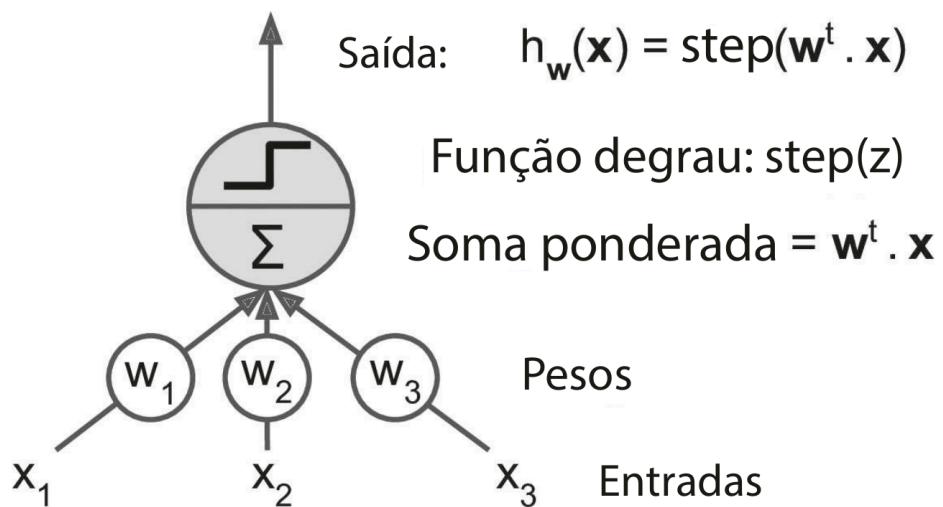
Géron (2021) coloca que a TLU calcula uma soma ponderada de suas entradas (Equação 7) e aplica uma função degrau que gera o resultado ou hipótese (Equação 8). As siglas x e w da Equação 7 representam vetores.

$$z = w_1 x_1 + w_2 x_2 + \dots + w_n x_n = w^T x \quad (7)$$

$$h_w(x) = degrau(z) \quad (8)$$

As equações 7 e 8 podem ser modeladas visualmente na Figura 5.

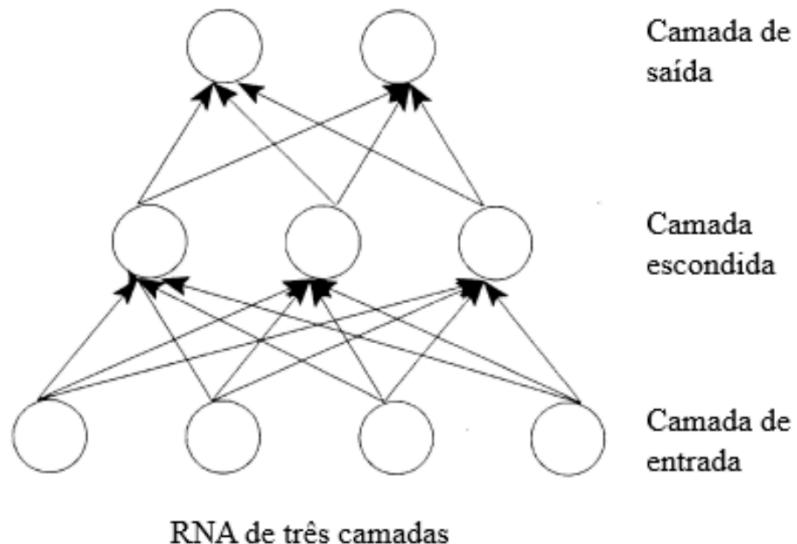
Figura 5 - Unidade lógica de limiar: um neurônio artificial que calcula uma ponderada de suas entradas e aplica uma função de grau.



Fonte: Géron (2019).

O perceptron não é capaz de resolver problemas não-lineares e por isso criaram uma nova estrutura conectando vários neurônios entre si e criando camadas. Essa estrutura foi batizada de perceptron multicamada (*multilayer perceptron* - MLP). Géron (2021) diz que a classificação de uma RNA como DL é um tanto confusa, mas se a estrutura possuísse 2 camadas escondidas ela já entrava nessa categoria segundo definições de 1990. A Figura 6 mostra uma MLP com uma única camada escondida, uma camada de entrada e uma camada de saída.

Figura 6 - RNA com uma camada escondida.



Fonte: Krul (2021).

Existem também neurônios de viés que são constantes ligados a cada neurônio da camada de saída e que geralmente ficam explícitos.

Ainda segundo Géron (2021), hoje o aprendizado desse sistema é feito utilizando um algoritmo de otimização genérico chamado gradiente descendente que consegue identificar ótimas soluções para vários problemas. A ideia geral dele é ajustar iterativamente parâmetros com o intuito de minimizar uma função de custo procurando o mínimo global no espaço dos parâmetros (n -dimensional) mas podendo encalhar em um mínimo local.

Em 1986, David Rumelhart, Geoffrey Hinton e Ronald Williams apresentaram o algoritmo de treinamento de retropropagação (algoritmo de *backpropagation*) que é utilizado para atualizar os pesos w de cada neurônio.

Conforme Géron (2021) coloca, para cada instância (exemplo) de treinamento, o algoritmo de retropropagação primeiro faz uma predição (*forward pass*) e calcula o erro, depois passa por cada camada no sentido inverso a fim de calcular a contribuição do erro de cada conexão (*reverse pass*) e, por fim, ajusta os pesos (parâmetros) da conexão para reduzir o erro (etapa do gradiente descendente, a qual é baseada no cálculo de derivadas parciais através da regra da cadeia).

O algoritmo leva esse nome por iniciar a predição no sentido da camada de entrada para a camada de saída, calcula o erro e então volta atualizando os parâmetros da RNA no sentido oposto (do fim para o início) utilizando o gradiente descendente.

2.5.1 Funções de ativação

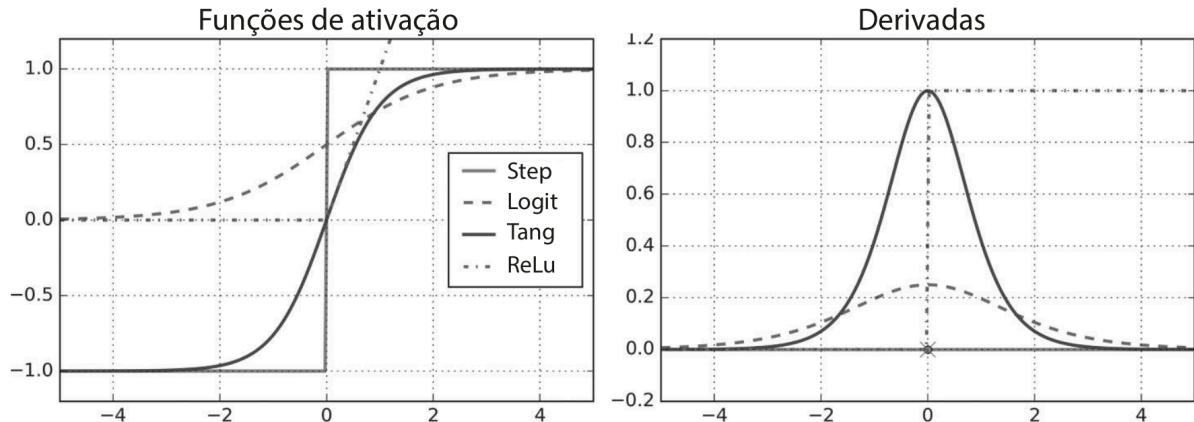
A função de ativação é o operador aplicado a soma ponderada das entradas dos neurônios com seus respectivos pesos. Caso o valor de entrada dessa função atinja determinado limiar, o neurônio é ativado e a informação é passada adiante por ser relevante para minimizar a função de custo. Caso o limiar de ativação não seja atingido acontece o oposto e o neurônio não passa a informação adiante.

Para o perceptron, inventado em 1957, a função de ativação era uma função degrau, sendo mais específico uma função *Heaviside* (função degrau unitário - equação 9). Quando a soma ponderada das entradas ultrapassa o limiar o resultado é 1 (ativação) senão é 0 (desativação) (GÉRON, 2021).

$$heaviside(z) = \begin{cases} 0 & \text{se } z < 0, \\ 1 & \text{se } z \geq 0. \end{cases} \quad (9)$$

Existem diversas outras funções de ativação, cada uma com comportamento diferente. A Figura 7 mostra algumas funções de ativação (a esquerda) e suas derivadas (a direita), no ponto $x = 0$ é o limiar que define a informação a ser passada para o próximo neurônio ou próximo estágio.

Figura 7 - Funções de ativação e suas derivadas.



Fonte: Géron (2019).

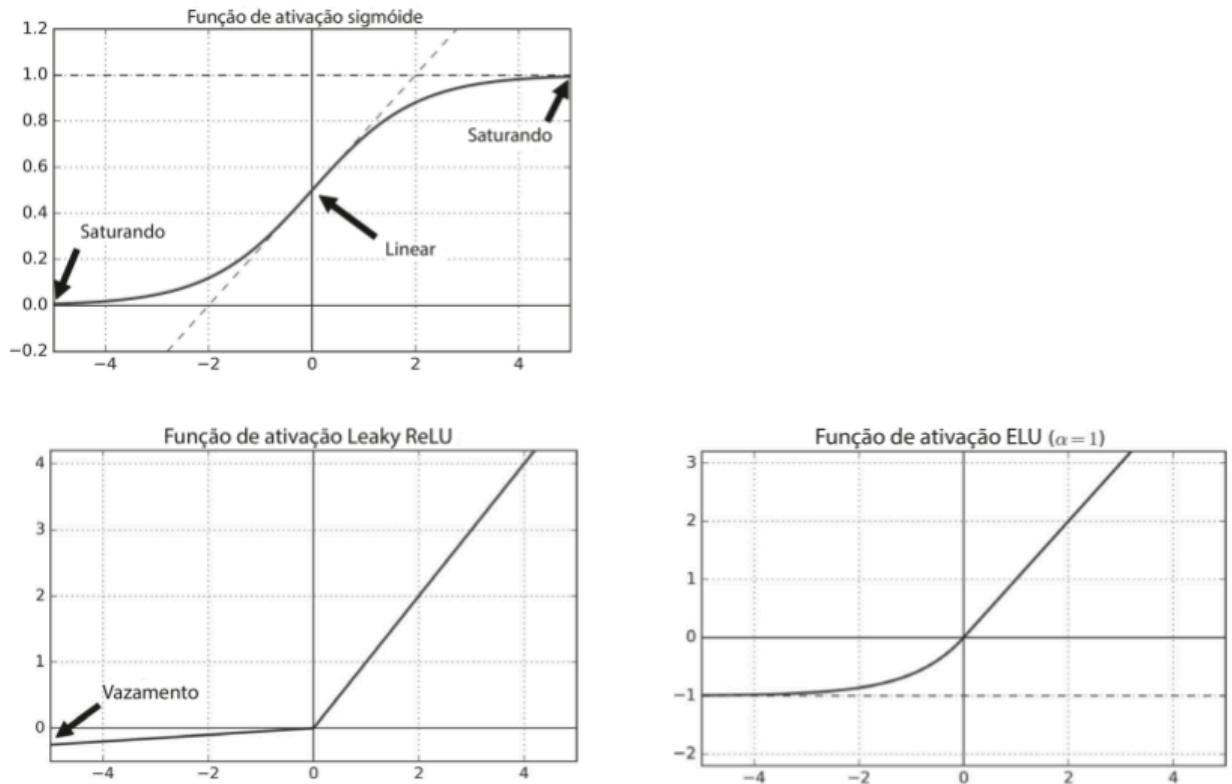
É interessante observar a derivada de cada uma delas. Enquanto a tangente hiperbólica (Tang) apresenta uma derivada contínua, a ReLu mostra um salto que vai de zero para um valor constante. Cada problema de ciência de dados requer diferentes funções de ativações. A função sigmóide (equação 10) que possui uma classificação binária suave, apesar de simples, apresenta bons resultados em casos gerais.

$$A(x) = \frac{1}{1+e^{-x}} \quad (10)$$

Existem outras funções de ativação que são ótimas em evitar problemas de gradientes de fuga e explosão de gradiente. Segundo Géron (2021) tais problemas ocorrem quando os gradientes ficam cada vez menores ou cada vez maiores, quando circulam de maneira reversa através da RNA durante o treinamento. Esses dois problemas dificultam o treinamento das camadas iniciais/inferiores.

Exemplos dessas funções são *Leaky ReLu* e *ELU*. São funções de ativação de não saturação pois não ficam presas a intervalos para entradas muito maiores ou muito menores que zero, diferente da função logística que sofre saturação como é mostrado na Figura 8. A ReLu, por exemplo, não zera a saída para valores de entrada menores que zero (vazamento).

Figura 8 - Funções de ativação sigmóide, Leaky ReLU e ELU.



Fonte: Géron (2019).

Uma última função de ativação criada é a SELU (*Scaled ELU*). É uma variação da ELU que oferece algumas vantagens em uma rede neural totalmente conectada como normalização dos dados que trafegam internamente, assim ela mantém a média em 0 e o desvio-padrão em 1 das arestas durante o treinamento evitando problemas do gradiente de fuga/explosão. (Géron, 2021). Além disso, SELU geralmente tem um desempenho substancial em relação às demais.

Assim, Géron (2021) indica a escolha da função de ativação na seguinte ordem: SELU, ELU, *leaky* ReLU (e variantes), ReLU, tanh (tangente hiperbólica), logística. Considerando as características das funções apresentadas, a ReLU será a escolhida para implementação da RNA no presente trabalho já que o pêndulo invertido não é um sistema de altíssima complexidade.

2.5.2 Métodos de aprendizado e otimização

O que caracteriza um sistema inteligente é a capacidade de aprender as regras que resolvem o problema de forma autônoma, sem que essas regras sejam escritas por humanos. Tais regras são chamadas de hipóteses e para problemas determinísticos são desenvolvidas manualmente por humanos.

Conforme Krul (2021) coloca em seu trabalho, no DL 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 minimizando a função custo J a partir, geralmente, do erro quadrático médio.

