Instituto Tecnológico de Aeronáutica

CT-213

Laboratório 7 - Imitation Learning com Keras

Aluno: Pedro Elardenberg Sousa e Souza
Professor: Marcos Ricardo Omena de Albuquerque Maximo

31 de dezembro de 2023

Conteúdo

1 Resumo

Este Laboratório é uma atividade que visa a introdução à ferramenta Keras para construção de redes neurais em *python* utilizando o *framework Tensorflow.* Ademais, esta atividade visa copiar um movimento de caminhar de um robô humanoide usando uma técnica chamada *imitation learning.*

Para o primeiro objetivo, foi utilizado o código *test_keras.py,* com o qual foi visto como a ferramenta cria uma rede neural, suas camadas, e sua compilação. Foram gerados gráficos de treinamento desse código para exemplificar o seu funcionamento.

Em seguida, no código *imitation_learning.py,* foi criada uma rede neural de 4 camadas que utiliza um modelo de movimento das juntas de um robô baseado em teoria de controle para aprender o seu movimento. Os resultados obtidos foram mostrados em imagens, para dois números de iterações do treinamento do algoritmo diferentes.

Os resultados foram então mostrados e discutidos neste relatório.

2 Introdução

Keras é uma Interface de Programação de Aplicação (API) feita em *python* e que é executada em *frameworks* como JAX, Pytorch e Tensorflow. Sua filosofia é ser simples, flexível e poderoso. Simples pois suaviza a curva de aprendizado no assunto e torna o desenvolvedor mais livre para atuar nas partes do problema que mais importam; Flexível pois é feito para tornar trabalhos simples mais diretos ao mesmo tempo em que permite que solções estado da arte sejam possíveis; e Poderoso porque é uma ferramenta que oferece performance e escalabilidade a nível industrial, sendo utilizada por grandes empresas no mundo [1].

TensorFlow é uma biblioteca de código aberto criada para aprendizado de máquina, computação numérica e muitas outras tarefas. Foi desenvolvido pelo Google em 2015 e rapidamente se tornou uma das principais ferramentas para machine learning e deep learning [3].

Para este laboratório, foi necessário instalar versões antigas do *numpy* e do *tensorflow*. Foi verificado que a versão do *numpy* compatível com o código era uma versão anterior à 1.20. As ferramentas, portanto, foram instaladas da seguinte forma:

```
pip install tensorflow==2.5.0
pip install numpy=1.19.5
```

Imitation Learning é uma técnica de *machine learning* em que o algoritmo busca imitar um comportamento observado na natureza. Para este laboratório, o objetivo é copiar um movimento de caminhar de um robô. Em abordagens tradicionais de *machine learning*, o agente aprende por tentativa e erro dentro de determinado ambiente, guiado por uma função de custo ou de recompensa. Em *Imitation Learning*, por sua vez, o agente aprende por meio de um *dataset* de demonstrações de uma entidade modelo, neste caso, o modelo de caminhada do robô utilizando teoria de controle. O objetivo é replicar o comportamento da entidade modelo em condições similares, senão em mesmas condições[2].

2.1 Como *Imitation Learning* funciona

Imitation Learning envolve observar a atividade realizada pela entidade modelo e aprender a imitar essas ações. O processo geralmente envolve três passos:

- *Data Collection:* A entidade modelo demonstra a atividade a ser aprendida. Neste caso, é a própria caminhada do robô. As ações e decisões dessa entidade são armazenadas como dados.

- *Learning:* Os dados coletados são utilizados para treinar o modelo de *machine learning*. O modelo aprende a política - o mapeamento de observações do ambiente a ações - que tenta replicar o comportamento da entidade modelo.

- *Evaluation:* O modelo treinado é testado no ambiente para verificar sua performance em comparação com a entidade modelo. O objetivo é minimizar a diferença entre a performance da entidade e a do agente.

2.2 Regularização L_2

Para uma rede profunda, o problema do *overfitting,* isto é, quando o modelo treinado se adequa demais aos exemplos de treinamento, tornando a representação viciada, é maior que numa rede com menos camadas, pois seriam necessários muitos dados para dar à rede uma capacidade de generalização satisfatória. Nesses casos, utiliza-se regularização. Esse artifício visa fazer a rede não depender muito de um único neurônio, o que pode favorecer muito o *overfitting*[4]. Um tipo de regularização muito utilizada é a chamada L_2:

$$J_{L_2} = \frac{\lambda_{L_2}}{2m} \sum_j \theta_j^2 \tag{1}$$

Em que J_{L_2} é o termo da função de custo associado à regularização, λ_{L_2} é um hiperparâmetro e θ_j são os vetores de pesos e *biases* da minha função de representação.

