UNIVERSIDADE FEDERAL DE JUIZ DE FORA
INSTITUTO DE CIÊNCIAS EXATAS / DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO (DCC)
PROGRAMA DE PÓS-GRADUAÇÃO EM MODELAGEM COMPUTACIONAL

Felipe Rafael de Souza

Aprendizado por Reforço Assistido por Imitação para Jogos Digitais

Juiz de Fora
2023

Felipe Rafael de Souza

Aprendizado por Reforço Assistido por Imitação para Jogos Digitais

Dissertação apresentada ao Programa de Pós-graduação em Modelagem Computacional da Universidade Federal de Juiz de Fora como requisito parcial à obtenção do título de Mestre em Modelagem Computacional.

Orientador: Prof. Dr. Heder Soares Bernardino

Juiz de Fora

2023

Ficha catalográfica elaborada através do Modelo Latex do CDC da UFJF
com os dados fornecidos pelo(a) autor(a)

Souza, Felipe Rafael de.
Aprendizado por Reforço Assistido por Imitação para Jogos Digitais / Felipe Rafael de Souza. – 2023.
72 f. : il.

Orientador: Heder Soares Bernardino
Dissertação (Mestrado) – Universidade Federal de Juiz de Fora, Instituto de Ciências Exatas / Departamento de Ciência da Computação (DCC). Programa de Pós-graduação em Modelagem Computacional, 2023.

1. Aprendizado por Reforço. 2. Aprendizado por Reforço Inverso. 3. Redes Neurais Profundas. 4. Aprendizado por Imitação.

Felipe Rafael de Souza

Aprendizado por Reforço Assistido por Imitação para Jogos Digitais

Dissertação apresentada ao Programa de Pós-Graduação em Modelagem Computacional da Universidade Federal de Juiz de Fora como requisito parcial à obtenção do título de Mestre em Modelagem Computacional. Área de concentração: Modelagem Computacional.

Aprovada em 15 de março de 2023.

BANCA EXAMINADORA

Prof. Dr. Heder Soares Bernardino - Orientador
Universidade Federal de Juiz de Fora

Prof. Dr. Leonardo Goliatt da Fonseca
Universidade Federal de Juiz de Fora

Prof. Dr. Eduardo Krempser da Silva
Fundação Oswaldo Cruz - Fiocruz

Juiz de Fora, 14/03/2023.

Documento assinado eletronicamente por **Leonardo Goliatt da Fonseca**, **Professor(a)**, em 15/03/2023, às 12:50, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Heder Soares Bernardino**, **Professor(a)**, em 15/03/2023, às 12:51, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Eduardo Krempser da Silva**, **Usuário Externo**, em 15/03/2023, às 12:52, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

A autenticidade deste documento pode ser conferida no Portal do SEI-Ufjf (www2.ufjf.br/SEI) através do ícone Conferência de Documentos, informando o código verificador **1186885** e o código CRC **3857340C**.

AGRADECIMENTOS

Agradeço a todos os amigos e familiares pelo apoio, assim como aos colegas de curso pelos momentos compartilhados. Em especial, expresso minha gratidão aos meus pais pelo incentivo e suporte. Não posso deixar de reconhecer e expressar minha imensa gratidão à Gabriella por todo o carinho, dedicação e auxílio nesta jornada, seu suporte constante, incentivo e compreensão foram fundamentais para superar os desafios dessa caminhada.

Gostaria também de agradecer aos professores do Programa de Pós-Graduação em Modelagem Computacional (PPGMC) pela dedicação, bem como a todos os funcionários da Universidade Federal de Juiz de Fora (UFJF) que tornaram esta trajetória mais fácil. Um agradecimento especial ao Heder por seu apoio e orientação em todas as etapas e mudanças que ocorreram durante o meu percurso acadêmico.

Por fim, expresso minha gratidão à Fundação de Amparo à Pesquisa do Estado de Minas Gerais (FAPEMIG), à Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) e à UFJF.

Porque a ciência tem que ser divertida! (Scicast)

RESUMO

O Aprendizado por Reforço (RL) e o Aprendizado por Imitação (IL) são ramos da Inteligência Artificial que possibilitam o aprendizado através da interação com o ambiente e através da observação de exemplos, respectivamente. Eles possuem aplicações em diversas áreas, tais como: veículos autônomos, controle de robôs e jogos. Os jogos são amplamente utilizados para testar o desempenho de modelos de Aprendizado por Reforço, geralmente utilizando redes neurais profundas, pois proporcionam um ambiente controlado capaz de expor o modelo à uma ampla variedade de problemas e contextos. Dessa forma, o presente trabalho tem como objetivo propor modelos de controle para o jogo *Sonic The Hedgehog* utilizando Aprendizado por Imitação e Aprendizado por Reforço Profundo. Além disso, busca-se analisar o desempenho de modelos de imitação baseados em estratégias adversariais, investigar o impacto da imitação no comportamento e desempenho do modelo, e verificar se o Aprendizado por Imitação pode ser uma alternativa viável à criação de funções de recompensa. Foram realizados experimentos comparando diversos métodos de IL, a fim de verificar se o mesmo seria capaz de gerar bons controladores para o jogo. Em seguida, os métodos de IL de clonagem comportamental, Aprendizado por Imitação Generativo Adversarial e Aprendizado por Reforço Inverso Adversarial foram utilizados para iniciar o RL, com a hipótese de que o conhecimento prévio de domínio disponibilizado pela imitação auxilie o modelo a atingir melhores resultados. Os resultados obtidos mostraram que o IL pode ser utilizado para gerar controladores de jogos digitais e que a inicialização da etapa de RL com o Aprendizado por Imitação pode ajudar o modelo a obter melhor desempenho.

Palavras-chave: Aprendizado por Reforço Profundo, Redes Neurais Convolucionais, Redes Neurais Adversariais, Aprendizado por Imitação, Aprendizado por Reforço Inverso, Otimização de Política Proximal

ABSTRACT

Reinforcement Learning (RL) and Imitation Learning (IL) are branches of Artificial Intelligence that enable learning through interaction with the environment and through observation of examples, respectively. They have applications in several areas, such as: autonomous vehicles, robot control and games. Games are widely used to test the performance of Reinforcement Learning models, usually using deep neural networks, as they provide a controlled environment capable of exposing the model to a wide variety of problems and contexts. Thus, the present work aims to propose control models for the game *Sonic The Hedgehog* using Imitation Learning and Deep Reinforcement Learning. In addition, we seek to analyze the performance of imitation models based on adversarial strategies, investigate the impact of imitation on the model's behavior and performance, and verify whether Imitation Learning can be a viable alternative to creating reward functions. Experiments were carried out comparing different IL methods, in order to verify if it would be able to generate good controllers for the game. Then, the IL methods of behavioral cloning, Adversarial Generative Imitation Learning and Adversarial Inverse Reinforcement Learning were used to start the RL, with the hypothesis that the prior domain knowledge provided by imitation helps the model to achieve better results. The obtained results showed that the IL can be used to generate digital game controllers and that the initialization of the RL step with Imitation Learning can help the model to obtain better performance.

Keywords: Deep Reinforcement Learning, Convolutional Neural Networks, Adversarial Neural Networks, Imitation Learning, Inverse Reinforcement Learning, Proximal Policy Optimization

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

RL	*Reinforcement Learning*
DP	*Deep Learning*
IRL	*Inverse Reinforcement Learning*
MLP	*Multilayer Perceptron*
PPO	*Proximal Policy Otimization*
BC	*Behavior Cloning*
GAIL	*Generative Adversarial Imitation Learning*
AIRL	*Adversarial Inverse Reinforcement Learning*
GAN	*Generative Adversarial Network*

SUMÁRIO

1 INTRODUÇÃO

O desenvolvimento de agentes inteligentes capazes de aprender diretamente de imagens tem sido um grande desafio para o Aprendizado por Reforço, *Reinforcement Learning* (RL), nos últimos anos, e possui aplicações em diversas áreas, tais como: veículos autônomos (Aradi, 2020), controle de robôs (Nguyen and La, 2019), diagnóstico médico (Zhou et al., 2021) e jogos (Mnih et al., 2013). Os jogos são amplamente utilizados para testar o desempenho de modelos de Aprendizado por Reforço e Aprendizado por Reforço Profundo, pois proporcionam um ambiente controlado capaz de expor o modelo à uma ampla variedade de problemas e contextos.

O Aprendizado por Reforço é um ramo da Inteligência Artificial, derivado dos estudos de aprendizagem da Psicologia (Kaelbling et al., 1996), onde os comportamentos de um agente tendem a se repetir ou extinguir de acordo com as respostas às suas ações. Diferente de outros tipos de aprendizado, como o supervisionado e não supervisionado, em que é necessário ter acesso a um conjunto de dados para treinar o modelo, no Aprendizado por Reforço o aprendizado se dá através da interação com o ambiente, onde o agente executa ações e recebe recompensas ou punições como respostas, tendo como objetivo maximizar a recompensa recebida. Porém, em diversos problemas, é difícil definir uma função de recompensa adequada (Dulac-Arnold et al., 2019). Existem diversas alternativas para tentar amenizar o problema de definição da função de recompensa, entre elas o Aprendizado por Imitação, *Imitation Learning* (IL) (Hussein et al., 2017) e o Aprendizado por Reforço Inverso, *Inverse Reinforcement Learning* (IRL) (Zhifei and Meng Joo, 2012). No IL, o agente aprende a partir de observações de comportamento de um especialista, permitindo que comportamentos sejam aprendidos mais rapidamente (Bandura, 1962). Já no IRL, o agente utiliza as observações do comportamento para gerar uma função de recompensa.

O Aprendizado por Reforço Profundo, *Deep Reinforcement Learning* (DRL) (Huang et al., 2012), surge da combinação do RL com Redes Neurais Profundas, *Deep Neural Networks* (DNN). O DRL permitiu a utilização do RL em ambientes de alta dimensão com informações não estruturadas, como criar agentes para tomar decisões a partir do uso direto de imagens.

Diversos trabalhos exploraram a criação de controladores para jogos utilizando DRL, como (Mnih et al., 2013), (Bhonker et al., 2016), (Ibarz et al., 2018) e (Berner et al., 2019). Porém, a utilização de técnicas adversariais de IL, como: *Generative Adversarial Imitation Learning* (GAIL) (Ho and Ermon, 2016) *Adversarial Inverse Reinforcement Learning* (AIRL) (Fu et al., 2017), combinadas com o Reforço Profundo são pouco exploradas.

Sabendo que o Aprendizado por Imitação possibilita que comportamentos sejam aprendidos de forma mais rápida através da observação de exemplos, criando, assim, um

repertório comportamental sem a necessidade da interação direta com o ambiente exigida na aprendizagem por reforço, a contribuição dessa pesquisa se dá ao buscar combinar estratégias de imitação adversariais e comportamentais à de reforço, com a hipótese de que o agente já começará sua interação com o ambiente tendo algum conhecimento de domínio prévio, facilitando o processo de busca e permitindo que o agente atinja melhores resultados.

1.1 Objetivos

O presente trabalho tem como objetivo geral desenvolver modelos de Aprendizado por Reforço e Aprendizado por Imitação para controle de jogos digitais. Como objetivos específicos, pretende-se:

- Analisar o desempenho do uso do Aprendizado por Imitação para inicialização dos modelos de Aprendizado por Reforço no jogo *Sonic The Hedgehog*;
- Explorar o desempenho de modelos de imitação baseados em estratégias adversariais;
- Investigar o impacto da imitação no comportamento e desempenho do modelo;
- Verificar se o Aprendizado por Imitação fornece uma alternativa viável à criação de funções de recompensa.
- Disponibilizar dados de demonstração para futuros trabalhos.

1.2 Organização do texto

Esse trabalho é composto de 7 capítulos e está organizado da seguinte forma: o capítulo 1 contém a introdução, no capítulo 2, é apresentado o ferramental teórico necessário para entendimento desse trabalho; o capítulo 3 apresenta uma revisão da literatura; o capítulo 4 conta com a definição do problema, além de detalhes de funcionamento do emulador. Detalhes da proposta apresentada são encontrados no capítulo 5. A metodologia e discussão sobre os resultados são apresentadas no capítulo 6, enquanto o capítulo 7 contém as conclusões, discussões finais e a proposta de trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são apresentados os conceitos, métodos e modelos utilizados como base para esse trabalho. O objetivo é familiarizar o leitor com toda a base teórica necessária para o entendimento desta pesquisa.

2.1 Redes Neurais Artificiais

Rede neural artificial (RNA), ou em inglês *Artificial Neural Network (ANN)*, é uma técnica de Inteligência Artificial inspirada no funcionamento do cérebro. Uma RNA é constituída da interligação de neurônios artificiais, que permite mapear valores de entrada em valores de saída (Abdi et al., 1999).

Biologicamente, uma rede neuronal é a interligação de diversos neurônios que interagem entre si através da passagem de sinais elétricos dos seus axônios (terminais transmissores) para os dendritos (terminais receptores) do próximo neurônio. Essa passagem do impulso elétrico entre axônios e dendritos é chamada de sinapse, e ocorre quando a combinação dos impulsos de entrada atingem um nível mínimo, estado chamado de potencial de ação ou potencial de ativação (Moreira, 2013). Na Figura 1, pode-se ver a representação de um neurônio biológico.

No ano de 1943, o matemático Walter Pitts e o psiquiatra Warren McCulloch publicaram o artigo (McCulloch and Pitts, 1943). Esse trabalho, considerado o ponto de partida das redes neurais artificiais, demonstra como é possível modelar o funcionamento neuronal através de circuitos digitais. Anos mais tarde, o psicólogo Frank Rosenblatt

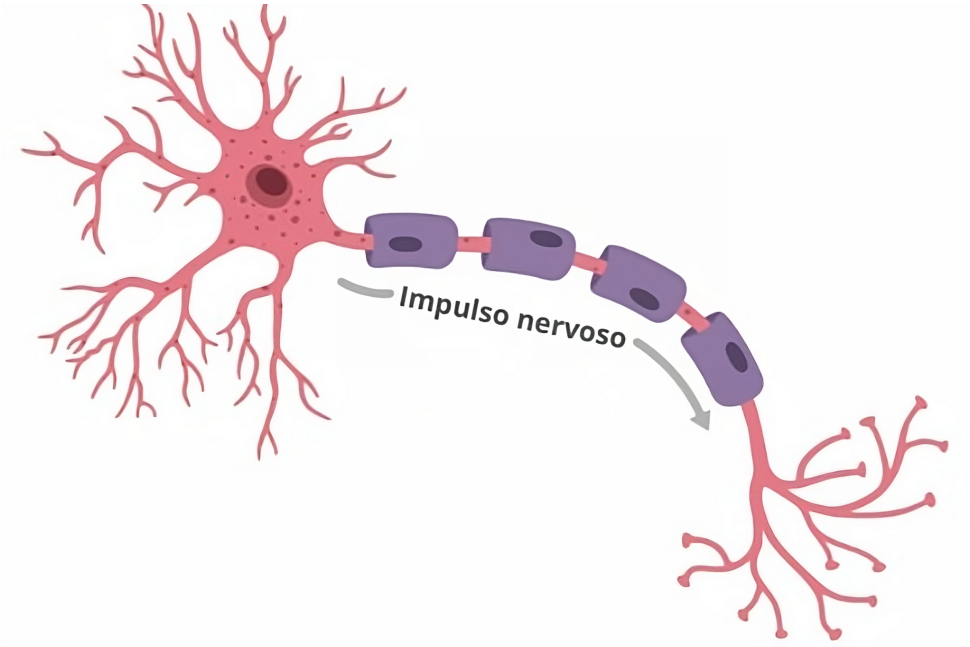


Figura 1 – Figura retirada de (Franco, 2020)

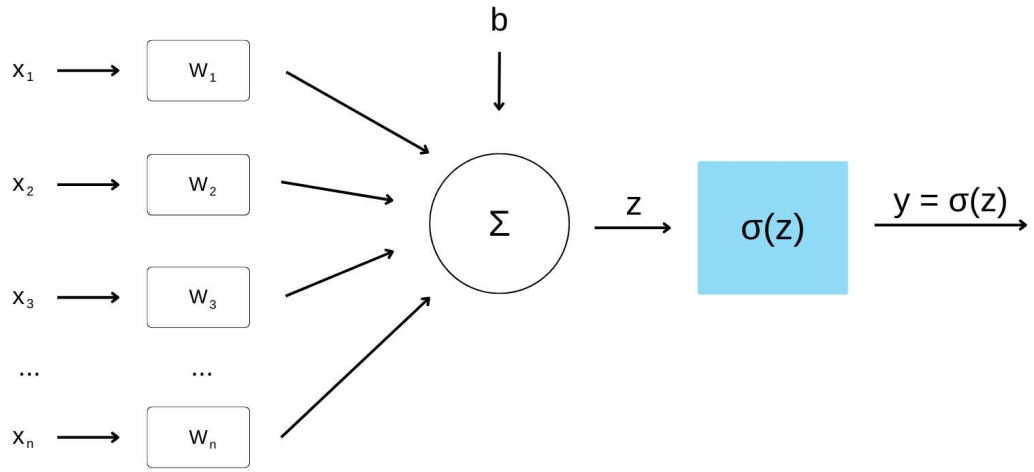


Figura 2 – Representação do modelo Percetron de neurônio artificial.

criou um modelo de rede neural artificial chamado Perceptron (Rosenblatt, 1958). O modelo consiste em neurônios artificiais que recebem entradas e produzem uma saída ponderada através de uma função de ativação, onde, se o valor da entrada ultrapassa um certo limiar de ativação, essa saída é verdadeira, e, caso o valor da entrada não seja suficiente para ultrapassar o limiar de ativação, essa função retornará falsa. O modelo é treinado utilizando um algoritmo de otimização que ajusta os pesos das conexões entre as entradas e a saída de forma a minimizar o erro em relação aos dados apresentados. A Figura 2 mostra a representação matemática do modelo.

- $X = x_1, x_2, x_3, ..., x_n$: vetor contendo as entradas da rede;
- $W = w_1, w_2, w_3, ..., w_n$: vetor contendo os pesos da rede, geralmente iniciado com valores aleatórios;
- b: chamado de viés, é um valor real adicionado ao somatório do vetor de entrada X ponderados pelo vetor de pesos W. Esse valor permite deslocar a função de ativação no eixo y, aumentando ou diminuindo o limiar para sua ativação;
- Σ: Combinador linear que agrega os sinais de entrada ponderados pelos pesos, produzindo um valor z, chamado de potencial de ativação;
- $z = \sum_{i=1}^{n} x_i \times w_i + b$, chamado de potencial de ativação;
- $\sigma(z) = \begin{cases} 1, & z >= 0 \\ 0, & z < 0 \end{cases}$, chamado de função de ativação; e
- y: saída binária do neurônio.

2.1.1 Perceptron Multi Camadas

O Perceptron Multi Camadas (em inglês, *Multilayer Percetron (MLP)*) é um modelo de Aprendizado Supervisionado proposto por Werbos em (Werbos, 1974). Esse modelo consiste em combinar uma ou mais camadas internas, de modo que a saída dos neurônios de uma camada alimente a próxima camada.

Diferente do Perceptron tradicional, que possui apenas uma saída binária, o MLP pode possuir diversas saídas, e é capaz de modelar problemas não lineares, podendo ser utilizado para a resolução de uma extensa gama de problemas, tais como: reconhecimento de padrões, processamento de imagens, previsão de séries temporais e aprendizado por reforço.

A arquitetura do MLP é composta por três tipos de camadas: a camada de entrada; uma ou mais camadas ocultas, e camada de saída, que pode conter um ou mais neurônios de saída. A camada de entrada recebe os dados de entrada e os repassa para a primeira camada oculta. Cada neurônio na camada oculta processa a informação recebida e a envia para a próxima camada oculta ou para a camada de saída (no caso da última camada oculta). A camada de saída, por sua vez, produz a saída final do modelo. A Figura 3 ilustra um exemplo de arquitetura MLP.

O MLP é considerado uma rede neural *feedforward*, porque a informação se move apenas em uma direção, da camada de entrada para a camada de saída (Rosenblatt, 1958). Nesse tipo de rede, cada camada de neurônios se conecta exclusivamente à camada seguinte, permitindo que a informação flua unidirecionalmente através das camadas. Os neurônios de cada camada recebem as saídas dos neurônios da camada anterior e processam a informação, repassando o resultado para os neurônios da camada seguinte, até chegar a camada de saída, onde a resposta do modelo é gerada.

2.1.2 Funções de ativação

A função de ativação compõe parte essencial do neurônio artificial e, por sua vez, de toda a rede neural. A função de ativação consiste em uma transformação numérica aplicada à combinação dos sinais de entrada, já ponderada pelos respectivos pesos sinápticos do neurônio, podendo ser linear ou não linear. A função de ativação linear possui aplicação restrita, limitando-se ao tratamento de problemas lineares ou problemas onde a saída do neurônio deve ser proporcional à entrada. Nesse último caso, essa função é geralmente utilizada na camada de saída. Em geral, as funções não lineares podem ser utilizadas para modelar uma extensa variedade de problemas, já que sua não linearidade permite encontrar padrões complexos dos dados de entrada (Goodfellow et al., 2016).