A seguir, a Equação 11 indica a função de custo J onde N representa o número de exemplos de treinamento. \hat{y}_i é o valor predito e y_i o valor real.

$$J = \frac{1}{N} \sum_{i=1}^N (\hat{y}_i - y_i)^2 \quad (11)$$

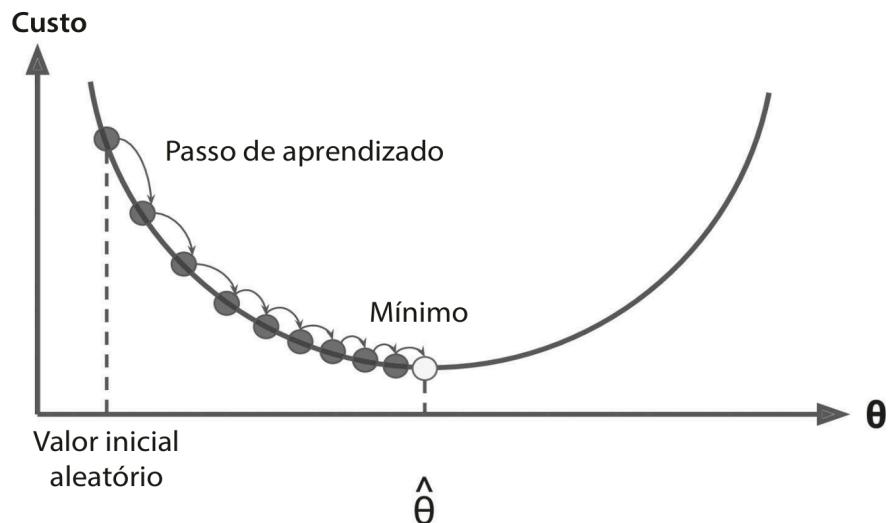
Esse é o processo de treinamento da RNA. Alguns métodos de otimização utilizados nele são o método do gradiente, Adagrad e Adam que atualizam os pesos da RNA.

Krul (2021) menciona que o método Gradiente Descendente utiliza a derivada de primeira ordem da função custo para calcular em qual sentido os pesos da RNA devem ser atualizados para que J atinja seu mínimo, de acordo com a Equação 12. Nessa equação α é um escalar denominado coeficiente de aprendizado e ele determina o tamanho da atualização dos pesos. $\nabla_w J(w)$ é o gradiente da função custo em relação aos pesos w_t .

$$w_{t+1} = w_t - \alpha \nabla_w J(w) \quad (12)$$

Imaginado que se trate de um problema unidimensional, o vetor de pesos w seja composto por um único elemento chamado de θ , é ilustrado na Figura 9 o processo de minimização da função de custo em que o passo de aprendizado é definido por α .

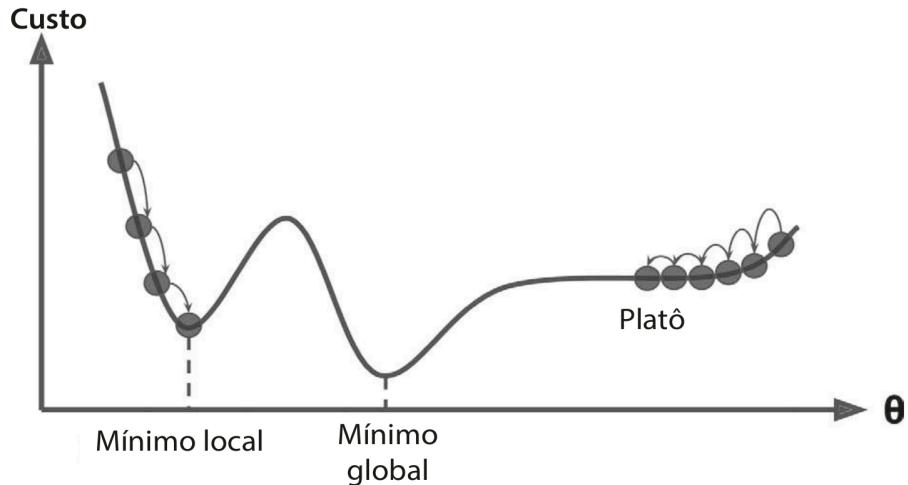
Figura 9 - Gradiente descendente.



Fonte: Géron (2019).

Na Figura 10 visualizamos problemas em que o gradiente pode ficar encalhado em um mínimo local podendo passar a falsa impressão de mínimo global. Existem também casos nos quais o espaço da função de custo seja muito pouco íngreme, fazendo com que o gradiente leve muito tempo para convergir, chamado de platô, como é observado na parte da direita da Figura 10 onde o gradiente descendente começa descendo rapidamente o declive mais íngreme, que não aponta diretamente para o ideal global, em seguida ele desce bem lentamente até o fim do vale (mínimo global).

Figura 10 - Armadilhas do gradiente descendente.

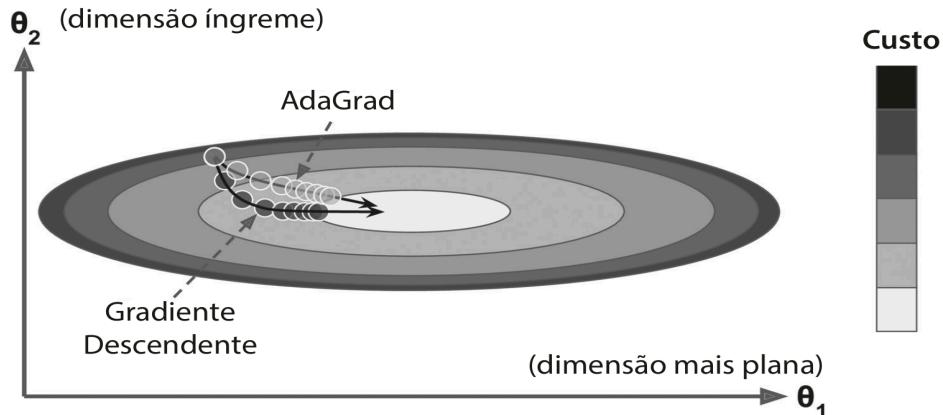


Fonte: Géron (2019).

O algoritmo de otimização Adagrad consegue corrigir a direção anterior do gradiente para apontar um pouco mais para o ideal global (COLOMBINI, 2019).

A Figura 11 ilustra a comparação do método gradiente com o Adagrad em um problema bidimensional com dois parâmetros a serem ajustados, θ_1 e θ_2 .

Figura 11 - AdaGrad versus Gradiente Descendente.



Fonte: Géron (2019).

A otimização Adam (estimativa adaptativa do momento), segundo Géron (2021), combina duas ideias de otimização:

1. uma bola de boliche descendo por um declive suave em uma superfície lisa que começa devagar e rapidamente ganha impulso até atingir a velocidade final (otimização *momentum*);

- acumular somente gradientes mais recentes no processo de atualização dos pesos que evita o risco de desacelerar rápido demais e nunca convergir para o ideal global (otimização *RMSProp*).

A Tabela 2 compara os otimizadores:

Tabela 2 - Comparaçāo dos otimizadores.

Convergência de classe	Velocidade	Qualidade
Gradiente descendente	ruim	bom
Adagrad	bom	ruim (para muito cedo)
Adam	bom	mediano ou bom

Fonte: Géron (2021).

Géron (2021) aponta que, via de regra, os métodos de otimização adaptativa (otimização Adam) são ótimos e convergem rapidamente para uma boa solução. Considerando isso, ele será adotado como otimizador.

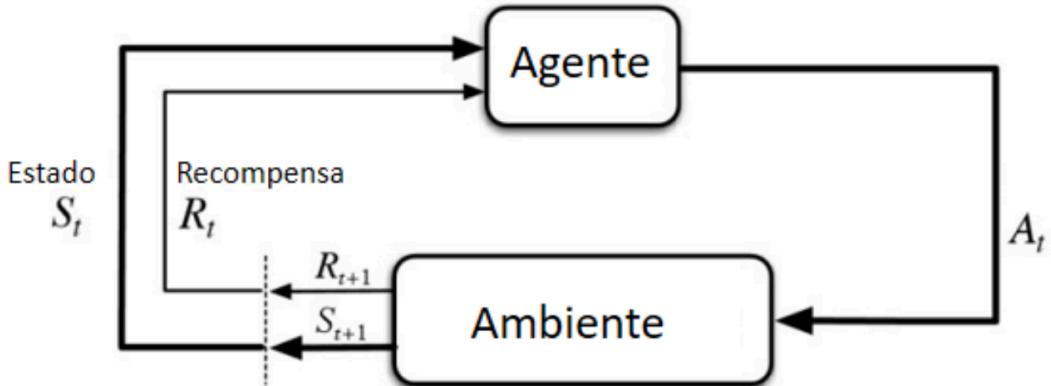
2.6 Aprendizagem por reforço

Segundo Géron (2021), no RL um agente de software faz observações e realiza ações dentro de um ambiente, e em troca recebe recompensas. Seu intuito é aprender a agir de forma a maximizar as recompensas esperadas ao longo do tempo. Pode-se considerar recompensas positivas como satisfação e recompensas negativas como dor. Resumindo, o agente atua no ambiente e aprende por tentativa e erro a maximizar sua satisfação e minimizar a sua dor. É uma definição bem geral e que pode ser usada em um leque amplo de tarefas, tais como: robótica, jogo *Ms. Pac-Man*, jogo *Go*, termostato (controle de temperatura), *trader* automático (mercado financeiro), etc. Pode não haver recompensa positiva; por exemplo, o agente pode circular em um labirinto, recebendo recompensa negativa a cada intervalo de tempo, logo, é melhor encontrar a saída o mais rápido possível.

Krul (2021) expõe que neste aprendizado existe um agente que opera em um ambiente com regras físicas próprias, e sua interação com o ambiente representa suas ações A_t como indica a Figura 12. 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. 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.

Figura 12 - Representação do problema de aprendizagem por reforço.



Fonte: Krul (2021).

Existe o conceito de retorno (Equação 13) que é definido como o acúmulo ponderado de todas as recompensas no tempo. Nessa equação γ é um parâmetro $0 \leq \gamma \leq 1$ 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 (KRUL, 2021).

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \quad (13)$$

Por fim, Krul (2021) expõe que um desafio encontrado no aprendizado por reforço é o de *exploration* (explorar) e *exploitation* (usufruir). 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 contornar esse problema de explorar e usufruir é necessário usar uma taxa de exploração que decai com o tempo fazendo com que o algoritmo comece a tomar melhores decisões (política) ao invés de manter uma alta exploração com ações aleatórias (GÉRON, 2021). Para isso se usa a política ε -greedy em que a cada intervalo o agente age aleatoriamente com probabilidade ε ou gananciosamente com probabilidade $1 - \varepsilon$ (usufruir de uma opção conhecida ao invés de explorar novos caminhos) (GÉRON, 2021).

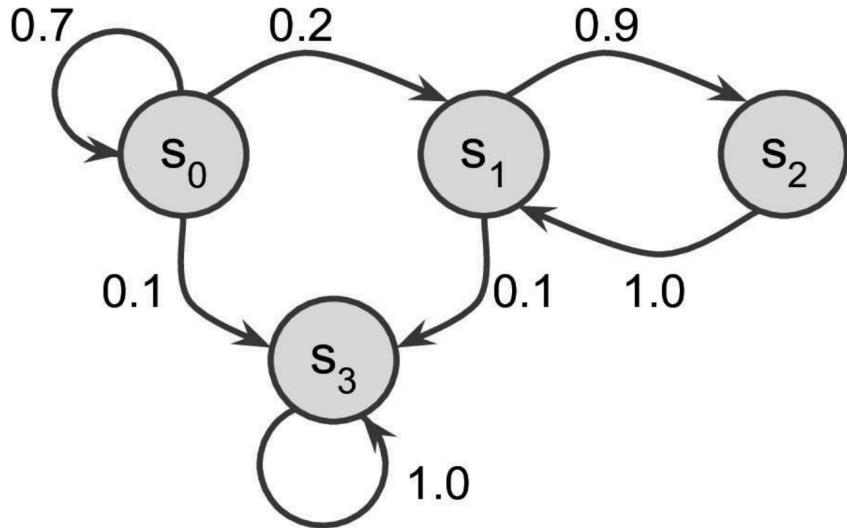
O estudo de RL envolve determinar funções de valor $V(s)$, que são representações matemáticas das expectativas de ganhos futuros de recompensa. Essas funções podem ser baseadas em estado ou estado-ação, e se relacionam ao potencial de ganhos de recompensa para um dado estado ou ação do agente. Como as recompensas futuras são influenciadas pelas ações futuras do agente, é necessário levar em conta a política atual do agente, que descreve a probabilidade de ações futuras em estados futuros. A resolução de um problema de RL se trata de encontrar uma política que maximize os ganhos de recompensa a longo prazo, considerando uma política ótima (COLOMBINI, 2019).

2.6.1 Processos de Decisão de Markov

Géron (2021) comenta que no século XX foram iniciados estudos sobre processos estocásticos sem memória, chamados de cadeias de Markov. Esses processos têm um número fixo de dados e evoluem aleatoriamente de um estado para outro a cada intervalo. A probabilidade de evoluir de um estado s para um

estado s' é fixa e depende apenas do par (s, s') , não de estados anteriores (por isso o sistema não tem memória). A Figura 13 mostra uma cadeia de Markov com quatro estados onde as arestas são as probabilidades de migrar de um estado para o outro tomando alguma ação.

Figura 13 - Exemplo de uma cadeia de Markov.



Fonte: Géron (2019).

Em RL, além de cada aresta (ação) possuir uma probabilidade de levar a um estado, ela possui recompensas que são acumuladas pelo agente.

Krul (2021) comenta que um problema de RL que satisfaz a propriedade de Markov é chamado de Processo de Decisão de Markov, ou simplesmente PDM (MDP - *Markov decision process*). Nesse processo, dado um estado s e ação a , a probabilidade de se alcançar cada novo estado s' e recompensa r é dada pela Equação 14 enquanto a recompensa obtida nessa transição é a Equação 15.

$$p(s', r | s, a) = \Pr \{R_{t+1} = r, S_{t+1} = s' | S_t = s, A_t = a\} \quad (14)$$

$$r(s, a) = E[R_{t+1} | S_t = s, A_t = a] = \sum_{r \in R} r \sum_{s' \in \mathbb{S}} p(s', r | s, a) \quad (15)$$

Segundo Géron (2021), Richard Bellman (1957) identificou uma forma de estimar o valor do estado ideal de qualquer estado s , notado por $V^*(s)$, que é a soma

de todas as recompensas futuras com desconto que o agente pode esperar em média (Equação 16), após alcançar um estado s , presumindo que ele se comporte de maneira ideal.

$$V^*(s) = \max_a \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma \cdot V^*(s')], \text{ para todo } s \quad (16)$$

Na equação 16, $T(s, a, s')$ é a probabilidade de transição do estado s para o estado s' dado que o agente escolheu a ação a . $R(s, a, s')$ é a recompensa que o agente obtém quando passa do estado s para o estado s' , dado que tenha escolhido a ação a . γ é a taxa de desconto. Em outras palavras, $V(s)$ pode ser chamado de valor de reforço imediato pois ele mede a qualidade do estado atual (COLOMBINI, 2019).