O gradiente da função de custo, portanto, é igual a:

$$\frac{\partial J}{\partial \theta_j} = \frac{\partial J_{errorL_2}}{\partial \theta_j} + \frac{\lambda_{L_2}}{m}\theta_j \tag{2}$$

Logo, a descida de gradiente será dada por:

$$\theta_{j+1} = \theta_j - \alpha \frac{\partial J_{L_2}}{\partial \theta_j} = \left(1 - \alpha \frac{\lambda_{L_2}}{2m}\right)\theta_j - \alpha \frac{\partial J_{errorL_2}}{\partial \theta_j} \tag{3}$$

3 Análise dos Resultados

Este laboratório possui um código inicial *test_keras.py*, destinado a apresentar a ferramenta Keras de rede neural. Todas as atividades realizadas utilizam-se das ideias exemplificadas nesse código para a sua realização.

3.1 Estudo de Implementação de Rede Neural com Keras

O algoritmo *test_keras.py* é inicializado com os seguintes valores iniciais:

Tabela 1: Valores iniciais do algoritmo *test_keras.py*

Variável inicial	**Valor**
λ_{L_2}	0 ou 0.02
num_cases	200
num_epochs	5000
inputs	$[num\_cases][2]$ $rand(\pm 10)$
expected_outputs	gtz(inputs) ou xor(inputs)
noise	$[num\_cases][2]$ $rand(\pm 2)$

Em que λ_{L_2} é um hiperparâmetro associado à regularização dos pesos da função de custo; num_cases e num_epochs são, respectivamente, a quantidade inicial dos casos autogerados e a quantidade de iterações de treino da rede; *inputs* é uma matriz [num_cases][2] com valores iniciais aleatórios de ± 10.

A variável expected_outputs é o modelo ao qual os valores iniciais serão adequados, que corresponde à função *gtz* ou *soma maior que zero*, que retorna o valor 1 caso os valores iniciais de um *input* seja maior que zero e retorna 0 caso contrário, e *xor* retorna o valor 1 caso um dos valores iniciais de um *input* seja maior que zero e o outro seja menor que zero e retorna 0 caso contrário.

O parâmetro *noise* é um ruído adicionado a *inputs* após a definição de expected_outputs para corromper os dados iniciais.

Após isso, a criação da rede neural utilizando Keras fica da seguinte forma:

Primeiramente, inicia-se a rede neural na variável *model:*

```
model = models.Sequential()
```

Depois, adiciona-se uma camada à rede. Aqui, foram utilizadas duas camadas, a primeira com 50 neurônios e a segunda com 1, funções de ativação sigmóide e regularização L_2.

```
model.add(layers.Dense(50, activation=activations.sigmoid,
    input_shape=(2,), kernel_regularizer=regularizers.l2
    (lambda_l2)))
    model.add(layers.Dense(1, activation=activations.sigmoid,
    kernel_regularizer=regularizers.l2(lambda_l2)))
```

Antes de treinar o modelo, é preciso compilá-lo com uma função de perda. Neste caso, foi utilizada a função entropia cruzada *(cross entropy)* com *Adam optimization:*

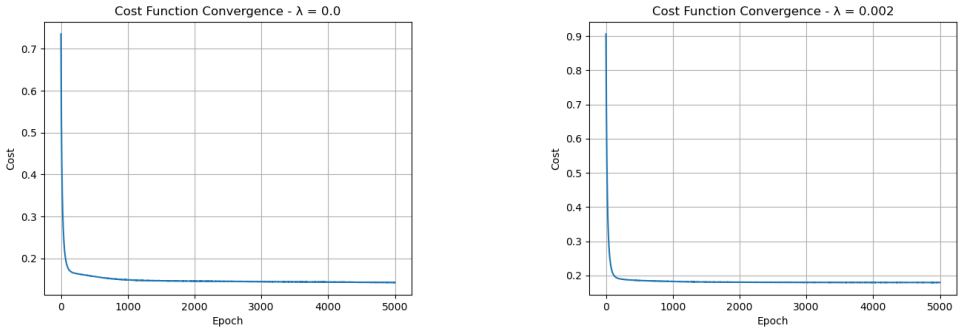
```
model.compile(optimizer=optimizers.Adam(), loss=losses.
    binary_crossentropy, metrics=[metrics.binary_accuracy])
```

