Politécnico de Leiria

Escola Superior de Tecnologia e Gestão

Departamento de Engenharia Eletrotécnica

Mestrado em Engº Eletrotécnica - Energia e Automação

COMPARAÇÃO DE MÉTODOS DE APRENDIZAGEM POR REFORÇO EM PROCESSOS INDUSTRIAIS DISCRETOS SEQUENCIAIS

Tiago Resende da Silva

Leiria, março de 2023

Politécnico de Leiria
Escola Superior de Tecnologia e Gestão
Departamento de Engenharia Eletrotécnica
Mestrado em Engº Eletrotécnica – Energia e Automação

COMPARAÇÃO DE MÉTODOS DE APRENDIZAGEM POR REFORÇO EM PROCESSOS INDUSTRIAIS DISCRETOS SEQUENCIAIS

Tiago Resende da Silva

Número: 2200086

Dissertação realizada sob a orientação do Doutor Luís Miguel Ramos Perdigoto (luis.perdigoto@ipleiria.pt) e do Doutor Luís Manuel Conde Bento (luis.conde@ipleiria.pt).

Leiria, março de 2023

Nota: Este documento foi escrito de acordo com as práticas idiomáticas, gramaticais e ortográficas do Português do Brasil. O autor fez todos os esforços para garantir que o idioma, o estilo e a pontuação sejam consistentes com as convenções do Português Europeu.

“Mares calmos não fazem um bom marinheiro.”

Franklin D. Roosevelt

Agradecimentos

Primeiramente, aos meus orientadores Dr. Luís Miguel Ramos Perdigoto e Dr. Luís Manuel Conde Bento que me derem suporte, orientação e condução na realização deste estudo, além de me apresentaram este tema de alta relevância e promissor no campo do controlo de processos industriais.

Aos demais professores do Politécnico de Leiria, pelos conhecimentos transmitidos, abrindo portas para novas oportunidades profissionais, criando um diferencial na minha carreira.

À minha esposa, que se manteve companheira durante esta longa jornada de pouco mais de dois anos e compreensiva durante minha ausência, sempre me incentivando e apoiando a concluir essa relevante etapa da minha vida.

Por fim, aos meus amigos, que me apoiaram, deram-me auxílio nas questões mais difíceis no transcorrer do mestrado e por terem tornado minha permanência em Leiria mais familiar.

Resumo

O processo de comissionamento de equipamentos durante a implementação de um novo sistema, ou na reconfiguração de um já existente, é uma etapa em que as empresas gastam dinheiro e tempo antes da entrada em operação. Baseando-se nesse problema, este trabalho analisa a utilização de Gémeos Digitais, que simulem o ambiente fabril, em conjunto com a utilização de técnicas de Aprendizagem por Reforço para permitir que o sistema se reconfigure e se programe de forma automática. Diferentemente das técnicas tradicionais de controlo, a Aprendizagem por Reforço encara o sistema como uma caixa negra, em que a interação entre o agente e o ambiente promove a sintonização dos parâmetros necessários para o funcionamento correto do processo industrial. Isso resulta na economia de dinheiro, na diminuição de tempo de produção e na busca de inúmeras possibilidades de operação até que se encontre a mais eficiente. Acrescenta-se a tudo isso a diminuição do tempo de exposição de pessoas ao processo de implementação inicial e consequente diminuição de acidentes, uma vez que o comissionamento ocorre no ambiente virtual.

Dessa forma, este trabalho desenvolve e aplica alguns dos diferentes algoritmos de *Deep Reinforcement Learning* a um sistema de empacotamente de latas. O principal objetivo é avaliar a viabilidade e o desempenho de utilizar estes tipos de algoritmos na aprendizagem e otimização automática das sequências de controlo em processos industriais de natureza sequencial e discreta. Dada a natureza sequencial dos processos, com necessidade inerente de efeito de memória, foram experimentadas diferentes arquiteturas de redes neuronais, realizando um estudo comparativo sobre a performance das redes neuronais LSTM (*Long Short-Term Memory*) frente à utilização de *buffers* de memória de estados anteriores de diferentes tamanhos. Por fim, os modelos com melhores resultados e maior estabilidade foram aplicados aos Gémeos Digitais, mostrando a capacidade que estes tipos de algoritmos de aprendizagem automática têm para ser aplicados no controlo de sistemas industriais.