Géron (2021) comenta que saber os valores de estados ideais pode ajudar muito, especialmente para avaliar uma política, mas não nos fornece a política ideal para o agente. Felizmente, Bellman (1957) encontrou um algoritmo para estimar os valores de ação de estado ideias, geralmente chamados de Q-valores (valores de qualidade) que medem a qualidade do estado dada uma ação (números que indicam a melhor ação em determinado estado). O Q-valor ideal do par estado-ação (s, a) , notado por $Q^*(s, a)$, é a soma das recompensas futuras com desconto que o agente pode esperar, em média, após alcançar o estado s e escolher a ação a , mas antes de ver o resultado dessa ação, supondo que atue de forma otimizada após ela. Definir a política ótima notada como $\pi^*(s)$, é trivial depois de ter os Q-valores ideais: quando o agente está no estado s , ele deve escolher a ação com o Q-valor mais alto para esse estado (Equação 17).

$$\pi^*(s) = \arg \max_a Q^*(s, a) \quad (17)$$

Treinar um modelo de RL é encontrar os valores $V(s)$ e $Q(s, a)$ computando o reforço acumulado (Equação 18) (COLOMBINI, 2019).

$$Q_{k+1}(s, a) \leftarrow \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma \cdot \max_{a'} Q_k(s', a')] \text{ para todo } (s, a) \quad (18)$$

2.6.2 Algoritmo Deep Q-Network

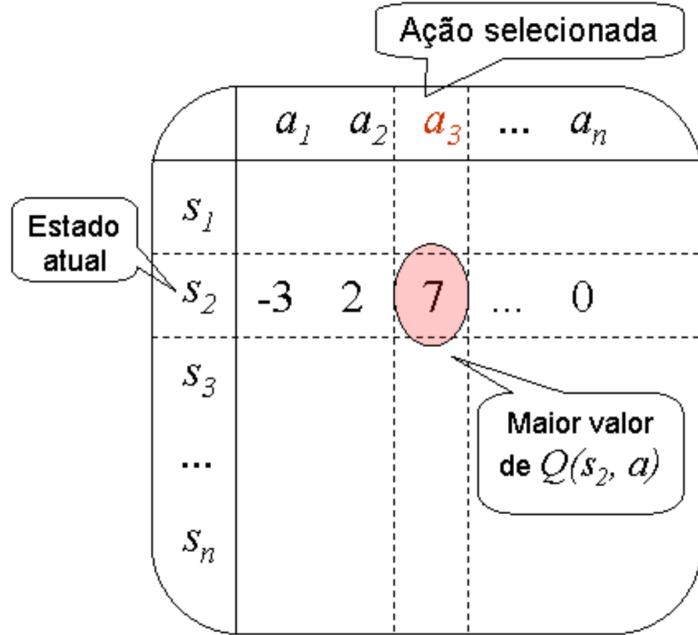
O algoritmo *deep Q-Network* é uma mistura de aprendizado por reforço (*Q-learning*) com rede neural profunda em que essa é usada para estimar os Q-valores utilizados para gerar a política ideal (GÉRON, 2021). Do ponto de vista de controle de sistemas eletromecânicos essa combinação é bastante útil pois garante robustez a imprevistos no sistema além do algoritmo conseguir aprender uma dinâmica complicada (COLOMBINI, 2019).

Segundo Géron (2021), o algoritmo *Q-learning* é uma adaptação do algoritmo de iteração Q-valor à situação em que as probabilidades de transição e as recompensas são inicialmente desconhecidas. O *Q-learning* funciona observando um agente jogar (por exemplo, aleatoriamente) e melhorando aos poucos suas estimativas dos Q-valores. Uma vez que ele tenha estimativas precisas dos Q-valores (ou próximo o bastante), a política ideal é escolher a ação que tem o Q-valor mais alto (ou seja, a política gananciosa) como mostra a Equação 19. Nela α corresponde a taxa de aprendizado que tende a zero no fim do treinamento para convergir para a política ideal (COLOMBINI, 2019).

$$Q_{k+1}(s, a) \leftarrow (1 - \alpha)Q_k(s, a) + \alpha(r + \gamma \cdot \max_{a'} Q_k(s', a')) \quad (19)$$

Q-learning aprende iterativamente testando o ambiente e atualizando os pares estado-ação em uma matriz (tabela que mapeia os estados-ações tendo os Q-valores) e, por isso, ele é considerado um método tabular como mostra a Figura 14 (COLOMBINI, 2019).

Figura 14 - Exemplo de uma matriz de Q-valores onde o algoritmo *Q-learning* mapeia os pares estado-ação.



Fonte: Colombini, 2019.

Esse algoritmo tenta explorar o ambiente o máximo possível e por isso leva conta a política ϵ -greedy que faz um balanceamento entre ganância por altas recompensas e espírito aventureiro (alta exploração independente da recompensa) (COLOMBINI, 2019).

Géron (2021) indica que o Q-learning é chamado de *off-policy*, já que a política que está sendo treinada não é necessariamente aquela que está sendo executada, ou seja, ele não se preocupa com qual política de decisão seguir. Por outro lado, o algoritmo gradiente de política é um algoritmo *on-policy*: explora o mundo usando a política de treinamento, ou seja, se preocupa com a política escolhida.

Géron (2021) também comenta que o principal problema do Q-learning é que ele não se adapta bem a grandes (ou mesmo médios) PDMs com muitos estados e ações, em alguns casos é absolutamente impossível monitorar uma estimativa para cada Q-valor. A solução é encontrar uma função $Q_\Theta(s)$, uma a que se aproxima do Q-valor de qualquer par estado-ação (s, a) usando um número gerenciável de parâmetros (dados pelo vetor de parâmetros). Isso se chama Q-learning aproximado. Em 2013 a DeepMind demonstrou que o uso de redes neurais profundas pode funcionar melhor para estimar os Q-valores, sobretudo para problemas complexos, não exigindo *feature engineering*. Uma RNA usada para

estimar os Q-valores se chama *deep Q-Network* (DQN), e usar uma DQN para o *Q-learning* Q aproximado se chama *Deep Q-learning* (*Q-learning* profundo).

Géron (2021) fecha esse tema explicando como treinar uma DQN. Basta considerar o Q-valor aproximado calculado pela DQN para um determinado par estado-ação (s, a) . Graças a Bellman (1957), sabemos que queremos que este Q-valor aproximado seja o mais próximo possível da recompensa r que, de fato, observamos depois de jogar a ação a no estado s , mais o valor descontado de jogar de forma otimizada desde então. Para estimar esta soma de futuras recompensas com desconto, podemos simplesmente rodar a DQN no próximo estado s' e em todas as ações possíveis a' . Obtemos um Q-valor futuro aproximado para cada ação possível. Depois escolhemos o mais alto (já que presumimos que estaremos jogando de maneira ideal) e o descontamos, e isso nos fornece uma estimativa da soma das recompensas descontadas futuras. Ao somar a recompensa r e a estimativa do valor futuro com desconto, obtemos um Q-valor alvo $y(s, a)$ para o par estado ação (s, a) , conforme mostrado na equação 20.

$$y(s, a) = r + \gamma \cdot \max_{a'} Q_\theta(s', a') \quad (20)$$

Por fim, Géron (2021) conclui que com este Q-valor alvo, podemos executar uma etapa de treinamento usando qualquer algoritmo de gradiente descendente. Em termos específicos, geralmente tentamos minimizar o erro quadrático entre o Q-valor estimado $Q(s, a)$ e o Q-valor alvo (ou a perda de Huber para reduzir a sensibilidade do algoritmo a grandes erros). Assim funciona o algoritmo básico de *deep Q-learning*.

Capítulo 3

Desenvolvimento dos controladores

Será executado os seguintes passos para desenvolvimento dos controladores:

1. Avaliar o problema do pêndulo invertido em um grau de liberdade;
2. Construir um controlador clássico PID (alocação de polos);
3. Implementar computacionalmente o modelo com *python* (*Gym CartPole-v1*);
4. Construir um controlador usando um algoritmo de RL;
5. Comparar os resultados e desempenhos obtidos pelos dois métodos utilizados;
6. Concluir sobre o resultado de cada controlador.

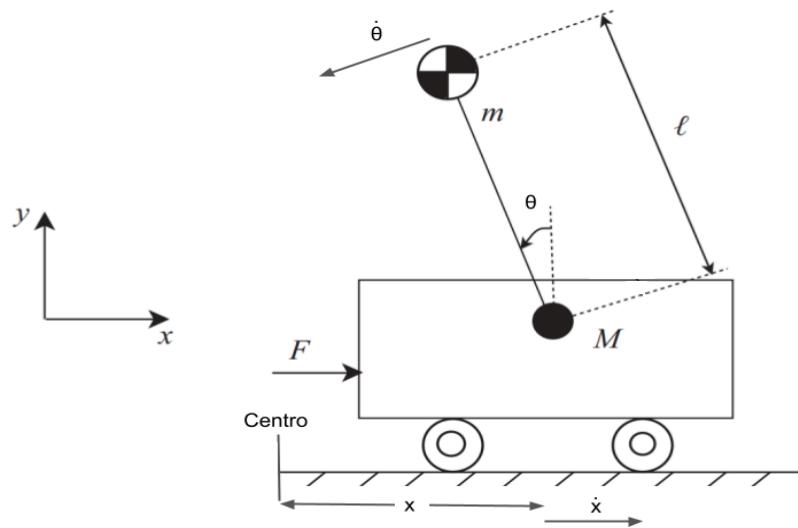
3.1 Controlador PID

O controlador PID é considerado o controlador clássico da literatura. Hoje ele é usado na grande maioria das aplicações reais.

Uma técnica bastante comum para desenvolvê-lo é a modelagem por espaço de estados que será implementada neste capítulo.

O modelo mecânico que se deseja controlar é o pêndulo invertido acoplado ao carro que é mostrado na Figura 15.

Figura 15 - Pêndulo invertido acoplado ao carro.



Fonte: Boubaker, Iriarte (2017) com adaptações feitas pelo autor.

Na Tabela 3 é possível encontrar os valores das variáveis do sistema utilizados na simulação.

Tabela 3 - Valores das variáveis físicas do sistema mecânico.

Constante	Descrição	Valor	Unidade
g	aceleração da gravidade	9.8	m/s ²
M	massa do carro	1.0	kg
m	massa concentrada da haste	0.1	kg
l	comprimento da haste	0.5	m

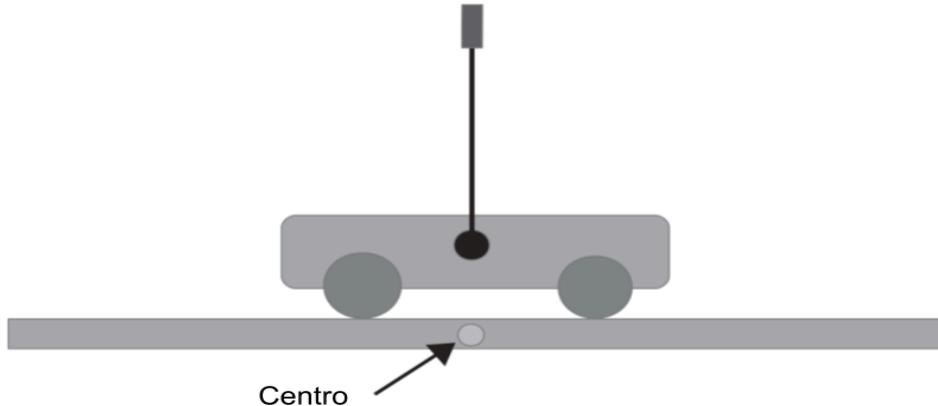
Fonte: Autor.

Na Figura 15 são apresentadas as variáveis que modelam o sistema. As equações ordinárias diferenciais de segunda ordem que definem a aceleração linear (Equação 3) e aceleração angular (Equação 4) são suficientes para caracterizar o sistema.

É desejado linearizar o sistema em um ponto de equilíbrio que se deseja manter o pêndulo equilibrado e o carro parado (Figura 16), sendo ele caracterizado pela Equação 21 com todos elementos iguais a zero. Esse ponto é instável e por isso é necessário fazer o controlador atuando para mantê-lo assim.

$$x = [x, x', \theta, \theta'] = [0, 0, 0, 0] \quad (21)$$

Figura 16 - Posição de equilíbrio do pêndulo.



Fonte: Boubaker, Iriarte (2017) com adaptações feitas pelo autor.

A linearização considera que o controle será feito partindo de ângulos iniciais pequenos em torno do ponto de equilíbrio. Para ângulos pequenos é possível fazer aproximações (Equações 22) que permitem simplificar as equações das acelerações angular (Equação 23) e linear (Equação 24).

$$\sin \theta = \theta \quad (22.1)$$

$$\cos \theta = 1 \quad (22.2)$$

$$\dot{\theta}^2 = 0 \quad (22.3)$$

$$\ddot{\theta}^2 = 0 \quad (22.4)$$

$$\ddot{\theta} = \frac{3(M+m)g\theta - 3F}{(4M+m)l} \quad (23)$$

$$\ddot{x} = \frac{-3mg\theta + 4F}{4M+m} \quad (24)$$

Dadas as equações simplificadas, a partir das equações 5.1, 5.2, 5.3 e 5.4 é possível obter os valores das matrizes A (Equação 25), B (Equação 26), C (Equação 27) e D (Equação 28) substituindo os valores da Tabela 3. A entrada do sistema é definida pelo escalar $u = F$ (força aplicada no carro).

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{-3mgl}{4M+m} & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 \end{bmatrix} \quad (25)$$

$$B = \begin{bmatrix} 0 \\ \frac{4}{4M+m} & 0 & \frac{-3}{(4M+m)l} \end{bmatrix}. \quad (26)$$

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad (27)$$

$$D = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad (28)$$

É necessário julgar a instabilidade do sistema. Então é calculado o determinante utilizando a matriz A e a variável s que representa x no domínio de laplace que pode ser vista na Equação 29.

$$\det(sI - A) = 0 \quad (29)$$

O sistema possui quatro polos, dois iguais a zero e dois com a parte imaginária nula como pode ser visto na Tabela 4. O fato de existir um zero no semi-plano direito (polo $3.97185 + 0j$) torna o sistema instável. (OGATA; YANG, 2002).