A escolha da função de ativação de uma rede neural artificial está diretamente ligada ao problema que será tratado, podendo ter grande influência no desempenho final do modelo (Nwankpa et al., 2018). A Figura 4 apresenta algumas das funções de ativações

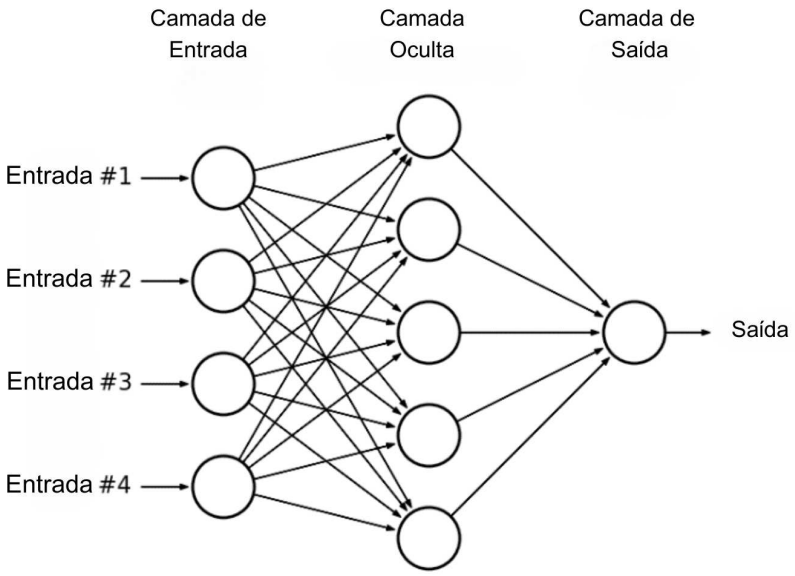


Figura 3 – Ilustração de uma arquitetura MLP em que a camada de entrada é composta por quatro neurônios. Há uma camada oculta, com cinco neurônios, e a camada de saída possui um neurônio. Figura adaptada de (Wikipedia, the free encyclopedia, 2017)

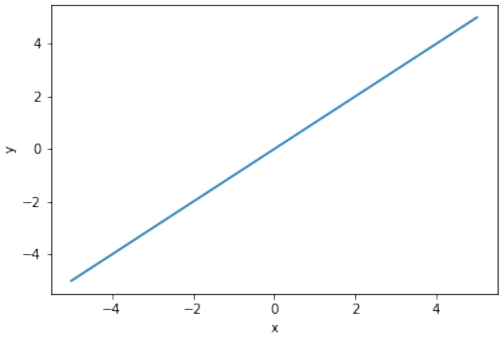
mais conhecidas e suas definições matemáticas.

2.1.3 Redes Neurais Profundas

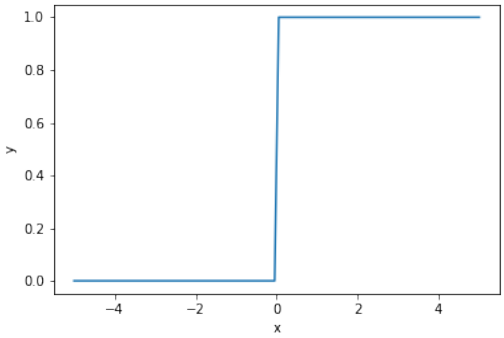
O termo Redes Neurais Profundas (*Deep Neural Network, DNN*) é usado para designar redes neurais com muitas camadas ocultas, em alguns casos chegando à centenas ou milhares de camadas. O grande número de camadas permite que as redes neurais profundas sejam capazes de aprender relações extremamente complexas. Nos últimos anos, o uso de redes profundas possibilitou diversos avanços nos mais diferentes campos, como em visão computacional, processamento de linguagem natural e reconhecimento de fala. Um dos primeiros trabalhos a descrever o uso de múltiplas camadas foi o de Ivakhnenko and Lapa (1966), no qual utilizaram um modelo com três camadas internas e uma otimização através de mínimos quadrados para cada camada. Porém, foi após o trabalho de Rumelhart et al. (1986) que se introduziu o uso do *backpropagation* para atualização dos pesos das redes neurais e que a utilização de redes de múltiplas camadas ganhou força.

Diversos trabalhos demostraram a eficiência da técnica de retro propagação para o aprendizado de relações complexas nos dados em diversos problemas, dentre eles a identificação de dígitos escrito à mão (LeCun et al., 1989) e (Martin and Pittman, 1991).

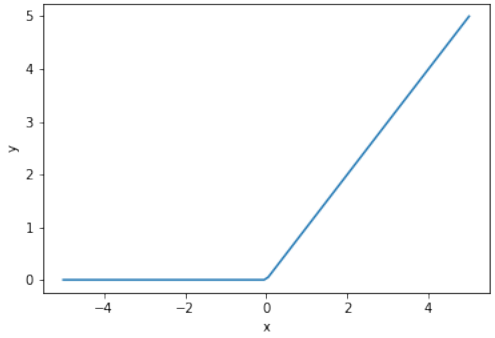
Figura 4 – Funções de ativação



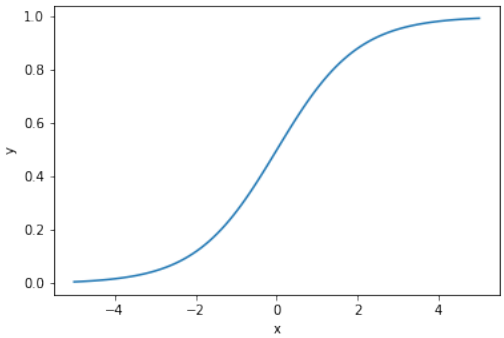
(a) $linear(z) = \{z, \quad \forall\, z \,\epsilon\, \mathbb{R}$



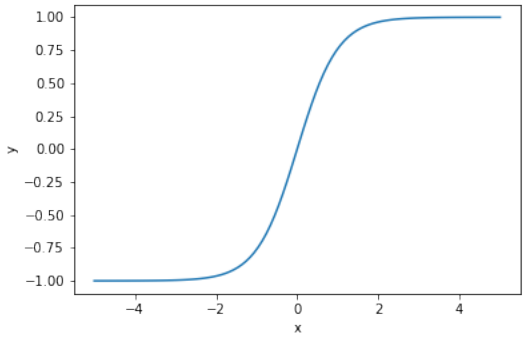
(b) $degrau(z) = \begin{cases} 1, & z >= 0 \\ 0, & z < 0 \end{cases}$



(c) $ReLu(z) = \begin{cases} z, & z >= 0 \\ 0, & z < 0 \end{cases}$



(d) $sigmoide(z) = \frac{1}{1+e^{-z}}$



(e) $\tanh(z) = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$

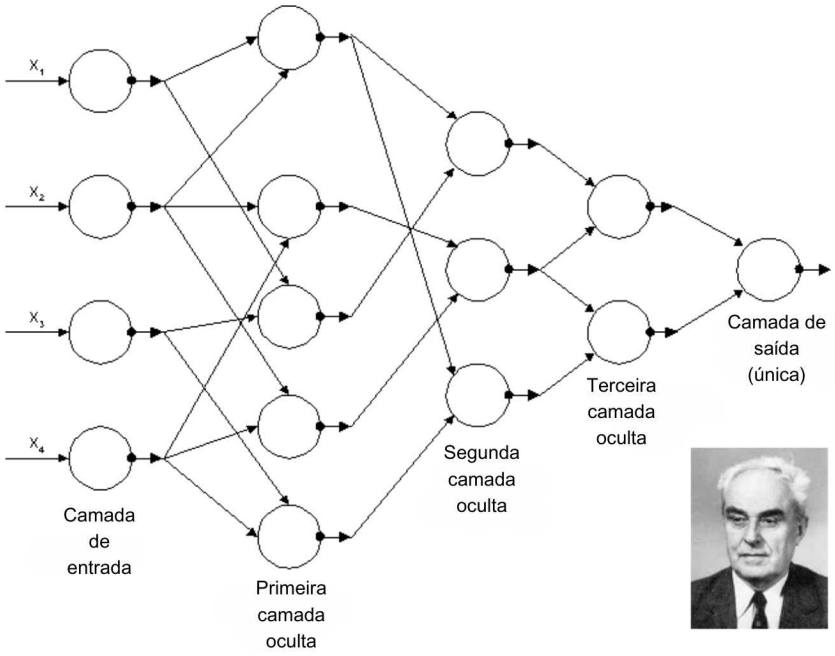


Figura 5 – Arquitetura da primeira rede neural profunda, proposta e treinada por Ivakhnenko and Lapa (1966). Figura adaptada de (Dettmers, 2015).

2.1.4 Redes Neurais Convolucionais

Redes Neurais Convolucionais, do inglês *Convolution Neural Networks* (CNN), é um modelo de Aprendizado de Máquina inspirado no funcionamento do córtex visual humano. As CNNs são projetadas de forma a extrair características de imagens, tais como linhas, bordas e curvas, de forma que a cada camada, padrões mais complexos podem ser identificados. São usualmente utilizadas em problemas de classificação em imagens, mas possuem ampla gama de aplicabilidade em problemas reais, como reconhecimento de voz, processamento de linguagem natural, e auxilio a diagnóstico médico e detecção de objetos.

O cerne do funcionamento das CNNs é a operação de convolução, que pode ser definida matematicamente como o somatório do produto entre duas funções ao longo da região de sobreposição. A operação de convolução entre duas funções $f(x)$ e $g(x)$ pode ocorrer no domínio discreto ou contínuo. A Equação 2.1 define uma operação de convolução entre duas funções contínuas.

$$(f * g)(x) = \text{conv}(x) = \int_{-\infty}^{\infty} f(u) \cdot g(x-u)\mathrm{d}u \tag{2.1}$$

O cálculo da convolução contínua é dado pela integral do produto das duas funções $f(x)$ e $g(x)$, desde que ambas sejam integráveis no intervalo. No cálculo discreto, a integral é substituída pelo somatório, como:

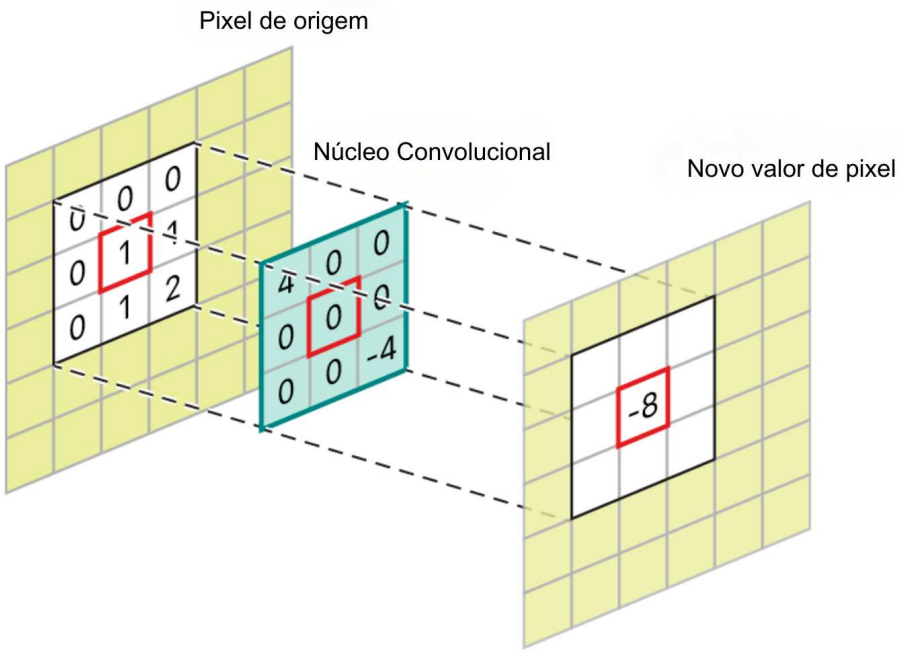


Figura 6 – O elemento central do filtro é posicionado sobre o pixel de origem, onde é aplicado a operação de convolução a seguir: $(4 \times 0 + 0 \times 0 + 0 \times 0) + (0 \times 0 + 0 \times 1 + 0 \times 1) + (0 \times 0 + 0 \times 1 + 2 \times -4) = -8$.

$$(f * g)(x) = \text{conv}(x) = \sum_{u=0}^{x} f(u) \cdot g(x - u)$$

Para utilização em imagens, a convolução descrita anteriormente deve ser adaptada para percorrer as duas dimensões espaciais da imagem (altura e largura), podendo ser descrita matematicamente como:

$$(f * g)(x, y) = \text{conv2d}(x, y) = \sum_{i=0}^{x} \sum_{j=0}^{y} f(i, j) \cdot g(x - i, y - j)$$

A Figura 6 mostra um exemplo da aplicação de uma convolução sobre uma imagem.

No contexto das CNNs, a convolução tem o papel de realizar uma extração de informações relevantes para a imagem. Através do uso de um filtro (também conhecido como *Kernel*), é possível extrair padrões da imagem, como: bordas, linhas, curvas, contrastes, texturas e etc. Para a aplicação de uma convolução sobre uma imagem, o filtro se desloca sobre a imagem, calculando a soma do produto escalar dos elementos correspondentes (O'Shea and Nash, 2015).

Como discutido anteriormente, as principais características das CNNs são as camadas convolucionais, capazes de identificar padrões nas imagens de entrada. Quanto mais profunda a camada de convolução, mais complexos são os padrões que os filtros dessa camada correspondente é capaz de detectar. Um exemplo é que, enquanto em camadas

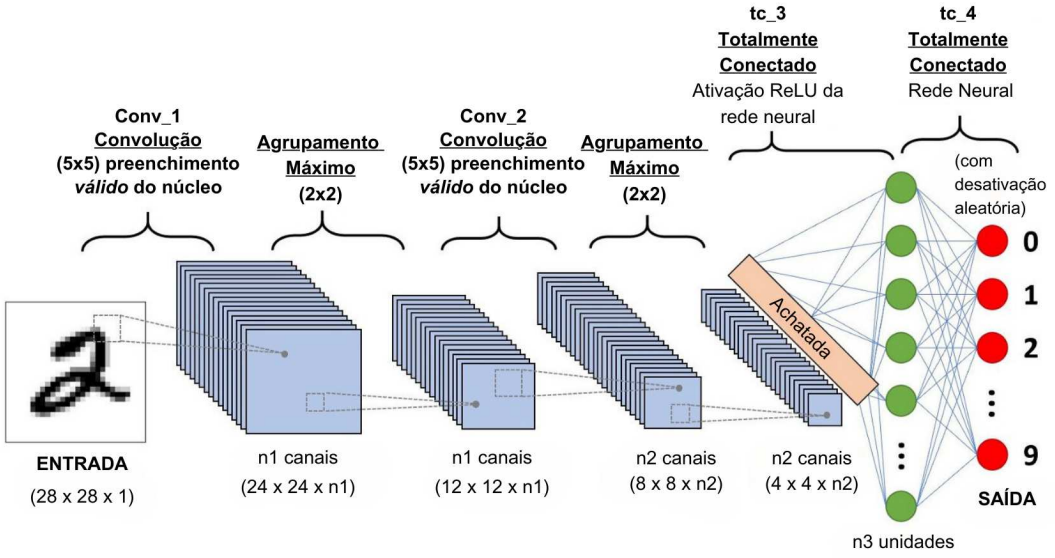


Figura 7 – A Figura mostra um exemplo de arquitetura para uma CNN. Adaptada de (Nadeem et al., 2020)

iniciais, os filtros são capazes de detectar bordas e linhas; em camadas mais profundas, os filtros de convolução são capazes de detectar olhos, boca, cabelos ou objetos complexos.

A arquitetura básica de uma CNN é composta por diferentes tipos de camadas, tais como:

- Camada de entrada: responsável por receber a informação e encaminhar para a próxima camada;
- Camadas de convolução: são encarregadas de realizar a aplicação dos filtros e extração de características da imagem;
- Camada de agrupamento (*polling*): esse tipo de camada reduz a dimensionalidade da entrada, de modo a não perder a informação;
- Camada totalmente conectada: responsável por converter a saída das camadas de convolução e *polling* para realizar a classificação; e
- Camada de Saída: produz a saída da rede.

Dois conceitos importantes para o entendimento do funcionamento das CNNs são os de *stride* e *padding*. O *stride* refere-se ao deslocamento do filtro sobre a imagem. Um *stride* de 1, por exemplo, significa que o filtro se move um pixel por vez durante sua aplicação, cobrindo todas as posições da imagem sem deixar lacunas. Já um *stride* de tamanho 2 significa que o filtro salta dois pixels a cada aplicação, resultado numa imagem

final menor. O *padding*, por sua vez, é uma técnica que consiste em adicionar bordas de pixels à imagem de entrada antes da aplicação da convolução. Essa técnica ajuda a reduzir a perda de informação da borda da imagem, permitindo que o filtro seja aplicado a todos os pixels da imagem original (Dwarampudi and Reddy, 2019).

2.2 Processo de Decisão de Markov

O processo de decisão de Markov, em inglês Markov *Decision Process (MDP)*, é um modelo matemático que descreve processos nos quais as transições entre estados são probabilísticas e os estados futuros dependem apenas do estado atual (Puterman, 1990). Nesse tipo de modelo, é possível observar o estado atual e interferir no processo executando ações (Puterman, 2014). O executor das ações no sistema é chamado de agente. Essas ações podem ter recompensa (ou custo), que depende do estado atual do processo. De forma alternativa, é possível definir a recompensa (ou custo) baseada apenas no estado. O processo de decisão de Markov é baseado no teorema de Markov, que afirma que o futuro de um processo depende apenas do seu estado atual, e não de seu passado (Russell, 2010). Este tipo de Processo é também denominado de (processo sem memória) (*memoryless process*), uma vez que o passado é 'esquecido' (Nogueira, 2008).

A Propriedade de Markov é uma característica fundamental de certos processos estocásticos, e pode ser matematicamente descrita pela Equação 2.2:

$$P(S_{t+1}|S_t) = P(S_{t+1}|S_1, S_2, S_3, ..., S_t) \tag{2.2}$$

Essa equação estabelece que a probabilidade de alcançar o estado S_{t+1}, dado o estado atual S_t, é igual à probabilidade de alcançar o estado S_{t+1}, considerando toda a sequência de estados desde S_1 até S_t. Em outras palavras, os estados S_1 até S_{t-1} não têm influência na probabilidade de atingir S_{t+1}. Portanto, o estado futuro S_{t+1} depende unicamente do estado anterior S_t, sendo independente dos eventos passados.

O processo de decisão de Markov pode ser definido como uma tupla (S,A,T,R) (Pellegrini and Wainer, 2007), onde:

- S é o conjunto de estados que o sistema pode assumir;
- A é o conjunto de ações que podem ser executadas sobre o sistema;
- $T : S \times A \times S \mapsto [0, 1]$ é a função de transição que atribui a probabilidade de uma ação $a \in A$ levar o sistema de um estado $s \in S$ para um estado $s' \in S$. Pode ser denotado como $T(s'|s, a)$;
- $R : S \times A \mapsto \mathbb{R}$ é a função que retorna a recompensa (ou custo) $r \in \mathbb{R}$ dado que fora executada uma ação $a \in A$ no estado $s \in S$.

O Processo de Decisão de Markov pode ser dividido em dois tipos: totalmente observável e parcialmente observável. No caso do Processo de Decisão de Markov Totalmente Observável, o agente tem acesso à informação completa do ambiente, conhecendo todos os possíveis estados, bem como as probabilidades de transição entre eles. Um exemplo de problema que pode ser modelado dessa forma é o jogo de Xadrez, onde todas as informações são conhecidas e é possível identificar todos os estados possíveis do jogo.

Por outro lado, no caso do Processo de Decisão de Markov Parcialmente Observável, o agente não possui informações completas sobre o ambiente, estando limitado a uma informação parcial que pode até ser afetada por ruído ou erros de leitura. Um exemplo de problema que pode ser modelado como um Processo de Decisão de Markov Parcialmente Observável é um jogo de cartas, no qual o agente só tem acesso às informações das cartas que estão na mesa e das cartas em sua própria mão.

2.2.1 Política para Processo de Decisão de Markov

A cada passo de tempo (ou época de decisão), o agente usa uma regra de decisão para escolher a próxima ação. Uma forma simples de regra de decisão é um mapeamento direto de estados em ações. O conjunto de todas as regras de decisão (uma para cada passo de tempo) é chamado de política e é representado pela letra grega π.