Palavras-chave: Aprendizagem por Reforço, LSTM, controlo discreto de sistemas, Gémeos Digitais.

Abstract

The equipment commissioning process during the implementation of a new system or the reconfiguration of an existing one is a step which companies spend money and time before starting operations. Based on this problem, this paper analyses the use of Digital Twins, that simulate the factory environment, together with Reinforcement Learning techniques to enable the system to automatically reconfigure and program itself. Unlike traditional control techniques, Reinforcement Learning sees the system as a black box, where the interaction between the agent and the environment causes the tuning of the necessary parameters for the correct operation in the industrial plant. This results in monetary savings and reduction of production time by searching a great number of possibilities of operation until the most efficient one is found. Besides that, there is a reduction in the exposure time of individuals to the initial implementation process and consequent reduction of accidents, because commissioning occurs in the virtual environment.

In this way, this dissertation develops and applies some of the different Deep Reinforcement Learning algorithms to a can packaging process. The main objective is to evaluate the feasibility and performance of using these types of algorithms in the automatic learning and optimization of control sequences in industrial processes of sequential and discrete nature. Given this sequential nature, that inherently requires a memory effect, different neural network architectures were experimented, conducting a comparative study on the performance of LSTM (Long Short-Term Memory) neural networks against using memory buffers of previous states with different sizes. Finally, the models with the best results and greater stability were applied to Digital Twins, showing the applicability of that class of machine learning algorithms in the control of industrial systems.

Keywords: Reinforcement Learning, LSTM, discrete control of systems, Digital Twin.

Índice

Índice de Figuras

Índice de Tabelas

Lista de Abreviaturas e Siglas

A2C *Advantage Actor Critic*

A3C *Asynchronous Advantage Actor Critic*

ANN *Artificial Neural Network*

CNN *Convolutional Neural Network*

CPU *Central processing Unit*

DDPG *Deep Deterministic Policy Gradient*

DBN *Deep Belief Network*

DP *Dynamic Programming*

DQN *Deep Q-Network*

DT *Digital Twin*

ECL Esteira de caixa de latas

GAN *Generative Adversarial Network*

GML Garra manipuladora de latas

GPU *Graphics Processing Unit*

IMC *Internal Model Control*

KL *Kullback-Leibler Divergence*

LSTM *Long Short-Term Memory*

MDP *Markovian Decision Process*

NARX *Nonlinear Autoregressive Exogenuous Model*

NLP *Natural Language Processing*

PPO *Proximal Policy Optimization*

RL *Reinforcement Learning*

RNN *Recurrent Neural Network*

SAC *Soft Actor Critic*

SARSA *State-action-reward-state-action*

SOM *Self Organazing Map*

TD *Temporal Difference*

TRPO *Trust Region Policy Optimization*

1. Introdução

Embora as ideias relativas à Aprendizagem por Reforço fossem introduzidas a partir da década de 40, somente nos últimos 20 anos que algoritmos dessa natureza começaram a ser implementados com maior frequência. Já em 1957, Richard Bellman, matemático estadunidense, publicou o livro *Dynamic Programming* (Bellman, 1957) que introduziu a famosa equação que leva o seu nome, *Bellman equation*, e fez uso das ideias do *Markov decicion process* (MDP) utilizado por Andrei Markov já no início do século. O que se buscava à época era a solução de problemas complexos económicos ou derivados da Teoria dos Jogos, em que a função matemática de otimização era presente.