Tabela 4 - Valores dos polos do sistema.

Polos

0 + 0j
0 + 0j
3.97185 + 0j
- 3.97185 + 0j

Fonte: Autor.

Para seguir em diante se faz necessário saber se o sistema é controlável através da equação 6. Calculando o posto da matriz T temos um valor igual a 4, exatamente o valor da dimensão do sistema em questão, logo ele é linear e controlável (OGATA; YANG, 2002). Na equação 30 é possível visualizar a matriz T .

$$T = \begin{bmatrix} 0 & 0.97560 & 0 & 1.04937 \\ 0.97560 & 0 & 1.04937 & 0 \\ 0 - 1.4634 & - 1.4634 & 0 - 23.08625 & - 23.086250 \\ 0 \end{bmatrix} \quad (30)$$

Dado que é possível realizar o controle do sistema, ele será controlado através da técnica de alocação de polos em que é criada uma matriz K que multiplica a entrada do sistema (Equação 31.1) e faz com que não exista polos no semiplano direito (parte real maior que zero). É necessário colocar uma limitação no atuador fazendo com que a força atuante se limite nos valores F e $-F$ com intuito de comparar os controladores já que o algoritmo *Deep Q-Network* não irá gerar uma força de atuação contínua, mas sim binária devido aspectos de implementação. No caso, a força será limitada em 10 newtons (equações 31.2 e 31.3).

$$u = -Kx \quad (31.1)$$

$$\text{se } u < 0 \text{ então } u = -F = -10N \quad (31.2)$$

$$\text{se } u \geq 0 \text{ então } u = F = 10N \quad (31.3)$$

Além da limitação na atuação do controlador, será necessário atingir dois requisitos de projeto que são obter a menor sobre-elevação possível MS e o menor tempo de estabilização T_s . A sobre-elevação corresponde ao primeiro pico que ultrapassa o valor final que se deseja atingir (valor $x = 0$ e $\theta = 0$) e tempo de estabilização é quanto que leva para atingir esse valor final.

O cálculo dos polos dominantes (Equação 32.1) é feito a partir do amortecimento (ξ), frequência natural do sistema (ω_n) e a frequência natural amortecida (ω_d Equação 32.4) que são funções da sobre-elevação (Equação 32.2) e do tempo de acomodação (equação 32.3).

$$Polos \Rightarrow s = -\xi \omega_n \pm \omega_n \sqrt{\xi^2 - 1} \quad (32.1)$$

$$MS = e^{\frac{-\xi\pi}{\sqrt{1-\xi^2}}} \quad (32.2)$$

$$T_s = \frac{4}{\xi\omega_n} \quad (32.3)$$

$$\omega_d = \omega_n \sqrt{1 - \xi^2} \quad (32.4)$$

Calculado os polos dominantes, os outros dois polos a serem adicionados precisam estar bem distantes à esquerda dos polos dominantes (OGATA; YANG, 2002). Será considerada uma distância de 8 vezes o valor de cada polo chegando nos seguintes resultados.

Tabela 5 - Polos e seus respectivos requisitos.

	MS	T_s	ξ	ω_n	ω_d	$p1$	$p2$	$p3$	$p4$
0	0.05	1.0	0.6901	5.7962	4.1948	-4.000000+4.194758j	-4.000000-4.194758j	-32.0000	-33.5581
1	0.08	1.5	0.6266	4.2559	3.3169	-2.666667+3.316896j	-2.666667-3.316896j	-21.3333	-26.5352
2	0.09	1.5	0.6083	4.3836	3.4791	-2.666667+3.479140j	-2.666667-3.479140j	-21.3333	-27.8331
3	0.10	2.0	0.5912	3.3832	2.7288	-2.000000+2.728753j	-2.000000-2.728753j	-16.0000	-21.8300
4	0.10	2.2	0.5912	3.0756	2.4807	-1.818182+2.480684j	-1.818182-2.480684j	-14.5455	-19.8455
5	0.10	2.5	0.5912	2.7066	2.1830	-1.600000+2.183002j	-1.600000-2.183002j	-12.8000	-17.4640
6	0.12	2.5	0.5594	2.8601	2.3707	-1.600000+2.370719j	-1.600000-2.370719j	-12.8000	-18.9657

Fonte: Autor.

A Tabela 5 mostra os valores calculados. Nela p_1 e p_2 são os polos dominantes, p_3 e p_4 são os polos adicionados manualmente a uma distância de 9 vezes à esquerda de p_1 e p_2 .

Após definir a disposição dos polos, a matriz de ganho K é identificada. Usando as matrizes de ganho listadas na tabela, o sistema é testado de forma semelhante ao processo de aprendizado por reforço. Para observar a resposta do sistema, realizamos uma simulação do modelo com discretização do tempo a cada 0,02 segundos e 200 instantes de tempo, semelhante à simulação dos modelos de aprendizado por reforço.

As condições iniciais para simulação estão na equação 33.

$$x = [0 \ 0 \ -0.021 \ 0] \quad (33)$$

3.2 Controlador *deep Q-Network*

Para treinar um algoritmo de aprendizado por reforço é necessário um ambiente (real ou simulado) para que ele treine e aprenda interagindo com ele dentro do sistema de recompensas. Esse ambiente pode ser real ou simulado (COLOMBINI, 2019).

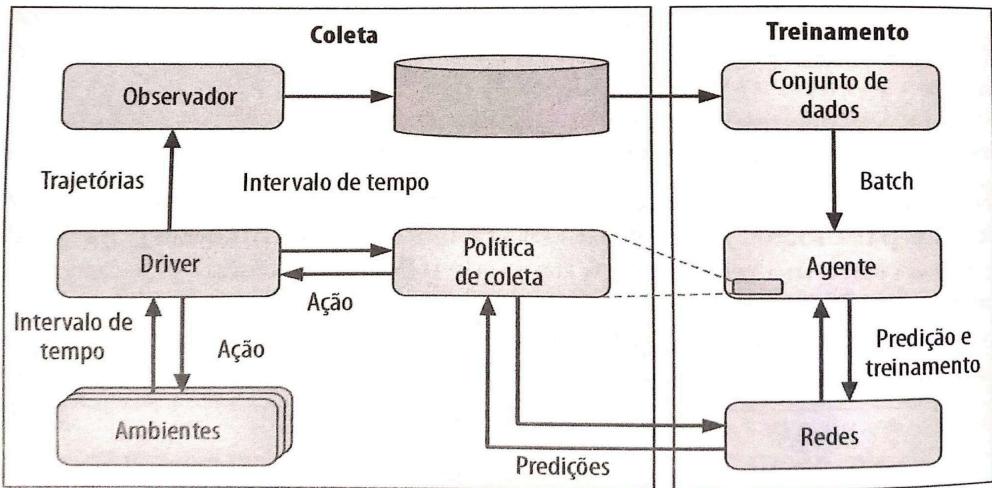
Ambientes reais são difíceis de serem utilizados pois no caso de um sistema mecânico os componentes se deterioram devido ao desgaste do elemento, podendo levar a quebra de algum material.

Por isso foi criado um ambiente simulado pela Google em 2018 chamado *TF-Agents*. Trata-se de uma biblioteca de RL *open-source* utilizada para treinar um agente que atua em algum ambiente.

O funcionamento de um programa de treinamento *TF-Agents* é dividido em duas etapas que ocorrem simultaneamente. Na primeira etapa chamada de coleta (parte esquerda da Figura 17), um "driver" explora o ambiente usando uma política de coleta para selecionar ações, gerando séries de transições chamadas "trajetórias" e as envia para um "observador", que as armazena em um "replay buffer". Em seguida, na fase de treinamento (parte direita da Figura 17), o "agente" acessa

essas trajetórias armazenadas no *buffer* e utiliza-as para treinar as redes neurais que a política de coleta utiliza.

Figura 17 - Arquitetura de treinamento.



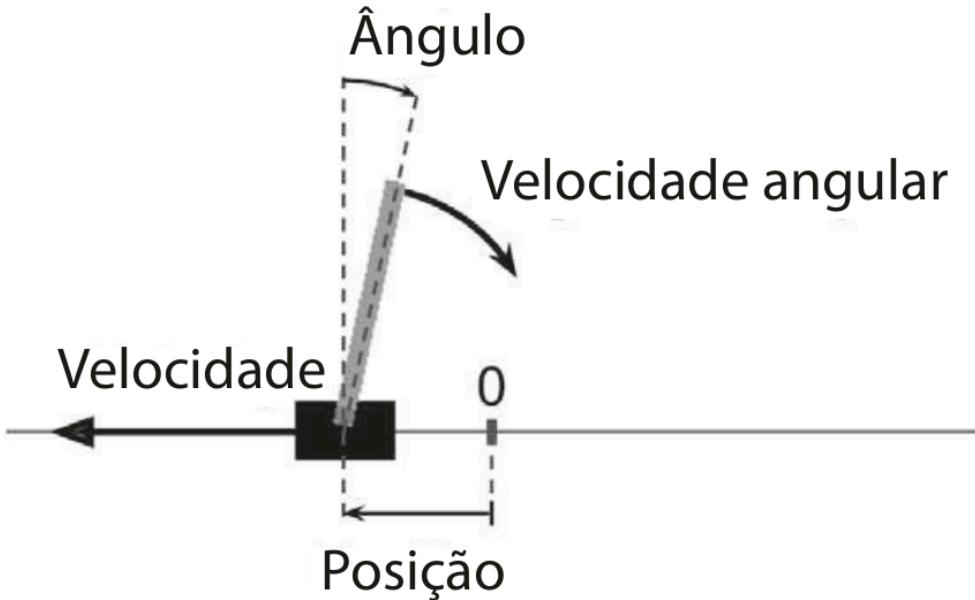
Fonte: *TF-Agents*.

É possível observar na Figura 17 que há vários ambientes simultaneamente, pois, em vez de explorar apenas um único ambiente, é comum que o driver explore múltiplas cópias do ambiente em paralelo, o que aproveita a capacidade de processamento de vários núcleos de CPU, mantém as GPUs ocupadas e fornece trajetórias com menor grau de correlação para o algoritmo de treinamento.

Cabe destacar que uma trajetória é uma representação compacta de uma transição de um determinado período para o seguinte, ou seja, em uma sequência de transições consecutivas de um dado período para o próximo, as trajetórias coletadas pelo driver são transmitidas para o observador que as salva no replay buffer e depois são amostradas pelo agente e utilizadas para treinamento.

O *TF-Agents* será utilizado para treinar o agente enquanto o ambiente de simulação utilizado é o ambiente computacional *Gym Cartpole-v1* do pacote *Open AI Gym* da linguagem *python*. Ele simula o modelo físico não-linear do pêndulo invertido e possui diversos outros ambientes que são utilizados pela comunidade para implementar e testar algoritmos de RL.

Figura 18 - Ambiente *CartPole*.



Fonte: Géron (2019).

Como representado na Figura 18, nesse ambiente aceleramos o carrinho para a esquerda ou para a direita a fim de equilibrar uma haste em cima. O sentido da velocidade é positivo para a direita, o ângulo cresce positivamente no sentido horário assim como a velocidade angular também.

Todos os parâmetros do sistema (massas, comprimentos, força, etc) são os mesmos utilizados no modelo do controlador PID (Tabela 3).

Definido o ambiente de simulação e as ferramentas, é necessário formular o problema de RL que será desenvolvido.

De forma genérica, queremos encontrar uma política ótima, que é uma função mapeando estados para ações, de forma a maximizar a esperança de recompensa futura. A função de valor, $V(s)$, é a esperança de recompensa futura a partir de um estado s , enquanto a função de valor de ação, $Q(s, a)$, é a esperança de recompensa futura a partir de um estado-ação (s, a) como mostra na Equação 34 onde E é a esperança matemática, γ é um coeficiente de aprendizado, t é o instante de tempo atual e T é o número máximo de instantes de tempo.

Para o problema específico do pêndulo invertido, o estado do agente s poderia ser representado como $S = [x, x', \theta, \theta']$ e a ação do agente como $a = \{-F, F\}$. A recompensa $R(s, a)$ poderia ser definida como +1 se estiver dentro

do limite e 0 fora dele. O objetivo seria maximizar a função de valor de ação $Q(s, a)$ ou a função de valor $V(s)$.

$$Q(s, a) = E \left[\gamma \sum_t^T R(s_t, a_t) \mid s_t = s, a_t = a \right] \quad (34)$$

Esse problema pode ser expressado como um processo de decisão de Markov pois nesse meio determinístico existe um conjunto finito de variáveis representando o estado do agente S e uma matriz de transição de estados P que leva uma ação a tomada em um estado s para um novo estado s' .

Será recebida uma recompensa +1 caso a haste esteja entre $+12^\circ$ e o carrinho esteja entre $+2,4$ metros. Para quaisquer outros valores a recompensa é zero como mostra a equação 35.

$$R(t) = \begin{cases} +1 & \text{se } -12^\circ \leq \theta \leq 12^\circ \text{ e } -2,4m \leq x \leq 2,4m, \\ 0 & \text{se qualquer outro caso.} \end{cases} \quad (35)$$

Existe a limitação de que só pode ser exercida sobre o carro uma força de $+10$ newtons (mover para esquerda ou para a direita) e o treino durará 200 intervalos ($T = 200$) de tempo podendo ser tomada no máximo 200 ações.

Definido isso se quer obter o maior ganho possível definido pela equação 36.

$$G_T = \sum_{t=1}^T R(t) \rightarrow \max \quad (36)$$

O agente será treinado usando valores aleatórios para as quatro variáveis de estado como condições iniciais, para evitar possíveis vieses na solução. O treinamento será encerrado quando o valor médio total de recompensa nos últimos 10 episódios é maior que 200, ou quando o número máximo de épocas é alcançado. Depois disso, é realizada uma avaliação do desempenho comparando o valor total das recompensas obtidas em cada episódio. A condição inicial θ é 0.021 radianos e outras variáveis de estado são inicializadas em zero.

Durante o treinamento os pesos (w_i) da rede neural serão atualizados minimizando o erro da ação $q(s, a, w)$ através do otimizador adam e um coeficiente de aprendizado de 0,004.

A RNA construída com *API TensorFlow Agents* possui 3 camadas: camada de entrada (4 neurônios), camada escondida (100 neurônios com função de ativação reLU) e camada de saída (2 neurônios com estimativa de $q(s, a, w)$).