Ao executar uma política, o agente receberá recompensas a cada passo de tempo. Para comparar duas políticas, é necessário adotar um critério de desempenho (Pellegrini and Wainer, 2007). É possível definir diversos critérios de desempenho para MDPs, e entre os mais conhecidos pode-se citar:

- Recompensa média por época de decisão: $\frac{1}{z}\sum_{t=0}^{z-1} r_t$;
- recompensa esperada total: $E\left[\sum_{t=0}^{z-1} r_t\right]$; e
- recompensa esperada descontada: $E\left[\sum_{t=0}^{z-1} \gamma^t r_t\right]$;

onde r_t é a recompensa no instante de tempo t, γ é chamado fator de desconto; sendo $\gamma \in]0,1[$, e z é chamado de horizonte. O horizonte de um MDP é o número de passos de tempo (ou épocas de decisão) disponíveis para a tomada de decisões. Existem três tipos de horizontes: finito (quando existe um número fixo de decisões); infinito (quando a tomada de decisão é repetida continuamente, sem a possibilidade de parada), e indefinido (análogo ao horizonte infinito, porém com a possibilidade do processo parar caso chegue em algum estado que tenha sido marcado como estado terminal).

2.2.2 Definição: Política para um MDP

Uma regra de decisão para um MDP em um passo de tempo t é uma função $d_t : S \mapsto A$ que determina a ação a ser executada dado o estado do sistema. Uma política

para um MDP é uma sequência de regras de decisão $\pi = \{d_0, d_1, d_2, ..., d_{z-1}\}$, uma para cada passo de tempo.

A função valor de uma política π para um MDP $M = (S, A, T, R)$ é uma função $V^\pi : S \mapsto \mathbb{R}$, tal que $V^\pi(s)$ dá o valor esperado da recompensa para esta política de acordo com o critério de otimalidade. Por exemplo, para horizonte finito com desconto seja $\pi = \{d_0, d_1, d_2, ..., d_{z-1}\}$, esta função pode ser definida como:

$$V_t^\pi(s) = R(s, d_t(s)) + \gamma \sum_{s' \in S} T(s, d_t(s), s') V_{t+1}^\pi(s') \tag{2.3}$$

onde $V^\pi(s) = 0$ para todas ações após o final do horizonte (porque após o final do horizonte não há mais ações que possam gerar recompensas).

2.3 Aprendizado por Reforço

No início do século XX, o fisiologista russo Ivan Pavlov estudava o processo digestivo de cães (Pavlov, 1897). Para isso, Pavlov realizava as medições da quantidade de saliva produzida por esses cães quando expostos a estímulos alimentares. Com o tempo, Ivan percebeu que os cães, que antes só começavam a salivar quando viam o estimulo alimentar (estímulos incondicionais), começaram a salivar antes, com outros estímulos do processo de experimentação (estímulos condicionais), como o som das tigelas sendo preparadas ou os passos do assistente pelo laboratório. Essas observações levaram então à enunciação do que hoje é conhecido como Condicionamento Clássico. As pesquisas de Pavlov serviram de pano de fundo para as teorias de aprendizado modernas, pavimentando o caminho para os estudos do psicólogo estadunidense Frederic Skinner, cuja maior contribuição foi publicada em (Ferster and Skinner, 1957), que traz análises de experimentações realizadas com o que ele chamava de 'Câmara de Condicionamento Operante', hoje conhecida como *Caixa de Skinner*.

No Condicionamento Clássico de Pavlov, o estímulo gera uma resposta no indivíduo. Já no Comportamento Operante de Skinner, as respostas do indivíduo geram uma consequência que tem chances de se repetir em um cenário semelhante, modificada pelo resultado dessa consequência. De modo geral, ações com consequências positivas tendem a serem repetidas, e ações com consequências negativas tendem a ser evitadas.

O Aprendizado por Reforço é um ramo do Aprendizado de Máquina inspirado na Psicologia Comportamental, que tem como base a premissa de que comportamentos podem ser aprendidos por meio de condicionamento, utilizando de reforços e punições (Kaelbling et al., 1996). Em Ciência da Computação, o Aprendizado por Reforço visa a geração de modelos para controle de agentes, utilizando a interação com o ambiente para treinar uma politica de controle que mapeie estados em ações, de forma a maximizar uma função de recompensa. O Aprendizado por Reforço tem sido amplamente utilizado

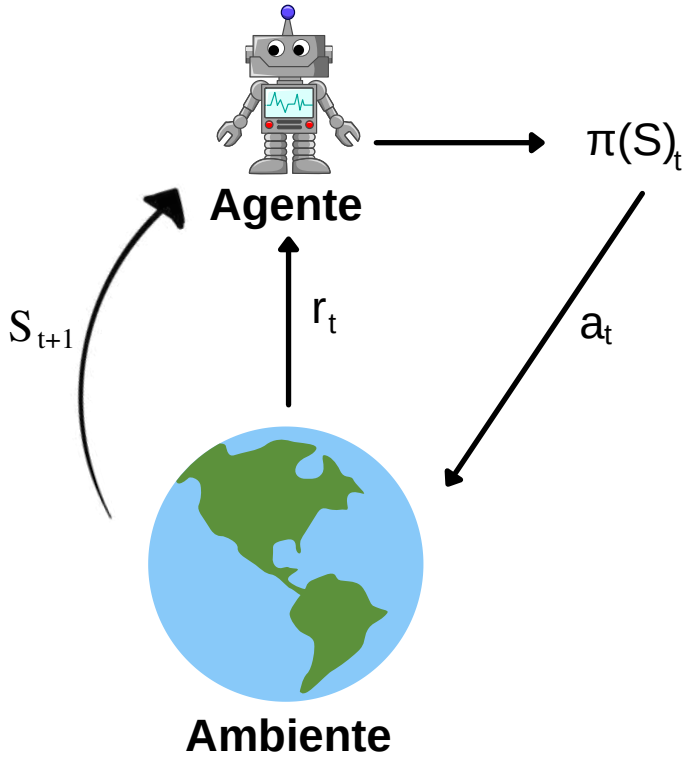


Figura 8 – Ilustração do Agente, representado pelo o robô, executando uma ação no ambiente (planeta) utilizando de uma política π e, por sua vez, o ambiente retorna uma recompensa r_t para a ação realizada no tempo t e um novo estado S_{t+1}.

para resolver uma série de problemas, tais como: controle de agentes robóticos (Johannink et al., 2019), controladores de jogos (Lample and Chaplot, 2017), carros autônomos (Aradi, 2020), telecomunicações (Peshkin and Savova, 2002) e etc.

2.3.1 Função de Recompensa

No Aprendizado por Reforço, a função de recompensa é responsável por guiar o aprendizado do agente, fornecendo um parecer a respeito das ações tomadas por ele (se são boas ou ruins). O treinamento de um modelo de Aprendizado por Reforço se dá pela geração de uma política de decisão que maximize a recompensa recebida pelo agente. Uma função de recompensa mal definida pode levar à não convergência do treinamento ou ao aprendizado de comportamentos indesejados. A definição de uma boa função de recompensa é primordial para a eficácia do aprendizado.

Em muitos problemas reais, é geralmente difícil produzir manualmente uma função de recompensa, como no caso da direção autônoma de veículos (Kiran et al., 2021), controle de robôs (Johannink et al., 2019) e sistemas de recomendações (Afsar et al., 2022). Em situações como essas, técnicas de Aprendizado por Reforço Inverso podem se tornar uma solução viável, visto que conseguem aprender a função de recompensa com base em demonstrações do comportamento desejado.

2.3.2 Gradiente de uma Política

O gradiente de uma política é utilizado para atualizar a política de decisão do Aprendizado por Reforço na direção da política ótima. A definição matemática do gradiente da política leva em consideração a hipótese da recompensa (Bowling et al., 2022), a qual postula que o Aprendizado por Reforço pode ser entendido como a maximização do valor esperado da soma cumulativa de um sinal escalar recebido, denominado recompensa (Sutton and Barto, 2018).

Seja τ, chamado de trajetória, a sequência de estados, ações e recompensas:

$$\tau = s_1, a_1, r_1, s_1, a_1, r_1, ...s_T, a_T, r_T$$

Visto que o objetivo do Aprendizado por Reforço é maximizar a recompensa esperada utilizando de uma política π, podemos representar essa função objetivo como:

$$J(\theta) = \mathbb{E}_{\pi_\theta}\left[r(\tau)\right] = \int \pi_\theta(\tau) r(\tau) \mathrm{d}\tau$$

Onde $r(\tau)$ representa a recompensa total em uma trajetória τ, θ representa os parâmetros da política, e $\pi_\theta(\tau)$ denota a probabilidade da política π seguir a trajetória τ com os parâmetros θ. Dessa forma, o objetivo do problema de aprendizado é encontrar o conjunto de parâmetros θ' que maximize a função J. Supondo que J seja uma função derivável, é possível atualizar os parâmetros θ na direção do maior deslocamento da função, utilizando o gradiente ascendente (Ruder, 2016). Assim, a atualização do conjunto de parâmetros θ pode ser descrita como segue:

$$\theta_{t+1} = \theta_t + \alpha \nabla J(\theta_t) \tag{2.4}$$

onde α é chamado de tamanho do passo ou taxa a aprendizado. Ao derivar $J(\theta)$, obtemos:

$$\nabla_\theta J(\theta) = \nabla_\theta \int \pi_\theta(\tau) r(\tau) d\tau$$

$$= \int \nabla_\theta \pi_\theta(\tau) r(\tau) d\tau$$

$$= \int \pi_\theta(\tau) \frac{\nabla_\theta \pi_\theta(\tau)}{\pi_\theta(\tau)} r(\tau) d\tau$$

$$\nabla_\theta J(\theta) = \nabla \mathbb{E}_{\pi_\theta}\left[r(\tau)\right] = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(\tau) r(\tau)\right]$$

Esse resultado demonstra que a derivada da recompensa esperada é igual à esperança do produto entre a recompensa e o gradiente do logaritmo da política π_θ.

Reescrevendo $\pi_\theta(\tau)$ como o produto das probabilidades de se começar em um estado arbitrário s_0 e, em seguida, multiplicando as probabilidades subsequentes, utilizando a regra do produto, uma vez que é assumida a independência entre as ações, temos:

$$\pi_\theta(\tau) = P(S_0) \prod_{t=1}^{T} \pi_\theta(a_t \mid s_t) p(s_{t+1}, r_{t+1} \mid s_t, a_t)$$

Em cada passo de tempo t, é utilizada a política π_θ para selecionar uma ação. Assim, a dinâmica do ambiente resulta em um novo estado e na recompensa associada à ação. Esse processo se repete por um número arbitrário de passos de tempo, denominado T. Tomando o logaritmo da equação acima e aplicando as propriedades do produto, chega-se a:

$$\log \pi_\theta(\tau) = \log P(S_0) + \sum_{t=1}^{T} \log \pi_\theta(a_t \mid s_t) + \sum_{t=1}^{T} \log p(s_{t+1}, r_{t+1} \mid s_t, a_t)$$

tomando o gradiente:

$$\nabla \log \pi_\theta(\tau) = \sum_{t=1}^{T} \nabla \log \pi_\theta(a_t \mid s_t)$$

$$\nabla_\theta J(\theta) = \nabla \mathbb{E}_{\pi_\theta} [r(\tau)] = \mathbb{E}_{\pi_\theta} \left[r(\tau) \left(\sum_{t=1}^{T} \nabla \log \pi_\theta(a_t \mid s_t) \right) \right] \tag{2.5}$$

Esse resultado demonstra que é possível representar o gradiente sem necessitar das informações sobre a dinâmica do ambiente ou o estado inicial da distribuição.

2.3.3 Método Ator-Crítico

O método Ator-Crítico, em inglês *Actor-Critic* (AC), representa uma classe de políticas de Aprendizado por Reforço que combina dois componentes principais: o ator e o crítico. O ator é responsável por definir a ação que o agente tomará dado um estado, enquanto o crítico é responsável por avaliar a qualidade das decisões tomadas pelo ator (Konda and Tsitsiklis, 1999).

Geralmente, o método Ator-Crítico é implementado utilizando uma rede neural para aproximar a função de cada um dos componentes: ator e crítico (Grondman et al., 2012). A rede neural do crítico fica encarregada de estimar uma função que mapeia a ação tomada para uma estimativa de recompensa esperada a longo prazo. Essa função é conhecida como função valor da ação (q-valor). Já a rede neural do ator fica responsável por mapear os estados em ações, fornecendo a política de controle do agente. Descrevemos matematicamente as funções aproximadas como:

- **Ator**, a função da política de controle do agente parametrizada por θ: $\pi_\theta(s, a)$
- **Critico**, função valor parametrizada por w: $q_w(s, a)$

As funções do ator e critico são otimizadas durante o treinamento da seguinte forma:

1. Para cada passo de tempo, t, o estado atual do ambiente s_t é passado como entrada para o ator e crítico;
2. A política atual realiza uma ação a_t com base no estado s_t;
3. O crítico utiliza o estado s_t e a ação a_t para calcular o q-valor, $q_w(s_t, a_t)$, da ação a_t no estado s_t
4. A ação a_t leva o ambiente para um novo estado s_{t+1} com uma recompensa r_{t+1} associada à transição;
5. O ator atualiza os parâmetros θ da política com base no q-valor. Utilizando os resultado das Equações 2.4 e 2.5 e substituindo a função r(τ) pela função valor $q_w(s, a)$, temos a atualização dos parâmetros da política do ator matematicamente descrita como:
$$\Delta\theta = \alpha\nabla_\theta \left(\log \pi_\theta(s, a)\right) q_w(s, a);$$
6. Utilizando a nova política o ator toma uma nova ação A_{t+1} que leva a um novo estado S_{t+1}; e
7. O crítico atualiza os parâmetros da seguinte forma:
$$\Delta w = \beta \left(R(s, a) + q_w(s_{t+1}, a_{t+1}) - q_w(s_t, a_t)\right) \nabla_w q_w(s_t, a_t)$$
Onde $R(s, a)$ é a recompensa R_{t+1} obtida e o termo entre parênteses é conhecido como erro de diferença temporal.

O processo é repetido até que um critério de parada, tal como o tamanho máximo do episódio ou a ocorrência de um estado terminal, seja atendido. A Figura 9 apresenta um diagrama do funcionamento da política Ator-Crítico.

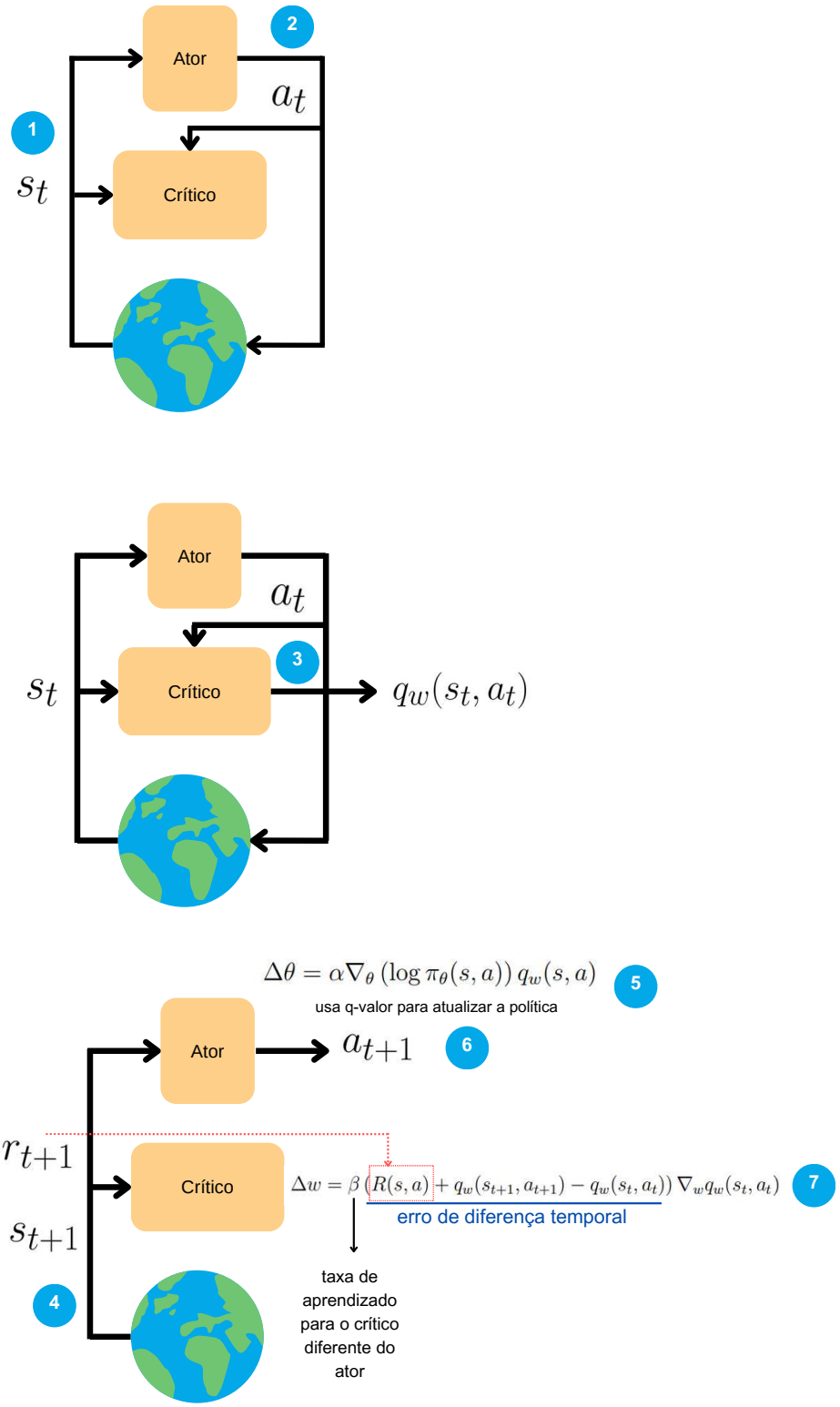


Figura 9 – Diagrama do processo de aprendizado por reforço utilizando o método Ator-Crítico.

2.3.4 Otimização de Política Proximal

A otimização de política proximal, também conhecida como *Proximal Policy Optimization* (PPO), é um algoritmo de Aprendizado por Reforço proposto por Schulman et al. (2017). Ele é baseado na ideia do algoritmo *Trust Region Policy Optimization* (TRPO) (Schulman et al., 2015). O PPO tem sido amplamente utilizado em diversos tipos de problemas, como robótica (Dang et al., 2022), jogos (Kaiser et al., 2019) e mercado de ações (Yang et al., 2020).

O PPO utiliza uma abordagem baseada em gradiente para a otimização da política. Ele trabalha de maneira iterativa, de forma que, em cada iteração, a política atual é comparada com uma nova política, buscando maximizar a recompensa recebida. A estratégia conta com uma função chamada *clipping*, que é responsável por permitir uma variação máxima na atualização da política. Dessa forma, a nova política não será muito diferente da anterior, evitando mudanças abruptas de comportamento do agente. O uso da função de *clipping* garante a estabilidade do PPO e leva a uma convergência mais rápida da política treinada (Hsu et al., 2020).

As estratégias de otimização por gradiente conduzem a política em direção à maximização da recompensa. No entanto, essas estratégias frequentemente enfrentam um desafio relacionado ao tamanho do passo. Se o passo em direção ao gradiente for muito pequeno, o processo de convergência pode se tornar excessivamente lento. Por outro lado, se o passo for muito grande, isso pode comprometer a estabilidade do processo, resultando em dificuldades para alcançar a convergência.

Seja $L(\theta)$ a função objetivo da política, descrita como:

$$L(\theta) = \mathbb{E}_t \left[\log \pi_\theta(a_t \mid s_t) A_t \right] \tag{2.6}$$

onde A é a função de vantagem, que descreve o quão melhor (ou pior) é realizar uma ação em um determinado estado em comparação com a média do valor desse estado. Dessa forma, o PPO propõe uma modificação na função objetivo para evitar atualizações prejudiciais ao processo de aprendizado. A função objetivo modificada é descrita matematicamente da seguinte forma:

$$L^{CLIP}(\theta) = \mathbb{E}_t \left[\min \left(r_t(\theta) A_t, clip \left(r_t(\theta), 1 - \varepsilon, 1 + \varepsilon \right) A_t \right) \right]$$

$r_t(\theta)$ é chamado de função de razão (*ratio function*) e é calculada da seguinte forma:

$$r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{antigo}}(a_t \mid s_t)}$$

onde $\pi_\theta(a_t \mid s_t)$ é a probabilidade de realizar a ação a_t no estado s_t utilizando a política atual, e $\pi_{\theta_{antigo}}(a_t \mid s_t)$ é a probabilidade de realizar a ação a_t no estado s_t utilizando a política antiga. Logo, $r_t(\theta)$ representa a taxa entre as políticas atual e antiga:

- Se $r_t(\theta) > 1$, a ação a_t no estado s_t é **mais** provável na política atual do que na antiga.
- Se $0 \geq r_t(\theta) \leq 1$, a ação a_t é **menos** provável de ocorrer no estado s_t na política atual do que na antiga.