Para treinar o modelo, usa-se o método *fit*:

```
history = model.fit(inputs, expected_outputs, batch_size=
    (num_cases // 4), epochs=num_epochs)
```

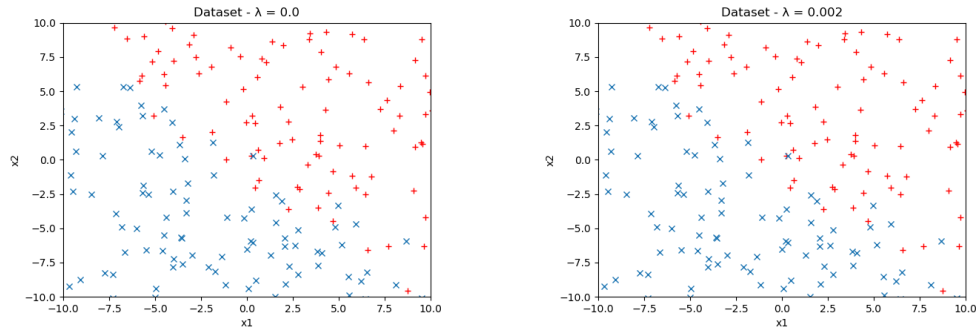
3.2 Análise do Efeito de Regularização

O algoritmo *test_keras.py* foi executado de acordo com os parâmetros descritos na tabela 1. Para a função de classificação *gtz*( ), os resultados para $\lambda_{L_2} = 0$ e $\lambda_{L_2} = 0.02$ são os seguintes:



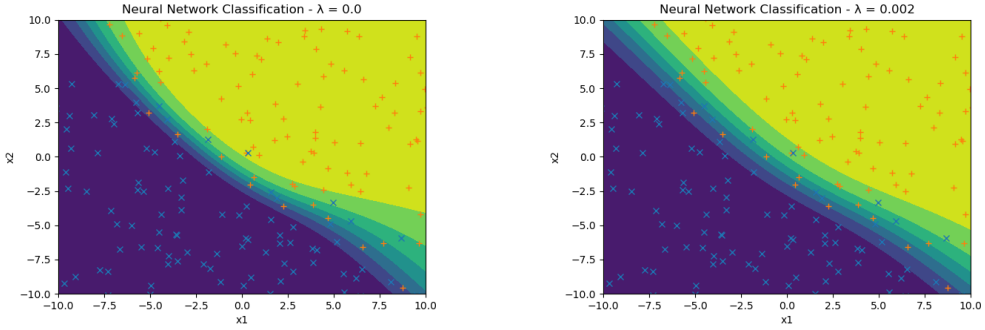
(a) (b)

Figura 1: Convergência da função *gtz* para $\lambda_{L_2} = 0$ e $\lambda_{L_2} = 0.02$



(a) (b)

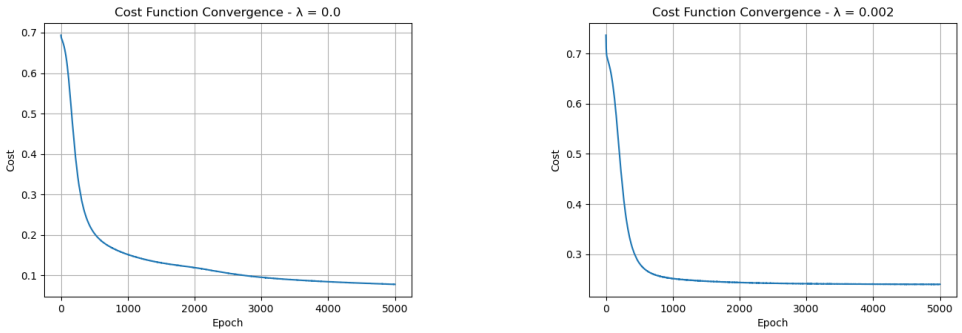
Figura 2: Conjunto de dados iniciais para a função *gtz* para $\lambda_{L_2} = 0$ e $\lambda_{L_2} = 0.02$



(a) (b)