Serão executados 10000 episódios. Durante o processo de treinamento, são realizadas várias etapas, na qual em cada uma delas é executado um segmento e é coletada uma trajetória. Essa trajetória inclui o estado anterior, as ações tomadas pelo agente de acordo com a política atual, e o novo estado resultante. Estas trajetórias são armazenadas em um *buffer*, e usadas como dados de entrada para a RNA para estimar o valor de diferentes ações. Enquanto os valores na matriz Q são atualizados, o erro é calculado e ajustado através do método de descida de gradiente estocástico.

Capítulo 4

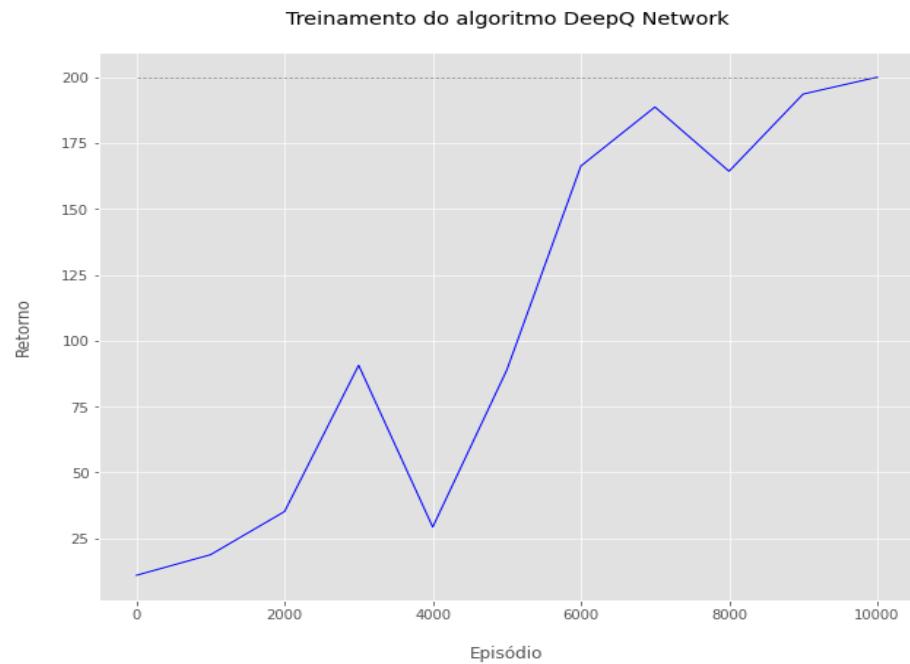
Resultados

Este capítulo apresenta os resultados obtidos ao controlar um pêndulo invertido usando o RL e o método de alocação de polos. Inicialmente, será descrito o processo de treinamento do algoritmo de RL e como o sistema se comporta sob o seu controle. Em seguida, serão apresentados os resultados obtidos com a resolução do problema através do método de alocação de polos. Por fim, haverá uma comparação entre os resultados dos dois métodos propostos. Os algoritmos foram implementados usando *Python 3.8.10* no *Google Colab (Colaboratory)*, um ambiente de *Jupyter Notebook* acessível sem necessidade de configuração prévia.

Para o algoritmo de aprendizado por reforço *deep Q-Network*, foi utilizada uma rede neural que estima a matriz valor-ação com o otimizador Adam durante o treinamento. A taxa de aprendizado α utilizada foi 0,004 e o treinamento realizado em 10000 episódios.

A Figura 19 apresenta os resultados da etapa de treinamento do algoritmo *deep Q-Networks* que utiliza Adam como função de otimização da rede neural.

Figura 19 - Treinamento utilizando o método do Otimizador Adam para taxa de aprendizado de 0,004.



Fonte: Autor.

No gráfico de treinamento (Figura 19) é possível notar que o algoritmo precisou das 10000 etapas para atingir a recompensa máxima de 200 pontos. É um resultado não muito interessante por se tratar de bastante episódios para atingir a recompensa máxima. Géron (2021) em seus experimentos atinge a recompensa máxima com 300 episódios seguindo hiperparâmetros parecidos com os daqui. Apesar disso, um bom sinal é que o algoritmo tende sempre continuar aumentando as recompensas com o decorrer dos episódios, diferentemente de Géron onde apesar de seu modelo atingir o máximo muito cedo ele drasticamente volta para recompensas baixas. No experimento realizado aqui houve apenas uma queda no retorno de recompensas entre os episódios 3000 e 4000.

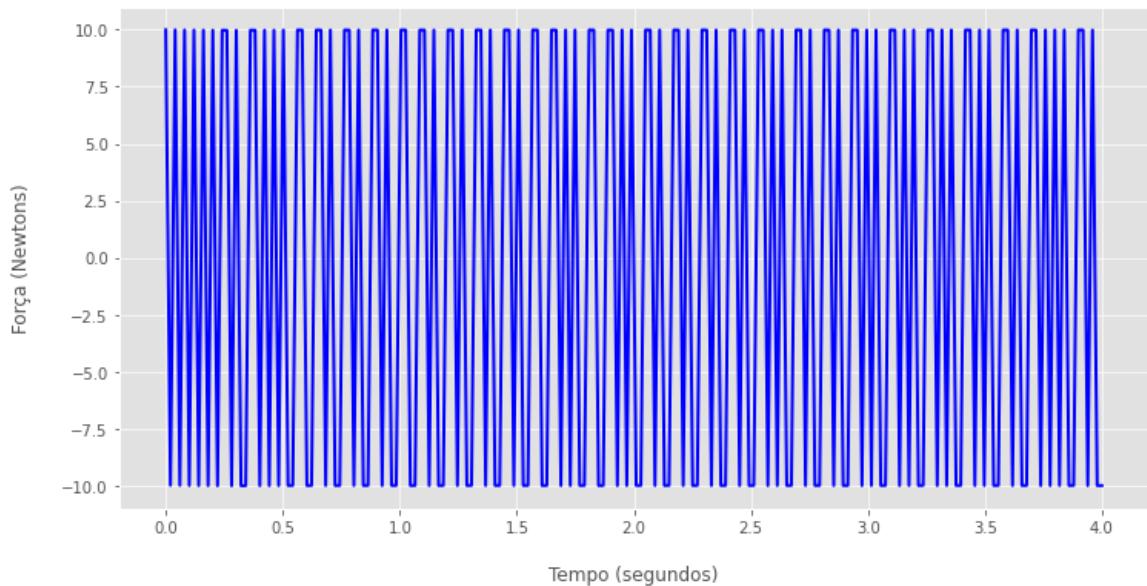
O tempo de treinamento durante todos os episódios levou 3 minutos e 18 segundos.

Com o modelo treinado, ele foi testado em um ambiente com as mesmas condições iniciais do controlador PID, todas variáveis zeradas exceto o ângulo inicial do pêndulo (-0,021 radianos) exatamente como na Equação 33.

Na Figura 20 está a força gerada pelo algoritmo para controlar o carro e manter a haste na vertical.

Figura 20 - Força de controle aplicada no carro pelo algoritmo.

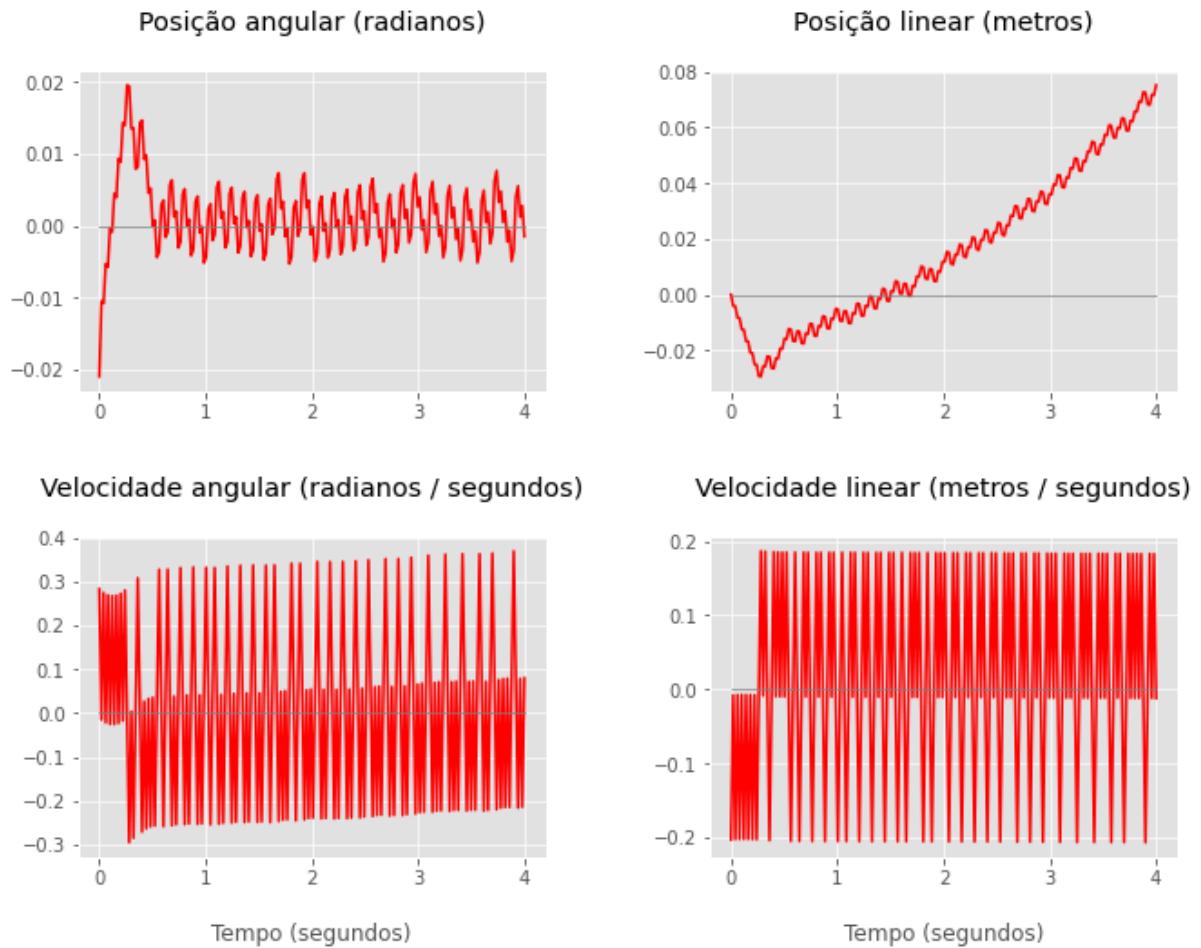
Esforço de controle aplicado pelo algoritmo DeepQ Network



Fonte: Autor.

A Figura 21 mostra as respectivas respostas do sistema com a força aplicada da Figura 20.

Figura 21 - Respostas do sistema após as forças aplicadas pelo controlador.



Fonte: Autor.

Na Figura 21 é possível notar que o algoritmo consegue equilibrar a haste na vertical (posição angular igual a zero). O ângulo máximo que o pêndulo atinge é 0,0195 radianos.

Apesar de manter o pêndulo equilibrado, o modelo não foi capaz de manter o carrinho em torno da condição inicial de 0 metros. Ele chegou a atingir a posição de 0,029 metros a direita e após isso mantém forças constantes que levam o carro para a esquerda (sentido positivo de x do sistema) infinitamente mesmo não sendo recompensado quando o carro está longe do centro.

A variação da velocidade angular do carro ficou entre [-0.294, 0.369] rad/s enquanto sua velocidade linear ficou entre [-0.207, 0.187] m/s.

Agora analisando o resultado do controlador gerado através do método de alocação de polos, na Tabela 6 existem as matrizes de ganho K obtidas de acordo com variações dos parâmetros de projeto sobre-elevação e tempo de acomodação. O intuito é conseguir os menores parâmetros para um esforço de controle que não ultrapasse em módulo 10 newtons e que assim seja justa a comparação com o controlador neural.

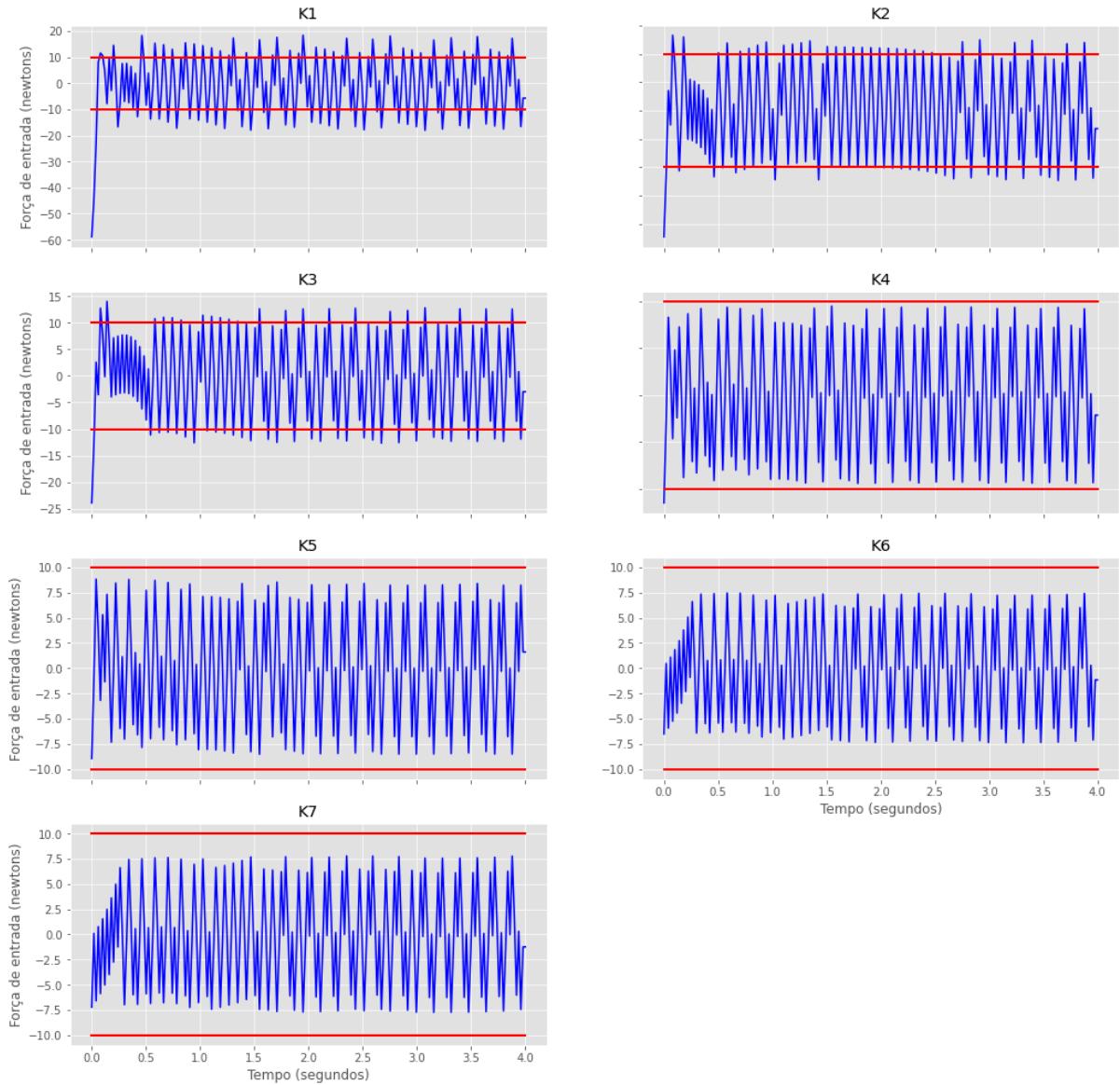
Tabela 6 - Matrizes de ganho com variação dos parâmetros do projeto.