A função razão pode substituir o log da probabilidade na Equação 2.6, gerando:

$$L(\theta)' = \mathbb{E}_t \left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{antigo}}(a_t \mid s_t)} A_t \right] = \mathbb{E}_t \left[r_t(\theta) A_t \right]$$

No entanto, se a ação tomada for muito mais provável na política atual do que na antiga, o gradiente de política iria apresentar um valor alto, gerando, então, uma atualização excessiva de política. Desse modo, é desejado manter a razão entre a política atual e antiga dentro de uma margem estabelecida. Para isso, é tomado o mínimo entre a função $r_t(\theta)A_t$ e a função $clip\left(r_t(\theta), 1-\varepsilon, 1+\varepsilon\right) A_t$, representada por:

$$\min\left(r_t(\theta)A_t, clip\left(r_t(\theta), 1-\varepsilon, 1+\varepsilon\right) A_t\right)$$

de modo que o objetivo final seja um limite inferior do objetivo não recortado. (Schulman et al., 2017) utiliza $\varepsilon = 0.2$, mantendo a razão entre 0.8 e 1.2). A Tabela 1 e a Figura 10 mostram os casos de aplicação da função de *clipping* do PPO, em que tem-se:

- Casos 1 e 2 (razão está dentro dos limites de $1-\varepsilon$ e $1+\varepsilon$, e, desse modo, a *clipping* não é aplicado):
  - Caso 1, vantagem A positiva: a ação é melhor que a média de todas as ações naquele estado. Portanto, é desejado aumentar a probabilidade da política atual de realizar essa ação nesse estado.
  - Caso 2, vantagem A negativa: a ação é pior que a média de todas as ações naquele estado. Portanto, é desejado diminuir a probabilidade da política atual de realizar essa ação nesse estado.
- Casos 3 e 4 (razão está abaixo de $1-\varepsilon$, ou seja, a probabilidade de tomar essa ação nesse estado é muito menor na política atual do que na antiga):
  - Caso 3, vantagem A positiva: é desejado aumentar a probabilidade de realizar essa ação nesse estado.
  - Caso 4, vantagem A negativa: não é desejado diminuir ainda mais a probabilidade de realizar essa ação nesse estado. Desse modo, o gradiente é zero (linha reta). Logo, as probabilidades de tomar a ação nesse estado não são alteradas (não são alterados os pesos da rede neural).

Tabela 1 – Adaptado de (Bick and Wiering, 2021)

	$r_t(\theta) > 0$	A_t	**Valor** min **retornado**	**Objetivo foi cortado?**	**Sinal do objetivo**	**Gradiente**
1	$r_t(\theta) \in [1-\varepsilon, 1+\varepsilon]$	+	$r_t(\theta)A_t$	não	+	✓
2	$r_t(\theta) \in [1-\varepsilon, 1+\varepsilon]$	-	$r_t(\theta)A_t$	não	-	✓
3	$r_t(\theta) < 1-\varepsilon$	+	$r_t(\theta)A_t$	não	+	✓
4	$r_t(\theta) < 1-\varepsilon$	-	$(1-\varepsilon)\,A_t$	sim	-	0
5	$r_t(\theta) > 1+\varepsilon$	+	$(1+\varepsilon)\,A_t$	sim	+	0
6	$r_t(\theta) > 1+\varepsilon$	-	$r_t(\theta)A_t$	não	-	✓

- Casos 5 e 6 (razão está acima do limite $1+\varepsilon$, ou seja, a probabilidade de realizar essa ação nesse estado é muito maior na política atual do que na antiga):
  - Caso 5, vantagem A positiva: a probabilidade de realizar a ação naquele estado já é maior na política atual do que na anterior. Para não ser ganancioso demais, o gradiente é zero e os pesos não são atualizados.
  - Caso 6, vantagem A negativa: nesse caso é desejado diminuir a probabilidade de realizar a ação naquele estado.

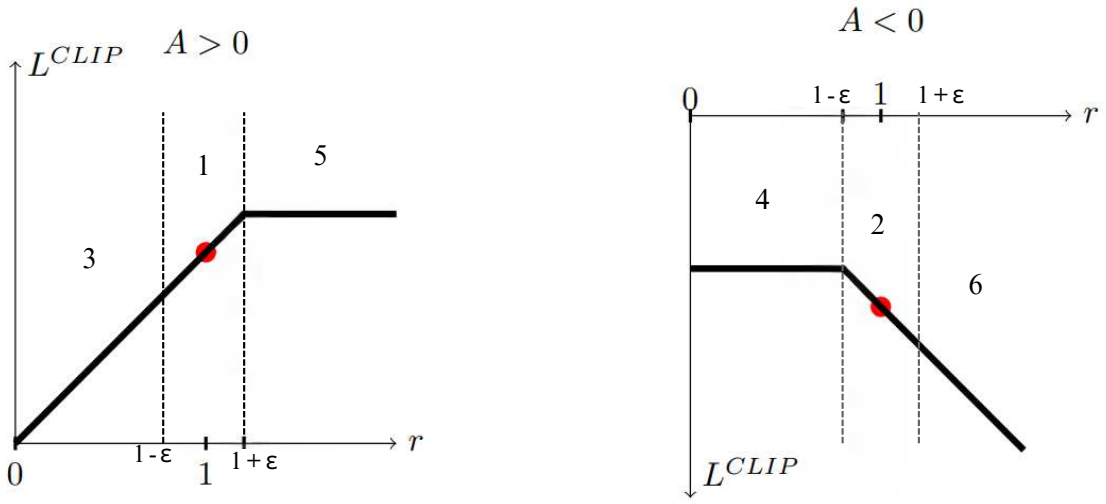


Figura 10 – Representação da função de corte do PPO,

2.4 Aprendizado por Reforço Inverso

Diferente do Aprendizado por Reforço, que busca encontrar uma política de decisão que maximize a função de recompensa, o Aprendizado por Reforço Inverso busca encontrar uma função de recompensa com base em uma política ou em observações do comportamento desejado. O Aprendizado por Reforço Inverso, também conhecido como

Inverse Reinforcement Learning (IRL) (Ng et al., 2000), é especialmente útil em casos em que é difícil definir uma função de recompensa manualmente (Zhifei and Meng Joo, 2012). Trabalhos recentes mostram que o uso do IRL pode melhorar a generalização e robustez do modelo, resultando em um agente mais adaptável às mudanças no ambiente (Arora and Doshi, 2021).

No IRL, a política π é aproximada por um conjunto D contendo $n \in \mathbb{N}$ trajetórias τ do especialista. A função de recompensa pode ser expressa como uma aproximação linear $R(s) = \alpha_1\phi_1(s) + \alpha_2\phi_2(s) + ... + \alpha_n\phi_n(s)$, onde $\phi(s)$ são funções que mapeiam o estado em um valor real, e os parâmetros α são os coeficientes que se deseja ajustar. O processo de Aprendizado por Reforço Inverso geralmente segue os seguintes passos:

- Coleta de dados: nessa etapa, as demonstrações de trajetórias do especialista, que consistem em pares de observações e ações, são coletadas e utilizadas como dados de treinamento;
- Inferência de recompensas: com base nos dados coletados, o processo de IRL busca inferir as recompensas. O objetivo é encontrar uma função de recompensa que melhor explique o comportamento observado do especialista nas trajetórias de demonstração;
- Treinamento do agente: uma vez que as recompensas tenham sido inferidas, o próximo passo é treinar um agente para aprender a melhor política com base na função de recompensa aproximada. Isso é realizado por meio de algoritmos de aprendizado por reforço tradicionais; e
- Avaliação: após o treinamento, o desempenho do agente é avaliado. Geralmente, o desempenho do agente é comparado diretamente com o desempenho das demonstrações do especialista. No entanto, outras abordagens, como a avaliação por meio de *feedbacks* humanos, também podem ser utilizadas.

Os passo de inferência da função de recompensa, treinamento do agente e avaliação são repetidos até que um critério de parada seja atingido, como um número máximo de iterações ou quando o agente alcança um valor específico de recompensa acumulada.

2.5 Aprendizado por Imitação

O Aprendizado por Imitação, em inglês *Imitation Learning* (IL), consiste em aprender uma tarefa a partir de exemplos. O Aprendizado por Imitação está presente no desenvolvimento humano. Seja no balbuciar das primeiras sílabas 'mama' de um bebê, após ouvir diversas vezes a palavra 'mamãe', ou em situações mais complexas, como aprender um novo golpe de karatê, aprender a ler e escrever, dominar um esporte ou tocar um instrumento. É considerado uma das principais ferramentas de aprendizado, uma

vez que permite a assimilação de um comportamento sem necessariamente passar pelo processo de tentativa e erro (Bandura, 1962).

Contudo, o Aprendizado por Imitação não é exclusividade da espécie humana (Galef Jr, 2013) e está presente em diversos animais, como outros primatas (Heyes, 1998), aves (Galef Jr et al., 1986) e felinos (Zentall, 1996), permitindo que esses aprendam novas habilidades mais rapidamente, aumentando sua adaptabilidade ao ambiente e fornecendo, assim, uma vantagem evolutiva que permite a perpetuação dos seus genes (Huber et al., 2009).

No contexto computacional, o Aprendizado por Imitação consiste em aprender a realizar uma tarefa com base em demonstrações executadas por um ou mais especialistas. Ao contrário do Aprendizado por Reforço, onde o agente interage com o ambiente e tem acesso a uma função de recompensa para guiar seu aprendizado, no Aprendizado por Imitação, o agente não possui acesso a tal função, guiando-se através das demonstrações fornecidas.

Através do Aprendizado por Imitação, é possível aprender tanto a política de decisão do agente, quanto a função de recompensa. Neste último caso, as técnicas capazes realizar a modelagem da função de recompensa recebem o nome de Aprendizado por Reforço Inverso (ou IRL, do ingês *Inverse Reinforcement Learning*), justamente porque, diferente do Aprendizado por Reforço, que utiliza a função de recompensa para gerar o modelo, ela utiliza as demonstrações para a geração da função de recompensa. Existem diversos tipos de técnicas de aprendizado por imitação, tais como, Clonagem Comportamental (*Behavior Cloning*, BC) (Bain and Sammut, 1995), Aprendizagem por Imitação Adversária Generativa (*Generative Adversarial Imitation Learning*, GAIL) e Aprendizagem Adversarial por Reforço Inverso (*Adversarial Inverse Reinforcement Learning, AIRL*). Nas subseções a seguir, será detalhado o funcionamento de cada uma dessas técnicas.

2.5.1 Clonagem Comportamental

A Clonagem Comportamental, do inglês *Behavior Cloning*, é uma técnica de Aprendizado por Imitação que permite que um modelo aprenda a reproduzir as decisões de especialistas com base em demonstrações. Ela usa o aprendizado supervisionado para mapear pares de observações e ações e, assim, treinar uma política que minimize o erro em relação aos dados apresentados. É importante ressaltar que os modelos de clonagem comportamental geralmente não conseguem se recuperar bem de erros e são dependentes da qualidade das demonstrações fornecidas (Codevilla et al., 2019), mas destacam-se por ser uma técnica simples e computacionalmente eficiente (Hussein et al., 2017). Além disso, ela tem sido amplamente utilizada para resolver diversos problemas de aprendizado, como em robótica (Fang et al., 2019), carros autônomos (Farag and Saleh, 2018), jogos (Pearce and Zhu, 2022) e processamento de linguagem natural (Wang et al., 2022).

2.5.2 Aprendizagem por Imitação Adversária Generativa

Aprendizagem por Imitação Adversária Generativa, em inglês *Generative Adversarial Imitation Learning* (GAIL) (Ho and Ermon, 2016), é uma técnica de Aprendizado por Imitação inspirada nas Redes Adversárias Generativas (GANs) (Goodfellow et al., 2020), utilizada para gerar modelos capazes de imitar as ações de um especialista. Diferente do *Behavior Cloning*, que compara diretamente as ações do especialista com as ações tomadas pelo modelo, o GAIL emprega um submodelo interno, chamado de descriminador adversário, para distinguir indiretamente ações tomadas pelo modelo daquelas tomadas pelo especialista.

Utilizando-se da combinação de conceitos das Redes Adversárias Generativas e do Aprendizado por Reforço Inverso, o GAIL consegue aprender a partir de um número reduzido de demonstrações do especialista. Durante o processo de treinamento, o discriminador serve como uma espécie de função de recompensa, avaliando se o comportamento apresentado pela política se aproxima do comportamento dos especialistas (Bhattacharyya et al., 2022). É importante ressaltar que, embora o discriminador possa funcionar como uma espécie de função de recompensa interna para o modelo, orientando o gerador a produzir políticas que se aproximem mais do comportamento do especialista, o GAIL não deve ser considerado uma técnica de Aprendizado por Reforço Inverso. Isso porque o objetivo do GAIL é aprender a política de controle, não a função de recompensa. O Aprendizado por Reforço Inverso, em contrapartida, tem como objetivo inferir a função de recompensa a partir das demonstrações do especialista, sem a necessidade de acessar a política de controle (Arora and Doshi, 2021). O GAIL tem se mostrado promissor na resolução de diversos problemas, tais como direção autônoma (Peng et al., 2022), geração automática de texto (Wu et al., 2021), jogos (Song et al., 2018) e robótica (Tsurumine et al., 2019).

O gerador do GAIL não é exposto diretamente às demonstrações do especialista, sendo responsável apenas pela criação de uma política π_i (política gerada na iteração i) capaz de imitar o comportamento do especialista de forma indistinguível para o discriminador. O discriminador, por sua vez, é treinado de forma a ser apto a identificar se uma dada trajetória do agente é de origem do especialista ou é fruto da política desenvolvida pelo gerador. Essa evolução mútua, junto ao fato do gerador não ser exposto diretamente às demonstrações do especialista, proporciona ao GAIL uma maior capacidade de generalização e recuperação de falhas em comparação a métodos como a clonagem comportamental, que utiliza diretamente as demonstrações do especialista para treinar a política (Chen et al., 2020).

O algoritmo de Aprendizagem por Imitação Adversária Generativa funciona inicialmente assumindo que a política do especialista $\hat{\pi}$ pode ser aproximada pelas trajetórias de demonstrações fornecidas $\tau \sim \hat{\pi}$, onde τ representa o conjunto de trajetórias. Em paralelo,

o gerador cria uma política π_1 e gera uma trajetória utilizando essa política. As duas trajetórias, uma do especialista e a outra da política do gerador, são então combinadas e apresentadas como entrada para o discriminador, que tem como objetivo identificar quais sub-trajetórias foram originadas utilizando a política do gerador. Por sua vez, o gerador recebe um *feedback* da qualidade da sua política com base no retorno do discriminador. Esse *feedback* é utilizado para otimizar a sua política, gerando uma nova política π_{i+1}. Esse ciclo é repetido até que algum critério de parada pré estabelecido seja atingido. Geralmente, o número máximo de interações do gerador com o ambiente é utilizado como critério de parada. A Figura 11 mostra um diagrama do funcionamento do GAIL.

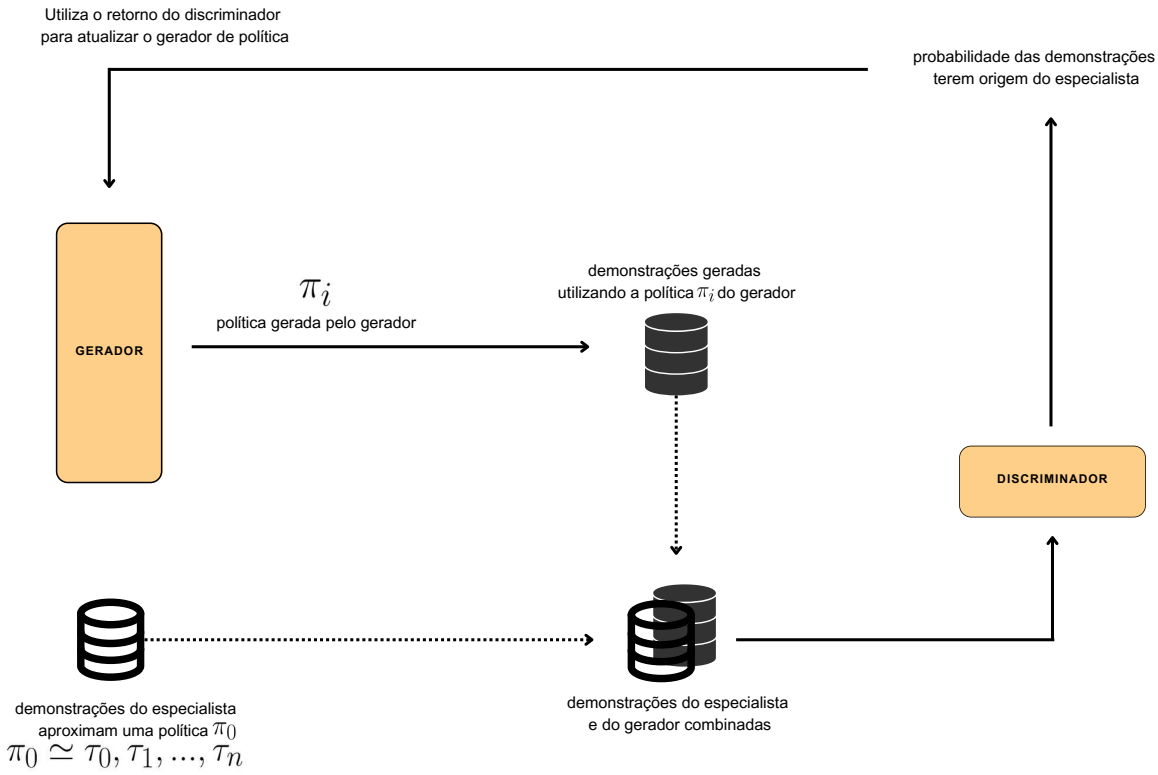


Figura 11 – Diagrama descrevendo o funcionamento do modelo de Aprendizado por Imitação Adversária Generativa.

2.5.3 Aprendizado por Reforço Inverso Adversarial

O Aprendizado por Reforço Inverso Adversarial, do inglês *Adversarial Inverse Reinforcement Learning* (AIRL) (Fu et al., 2017), é um algoritmo de Aprendizado por Reforço Inverso que busca aprender a função de recompensa para um ambiente a partir de demonstrações do comportamento desejado, sem ter acesso a função de recompensa real do ambiente, sendo especialmente útil quando a função de recompensa é difícil de ser modelada manualmente (Arora and Doshi, 2021). Assim como o GAIL, o AIRL utiliza de

uma abordagem *adversarial*, onde um discriminador é treinado para reconhecer a diferença entre ações providas das demonstrações do especialista e ações do agente, e a política do agente é treinada de forma a ser indistinguível das ações do especialista.

O processo de otimização da função de recompensa no AIRL tem como objetivo encontrar uma função de recompensa capaz de gerar comportamentos similares aos do especialista. Esse processo é geralmente realizado por meio de iterações alternadas entre o treinamento da política do agente e do modelo do discriminador.

O processo de otimização do AIRL ocorre da seguinte forma:

1. As demonstrações do especialista (pares de observações e ações) são utilizadas como dados de treinamento;

2. Geração de Trajetórias: o agente utiliza uma política de Aprendizado por Reforço para gerar trajetórias no ambiente. Essas trajetórias são geradas com base nas estimativas atuais da função de recompensa;

3. Treinamento do discriminador: o discriminador é treinado para distinguir entre as trajetórias do agente (geradas na etapa anterior) e as demonstrações reais do especialista. O objetivo é que o discriminador seja capaz de diferenciar o comportamento do especialista dos comportamentos gerados pelo agente;

4. Treinamento da política do agente: a política de decisão é treinada para gerar comportamentos que sejam indistinguíveis das demonstrações do especialista. Ela recebe os *feedbacks* do discriminador, que avalia a qualidade dos comportamentos gerados pela política atual. Assim, o objetivo é maximizar a habilidade de enganar o discriminador, gerando comportamentos que se aproximem ao máximo dos comportamentos reais do especialista; e

5. Atualização da função de recompensa: a função de recompensa estimada é atualizada buscando maximizar a probabilidade de, utilizando a política de decisão, gerar comportamentos que sejam indistinguíveis das demonstrações reais do especialista.

Os passos 3 a 5 são repetidos em iterações alternadas, permitindo a otimização sucessiva do agente e do discriminador. Essas iterações buscam encontrar um equilíbrio entre gerar comportamentos cada vez mais próximos do especialista e tornar o discriminador mais habilidoso em distinguir as demonstrações reais dos comportamentos do agente. O objetivo do processo de otimização é encontrar uma função de recompensa capaz de explicar e reproduzir as preferências do especialista através das demonstrações fornecidas.

3 TRABALHOS RELACIONADOS

Este capítulo apresenta, respectivamente, trabalhos relevantes e relacionados para o tema da presente dissertação.

(Mnih et al., 2013) apresenta um método para geração de controladores capazes de jogar diversos jogos de Atari 2600[1], incluindo jogos como *Space Invaders* e *Pong*. O modelo apresentado utiliza diretamente as imagens da tela do jogo e aprende a jogar através da interação com o ambiente, utilizando tentativa e erro, onde a pontuação do jogo é utilizada como recompensa, guiando um método de Aprendizado por Reforço Profundo. Ao todo, o modelo proposto foi treinado em 49 jogos, sendo capaz de atingir resultados melhores do que o de jogadores humanos em múltiplos jogos, como no *Pong* e *Breakout*.