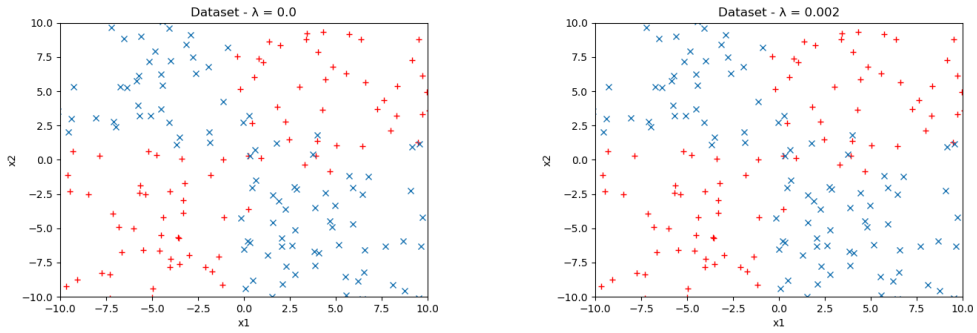
Figura 3: Resultado da função *gtz* para a classificação dos dados iniciais utilizando $\lambda_{L_2} = 0$ e $\lambda_{L_2} = 0.02$

Para essa função de classificação, percebe-se que a inclusão do parâmetro de regularização λ_{L_2} traz uma melhoria muito marginal nos resultados. A convergência do custo (figura 1) é apenas um pouco mais rápida com a inclusão e a figura 3, que mostra o resultado da classificação, apresenta uma divisão um pouco mais próxima de uma diagonal, que seria a solução do problema.



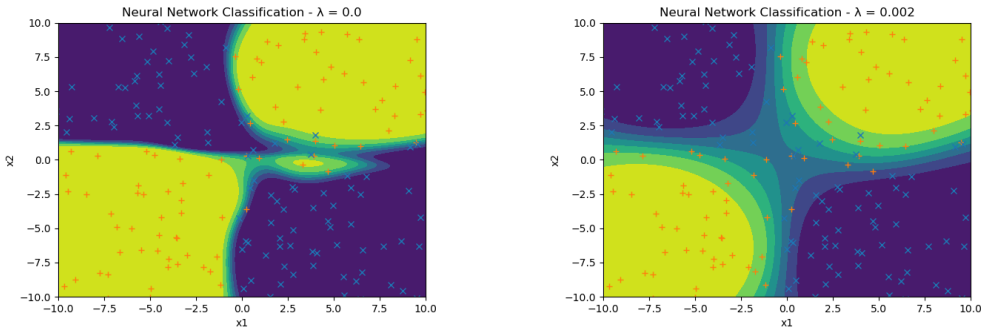
(a) (b)

Figura 4: Convergência da função *xor* para $\lambda_{L_2} = 0$ e $\lambda_{L_2} = 0.02$



(a) (b)

Figura 5: Conjunto de dados iniciais para a função *xor* para $\lambda_{L_2} = 0$ e $\lambda_{L_2} = 0.02$



(a) (b)

Figura 6: Resultado da função *xor* para a classificação dos dados iniciais utilizando $\lambda_{L_2} = 0$ e $\lambda_{L_2} = 0.02$

Para o caso da função *xor*, entretanto, a inclusão do parâmetro λ_{L_2} foi bem mais significativo, seja na velocidade da convergência da função de custo que, sem a regularização, demorou cerca de 3000 iterações para se obter um custo menor que 0.1, enquanto que, com a regularização, o algoritmo convergiu em menos de 1000 iterações (figura 4, seja no resultado da classificação, que se assemelha mais ao resultado esperado, que seria 1 nos quadrantes ímpares na figura 6 e 0 nos quadrantes pares.

Isso ocorre porque a função *xor* não é linearmente separável, e uma rede neural com poucas camadas com uma função de ativação como a sigmóide tende a gerar *overfitting*. Ao se fazer uma regularização, o modelo "força"a rede a não depender tanto de um único neurônio, o que dá resultados menos super-representados nas áreas mais próximas do limiar. Isso gera uma rede menos dependente do modelo de teste e mais próxima de um caso mais genérico de representação.