Sobre-elevação (%)	Tempo de acomodação (segundos)	Índice	Matriz de ganho K
0,05	1,00	K1	[-2516 -753 -2803 -552]
0,08	1,50	K2	[-715 -271 -1061 -217]
0,09	1,50	K3	[-796 -287 -1139 -228]
0,10	2,00	K4	[-279 -128 -547 -114]
0,10	2,20	K5	[-190 -96 -427 -90]
0,10	2,50	K6	[-114 -65 -311 -66]
0,12	2,50	K7	[-138 -72 -344 -72]

Fonte: Autor.

Para avaliar qual matriz K não precisa ser saturada sob condições de comparação com parâmetros iniciais, simulações foram realizadas com a entradas observadas $u = -Kx$ para cada matriz K .

Figura 22 - Esforços de controle (newtons) gerado pelo controlador com as respectivas matrizes de ganho K .



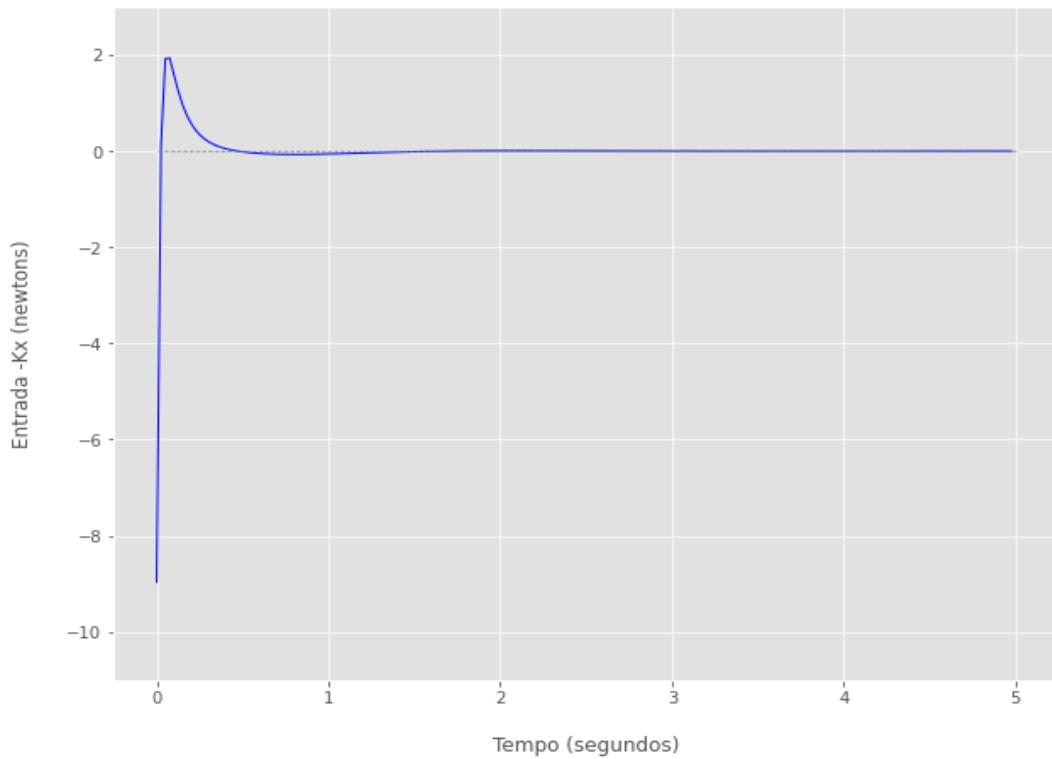
Fonte: Autor.

Na Figura 22 é possível notar que as 4 primeiras matrizes de ganho apresentam um esforço de controle acima ou abaixo de 10 N ou -10 N. K_4 seria a escolhida por possuir menores parâmetros de projeto caso não tivesse ultrapassado -10 N no esforço inicial.

Desta forma a matriz K_5 foi eleita o melhor ganho por não ultrapassar o limite da força de controle permitido. Em seguida foi analisada a resposta do sistema mecânico com controlador considerando a entrada $u = -K_5 x$ com o controlador livre do saturador [10] newtons.

Figura 23 - Esforço de controle com controlador livre do saturador.

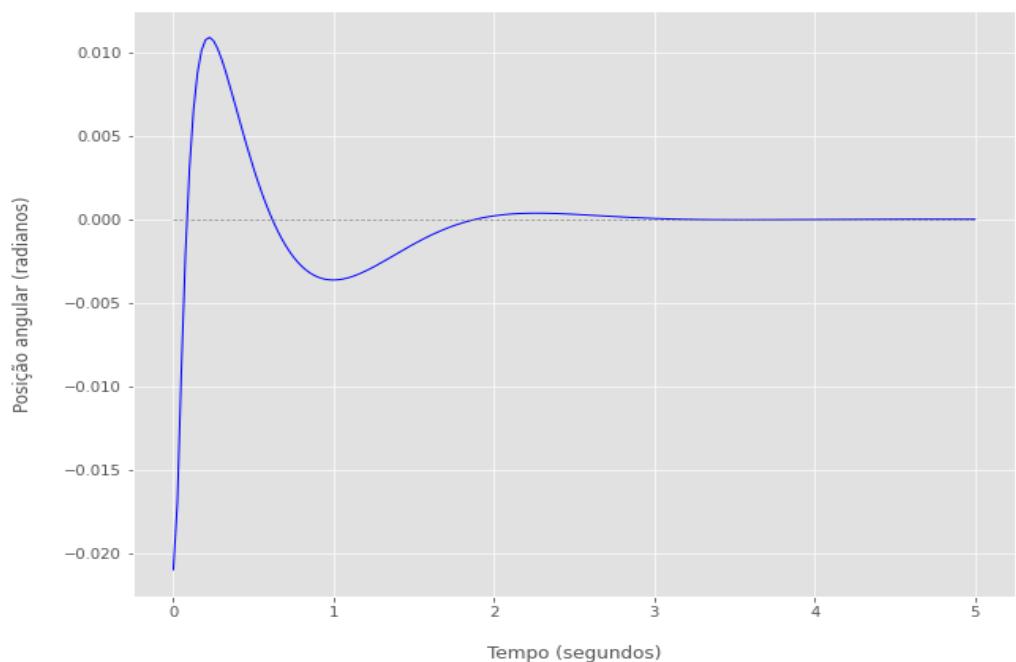
Sistema controlado sem limitação no esforço de controle



Fonte: Autor.

Figura 24 - Resposta da posição angular θ com controlador livre do saturador.

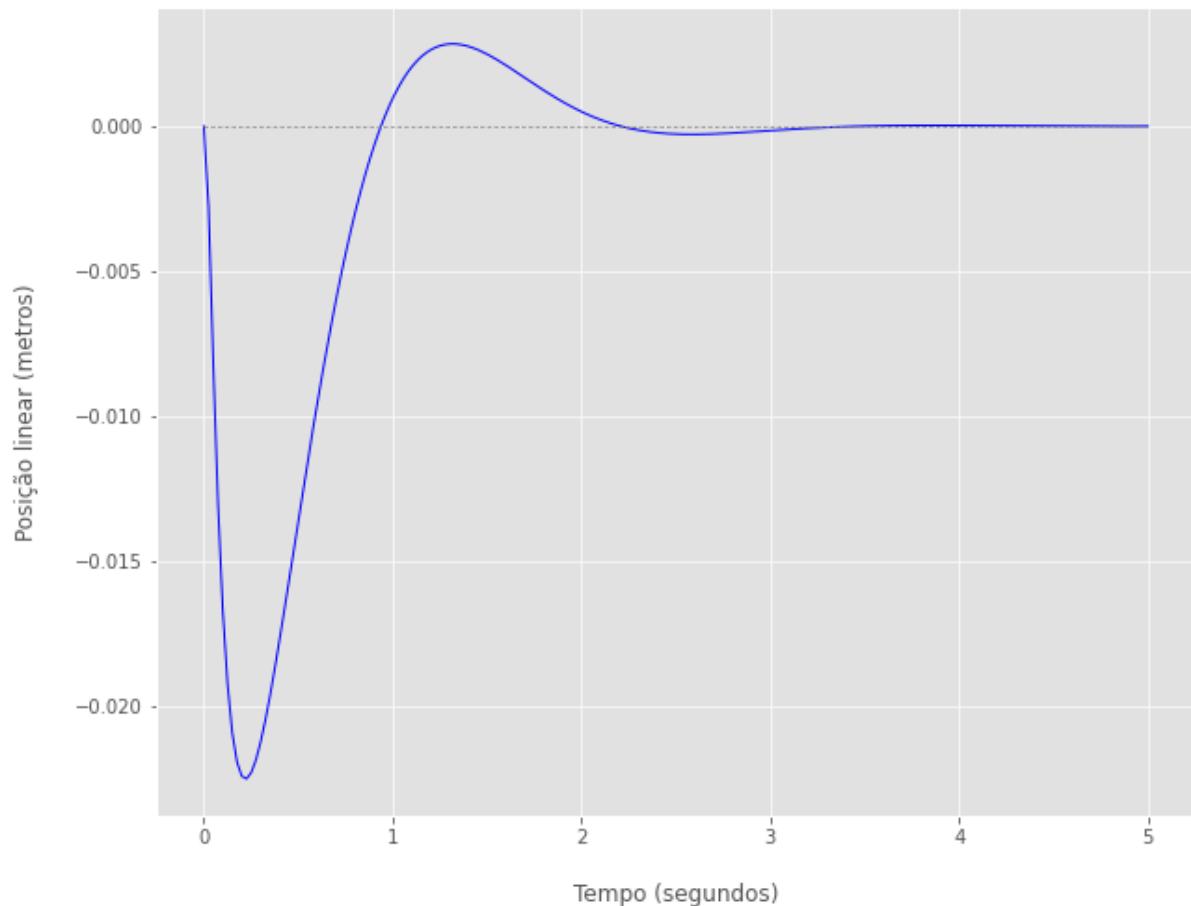
Sistema controlado sem limitação no esforço de controle



Fonte: Autor.

Figura 25 - Resposta da posição linear x com controlador livre do saturador.

Sistema controlado sem limitação no esforço de controle

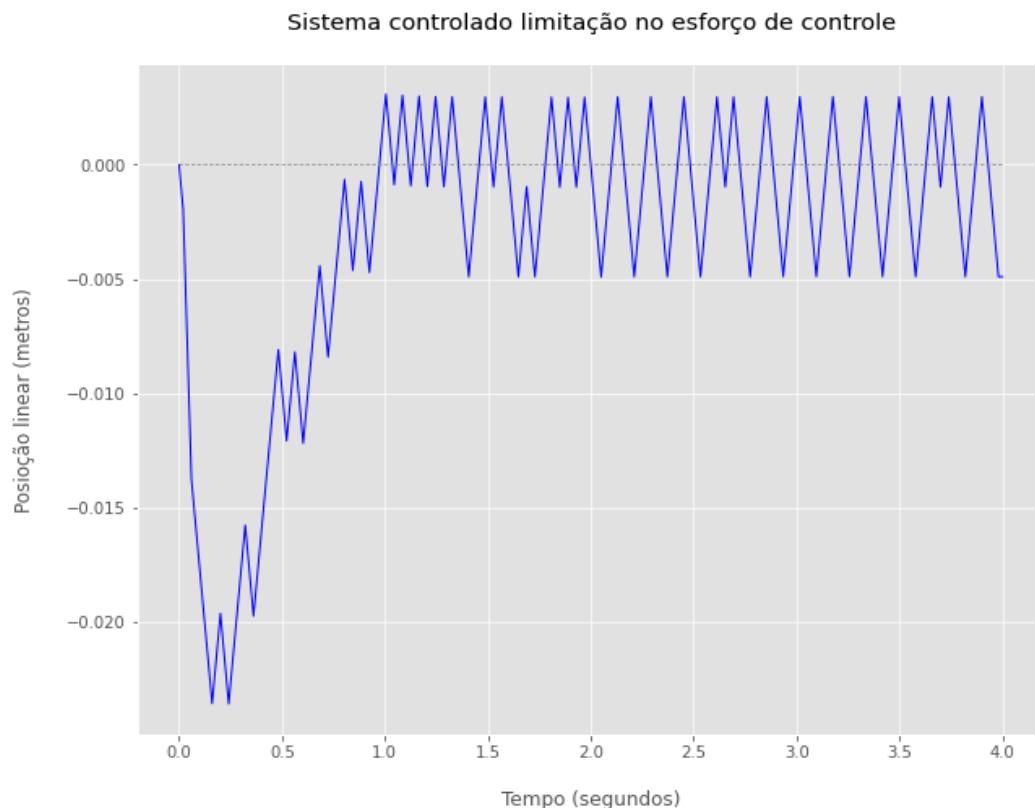


Fonte: Autor.

Na Figura 24 vemos que o controlador consegue manter o equilíbrio da haste na vertical e a Figura 25 mostra que o carrinho fica na posição de origem ($x = 0$). A Figura 23 mostra a força aplicada no carro pelo controlador.