Como o desenvolvimento computacional ainda era precoce no início da segunda metade do século XX, em que computadores possuiam baixo processamento e pouca memória, havia muita pesquisa no campo da Programação Dinâmica para desenvolvimento de métodos que pudessem solucionar problemas computacionais de forma eficiente e com baixo custo computacional.

Em paralelo, é também nessa época que se começa a "humanizar" o comportamento computacional, por meio da aprendizagem por tentativa e erro. Nesse contexto, podem ser citados os conceitos de recompensa e punição já descritos por Allan Turing no processo de aprendizagem de máquina (Turing, 1950), em que procedimentos indevidos deveriam ser punidos por um sinal de punição, enquanto uma recompensa aumentaria a probabilidade de procedimentos corretos. Por sua vez, em 1951, Marvin Minsky cria o primeiro neuro computador, chamado SNARC, que possuia 40 sinapses hebbianas e eram capazes de ajustar os pesos neuronais automaticamente (Kelemen, 2007). Os termos "*reinforcement"* e "*reinforcement learning" (*Aprendizagem por Reforço*)* começam a ser descritos na década de 60, podendo citar como exemplo o artigo de Minsky, "*Steps Toward Artificial inteligence*" (Minsky, 1961), onde além de descrever o "operador de reforço", Minsky trata das caracteríticas de previsões e expectativas da máquina sob aprendizagem.

Já os métodos de aprendizagem por Diferença Temporal foram introduzidos por Ian Witten (Witten, 1977), aplicados em controles adaptivos e que por conseguinte deu origem ao desenvolvimento de métodos de Q-learning (Watkins, 1989) e SARSA (Rummery, 1994). Estes algoritmos Q-learning e SARSA são a base dos algoritmos de "*reinforcement learning*" aplicados atualmente.

Ainda que as redes neuronais passem a ser grande objeto de estudo e ocorram avanços nesse setor na década de 80, com a concepção do algoritmo de *backpropagation* (Rumelhart et al., 1986) e (LeCun et al., 1989), somente a partir dos anos 2000 é que são verdadeiramente usadas em grande número de casos e aplicadas significativamente no ambiente computacional. Isso porque o processamento computacional expandiu-se exponencialmente a partir daí, proporcionando maior velocidade na execução de algoritmos de Aprendizagem por Reforço combinados com redes neuronais. Identificou-se também que a programação paralela, com utilização de placas gráficas GPU's, poderia possibilitar a realização de operações matemáticas de maneira mais rápida e eficiente, especialmente com vetores e matrizes.

É também a partir dos anos 2000 que o termo "*Digital Twin"* (Gémeo Digital) começa a ocorrer com frequência, quando começa a incidir no produto físico, produto virtual e na suas conexões (Tao et al., 2019) . Em 2012, a NASA revisou esse conceito, definindo *Digital Twin* como uma simulação multifísica, multiescala, probabilística e ultrafiel que refletia, de maneira oportuna, o estado do gémeo correspondente baseado em informações históricas, sensores de tempo real e modelo físico (Glaessgen and Stargel, 2012). Desde então há vários entendimentos sobre Digital Twin (DT), mas em todos o ponto comum está num modelo virtual que reflita as características de um produto. O DT é um conceito com crescente importância no âmbito da quarta revolução industrial, a "Indústria 4.0".

1.1. Motivação

A realização de controlo automático de processos é algo rotineiro e necessário no ambiente industrial. Eficiência produtiva é o que diferencia empresas quando os custos e a receita de uma companhia são levadas em consideração, sendo que controlos mais eficazes levam a uma maior rentabilidade empresarial.

Os processos de comissionamento, manutenção e reconfiguração de equipamentos existentes são geradores de custos, uma vez que requerem mão-de-obra qualificada, necessitam de tempo, expõem pessoas a riscos, podendo gerar contratempos e situações inusitadas para as companhias. Mais, possuem o agravante de interromper a produção para regulação desses equipamentos, diminuindo o lucro final. (Vinyals et al., 2017)

Este trabalho avalia diferentes algoritmos de Aprendizagem por Reforço aplicados à automatização do controlo de processos, recorrendo à simulação de tarefas industriais em Gémeos Digitais/Simuladores, de forma que a intervenção humana seja minimizada.