3.3 *Imitation Learning*

Usando o Keras, foi criada uma rede neural contendo os seguintes parâmetros:

Tabela 2: Arquitetura da rede neural usada para o *imitation learning.*

Layer	**Neurons**	**Activation Function**
Dense	75	Leaky ReLU ($\alpha = 0,01$)
Dense	50	Leaky ReLU ($\alpha = 0,01$)
Dense	20	Linear

Como função de otimização, foi utilizada *Adaptive Moment Estimation* (Adam), com o tamanho do batch igual ao tamanho do dataset. Depois de algumas tentativas, o número de iterações usado para este laboratório foi 1500000. Esse número pode ser menor caso a máquina utilizada para execução do programa possua GPU ou tenha uma melhor capacidade de processamento de dados (o processador da máquina utilizada para este laboratório é o Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz, com mais de 6 anos de uso).

Os resultados são mostrados nas imagens abaixo:

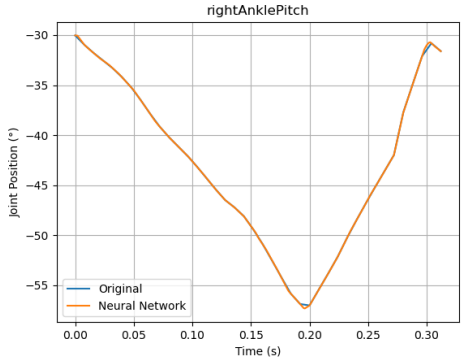


Figura 7: Inclinação do tornozelo direito do robô - Comparação

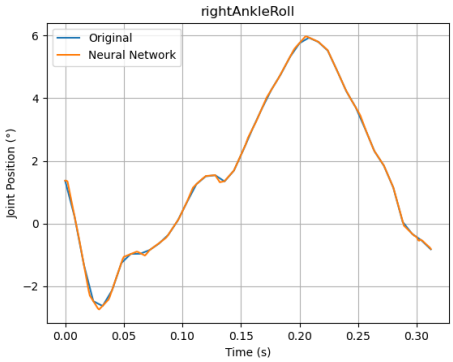


Figura 8: Rolagem do tornozelo direito do robô - Comparação

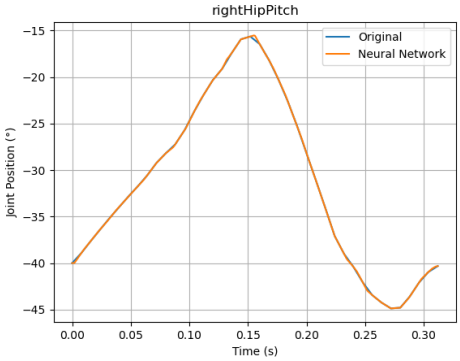


Figura 9: Inclinação do lado direito do quadril do robô - Comparação

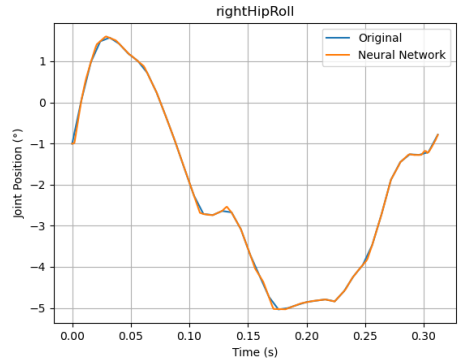


Figura 10: Rolagem do lado direito do quadril do robô - Comparação

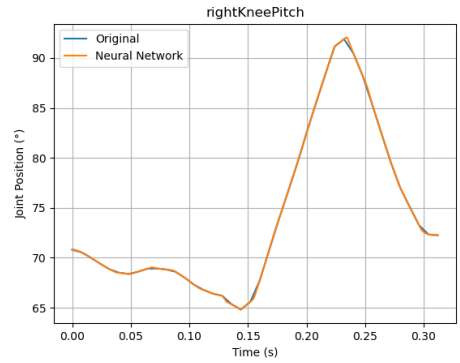


Figura 11: Inclinação do joelho direito do robô - Comparação

Para um número menor de treinamento, o programa tem dificuldades

para representar movimentos mais bruscos. As figuras abaixo mostram o desempenho para 30000 iterações, número sugerido na elaboração do laboratório.