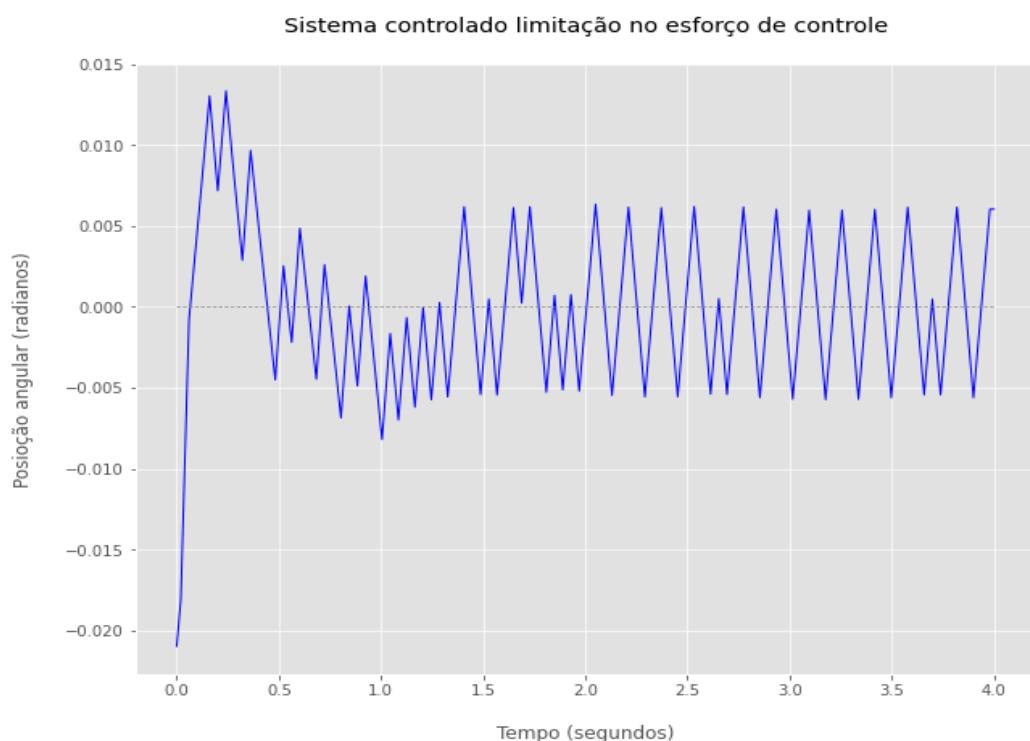
Na sequência foi analisado o comportamento do sistema agora limitando o controlador com esforços de controle de no máximo 10 newtons.

Figura 26 - Resposta da posição linear x com controlador sob efeito do saturador.



Fonte: Autor.

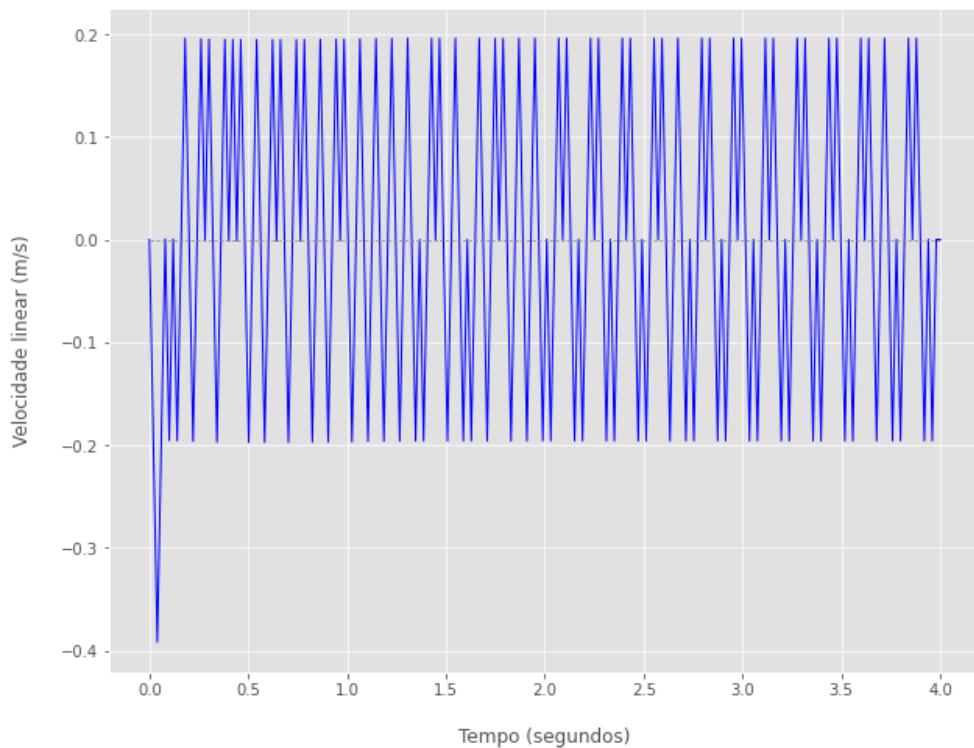
Figura 27 - Resposta da posição angular θ com controlador sob efeito do saturador.



Fonte: Autor.

Figura 28 - Resposta da velocidade linear com controlador sob efeito do saturador.

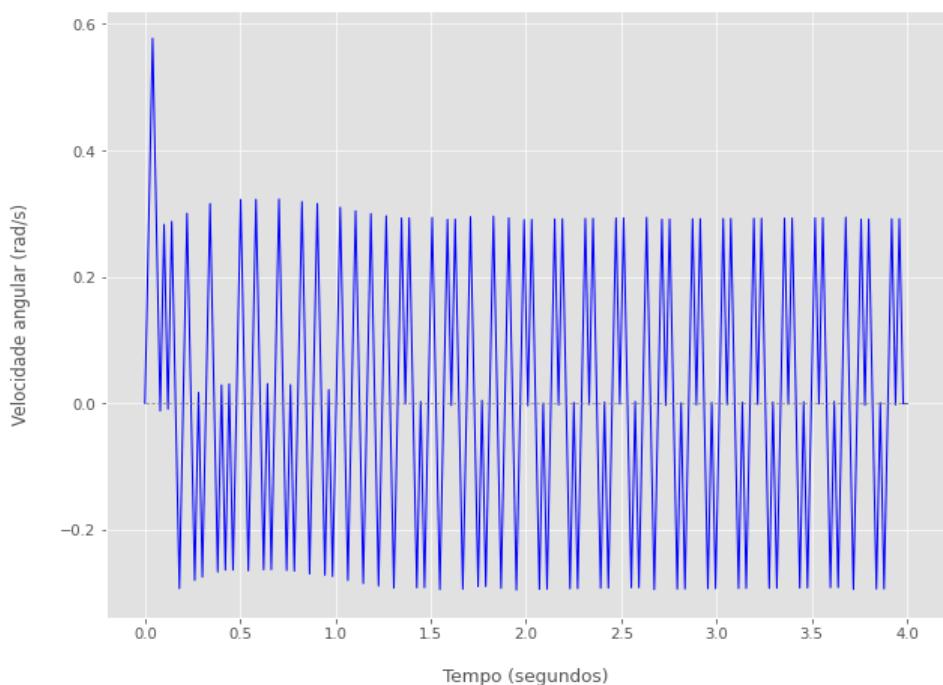
Sistema controlado limitação no esforço de controle



Fonte: Autor.

Figura 29 - Resposta da velocidade angular com controlador sob efeito do saturador.

Sistema controlado limitação no esforço de controle



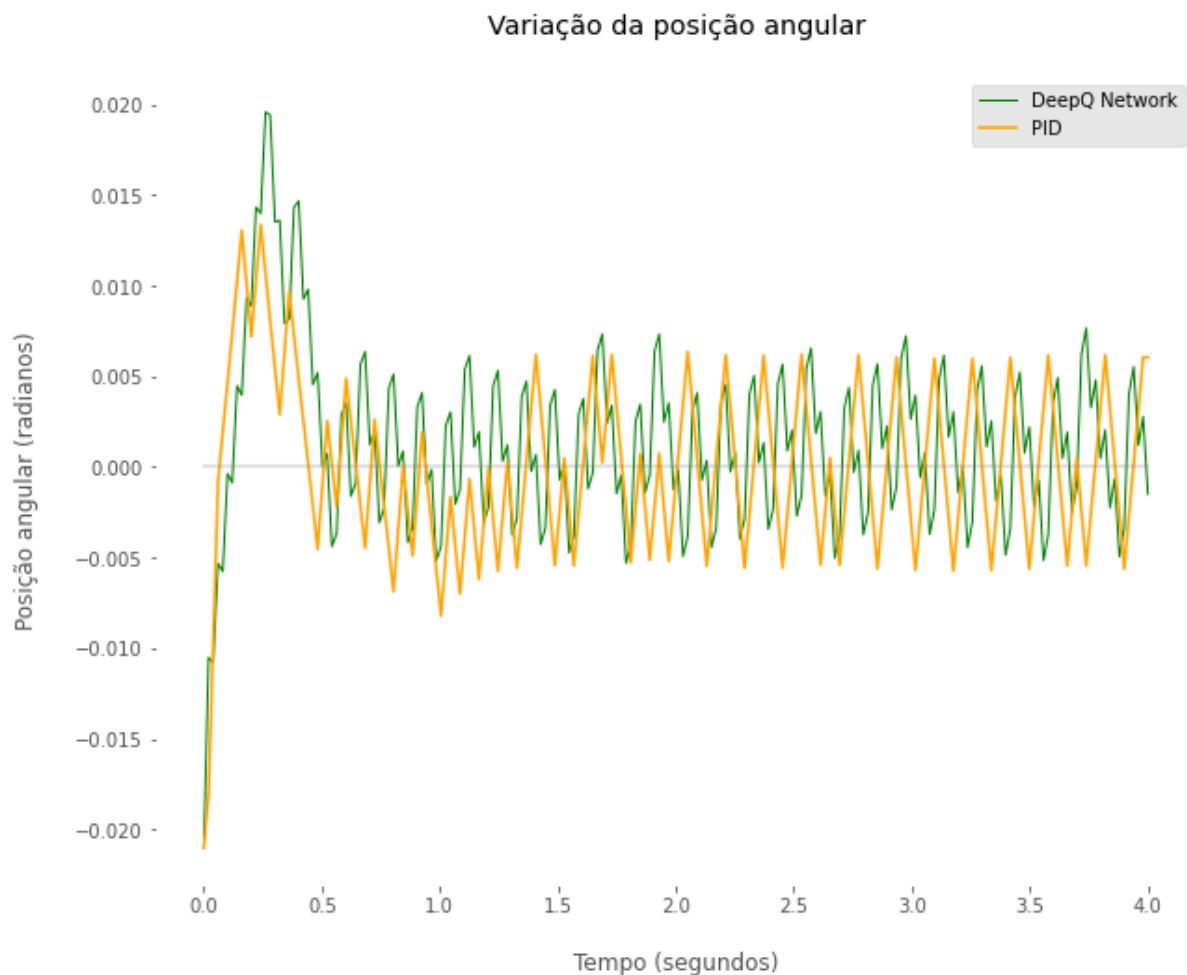
Fonte: Autor.

As Figuras 26, 27, 28 e 29 mostram a resposta do sistema controlado sob efeito do saturador que limita o esforço de controle.

Feita as análises das respostas do sistema e tendo em vista que a matriz de ganho K5 obteve os menores parâmetros de projeto atuando num limite de 10 newtons, ela foi o ganho escolhido para o controlador PID.

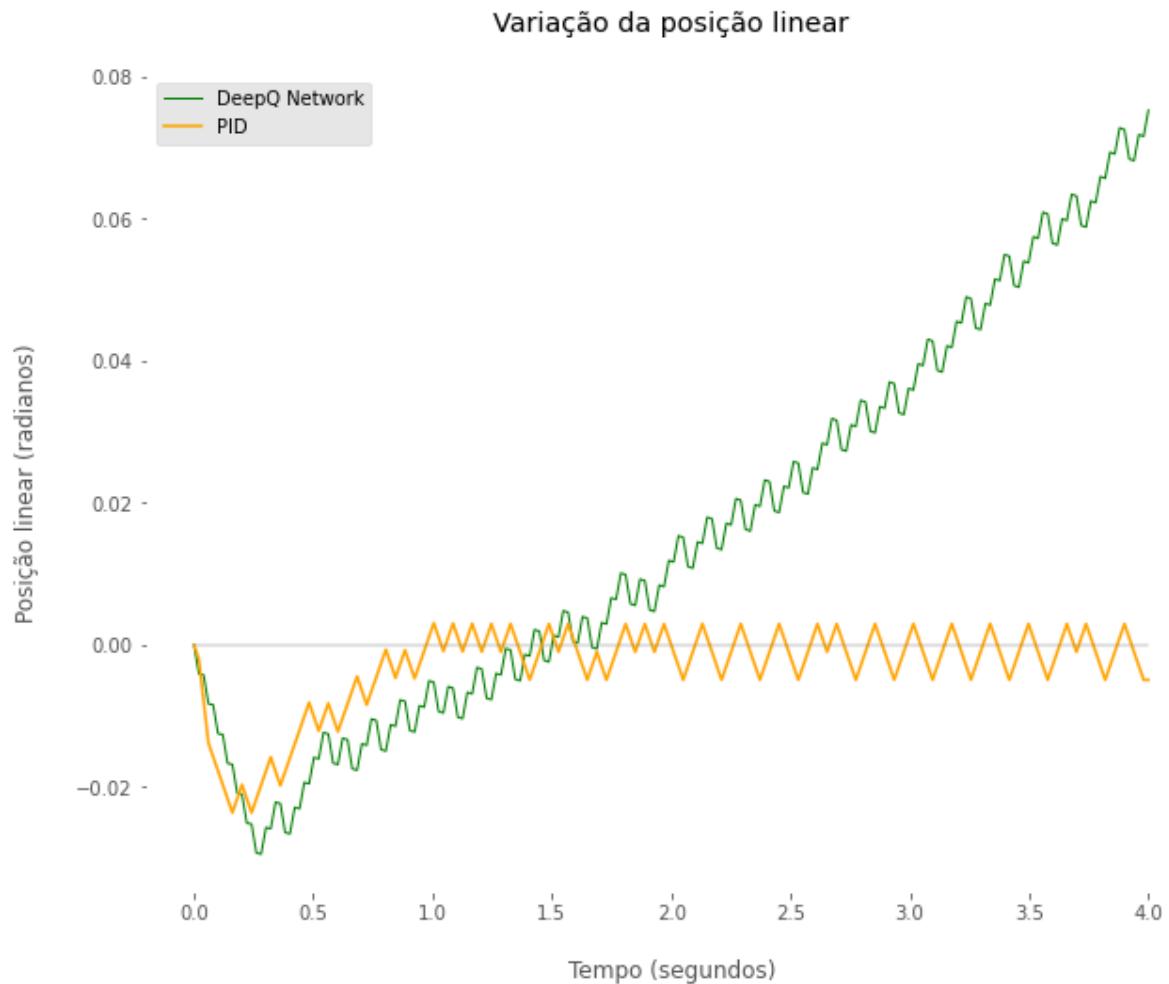
Desenvolvidos ambos os controladores foi realizada a comparação de performance entre eles.