A partir de 2010, algoritmos de Aprendizagem por Reforço têm sido implementados com maior frequência em diferentes cenários da digitalização, nomeadamente em jogos (Vinyals et al., 2017), economia (Mosavi et al., 2020), pesquisas biológicas (Neftci and Averbeck, 2019) e no controlo automático de máquinas e robôs (Nguyen and La, 2019). A habilidade de adequação a diferentes ambientes, o reconhecimento de diferentes padrões de estados, a adaptação a mudanças de ambiente e a extrapolação de estratégias de políticas de ações, com por exemplo no desempenho excepcional no jogo AlphaGo Zero (Silver et al., 2016), são os principais fatores que motivam pesquisadores e que motiva este trabalho a utilizar essa classe de algoritmos.

Conceber Gémeos Digitais/Simuladores em realidade virtual para simulação desses processos industriais são cruciais para o ganho de tempo no treino e avaliação de algoritmos, evitando-se acidentes, interrupções e custos que tornariam tais práticas inviáveis em ambiente real. Além disso, o ambiente virtual permite a experimentação de diferentes cenários e visualizações de problemas que em um processo industrial poderia não ser visível.

1.2. Objectivos

Pretende-se pesquisar diferentes algoritmos de Aprendizagem por Reforço e redes neuronais investigando o estado do arte, no requesito de métricas de performance, seja em controlos industriais, seja em outros contextos, de maneira que possam ser aplicados aos processos automáticos descritos neste trabalho, com o objetivo de obter através de aprendizagem automática as sequências de controlo desses processos

Pretende-se também desenvolver tais algoritmos, integrando-os em Gémeos Digitais/Simuladores, de maneira que a comunicação entre o agente e o ambiente virtual ocorra com integridade e segurança.

Finalmente, será avaliado o desempenho desse desenvolvimento, realizando principais conclusões acerca da aplicação de Aprendizagem por Reforço no controlo automático de processos industriais discretos sequenciais.

Essas implementações podem ser encontradas no seguinte repositório do GitHub: https://github.com/ipleiria-robotics/ARPIDS

1.3. Recursos Utilizados

Todo trabalho foi desenvolvido por meio de um computador com processador Intel Core I7-6500, 2,50 Ghz, 8 GB de memória ram e placa de vídeo Nvidia GeForce 940MX, que conta com 1,25 Ghz de processamento dedicado a vídeo.

Foram usados os *frameworks* Jupyter e Visual Studio para desenvolvimento dos algoritmos em conjuntos com as bibliotecas Keras (Deep Reinforcement Learning for Keras, 2023) e Stable Baselines 3 (Stable Baselines3, 2023), que permitem a criação dos modelos em Aprendizagem por Reforço. Além disso, para maior rapidez no processamento dos dados de treinamento, foi instalada a API CUDA 11.7, que integra a placa de vídeo com o processador da placa mãe, facilitando os cálculos matriciais recorrentes nesse processo de aprendizagem.

Por fim, os simuladores foram desenvolvidos no *framework* Unity, que permite a criação de realidades virtuais em gráficos de 3 dimensões com qualidade, tornando a experiência do usuário mais próxima do real.

1.4. Estrutura da dissertação

A dissertação possui 6 capítulos. O primeiro capítulo introduz o tema, contextualizando-o historicamente, identificando aplicações, usos, pesquisas e o porquê do aumento significativo dessas técnicas na atualidade. Aqui são descritos os recursos utilizados, as principais motivações e os principais objectivos que justificaram a realização deste trabalho.

O segundo capítulo realiza o enquadramento teórico das técnicas e algoritmos apresentados, descrevendo as arquiteturas de redes neuronais, os algoritmos de Aprendizagem por Reforço e os Gémeos Digitais.