Publicado por um grupo de pesquisadores da empresa DeepMind, subsidiária do Google, (Silver et al., 2016) deu origem ao AlphaGo. O AlphaGo é um modelo desenvolvido para jogar o jogo de tabuleiro Go[2]. O modelo proposto usa do Aprendizado por Reforço Profundo e técnicas de Aprendizado por Imitação, dispondo de exemplos jogadas realizadas por especialistas humanos. O trabalho também utilizou um processo de busca local que permitia ao modelo simular diversas sequências de jogadas e escolher a melhor opção. O fato do Go[2] ser um jogo de soma zero (para um jogador ganhar, o outro necessariamente precisa perder) possibilitou que o modelo jogasse contra ele próprio, melhorando seu desempenho e desenvolvendo estratégias não vistas. O trabalho ganhou notoriedade, pois foi capaz de superar o campeão mundial de Go[2], Lee Sedol, por 4 a 1 em 2016.

Em 2017, um ano depois do AlphaGo, o mesmo grupo de pesquisa da DeepMind publicou outra pesquisa (Silver et al., 2017), responsável pela criação do AlphaGo Zero. Diferente do trabalho anterior, onde existia uma etapa de aprendizado por demonstrações humanas, o AlphaGo Zero iniciou o seu treinamento sem nenhuma informação de como o jogo funcionava, aprendendo através de tentativa e erro por meio de métodos de Aprendizado por Reforço Profundo. O AlphaGo Zero também dispensou o uso da etapa de busca local para simular e escolher entre diversas jogadas possíveis, tornando o modelo final computacionalmente mais eficiente que o anterior. Durante o treinamento, a versão Zero do modelo utilizou a mesma abordagem de treinar contra si mesmo e foi capaz de desenvolver estratégias que não eram conhecidas por especialistas do jogo, superando a versão anterior em taxa de vitória.

Em (Vinyals et al., 2019) é apresentada uma abordagem de Aprendizado por

[1] Atari 2600 foi um console de video-game projetado por Jay Miner e lançado 1977 pela empresa Atari Inc. nos Estados Unidos, alcançando grande sucesso na década de 1980.

[2] O Go é um jogo de tabuleiro para dois jogadores que se originou na China há mais de 2.500 anos. O objetivo do jogo é controlar mais território no tabuleiro do que o seu oponente, colocando pedras em um tabuleiro tradicionalmente feito de madeira e dividido numa grade de 19x19 linhas. O número de jogadas possíveis no Go é de aproximadamente 2×10^{170}.

Reforço Profundo baseada em múltiplos agentes para jogar *StarCraft II*, considerado um jogo de estratégia em tempo real, em inglês *Real Time Strategy* (RTS). O trabalho utilizou os quadros da tela do jogo para treinar uma política do tipo Ator-Crítico, além de utilizar uma busca local em árvore para apoiar a escolha de decisões estratégicas. O modelo treinado foi capaz de vencer jogares humanos profissionais, mostrando ser apto a realizar planejamento estratégico de jogadas e gerenciamento de recursos, além de se adaptar a diferentes estilos de jogos do oponente.

(Berner et al., 2019) introduziu um modelo multiagente para jogar o jogo *Dota 2*, chamado de *OpenAI Five*. *Dota 2* é um jogo de estratégia em tempo real composto por dois times de cinco jogadores cada, que requer coordenação de equipe e habilidade individual para vencer. Os autores utilizaram de uma abordagem de Aprendizado por Reforço distribuído, na qual é possível treinar múltiplos agentes em paralelo. O modelo é composto por submodelos independentes para cada agente (cada um dos cinco jogadores do time), onde cada um desses agentes conta com uma política independente otimizada com o algoritmo *Proximal Policy Otimization*. O treinamento do modelo foi realizado colocando o modelo para jogar contra si próprio e durou aproximadamente seis meses, sendo equivalente a mais de 180 anos de jogos humanos por dia de treinamento para cada um dos agentes (aproximadamente 900 anos de treinamento contando cada um dos cinco agentes). O *OpenAI Five* venceu uma equipe profissional de jogadores humanos, sendo capaz de coordenar jogadas complexas e comunicar ações entre os seus agentes.

Em 2022, pesquisadores da *OpenAI* propuseram um método capaz de aprender através de vídeos do jogo *Minecraft*[3] (Baker et al., 2022). O trabalho utilizou um modelo chamado *Inverse Dynamics Model* (IDM) capaz de, dado um vídeo do jogo, mapear quais ações estão sendo realizadas. Para isso, o modelo foi treinado utilizando uma base de dados de mais de duas mil horas de vídeos com ações mapeadas. Após o treinamento, o IDM foi utilizado para mapear as ações realizadas por jogadores em mais de 70 mil horas de vídeos de *Minecraft*. Os dados obtidos foram então utilizados para treinar um modelo de Aprendizado por Imitação de clonagem comportamental capaz de realizar diversas ações complexas, como coletar recursos e caçar. Os autores ainda propuseram uma etapa final de ajuste fino utilizando Aprendizado por Reforço, o que melhorou o desempenho do agente em tarefas de coleta de recursos e na criação de itens complexos.

A seguir os trabalhos mais diretamente relacionados ao presente estudo:

Bhonker et al. (2016) propõem a utilização de DRL para a criação de agentes capazes de jogar jogos de Super Nintendo (SNES) utilizando uma abordagem semelhante

[3] Minecraft, lançado em 2011, é considerado um dos jogos mais populares do mundo. O jogador é posto em um mundo aberto, onde tem acesso a diversos cubos dispostos de forma aleatória, e pode coletar recursos e construir itens. Desde seu lançamento, o jogo atraiu públicos de diferentes faixas etárias, possuindo milhões de horas de *gameplay* e tutoriais em plataformas de vídeo, como o Youtube.

à de Mnih et al. (2013), onde o agente aprende utilizando os pixels da tela. O trabalho se valeu do uso de redes convolucionais para treinar a política de controle do agente. O desempenho do modelo foi avaliado em cinco diferentes jogos: *Super Mario World, F-Zero, Gradius 3, Mortal Kombat e Wolfenstein*, obtendo desempenho equivalente ao de jogadores humanos em três dos jogos (*Super Mario World, F-Zero e Gradius 3*) e desempenho superior ao humano no jogo *Mortal Kombat*.

Uma abordagem para a geração de controladores capazes de jogar jogos de tiro em primeira pessoa, *First Person Shotting* (FPS), foi proposta por Lample and Chaplot (2017). O estudo utilizou o jogo *Doom*, clássico jogo de tiro lançado no inicio da década de 1990, como ambiente de teste. Assim como trabalhos anteriores que se propuseram a desenvolver controlares para jogos, o estudo utilizou os quadros da tela do jogo como entrada para uma rede neural convolucional combinada com uma rede *Long Short Term Memory* (LSTM) como política do modelo de Aprendizado por Reforço Profundo. O resultado obtido foi similar aos resultados de jogadores humanos, mostrando a efetividade do Aprendizado por Reforço Profundo em ambientes complexos.

O trabalho de Ibarz et al. (2018) propõe uma abordagem para o treinamento de agentes por Aprendizado por Reforço Profundo utilizando demonstrações de especialistas e preferências de trajetórias. Para obter as preferências de trajetórias, dois estados do jogo foram apresentados a um juiz humano, que escolhia qual considerava ser a melhor opção. As preferências coletadas foram então usadas para ajustar a função de recompensa do aprendizado por reforço. Os autores treinaram o modelo em nove jogos diferentes de Atari e obtiveram desempenhos superiores aos humanos em dois desses jogos.

A pesquisa de Motta (2019) utilizou de Aprendizado por Reforço e Aprendizado por Imitação para treinar agentes capazes de jogar *Bomberman*. O trabalho utilizou de políticas de redes neurais do tipo MLP e *Long Short Term Memory* (LSTM) treinadas com a otimização de política proximal. O autor também realizou uma comparação entre diversas formas de representação do jogo em uma matriz de posições de obstáculos de inimigos. Os resultados obtidos demonstraram que a utilização da Clonagem Comportamental pode influenciar positivamente o treinamento da política de Aprendizado por Reforço.

O artigo de Chiang et al. (2020) propõe um método para explorar o espaço de busca de jogos de plataforma do tipo *Side-Scrolling*[4]. Os autores propuseram uma técnica chamada *Trajectory Replay* com o objetivo de explorar rapidamente o ambiente. A técnica consiste em armazenar a melhor trajetória do agente e utilizá-la para compor a função de recompensa do Aprendizado por Reforço, permitindo que o agente utilize do conhecimento de experiências anteriores para guiar seu processo de aprendizado, gastando mais recursos em novas áreas não exploradas. Para avaliar o método proposto, os autores utilizaram o

[4] Side-Scrolling é uma modalidade de jogo eletrônico na qual a jogabilidade é vista do ângulo de visão lateral, e à medida que o personagem do jogador se move lateralmente, a tela o acompanha.

jogo *Sonic The Hedgehog* e *Super Mario Bros*, obtendo resultados superiores ao uso do Aprendizado por Reforço tradicional.

O trabalho de Souza et al. (2022) utiliza a otimização de política proximal para treinar uma política de Redes Neurais Convolucionais capaz de jogar *Sonic The Hedgehog*. Os autores propõem uma abordagem para a modelagem da função de recompensa, utilizando processamento de imagem chamado *Image-Based Reward Calculation Module* (IBRCM), no qual é levado em consideração o deslocamento do agente para compor parte da recompensa. A recompensa modificada é então utilizada para treinar o modelo de Aprendizado por Reforço. O modelo foi testado em seis níveis do jogo, sendo capaz de superar o desempenho da otimização de política proximal sem a modificação em três dos cenários de testes e apresentando desempenho equivalente em outros dois cenários.

O presente trabalho se distingue da literatura ao explorar a utilização de diferentes técnicas de Aprendizado por Imitação, em especial técnicas adversariais, em combinação com o Aprendizado por Reforço aplicadas a problemas de jogos digitais com alta dimensionalidade.

Em comparação às pesquisas citadas nessa revisão, essa dissertação não possui a pretensão de gerar modelos melhores que humanos, uma vez que o custo computacional para a realização de tais experimentos é ordens de grandeza maior que os recursos aqui disponíveis, limitando-se, então, a comparar a diferença entre os resultados obtidos.

4 DEFINIÇÃO DO PROBLEMA

Este capítulo apresenta o jogo *Sonic The Hedgehog*™, além de detalhes do funcionamento do simulador e das modificações realizados para o desenvolvimento desse trabalho. *Sonic The Hedgehog*™ é um jogo produzido pela empresa japonesa Sega. Lançado em Junho de 1991, rapidamente se tornou popular. O jogo conta a história de um ouriço chamado *Sonic*, que tem que proteger o mundo de um vilão chamado *Dr. Eggman* que transformou todos os animais do mundo em robôs. Para isso, o personagem deve percorrer um cenário com diversos inimigos e obstáculos que se tornam cada vez mais desafiadores ao longo do jogo.

O jogo tem se mostrado especialmente desafiador para técnicas de Inteligência Artificial devido à grande variedade de inimigos, texturas e objetos, além de possuir pequenos quebra-cabeças a serem resolvidos, como em alguns níveis onde o agente deve pressionar uma alavanca ou botão para liberar o caminho. Nichol et al. (2018) propõe o uso do *Sonic* como um benchmark para avaliar o desempenho de diversas técnicas de Aprendizado por Reforço.

4.1 Descrição do Jogo e Níveis

O jogo é dividido em sete diferentes níveis e cada nível é subdividido em atos. De forma geral, cada nível tem diferentes conjuntos de inimigos, texturas e objetos, e cada ato compartilha as mesmas texturas, objetos e inimigos, porém distribuídos de formas diferentes pelo cenário. A Tabela 2 apresenta lista com todos os níveis e quantidade de atos por nível para o jogo.

Tabela 2 – Lista de níveis com a respectiva quantidade de atos para o jogo *Sonic the Hedgehog*™.

Nível	**Número de atos**
Green Hill Zone	3
Marble Zone	3
Spring Yard Zone	3
Labyrinth Zone	3
Star Light Zone	3
Scrap Brain Zone	3
Final Zone	1

4.2 Controles

O controle do *Mega Drive*, console de video-game para o qual jogo *Sonic* foi desenvolvido, possui doze botões. Uma representação do mesmo pode ser visto na Figura 12. Para representar cada um dos botões do controle, o emulador utiliza um vetor binário

Figura 12 – Controle do video-game Mega Drive.

de tamanho doze. Quando um botão está pressionado, a posição relativa ao mesmo é marcada com 1.

4.3 Observações

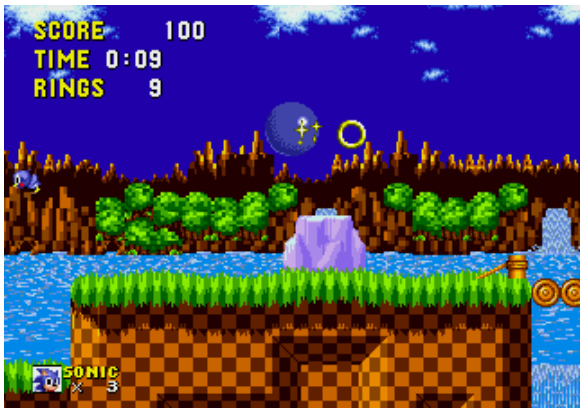
Foi utilizada a biblioteca *Gym* (Brockman et al., 2016), disponível na linguagem *Python*, como interface entre os métodos de aprendizado e o emulador. O emulador fornece uma matriz de 320x224 pixels em RGB e atualiza a uma taxa de 60 quadros por segundo. A Figura 13 mostra exemplos de imagens fornecidas pelo emulador.

4.4 Episódio

A interação do agente com o jogo é dividida em episódios (também chamado de trajetórias). Cada episódio corresponde a uma vida do personagem. Ao final de cada episódio, o emulador reinicia o nível. O episódio é considerado terminado quando:

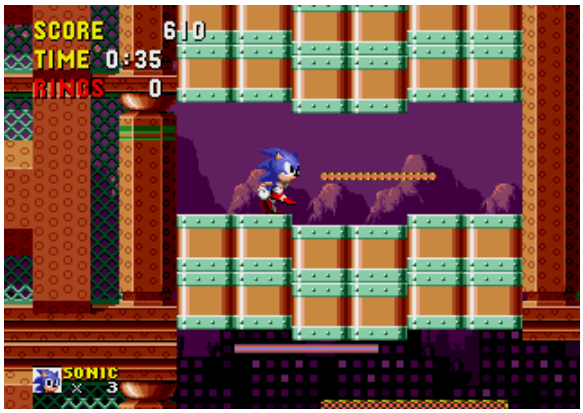
- o agente completa o nível;
- o agente morre; ou
- se passam mais de 4500 passos de tempo, equivalente a cinco minutos de jogo.

Durante um episódio, o agente recebe recompensa proporcional ao seu deslocamento horizontal. No *Sonic*, o objetivo final sempre se encontra à direita do ponto de início do nível. Entretanto, em alguns níveis, é necessário se deslocar para a esquerda (ou seja, na direção oposta à recompensa) por determinados momentos, seja pelo design do nível ou para ganhar aceleração suficiente para superar algum obstáculo. A Figura 14 ilustra essa situação. A recompensa final do agente é composta pelo deslocamento horizontal, normalizado entre 0 e 9000, e um bônus que decresce linearmente conforme a duração

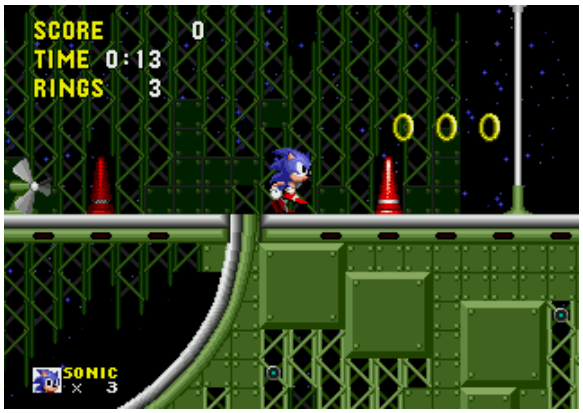


(a) Green Hill Zone, Ato 1

(b) Green Hill Zone, Ato 2



(c) SpringYard Zone, Ato 2



(d) StarLight Zone, Ato 2

(e) StarLight Zone, Ato 3

(f) ScrapBrain Zone, Ato 1

Figura 13 – Figura mostrando imagens de cada um dos níveis utilizados para treinamento.

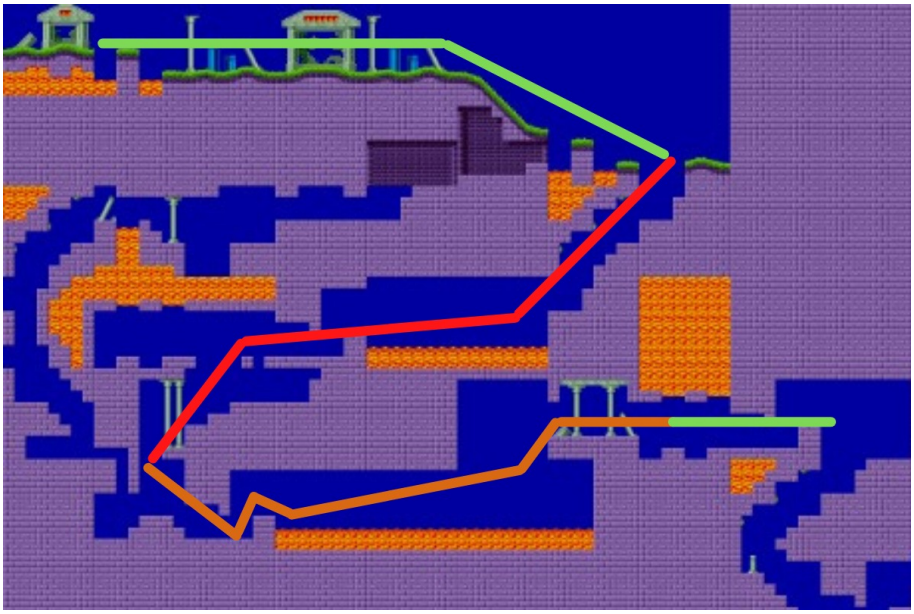


Figura 14 – A linha multicolorida representa a trajetória do *Sonic* no mapa *Marble Zone* ato 3. O agente inicia se movimentando para direita e ganhando recompensa positiva, demonstrado pela linha na cor verde. Porém em determinado ponto, o agente é obrigado a se deslocar para a esquerda para conseguir avançar, sendo punido por esse comportamento (recompensa negativa), representado pela cor vermelha. Durante o segmento laranja, o agente passa a ganhar recompensa positiva novamente, porém a recompensa acumulada ainda não é tão alta quanto à representada pelo primeiro segmento verde. Finalmente, no último segmento verde, o agente supera a recompensa acumulada no segmento verde inicial.

do episódio, iniciando em 1000 para episódios terminados instantaneamente e chegando a zero após 4500 passos de tempo. A recompensa máxima teórica que um agente pode receber é de 10000.

5 PROPOSTA

Este capítulo apresenta a contextualização da proposta e a descrição dos modelos elaborados para o desenvolvimento deste trabalho. Será detalhado como foram realizados os pré-processamentos necessários para os experimentos, bem como as arquiteturas das redes neurais utilizadas.

5.1 Criação das demonstrações

Para a criação das demonstrações usadas para treinar os modelos de Aprendizado por Imitação, foi empregada a biblioteca *pygames*, onde cada nível foi jogado por um jogador humano até ser finalizado por 5 vezes. A coleta de dados foi feita em dias diferentes para cada nível do jogo. Foram salvas as observações e ações do jogador humano das trajetórias que completaram o nível. Os dados obtidos foram pré-processados conforme descrito na Seção 5.2 e então salvos no formato .pckl utilizando a biblioteca *Pickle* da linguagem *Python*. A Tabela 3 apresenta a média, desvio padrão e o tempo total de observações em cada um dos níveis.

Tabela 3 – Número de observações e tempo total de demonstração para cada um dos níveis de teste.

Nível	Número de observações	Tempo total de demonstração
GreenHillZone.Act1	579 ± 53	3 minutos e 13 segundos
GreenHillZone.Act2	637 ± 33	3 minutos e 32 segundos
SpringYardZone.Act1	1640 ± 84	9 minutos e 7 segundos
StarLightZone.Act2	1205 ± 148	6 minutos e 42 segundos
StarLightZone.Act3	1440 ± 388	8 minutos
ScrapBrainZone.Act1	2095 ± 163	11 minutos e 38 segundos

Analisando os dados da Tabela 3, os níveis com tempo de demonstração maior, como *SpringYardZone.Act1* e *ScrapBrainZone.Act1*, podem ser entendidos como níveis mais complexos, visto que jogadores humanos demoraram cerca de três vezes mais tempo para completar em comparação com níveis mais rápidos.