Figura 30 - Comparação da resposta da posição angular θ entre os algoritmos.



Fonte: Autor.

Figura 31 - Comparação da resposta da posição linear x entre os algoritmos.



Fonte: Autor.

A Figura 30 mostra que a resposta angular θ usando o controle por alocação de polos é parecida ao resultado do algoritmo DQN, com este último levando o pêndulo a 0,02 radianos positivos antes de retornar ao equilíbrio vertical.

Ao avaliar a resposta da posição linear x , o controle por alocação de polos resulta em uma resposta centrada no ponto inicial diferente do algoritmo DQN que leva o carrinho infinitamente para esquerda.

Foi calculado o erro quadrático médio RMSE para ambos controladores considerando o valor correto θ e x iguais a zero como mostra a Tabela 7.

Tabela 7 - Erros em relação à $\theta = 0$ e $x = 0$.

Controlador	$RMSE_{\theta}$	$RMSE_x$
<i>Deep Q-Network</i>	0,0052	0,0320
Alocação de polos matriz K5	0,0047	0,0068

Fonte: Autor.

O controlador de alocação de polos conseguiu um erro em relação a θ menor por possuir um pico inicial menor que o controlador neural. Além disso, ele teve um erro 4,7 vezes menor em relação a posição linear do DQN pelo fato deste último não conseguir manter o carro parado em torno da origem.

Capítulo 5

Discussão

Análise dos resultados mostra que o DQN tem uma performance inferior à abordagem de controle convencional. Quanto ao treinamento do algoritmo DQN, o método de otimização Adam foi eficiente para o treinamento da rede neural atingindo a recompensa máxima devido a sua natureza recente e aperfeiçoamentos a partir de técnicas como gradiente descendente e Adagrad. Apesar disso, o agente necessitou de muitas etapas de treinamento por requerer um grande conjunto de dados exploratórios para a estimativa da função ação-valor. Houve um tempo de treinamento relativamente baixo comparado a outros algoritmos e aos experimentos de Krul (2021) e Géron (2021) já que nesse método a função ação-valor orienta as ações do agente sem precisar ser usada para estimativas adicionais.

Ao comparar as respostas das variáveis de estado a partir da mesma posição angular inicial, o algoritmo DQN teve erros maiores comparado ao controlador clássico apresentando maiores amplitudes nas variações de posição angular e linear. Estes resultados eram esperados devido à dificuldade excessiva de colocar RL em prática, em grande parte devido às instabilidades de treinamento e à enorme sensibilidade à escolha de valores de hiperparâmetros e sementes aleatórias (*random seeds*). Como afirmou o pesquisador Andrej Karpathy: "[O aprendizado supervisionado] quer trabalhar [...] O aprendizado por reforço deve ser obrigado a trabalhar." Portanto, em muitos casos, é necessário tempo e vários experimentos com diferentes hiperparâmetros para alcançar o objetivo desejado, no caso, manter o pêndulo na vertical. Este é um dos principais motivos pelos quais o aprendizado por reforço não é tão amplamente adotado quanto o aprendizado supervisionado comum (por exemplo, redes convolucionais). No entanto, ele é usado na vida real, como em aplicações de robótica, sistemas de recomendação e balanceamento de carga em centros de dados (GÉRON, 2021).

A função de custo para RL não foi planejada nem discutida por ser um indicador de desempenho insuficiente para problemas de RL. A perda pode diminuir, mas o desempenho do agente pode piorar (por exemplo, quando o agente fica preso em uma pequena região do ambiente e a rede neural começa a se ajustar excessivamente a essa região). Por outro lado, a perda pode aumentar, mas o

desempenho do agente pode melhorar (por exemplo, se a rede estiver subestimando os valores Q e começar a aumentar suas previsões corretamente, o agente provavelmente terá um desempenho melhor, recebendo mais recompensas, apesar de a perda pode aumentar, já que a DQN define os alvos, que também serão maiores) (GÉRON, 2021).

Na construção do controlador de alocação de polos, foi identificado que, devido à limitação de ação do sistema, nem todos os valores desejados de sobre-elevação e tempo de acomodação eram alcançáveis. Isso ocorre porque para estabilizar o sistema em curtos períodos, a ação precisa ser intensa, mas o atuador não tem capacidade para tal.

A grande vantagem do RL nesse caso é não ter que conhecer a dinâmica do sistema nem linearizar suas equações evitando a perda de uma informação matemática importante para o controle, entretanto ainda é necessário um ambiente de simulação. Já o controle clássico necessita da linearização e conhecimento prévio das equações de movimento do sistema.

Capítulo 6

Conclusão

O trabalho trata de duas abordagens para solucionar o problema do pêndulo invertido de um grau de liberdade. A primeira abordagem usa o aprendizado por reforço para controlar o pêndulo, enquanto a segunda utiliza o controle de estados com alocação de polos, amplamente conhecido e estudado.

A primeira abordagem foi implementada usando a linguagem *Python* e o ambiente *Cartpole-v1* da *OpenAI Gym*, comum para comparar algoritmos de controle com aprendizado por reforço, veja o Código Fonte (*link* disponível nas referências). O método DQN foi usado com a API *TensorFlow Agents*. Durante a etapa de treinamento, o agente aprendeu a equilibrar o pêndulo em vários episódios usando um sistema de recompensas. O objetivo era coletar o maior número de recompensas possível em um período de tempo.

O treinamento do método DQN foi realizado com sucesso, apesar de ser computacionalmente intenso e levar muitos passos em relação a outros experimentos da comunidade. O método de otimização Adam mostrou ser eficiente durante o ajuste dos pesos da rede neural. Após o treinamento, as simulações do comportamento do agente foram realizadas a partir de uma posição angular de 1,203°. No entanto, os resultados não foram satisfatórios, pois apesar de equilibrar o pêndulo com erro aceitável, o controlador não conseguiu manter o carro próximo ao centro.

Para a abordagem de alocação de polos, o sistema foi modelado e linearizado em espaço de estados a partir de equações diferenciais ordinárias conhecidas na literatura. As matrizes de ganho para estabilizar o sistema foram encontradas com base em sete combinações diferentes de sobre-elevação e tempo de acomodação. Três das sete tentativas alcançaram o objetivo, sendo que a solução com menor sobre-elevação e tempo de acomodação controlou o sistema com resposta satisfatória. A limitação de atuação do sistema não permitiu que todos os controladores clássicos resolvessem o problema. Especificações de engenharia que requerem uma resposta mais rápida exigiam uma atuação acima da capacidade do atuador. Além disso, a solução com o menor tempo de acomodação apresentou uma

resposta às perturbações do sistema mais lenta do que os controladores onde o esforço de controle ultrapassou o limite permitido.

Então foram comparadas as respostas das duas abordagens. A análise mostrou que o método de alocação de polos teve uma resposta melhor do que o DQN, tanto no equilíbrio da haste quanto em manter o carro na origem zero. O erro da posição linear central foi consideravelmente menor na resposta obtida com o método de alocação de polos. Isso pode ser explicado pela modelagem de recompensas do agente de RL em que os limites da posição linear para obter a recompensa foram amplos.

Em conclusão, os objetivos do trabalho foram atingidos, pois o principal objetivo era avaliar o problema dinâmico do pêndulo invertido com um grau de liberdade, propor o desenvolvimento de um controlador utilizando o algoritmo de aprendizado por reforço e comparar com um método de controle contínuo clássico. A comparação dos métodos mostrou que o algoritmo de aprendizado por reforço apresentou respostas um pouco insatisfatórias, mas que poderiam ser melhoradas alterando os hiperparâmetros do modelo e as variáveis de treinamento.

O treinamento de modelos de RL é frequentemente desafiador devido a vários fatores, incluindo a instabilidade do treinamento, a escolha inadequada de hiperparâmetros e a dependência da semente aleatória. Algumas abordagens que podem ser úteis para lidar com esses desafios incluem:

1. Treinamento de amostra dupla: o uso de dois modelos, um como ator e outro como crítico, pode ajudar a estabilizar o treinamento;
2. Estratégias de hiperparâmetros: a busca por grade e a otimização bayesiana são duas abordagens comuns para escolher valores ótimos de hiperparâmetros;
3. Ajuste de sementes aleatórias: a utilização de múltiplas sementes aleatórias durante o treinamento e a seleção da semente com o desempenho mais alto pode ajudar a lidar com a dependência da semente aleatória;
4. Regularização: a adição de regularização, como dropout ou penalização L2, pode ajudar a prevenir *overfitting* e tornar o treinamento mais estável;
5. Modificações na arquitetura da rede: experimentar com diferentes tipos de redes neurais, como redes neurais convolucionais ou redes neurais recorrentes, pode ajudar a melhorar o desempenho;

6. Modelagem do sistema de recompensas: testar diferentes intervalos de recompensas para a posição linear e posição angular, além de adicionar recompensas negativas em diferentes estados.

Em geral, a abordagem para solucionar esses desafios é experimentar com diferentes combinações de técnicas e encontrar a que funciona melhor para um problema específico.

Uma das vantagens do ML é a não necessidade de conhecimento da física do problema, já que o agente aprende por conta própria a atingir o objetivo definido. Para a engenharia de controle e automação, esta abordagem se combina com uma ampla gama de técnicas de controle para ampliar as opções disponíveis na resolução de problemas.

Uma sugestão para trabalhos seguintes é criar um controlador com aprendizado supervisionado a partir de um banco de dados de um controlador clássico que possa aprender a comportamento do sistema controlado podendo prever as saídas das variáveis de estado com a respectiva força aplicada.

Referências Bibliográficas

ALVARENGA, V. L. **Identificação de uma plataforma de um pêndulo invertido e simulação de técnicas de controle.** 2019. 79 f. Dissertação (Mestrado em Modelagem de Sistemas Fuzzy) - Departamento de Engenharia de Sistemas, Universidade Federal de Lavras, Lavras, 2013.

BELLMAN, R. **Dynamic Programming.** Princeton: Princeton University Press, 1957.

BOUBAKER, O.; IRIARTE, R. **The Inverted Pendulum in Control Theory and Robotics:** From theory to new innovations. The Institution of Engineering and Technology, 2017.

CARVALHO, D. B.; JOTA, F. G. **Controle de um pêndulo invertido por meios de redes neurais artificiais treinadas evolutivamente.** In: 40. SBAI - Simpósio Brasileiro de Automação Inteligente, São Paulo, Brasil, 1999.

CAVALCANTI, J. H. F.; ALSINA, P. J.; FERNEDA, E. Posicionamento de um pêndulo invertido usando algoritmos genéticos. **SBA Controle e Automação**, Campinas, v. 10, n. 1, p. 31-39, 1999.

COLOMBINI, E. L. **Notas de Aula: Introduction to AI: Lecture 16 – Machine Learning – Reinforcement Learning.** Instituto de Computação, UNICAMP, 2019. Disponível em: <https://github.com/CarlosChiarelli/toria-ml/tree/master/disciplina-IC-UNICAMP/6_aprendizadoReforco>. Acesso em: 19 jun. 2022.

DRUMMOND, A. C.; OLIVEIRA, K. C.; BAUCHSPIESS, A. **Estudo do controle de pêndulo inverso sobre carro utilizando rede neural de base radial.** In: IV Congresso Brasileiro de Redes Neurais. São José dos Campos, Brasil, 1999.

GÉRON, A. **Mãos à obra: aprendizado de máquina com Scikit-Learn, Keras & TensorFlow**: Conceitos, ferramentas e técnicas para a construção de sistemas inteligentes. 2. ed. Rio de Janeiro: Alta Books. 2021.

GÉRON, A. **Mãos à obra: aprendizado de máquina com Scikit-Learn, Keras & TensorFlow**. 1. ed. Rio de Janeiro: Alta Books. 2019.

GORI, M.; TESI, A. On the problem of local minima in backpropagation. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, New York, v. 14, n. 1, p. 76-86, jan. 1992.

GUPTA, M. M.; SINHA, N. K.; **Intelligent Control Systems**: Theory and Applications. Piscataway: IEEE Press, 1999.

HETCHT-NIELSEN R. Theory of the backpropagation neural network. **IEEE: International Joint Conference on Neural Networks**. Washington, v. 1, p. 593-605. 1989.

KRUL, A. M. **Aprendizagem por reforço como técnica de controle para o problema do pêndulo invertido**. 2021. 99 f. Trabalho de Conclusão de Curso (Graduação em Engenharia Mecânica) - Faculdade de Tecnologia, Universidade Federal do Amazonas, Manaus, 2021.

MITCHELL, T. M. **Machine Learning**: A multistrategy approach. New York: McGraw-Hill, 1997.

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

NISE, N. S. **Control Systems Engineering**. [S.I.]: John Wiley & Sons, 2007.

OGATA, K.; YANG, Y. **Modern control engineering**. India: Prentice Hall, v. 4. 2002.

PEREIRA. M. C. **Notas de aula de Técnicas em Teoria de Controle**: Sistemas Lineares de Controle - Controlabilidade. USP, ANO. Site: Disponível em: <[https://github.com/CarlosChiarelli/eng-controle-automacao/blob/master/06_controle_avancado/class/09_obs_estado/%5B13%5D%20Controlabilidade_Observabilidade%20\(1\).pdf](https://github.com/CarlosChiarelli/eng-controle-automacao/blob/master/06_controle_avancado/class/09_obs_estado/%5B13%5D%20Controlabilidade_Observabilidade%20(1).pdf)>. Acesso em: 5 jul. 2022.

Sem autor: **Colaboratory**. Google Colab. Disponível em <<https://research.google.com/colaboratory/intl/pt-BR/faq.html#:~:text=Mais%20tecnicamente%2C%20o%20Colab%20%C3%A9,recursos%20de%20computa%C3%A7%C3%A3o%20como%20GPUs>>. Acesso em: 24 jan. 2023.

Sem autor: **TF-Agents**. Disponível em: <<https://github.com/tensorflow/agents>>. Acesso em: 10 ago. 2022.

Sem autor: **Código fonte**. Disponível em <https://github.com/CarlosChiarelli/eng-controle-automacao/tree/master/07_tcc>. Acesso em: 31 jan. 2023.

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