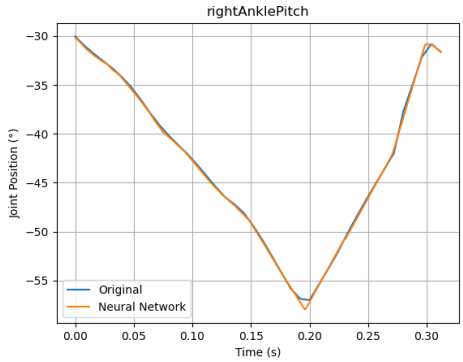


Figura 12: Inclinação do tornozelo direito do robô - Comparação 30000 iterações

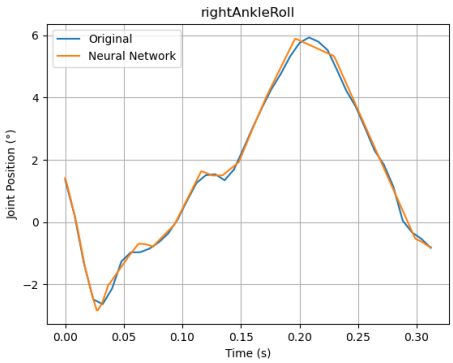


Figura 13: Rolagem do tornozelo direito do robô - Comparação

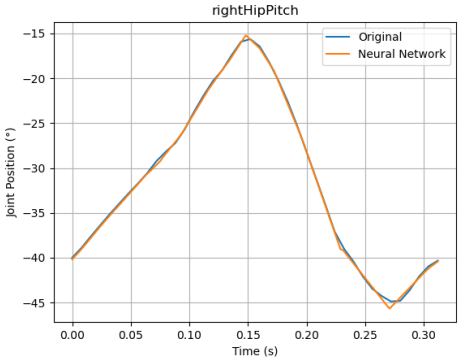


Figura 14: Inclinação do lado direito do quadril do robô - Comparação 30000 iterações

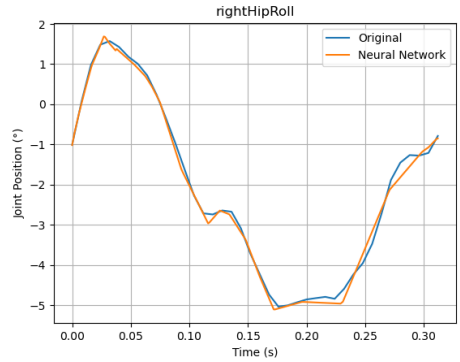


Figura 15: Rolagem do lado direito do quadril do robô - Comparação 30000 iterações

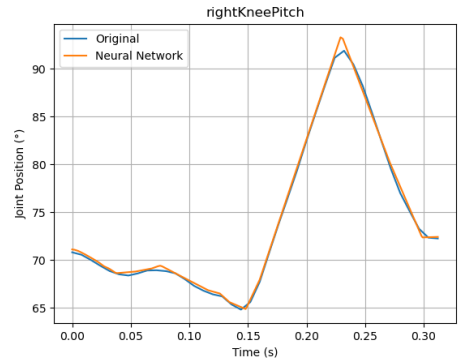


Figura 16: Inclinação do joelho direito do robô - Comparação 30000 iterações

4 Conclusão

Com este laboratório, foi possível mostrar que o Keras é uma poderosa ferramenta para construção e implementação de redes neurais, principalmente se o problema demandar redes profundas e mais complexas. Pôde-se ver, também, que a técnica *imitation learning* é muito eficaz para o aprendizado de máquina.

Ademais, pôde-se perceber o quão fácil e prático é usar essa ferramenta, em comparação com implementar todos os métodos de classificação, ativação e otimização, o que torna os avanços nos estudos nessa área muito mais rápidos e diversos.

Por sua vez, é necessário uma máquina com boa capacidade de processamento para se obter resultados mais práticos em treinamentos de redes neurais profundas.

Referências

[1] About keras 3. Acessado em 23 de Dezembro de 2023. URL: `https://keras.io/about/`.

[2] O que é imitation learning? Acessado em 23 de Dezembro de 2023. URL: `https://deepai.org/machine-learning-glossary-and-terms/imitation-learning#:~:text=Imitation%20Learning%2C%20also%20known%20as,guided%20by%20a%20reward%20function`.

[3] O que é tensorflow? Acessado em 23 de Dezembro de 2023. URL: `https://didatica.tech/o-que-e-tensorflow-para-que-serve/`.

[4] Marcos Ricardo Omena de Albuquerque Maximo. Ct-213 - aula 8 - aprendizado de máquina profundo (deep learning). Apostila, 2020. Curso CT-213 - Inteligência Artificial para Robótica Móvel, Instituto Tecnológico de Aeronáutica.