5.2 Pré-Processamento

Visto que os movimentos do jogo *Sonic* não utilizam todos os botões do controle do *Mega Drive*, algumas combinações de botões podem ser ambíguas ou não fazer sentido para o jogo. Desse modo, foi realizado um pré-processamento na entrada das ações do emulador que antes era representado por um vetor binário de tamanho 12, conforme descrito na Sessão 4.2. O número de ações possíveis foi reduzido para 10 ações que fazem sentindo no jogo (de Souza et al., 2022), sendo representadas como por um conjunto discreto de tamanho 10. O conjunto de ações é descrito na Tabela 4.

Tabela 4 – Tabela com as 10 combinações possíveis de botões e ações correspondentes.

Ação	**Botões**	**Representação discreta**
Parado		0
Andar para esquerda	Esquerda	1
Andar para direita	Direita	2
Rolar para direita	Direita, Baixo	3
Rolar para esquerda	Esquerda, Baixo	4
Olhar para baixo	Baixo	5
Rolar	Baixo, B	6
Pular	B	7
Pular movendo-se para direita	B, Direita	8
Pular movendo-se para esquerda	B, Esquerda	9

As imagens provenientes do emulador (320×224 RGB) foram convertidas em uma matriz de escala de cinza e redimensionadas para (96×96). A observação do agente é gerada por 4 quadros sequenciais do jogo (4×96×96). O emulador roda nativamente a 60 quadros por segundo, sendo necessário que o agente realize uma ação a cada um desses quadros. Utiliza-se aqui a abordagem proposta em (Braylan et al., 2015), chamada *Frame Skip*, que mantém a ação por 4 quadros, reduzindo a taxa de atualização virtualmente para 15 quadros por segundo.

5.3 Modelos de Políticas

O presente trabalho assume como hipótese que o problema pode ser modelado como um Processo de Markov Parcialmente Observável e utilizou dois tipos de políticas da categoria *Actor-Critic* (Seção 2.3.3), uma baseada em CNN e a outra baseadas em MLP.

A política baseada em CNN é uma rede neural convolucional, com quatro camadas ocultas, utilizada como extrator de características. Essa rede é compartilhada entre o ator e o crítico, com entrada de 4 imagens de tamanho (96×96), correspondentes a quatro quadros do jogo pré processados (Seção 5.2). A primeira camada oculta é composta de 32 filtros de convolução com tamanho (8×8), *stride* de 4. A segunda camada possui 64 filtros de tamanho (4×4) com *stride* de 2. A terceira camada convolucional é composta por 64 filtros de tamanho (3×3) e *stride* de 3. A última camada é totalmente conectada e é formada por 512 neurônios. Entre cada uma das camadas ocultas, utilizou-se a função de ativação ReLu. Para o ator, a camada de saída é totalmente conectada à última camada oculta, utilizando de uma ativação linear composta por 10 neurônios, onde cada neurônio representa uma ação possível pelo agente. Já para o crítico, a camada de saída possui apenas um neurônio, também sendo totalmente conectada e utiliza de uma ativação linear (representando o v-valor da entrada). O modelo apresenta ao todo 2.181.291 parâmetros e sua arquitetura é ilustrada na Figura 15.

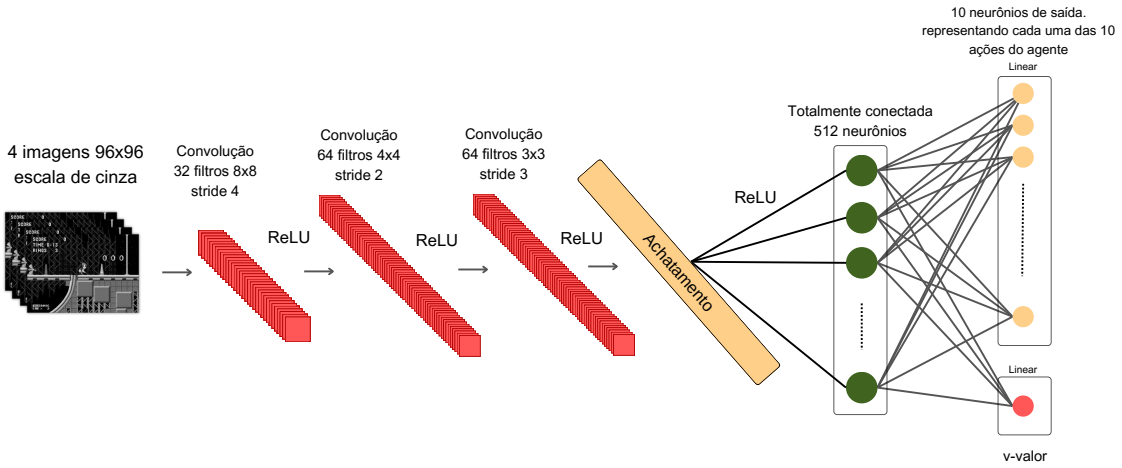


Figura 15 – Arquitetura da política utilizando CNN.

A política baseada em MLP consiste em duas redes Perceptron multi camadas com duas camadas ocultas cada uma, também utilizadas como extrator de características. A entrada é composta de 36.864 valores, correspondente às observações processadas do jogo transformadas em um vetor (4×96×96=36.864). Ambas as camadas ocultas têm 64 neurônios e utilizam a função de ativação tangente hiperbólica. Para o ator, a camada de saída é totalmente conectada à última camada oculta, utilizando de uma ativação linear, composta por 10 neurônios. Para o crítico, a camada de saída possui apenas um neurônio também conectado por uma ativação linear. O modelo baseado em MLP apresenta 4.727.755 de parâmetros e uma representação da sua arquitetura pode ser vista na Figura 16.

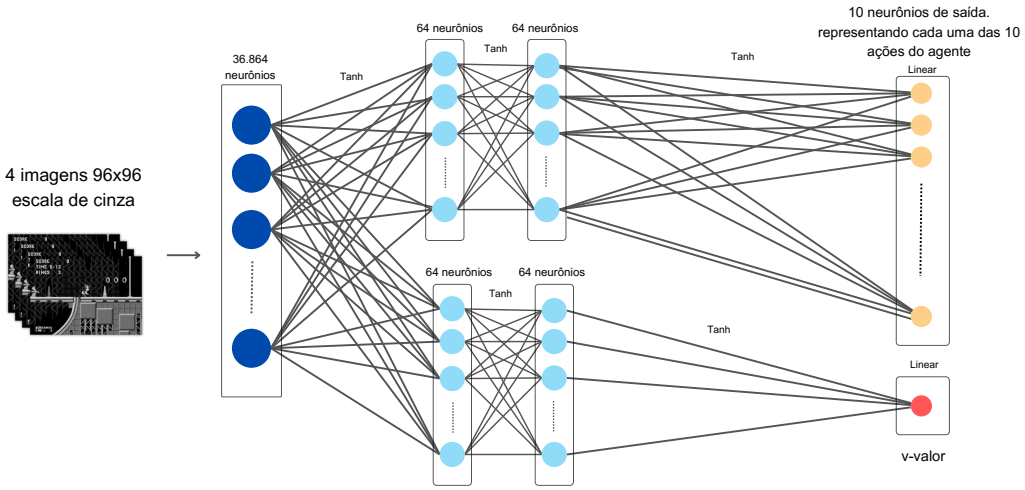


Figura 16 – Arquitetura da política utilizando MLP.

5.4 Discriminador

Para implementar o discriminador utilizado pelos modelos GAIL e AIRL, foi utilizado um MLP com duas camadas ocultas. A camada de entrada é composta pela observação vetorizada do ambiente, e contém 36.864 neurônios. As duas camadas ocultas possuem 32 neurônios cada, e utilizam a função de ativação ReLu. A camada de saída é totalmente conectada à última camada oculta e possui um neurônio, que é responsável por retornar o valor do discriminador. Essa rede neural possui 1.180.992 parâmetros.

6 EXPERIMENTOS E RESULTADOS

Este capítulo apresenta a metodologia experimental utilizada e a discussão dos resultados obtidos.

O desenvolvimento e implementação dos modelos aqui descritos, foram realizados utilizando a linguagem *Python* e as bibliotecas *Pytorch* (Paszke et al., 2019), *Gym* (Brockman et al., 2016), *Retro* (Nichol et al., 2018), StableBaseline3 (Raffin et al., 2021) e *Imitation* (Gleave et al., 2022). O código fonte e os dados de demonstrações estão disponíveis em https://github.com/feliperafael.

Os experimentos realizados nesse trabalho foram divididos em duas etapas. Primeiramente, foram gerados modelos de aprendizado por imitação. Na segunda etapa, os modelos gerados por IL foram submetidos a um treinamento de aprendizado por reforço.

Para treinamento dos modelos de IL, foram utilizadas as técnicas *Behavior Cloning*, GAIL e AIRL. Para os modelos foram testados com dois tipos de política, uma baseada em MLP e uma baseada em CNN, conforme visto na Seção 5.3. Para realizar a distinção das políticas utilizadas, modelos que empregam MLP são apresentados com um sufixo _MLP, assim como os modelos baseados em CNN são apresentados com o sufixo _CNN. Para a otimização da política, foi aplicado o algoritmo PPO, com os parâmetros descritos na Tabela 5. Os modelos baseados nas técnicas GAIL e AIRL foram treinados por 1 milhão de passos de tempo, equivalentes à aproximadamente 18 horas de jogo. Todos os modelos foram treinados nos seis níveis do *Sonic The Hedgehog* utilizados em (Chiang et al., 2020). Para cada experimento, foram realizadas dez execuções independentes. Os parâmetros para os modelos de BC, GAIL e AIRL são mostrados nas Tabelas 6 e 7.

Na segunda etapa, os modelos de BC, GAIL e AIRL, foram utilizados para iniciar a política de Aprendizado por Reforço, sendo comparados com modelos que tiveram

Tabela 5 – Parâmetros utilizados para a Otimização de Política Proximal

Parâmetro	valor
n_steps	512
gae_lambda	0.95
gamma	0.99
n_epochs	4
ent_coef	0.001
learning_rate	2e-4
clip_range	0.2

Tabela 6 – Parâmetros utilizados para o BC

Parâmetro	valor
batch_size	32
ent_weight	0.001

Tabela 7 – Parâmetros utilizados para o GAIL e AIRL

Parâmetro	valor
Batch de demonstração	1024
Capacidade do buffer de replay	2048
Atualizações do discriminador por iteração	4

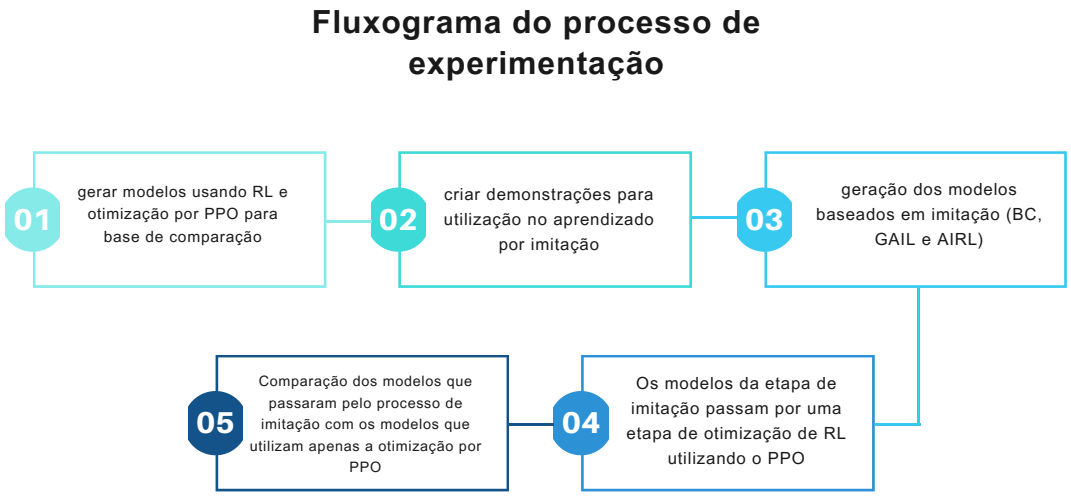


Figura 17 – Fluxograma do processo de experimentação

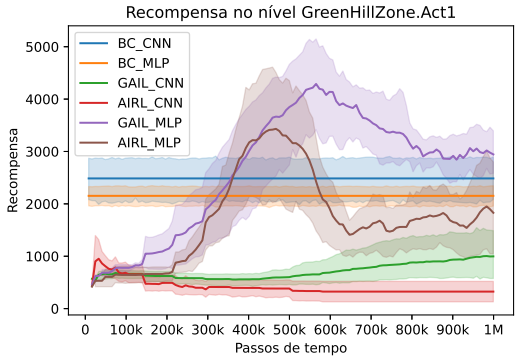
sua política iniciada aleatoriamente. O PPO foi usado para otimização da política e os modelos foram treinados por 1 milhão de passos de tempo. Os modelos que foram iniciados com o Aprendizado por Imitação recebem os sufixos: BC, GAIL e AIRL, representando, respectivamente, os métodos de imitação utilizados em cada um dos casos. Foram efetuadas 10 execuções independentes.

Os resultados obtidos foram comparados utilizando como métrica a pontuação descrita na Seção 4.4. As Tabelas 8, 9 e 10 apresentam as médias e desvio padrão para cada um dos níveis avaliados. Esses resultados foram obtidos após a execução de 50 episódios, ou seja, o agente jogou cada nível por 50 vezes, em cada uma das 10 execuções independentes. O fluxograma do processo de experimentação pode ser visto na Figura 17

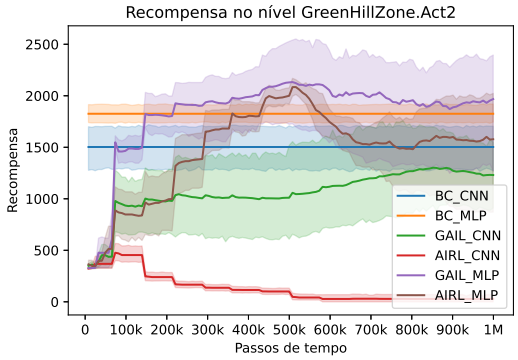
Os experimentos foram conduzidos no sistema operacional *Ubuntu* 20.04, em um computador equipado com processador *AMD Ryzen* 3600 e 32GB de RAM rodando a 3000mhz, e uma placa de vídeo GTX 1070 Ti.

6.1 Resultados e Discussão

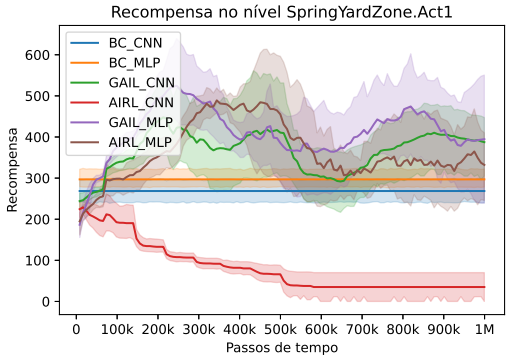
A Figura 18 mostra os resultados de treinamento para os modelos de imitação durante 1 milhão de passos de tempo de interação com o ambiente. Note que os modelos



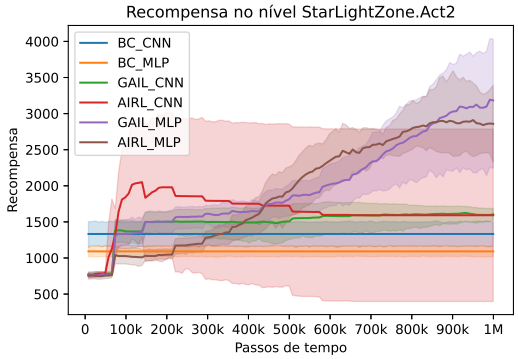
(a) Green Hill Zone, Ato 1



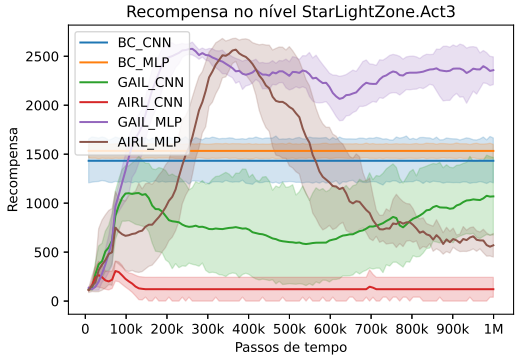
(b) Green Hill Zone, Ato 2



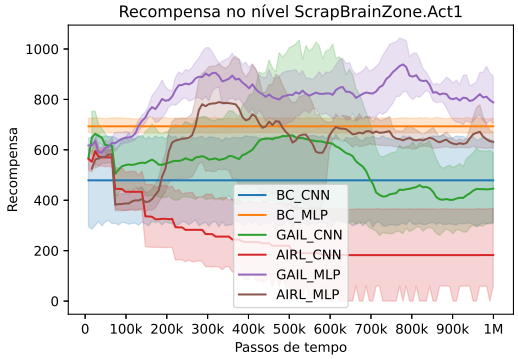
(c) SpringYard Zone, Ato 2



(d) StarLight Zone, Ato 2



(e) StarLight Zone, Ato 3



(f) ScrapBrain Zone, Ato 1

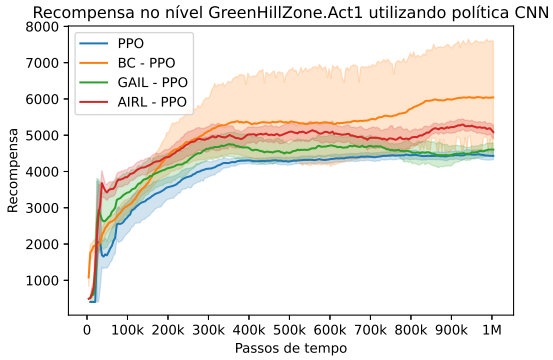
Figura 18 – Figura mostrando gráficos de treinamento dos modelos de imitação.

de BC são treinados de maneira a não possuir interação com o ambiente, logo ele é representado como uma valor constante de recompensa (eixo y do gráfico), apenas para fins de comparação.

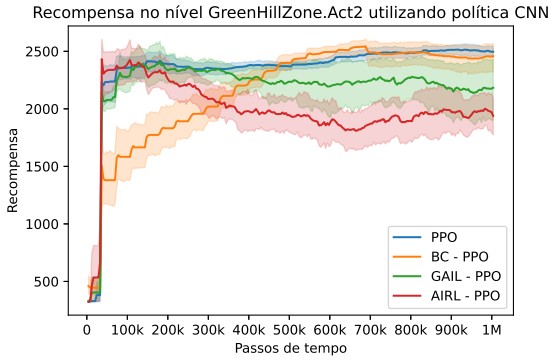
Ao analisar os gráficos apresentados na Figura 18, observa-se que os modelos adversariais que empregaram a política de MLP obtiveram melhor desempenho em relação aos modelos que utilizaram a política baseada em CNN. Uma explicação plausível para essa diferença de desempenho pode ser atribuída à complexidade das redes neurais utilizadas. A

Tabela 8 – Valor médio e desvio do desempenho dos modelos de aprendizado por imitação.

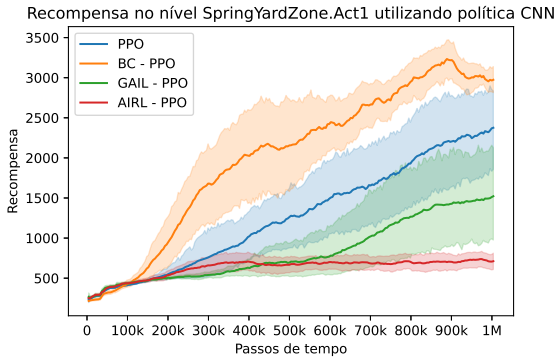
Nível	BC-MLP	BC-CNN	GAIL-MLP	GAIL-CNN	AIRL-MLP	AIRL-CNN
GreenHillZone.Act1	2153.13 ± 1994.62	2485.61 ± 2007.73	**2987.97 ± 2521.09**	859.1 ± 841.56	1830.96 ± 2356.6	324.88 ± 324.88
GreenHillZone.Act2	1825.44 ± 1064.9	1502.71 ± 1089.84	**2005.8 ± 2077.32**	949.52 ± 809.05	1589.44 ± 1280.44	27.84 ± 55.68
SpringYardZone.Act1	296.92 ± 211.68	268.58 ± 169.25	**562.18 ± 629.81**	349.82 ± 253.9	329.45 ± 452.69	35.07 ± 53.57
StarLightZone.Act2	1091.76 ± 598.8	1332.71 ± 586.02	**2981.99 ± 2426.64**	1371.76 ± 560.72	2635.12 ± 2030.5	1592.07 ± 1949.88
StarLightZone.Act3	1533.58 ± 734.32	1432.57 ± 883.48	**2194.55 ± 1076.29**	1255.77 ± 1096.89	548.09 ± 542.76	121.34 ± 185.35
ScrapBrainZone.Act1	693.33 ± 267.85	479.02 ± 366.62	**722.22 ± 250.72**	471.43 ± 180.35	627.17 ± 93.74	182.53 ± 278.85



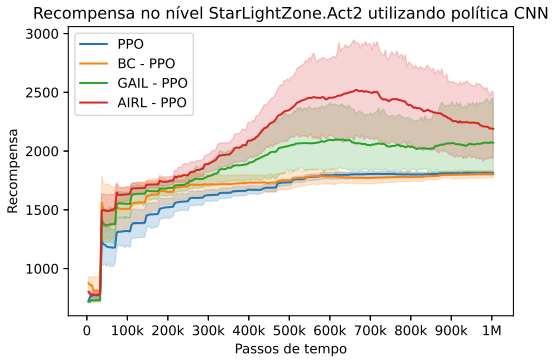
(a) Green Hill Zone, Ato 1



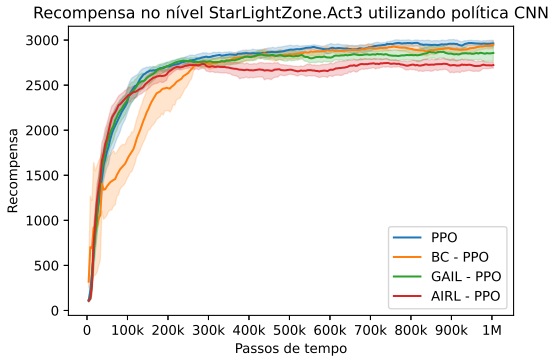
(b) Green Hill Zone, Ato 2



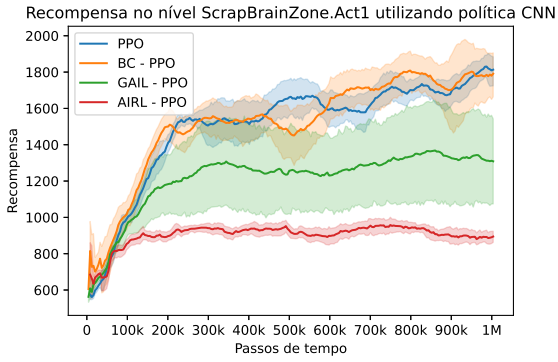
(c) SpringYard Zone, Ato 1



(d) StarLight Zone, Ato 2

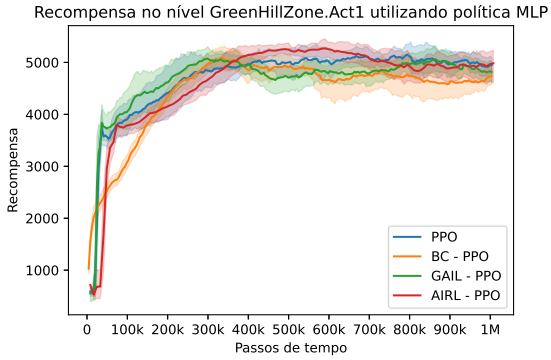


(e) StarLight Zone, Ato 3

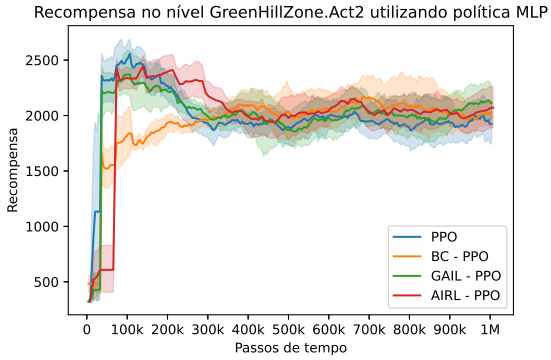


(f) ScrapBrain Zone, Ato 1

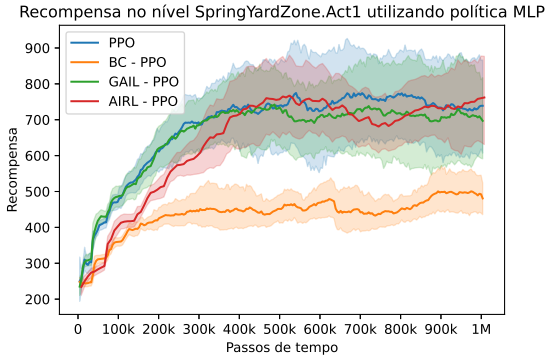
Figura 19 – Figura mostrando gráficos de treinamento dos modelos de aprendizado por reforço que utilizam a política baseada em CNN. Os modelos representados com os prefixos BC, GAIL e AIRL utilizaram os respectivos métodos de imitação para inicialização.



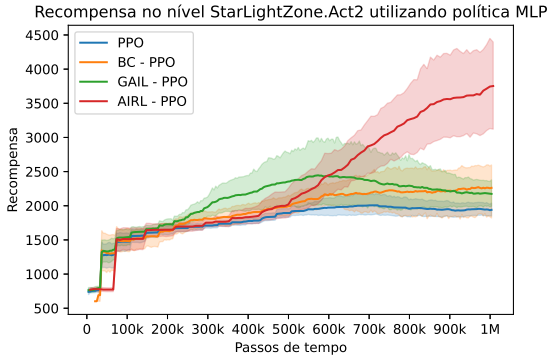
(a) Green Hill Zone, Ato 1



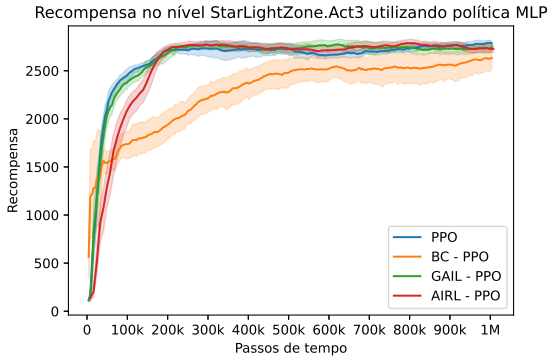
(b) Green Hill Zone, Ato 2



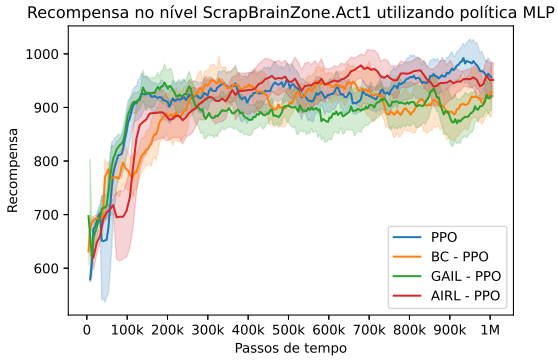
(c) SpringYard Zone, Ato 1



(d) StarLight Zone, Ato 2



(e) StarLight Zone, Ato 3



(f) ScrapBrain Zone, Ato 1

Figura 20 – Figura mostrando gráficos de treinamento dos modelos de aprendizado por reforço que utilizam a política baseada em MLP. Os modelos representados com os prefixos BC, GAIL e AIRL utilizaram os respectivos métodos de imitação para inicialização.

Tabela 9 – Comparação entre os modelos que utilizam apenas otimização com PPO e os modelos que utilizam inicialização com aprendizado por imitação antes da etapa de otimização com PPO para políticas baseadas em CNN. Os resultados marcados com asterisco são os que não apresentaram diferença estatística com a base de comparação (PPO-CNN), os testes estatísticos podem ser observados no Apêndice A.

Nível	PPO-CNN	BC-PPO-CNN	GAIL-PPO-CNN	AIRL-PPO-CNN
GreenHillZone.Act1	4515.65 +- 978.29	**6082.54 +- 2489.9**	5046.64 +- 2789.3	4961.07 +- 2887.0*
GreenHillZone.Act2	2531.43 +- 632.72	**2587.28 +- 442.69***	2107.55 +- 1899.89	2140.06 +- 1978.37
SpringYardZone.Act1	2707.08 +- 1890.37	**3106.87 +- 1910.13**	653.79 +- 645.93	677.57 +- 640.04
StarLightZone.Act2	1807.35 +- 184.98	1805.76 +- 188.34*	**2332.83 +- 1404.19**	2000.61 +- 901.04
StarLightZone.Act3	**2917.83 +- 530.75**	2915.74 +- 438.46	2703.84 +- 692.25	2700.27 +- 719.6
ScrapBrainZone.Act1	1775.76 +- 484.49	**1810.83 +- 522.5***	925.35 +- 446.8	897.88 +- 418.21

Tabela 10 – Comparação entre os modelos que utilizam apenas otimização com PPO e os modelos que utilizam inicialização com aprendizado por imitação antes da etapa de otimização com PPO para políticas baseadas em MLP. Os resultados marcados com asterisco são os que não apresentaram diferença estatística com a base de comparação (PPO-MLP), os testes estatísticos podem ser observados no Apêndice A.

Nível	PPO-MLP	BC-PPO-MLP	GAIL-PPO-MLP	AIRL-PPO-MLP
GreenHillZone.Act1	4731.29 +- 2805.88	**5009.36 +- 2663.18***	4945.86 +- 2875.67*	5004.37 +- 2658.21*
GreenHillZone.Act2	1944.51 +- 1798.55	2147.08 +- 1911.42	1909.06 +- 1874.67*	**2233.49 +- 1814.99**
SpringYardZone.Act1	693.19 +- 669.77	488.42 +- 443.94	683.47 +- 641.14*	**787.65 +- 739.7***
StarLightZone.Act2	1922.38 +- 713.32	2354.4 +- 1624.89*	2109.8 +- 1072.93	**2851.66 +- 2046.21**
StarLightZone.Act3	2673.39 +- 773.71	2694.31 +- 755.65	2706.56 +- 760.47*	**2744.49 +- 691.64***
ScrapBrainZone.Act1	951.15 +- 452.11	919.65 +- 413.46*	957.8 +- 464.11*	**958.76 +- 457.09***

política de CNN utilizada no experimento possui 2,1 milhões de parâmetros, o que a torna menos complexa do que a MLP, que possui 4,7 milhões. Sugerindo a possibilidade de que a maior quantidade de parâmetros da MLP pode ter contribuído para seu melhor desempenho, uma vez que a rede neural pode ter sido capaz de capturar características mais refinadas. No entanto, mais pesquisas são necessárias para confirmar essa hipótese e determinar outros fatores que possam ter influenciado esse resultado, já que esse comportamento parece não ter afetado da mesma forma os modelos de BC.

Outro ponto a se destacar é o fato dos modelos AIRL utilizando MLP apresentarem um pico de recompensa maior em etapas intermediárias do treinamento, como demonstrado nos gráficos dos níveis *GreenHillZone.Act1* e *StarLightZone.Act3*, levando a acreditar que esses modelos desaprenderam alguma regra relevante durante as iterações. Esse fenômeno é também observado ao analisar o gráfico do modelo GAIL com MLP no nível *GreenHillZone.Act1*, podendo indicar um efeito da abordagem adversarial, onde nem sempre a otimização em relação ao discriminador leva à uma otimização da pontuação do agente.

Ao examinar os resultados apresentados na Tabela 8, observa-se que o modelo GAIL, quando utilizado com política de MLP, obteve melhores desempenhos médios em todos os níveis avaliados. Os modelos baseados em BC e AIRL com MLP tiveram desempenho similares. Contudo, é possível destacar o desempenho superior do BC nos

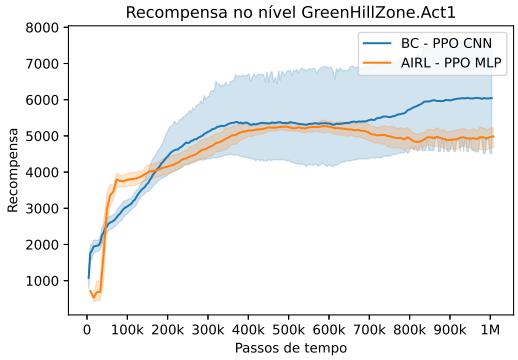
níveis *GreenHillZone.Act1* e *StarLightZone.Act3*, bem como do AIRL com MLP no nível *StarLightZone.Act2*.

Observa-se que a hipótese levantada na Seção 5.1, acerca de níveis com maior número de demonstrações serem mais desafiadores, foi confirmada pelos resultados obtidos. Especialmente, quando se avalia os níveis *ScrapBrainZone.Act1* e *SpringYardZone.Act1*, que apresentaram as maiores quantidades de demonstrações. Esses resultados sugerem que o desempenho inferior dos modelos nesses níveis pode ser atribuído à maior dificuldade apresentada por esses cenários.

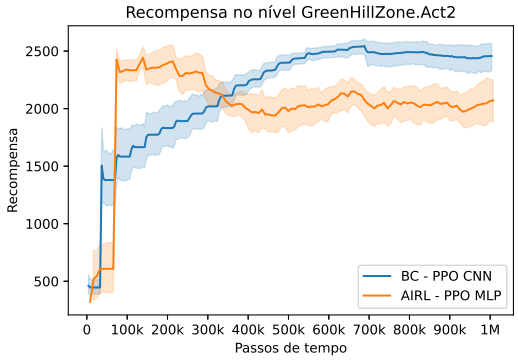
Analisando o gráfico apresentado na Figura 19, pode-se observar que os modelos de RL, que tiveram a política iniciada com o BC, se destacaram nos níveis *GreenHillZone.Act1* e *SpringYardZone.Act2* para as políticas baseadas em CNN. No entanto, para os demais níveis, a inicialização com BC parece não ter tido um efeito tão acentuado. Já com base na Tabela 9 é possível perceber que a inicialização com o BC resultou em políticas com médias superiores em cinco dos seis níveis. Sendo que no nível em que a média do PPO inicializado de forma aleatória foi melhor, a diferença na média entre eles foi de apenas 24 pontos. Com base na Figura 20 e Tabela 10, observa-se que os modelos de RL, iniciados com AIRL, apresentaram resultados superiores aos demais em 5 dos níveis, e no nível GreenHillZone.Act1 apresentou uma diferença de apenas 4.99 pontos na média em comparação com o segundo melhor resultado (BC-PPO-MLP).

A Figura 21 mostra a comparação entre o melhor modelo utilizando CNN e o melhor modelo utilizando MLP. analisando os gráfico podemos ver que apensar de as políticas utilizando MLP apresentar melhor desempenho nos experimentos utilizando apenas Aprendizado por Imitação, quando aplicamos a etapa de RL o modelo utilizando CNN torna-se competitivo.

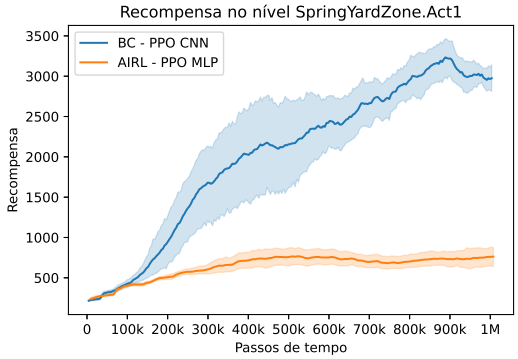
As Figuras 22 e 23 mostram quadrados brancos e esverdados representando a trajetória realizada pelo agente nas proximidades de um obstáculo de *loop*, presente no nível *GreenHillZone.Act1*. Podemos notar que o agente utilizando o algoritmo PPO iniciado aleatoriamente não consegue avançar na fase, ficando preso antes do *loop*. Esse obstáculo requer que o personagem tenha uma aceleração suficiente para superá-lo, e qualquer movimento que faça com que o agente perca aceleração resulta na impossibilidade de avançar. Por outro lado, ao analisar a trajetória do agente que foi inicializado com o BC, verifica-se que ele é capaz de superar o obstáculo, avançando em direção ao final da fase. Isso sugere que o agente utilizando imitação parece incorporar melhor as regras do jogo.



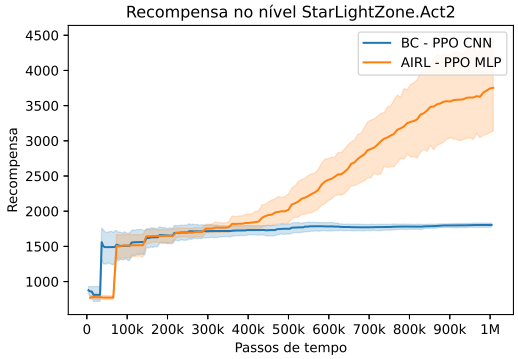
(a) Green Hill Zone, Ato 1



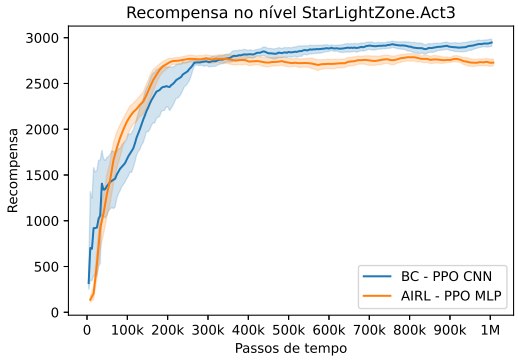
(b) Green Hill Zone, Ato 2



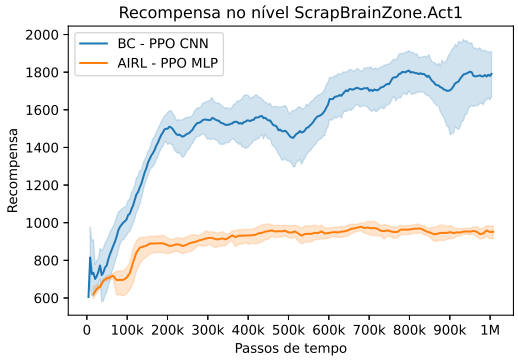
(c) SpringYard Zone, Ato 1



(d) StarLight Zone, Ato 2



(e) StarLight Zone, Ato 3



(f) ScrapBrain Zone, Ato 1

Figura 21 – Figura mostrando os melhores modelos utilizando a política CNN e MLP.

Figura 22 – Agente utilizando o PPO inicializado aleatoriamente não consegue superar o loop

Figura 23 – Agente utilizando o PPO inicializado com BC superando o loop

7 CONCLUSÃO E TRABALHOS FUTUROS

Diversos estudos têm explorado a criação de controladores de jogos usando DRL. No entanto, a combinação de técnicas de IL com Reforço Profundo tem sido pouco explorada, especialmente quando se trata do uso de técnicas adversariais, como o *Generative Adversarial Imitation Learning* e *Adversarial Inverse Reinforcement Learning*. O presente estudo se propôs a avaliar métodos de Aprendizado por Imitação e Aprendizado por Reforço para criação de controladores capazes de jogar jogos digitais, especificamente utilizando o jogo *Sonic The Hedgehog*.

Os resultados obtidos indicaram que modelos de imitação adversariais baseados em MLP obtiveram melhor desempenho que os modelos que utilizaram política de CNN. No entanto, não foi possível determinar se esse fato ocorreu devido à maior complexidade do modelo MLP, que pode ter sido capaz de capturar características mais relevantes para o problema. Observou-se, ainda, que os modelos AIRL, baseados em Aprendizado por Reforço Inverso, apresentaram maior sensibilidade ao treinamento em determinados níveis, sugerindo que, em alguns casos, a função de recompensa gerada não foi capaz de representar adequadamente o problema em questão.

Ao analisar a inicialização do treinamento de RL utilizando as políticas treinadas por BC, GAIL e AIRL, constatou-se que os modelos que passaram pela etapa de imitação antes do treinamento da política por RL obtiveram melhor desempenho, dando suporte à ideia de que a utilização de conhecimento de domínio prévio pode propiciar que o agente obtenha uma melhor performance. Esse resultado é promissor e pode indicar uma estratégia eficaz para acelerar o treinamento de agentes em ambientes mais complexos.

Em trabalhos futuros, planeja-se aprofundar a análise do uso de políticas baseadas em CNN nos métodos adversariais, com o objetivo de avaliar o desempenho dessas técnicas em um conjunto mais abrangente de demonstrações de especialistas e com recursos computacionais ampliados. Além disso, pretende-se explorar a capacidade de generalização dos modelos treinados em diferentes cenários e níveis do jogo, permitindo a transferência de conhecimento de um nível para outro. Deseja-se também estender essa abordagem para problemas do mundo real, tais como aplicações em robótica e veículos autônomos.

REFERÊNCIAS

Hervé Abdi, Dominique Valentin, and Betty Edelman. *Neural networks*. Number 124. Sage, 1999.

M Mehdi Afsar, Trafford Crump, and Behrouz Far. Reinforcement learning based recommender systems: A survey. *ACM Computing Surveys*, 55(7):1–38, 2022.

Szilárd Aradi. Survey of deep reinforcement learning for motion planning of autonomous vehicles. *IEEE Transactions on Intelligent Transportation Systems*, 23(2):740–759, 2020.

Saurabh Arora and Prashant Doshi. A survey of inverse reinforcement learning: Challenges, methods and progress. *Artificial Intelligence*, 297:103500, 2021.

Michael Bain and Claude Sammut. A framework for behavioural cloning. In *Machine Intelligence 15*, pages 103–129, 1995.

Bowen Baker, Ilge Akkaya, Peter Zhokhov, Joost Huizinga, Jie Tang, Adrien Ecoffet, Brandon Houghton, Raul Sampedro, and Jeff Clune. Video pretraining (vpt): Learning to act by watching unlabeled online videos. *arXiv preprint arXiv:2206.11795*, 2022.

Albert Bandura. Social learning through imitation. 1962.

Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

Raunak Bhattacharyya, Blake Wulfe, Derek J Phillips, Alex Kuefler, Jeremy Morton, Ransalu Senanayake, and Mykel J Kochenderfer. Modeling human driving behavior through generative adversarial imitation learning. *IEEE Transactions on Intelligent Transportation Systems*, 2022.

Nadav Bhonker, Shai Rozenberg, and Itay Hubara. Playing snes in the retro learning environment. *arXiv preprint arXiv:1611.02205*, 2016.

Daniel Bick and MA Wiering. *Towards Delivering a Coherent Self-Contained Explanation of Proximal Policy Optimization*. PhD thesis, 2021.

Michael Bowling, John D Martin, David Abel, and Will Dabney. Settling the reward hypothesis. *arXiv preprint arXiv:2212.10420*, 2022.

Alex Braylan, Mark Hollenbeck, Elliot Meyerson, and Risto Miikkulainen. Frame skip is a powerful parameter for learning to play atari. In *Workshops at the twenty-ninth AAAI conference on artificial intelligence*, 2015.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

Minshuo Chen, Yizhou Wang, Tianyi Liu, Zhuoran Yang, Xingguo Li, Zhaoran Wang, and Tuo Zhao. On computation and generalization of generative adversarial imitation learning. *arXiv preprint arXiv:2001.02792*, 2020.

I-Huan Chiang, Chung-Ming Huang, Nien-Hu Cheng, Hsin-Yu Liu, and Shi-Chun Tsai. Efficient exploration in side-scrolling video games with trajectory replay. *The Computer Games Journal*, 9:263–280, 2020.

Felipe Codevilla, Eder Santana, Antonio M López, and Adrien Gaidon. Exploring the limitations of behavior cloning for autonomous driving. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9329–9338, 2019.

Fengying Dang, Dong Chen, Jun Chen, and Zhaojian Li. Event-triggered model predictive control with deep reinforcement learning for autonomous driving. *arXiv preprint arXiv:2208.10302*, 2022.

Felipe Rafael de Souza, Thiago Silva Miranda, and Heder Soares Bernardino. A reward function using image processing for a deep reinforcement learning approach applied to the sonic the hedgehog game. In *Intelligent Systems: 11th Brazilian Conference, BRACIS 2022, Campinas, Brazil, November 28–December 1, 2022, Proceedings, Part II*, pages 181–195. Springer, 2022.

Tim Dettmers. This is how you cite a website in latex. `https://developer.nvidia.com/blog/deep-learning-nutshell-history-training/`, 2015.

Gabriel Dulac-Arnold, Daniel Mankowitz, and Todd Hester. Challenges of real-world reinforcement learning. *arXiv preprint arXiv:1904.12901*, 2019.

Mahidhar Dwarampudi and NV Reddy. Effects of padding on lstms and cnns. *arXiv preprint arXiv:1903.07288*, 2019.

Bin Fang, Shidong Jia, Di Guo, Muhua Xu, Shuhuan Wen, and Fuchun Sun. Survey of imitation learning for robotic manipulation. *International Journal of Intelligent Robotics and Applications*, 3:362–369, 2019.

Wael Farag and Zakaria Saleh. Behavior cloning for autonomous driving using convolutional neural networks. In *2018 International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT)*, pages 1–7. IEEE, 2018.

Charles B Ferster and Burrhus Frederic Skinner. Schedules of reinforcement. 1957.

Norma M. S. Franco. Fundamentos da neurociência. *Notas de Aula Bioeletrogênese PUC-RIO*, 2020.

Justin Fu, Katie Luo, and Sergey Levine. Learning robust rewards with adversarial inverse reinforcement learning. *arXiv preprint arXiv:1710.11248*, 2017.

Bennett G Galef Jr. • imitation in animals: History, definition, and interpretation of data from the psychological laboratory. In *Social learning*, pages 15–40. Psychology Press, 2013.

BG Galef Jr, LA Manzig, and RM Field. Imitation learning in budgerigars: Dawson and foss (1965) revisited. *Behavioural Processes*, 13(1-2):191–202, 1986.

Adam Gleave, Mohammad Taufeeque, Juan Rocamonde, Erik Jenner, Steven H. Wang, Sam Toyer, Maximilian Ernestus, Nora Belrose, Scott Emmons, and Stuart Russell. imitation: Clean imitation learning implementations. arXiv:2211.11972v1 [cs.LG], 2022. URL `https://arxiv.org/abs/2211.11972`.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

Ivo Grondman, Lucian Busoniu, Gabriel AD Lopes, and Robert Babuska. A survey of actor-critic reinforcement learning: Standard and natural policy gradients. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42 (6):1291–1307, 2012.

Cecilia M Heyes. Theory of mind in nonhuman primates. *Behavioral and brain sciences*, 21(1):101–114, 1998.

Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. *Advances in neural information processing systems*, 29, 2016.

Chloe Ching-Yun Hsu, Celestine Mendler-Dünner, and Moritz Hardt. Revisiting design choices in proximal policy optimization. *arXiv preprint arXiv:2009.10897*, 2020.

Xiaowei Huang, Gaojie Jin, and Wenjie Ruan. Deep reinforcement learning. In *Machine Learning Safety*, pages 219–235. Springer, 2012.

Ludwig Huber, Friederike Range, Bernhard Voelkl, Andrea Szucsich, Zsofia Viranyi, and Adam Miklosi. The evolution of imitation: what do the capacities of non-human animals tell us about the mechanisms of imitation? *Philosophical Transactions of the Royal Society B: Biological Sciences*, 364(1528):2299–2309, 2009.

Ahmed Hussein, Mohamed Medhat Gaber, Eyad Elyan, and Chrisina Jayne. Imitation learning: A survey of learning methods. *ACM Computing Surveys (CSUR)*, 50(2):1–35, 2017.

Borja Ibarz, Jan Leike, Tobias Pohlen, Geoffrey Irving, Shane Legg, and Dario Amodei. Reward learning from human preferences and demonstrations in atari. *Advances in neural information processing systems*, 31, 2018.

Aleksei Grigorevich Ivakhnenko and Valentin Grigorevich Lapa. Cybernetic predicting devices. Technical report, PURDUE UNIV LAFAYETTE IND SCHOOL OF ELECTRICAL ENGINEERING, 1966.

Tobias Johannink, Shikhar Bahl, Ashvin Nair, Jianlan Luo, Avinash Kumar, Matthias Loskyll, Juan Aparicio Ojea, Eugen Solowjow, and Sergey Levine. Residual reinforcement learning for robot control. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6023–6029. IEEE, 2019.

Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, et al. Model-based reinforcement learning for atari. *arXiv preprint arXiv:1903.00374*, 2019.

B Ravi Kiran, Ibrahim Sobh, Victor Talpaert, Patrick Mannion, Ahmad A Al Sallab, Senthil Yogamani, and Patrick Pérez. Deep reinforcement learning for autonomous driving: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 23(6): 4909–4926, 2021.

Vijay Konda and John Tsitsiklis. Actor-critic algorithms. *Advances in neural information processing systems*, 12, 1999.

Guillaume Lample and Devendra Singh Chaplot. Playing fps games with deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

Gale L Martin and James A Pittman. Recognizing hand-printed letters and digits using backpropagation learning. *Neural Computation*, 3(2):258–267, 1991.

Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5:115–133, 1943.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Catarina Moreira. Neurónio. *Revista de Ciência Elementar*, 1(1), 2013.

Ícaro Goulart Faria Motta França Motta. Aprendizagem por reforço e por imitação com redes neurais aplicada ao jogo bomberman. *Universidade Federal Fluminense*, 2019.

Muhammad Waqas Nadeem, Hock Guan Goh, Abid Ali, Muzammil Hussain, Muhammad Adnan Khan, and Vasaki a/p Ponnusamy. Bone age assessment empowered with deep learning: a survey, open research challenges and future directions. *Diagnostics*, 10 (10):781, 2020.

Andrew Y Ng, Stuart Russell, et al. Algorithms for inverse reinforcement learning. In *Icml*, volume 1, page 2, 2000.

Hai Nguyen and Hung La. Review of deep reinforcement learning for robot manipulation. In *2019 Third IEEE International Conference on Robotic Computing (IRC)*, pages 590–595. IEEE, 2019.

Alex Nichol, Vicki Pfau, Christopher Hesse, Oleg Klimov, and John Schulman. Gotta learn fast: A new benchmark for generalization in rl. *arXiv preprint arXiv:1804.03720*, 2018.

Fernando Nogueira. Modelagem e simulação-cadeias de markov. *Notas de Aula UFJF Juiz de Fora-2008*, 2008.

Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation functions: Comparison of trends in practice and research for deep learning. *arXiv preprint arXiv:1811.03378*, 2018.

Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*, 2015.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

IP Pavlov. The work of the digestive glands (wh thompson, trans.) london: Griffin. *Original work published in*, 1897.

Tim Pearce and Jun Zhu. Counter-strike deathmatch with large-scale behavioural cloning. In *2022 IEEE Conference on Games (CoG)*, pages 104–111. IEEE, 2022.

Jerônimo Pellegrini and Jacques Wainer. Processos de decisão de markov: um tutorial. *Revista de Informática Teórica e Aplicada*, 14(2):133–179, 2007.

Yanfei Peng, Guozhen Tan, and Huaiwei Si. Hkgail: Policy shaping via integrating human knowledge with generative adversarial imitation learning. *IET Intelligent Transport Systems*, 2022.

Leonid Peshkin and Virginia Savova. Reinforcement learning for adaptive routing. In *Proceedings of the 2002 International Joint Conference on Neural Networks. IJCNN'02 (Cat. No. 02CH37290)*, volume 2, pages 1825–1830. IEEE, 2002.

Martin L Puterman. Markov decision processes. *Handbooks in operations research and management science*, 2:331–434, 1990.

Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021. URL `http://jmlr.org/papers/v22/20-1364.html`.

Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

Stuart J Russell. *Artificial intelligence a modern approach*. Pearson Education, Inc., 2010.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

Jiaming Song, Hongyu Ren, Dorsa Sadigh, and Stefano Ermon. Multi-agent generative adversarial imitation learning. *Advances in neural information processing systems*, 31, 2018.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

Yoshihisa Tsurumine, Yunduan Cui, Kimitoshi Yamazaki, and Takamitsu Matsubara. Generative adversarial imitation learning with deep p-network for robotic cloth manipulation. In *2019 IEEE-RAS 19th International Conference on Humanoid Robots (Humanoids)*, pages 274–280. IEEE, 2019.

Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575 (7782):350–354, 2019.

Ruoyao Wang, Peter Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. Behavior cloned transformers are neurosymbolic reasoners. *arXiv preprint arXiv:2210.07382*, 2022.

Paul Werbos. Beyond regression: New tools for prediction and analysis in the behavioral sciences. *PhD thesis, Committee on Applied Mathematics, Harvard University, Cambridge, MA*, 1974.

Wikipedia, the free encyclopedia. Diagrama mlp, 2017. URL `https://commons.wikimedia.org/wiki/File:Neural_Network.gif`. [Online; Acessado 25 de fevereiro, 2023].

Qingyang Wu, Lei Li, and Zhou Yu. Textgail: Generative adversarial imitation learning for text generation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 14067–14075, 2021.

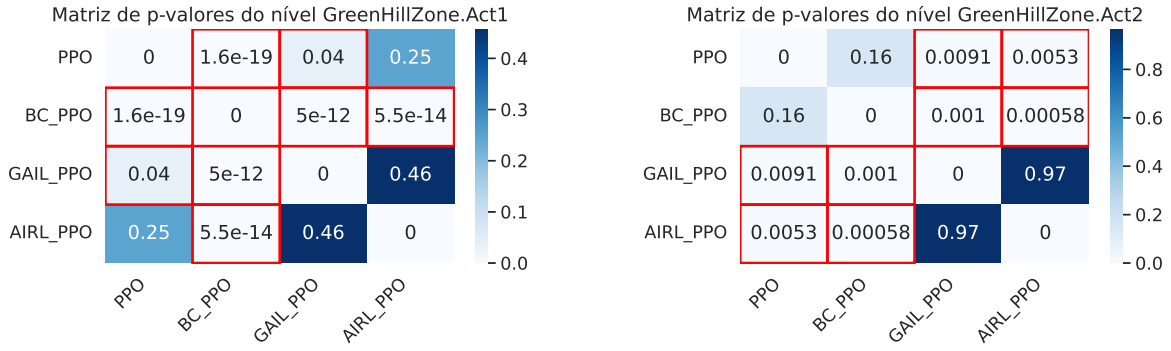
Hongyang Yang, Xiao-Yang Liu, Shan Zhong, and Anwar Walid. Deep reinforcement learning for automated stock trading: An ensemble strategy. In *Proceedings of the first ACM international conference on AI in finance*, pages 1–8, 2020.

Thomas R Zentall. An analysis of imitative learning in animals. *Social learning in animals: The roots of culture*, pages 221–243, 1996.

Shao Zhifei and Er Meng Joo. A survey of inverse reinforcement learning techniques. *International Journal of Intelligent Computing and Cybernetics*, 5(3):293–311, 2012.

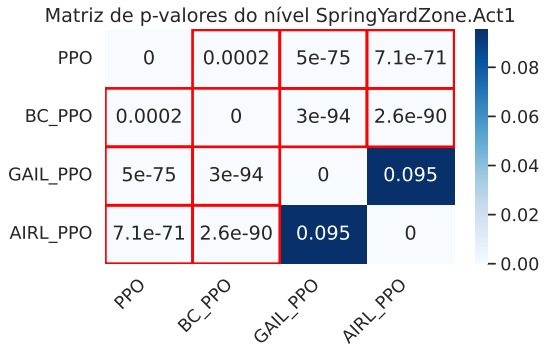
S Kevin Zhou, Hoang Ngan Le, Khoa Luu, Hien V Nguyen, and Nicholas Ayache. Deep reinforcement learning in medical imaging: A literature review. *Medical image analysis*, 73:102193, 2021.

APÊNDICE A – Teste Estatístico de Mann–Whitney Para Comparação dos Modelos

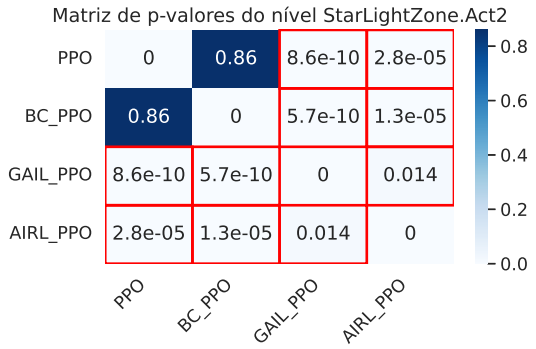


(a) Green Hill Zone, Ato 1

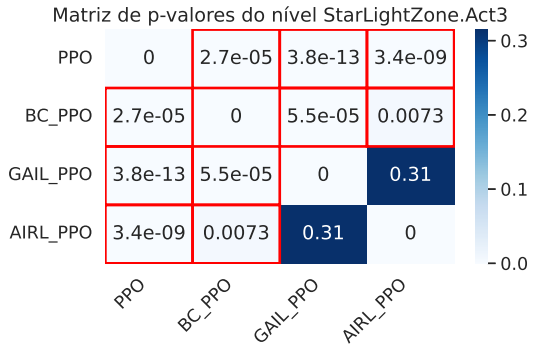
(b) Green Hill Zone, Ato 2



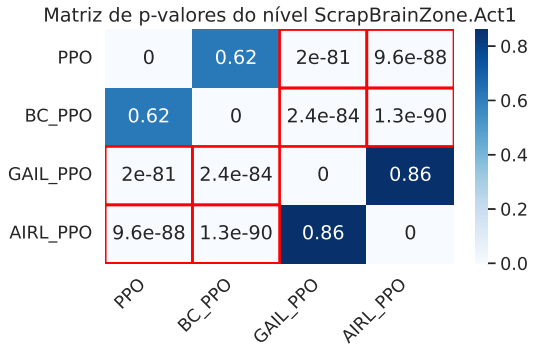
(c) SpringYard Zone, Ato 1



(d) StarLight Zone, Ato 2

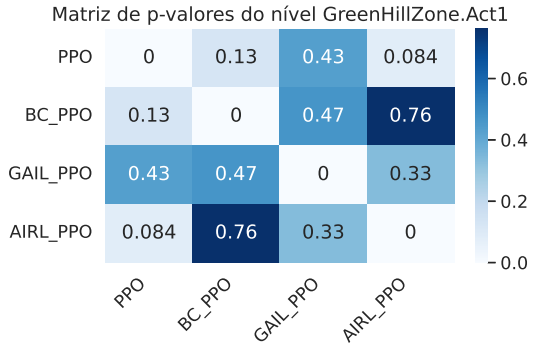


(e) StarLight Zone, Ato 3

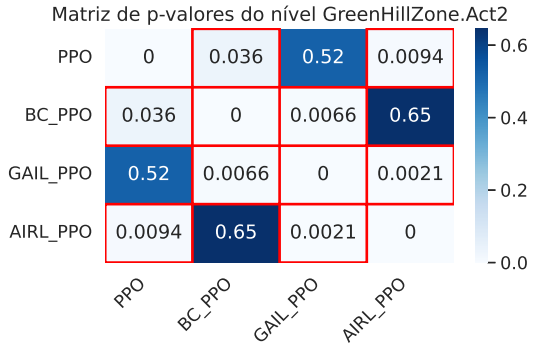


(f) ScrapBrain Zone, Ato 1

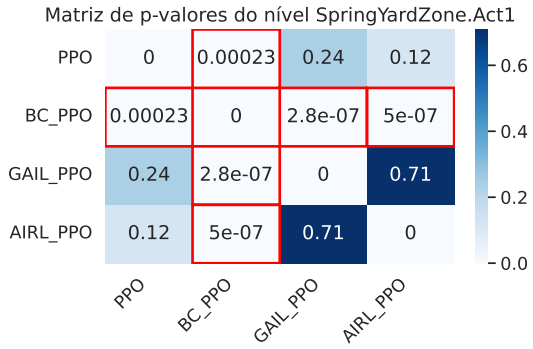
Figura 24 – Figura mostrando os p-valores entre os modelos de Aprendizado por Reforço iniciados com o Aprendizado por Imitação que utilizam políticas baseadas em CNN. Destacado em vermelho os p-valores menores que 0.05



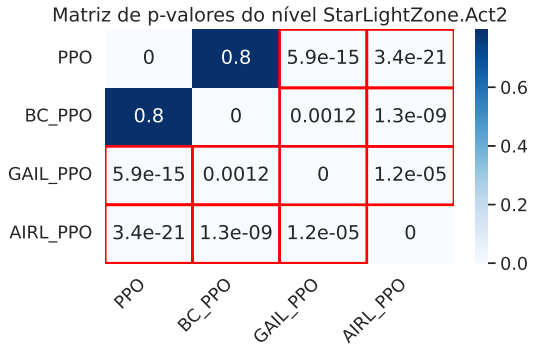
(a) Green Hill Zone, Ato 1



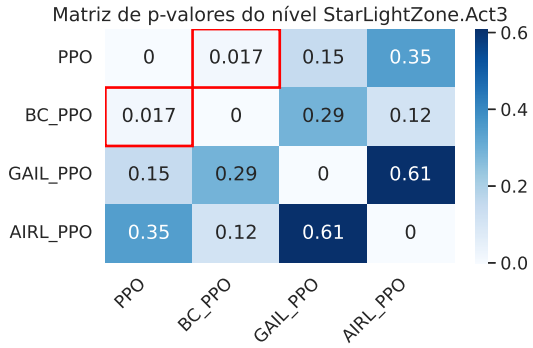
(b) Green Hill Zone, Ato 2



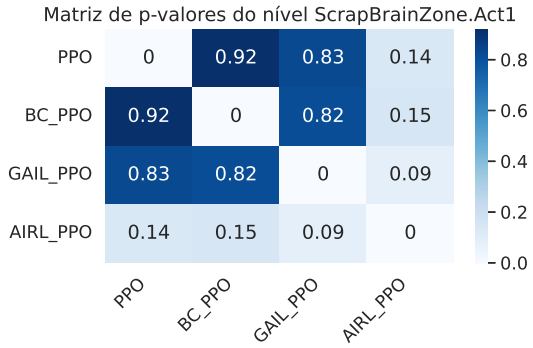
(c) SpringYard Zone, Ato 1



(d) StarLight Zone, Ato 2



(e) StarLight Zone, Ato 3



(f) ScrapBrain Zone, Ato 1

Figura 25 – Figura mostrando os p-valores entre os modelos de Aprendizado por Reforço iniciados com o Aprendizado por Imitação que utilizam políticas baseadas em MLP. Destacado em vermelho os p-valores menores que 0.05