O terceiro capítulo investiga o estado da arte sobre o tema, mostrando exemplos de aplicações no controlo de processos industriais e em Gémeos Digitais.

O quarto capítulo apresenta dois processos industriais que são desenvolvidos e concebidos como Gémeos Digitais/Simuladores para aplicação de Aprendizagem por Reforço: a esteira de caixa de latas (ECL) e a garra manipuladora de latas (GML).

O quinto capítulo descreve os treinos, testes e resultados obtidos da aplicação desses modelos nesses processos, avaliando e comparando os resultados dos diferentes algoritmos no controlo dos processos industriais desenvolvidos nos Gémeos Digitais/Simuladores.

No sexto e último capítulo são apresentadas as conclusões da aplicação da Aprendizagem por Reforço no controlo de processos industriais discretos sequenciais e apresenta possibilidades de estudo e de trabalhos promissores nesse assunto para futuro desenvolvimento.

2. Enquadramento Teórico e Tecnologias de Suporte

Este capítulo conceitua e descreve os principais métodos, modelos, arquiteturas, ferramentas e métricas de performance aplicadas ao processo industrial proposto neste trabalho. O capítulo explica o funcionamento e os elementos das redes neuronais, a memorização de dados e aprendizagem, os métodos de treinamento por Aprendizagem por Reforço, e finaliza com os sistemas de simulação de processos industriais, nomeadamente Gémeos Digitais.

2.1. Neurónio e Redes Neuronais

As redes neuronais são modelos computacionais inspirados em neurónios capazes de reconhecer padrões escondidos, correlacionar dados, classificá-los e agrupá-los, estabelecendo aí um aprendizado de máquina. Como nos neurónios biológicos, McCullock e Pitts propôem um modelo onde as sinapses (região de proximidade entre o neurónio e uma célula onde ocorre o impulso nervoso) são modeladas por pesos matemáticos que quando multiplicados pelas entradas resultam em valores de saída para o corpo celular seguinte (Mcculloch and Pitts, 1943). Os valores resultantes dos pesos são adicionados em uma função de ativação que, ao ser comparada com um threshold (bias), determinará se a saída será ativada ou não, resultando em um valor final, conforme figura 1 abaixo:

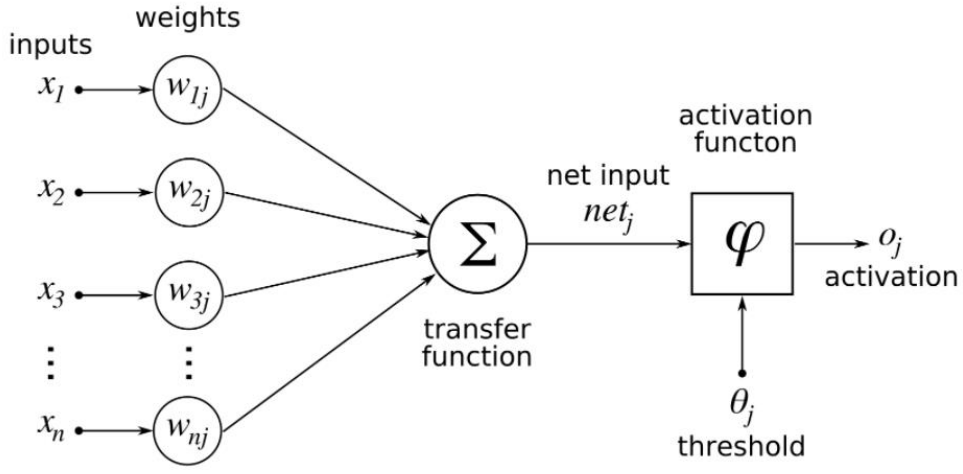

Figura 1: Neurónio artificial proposto por McCullock e Pitts (Artificial Neural Networks, 2021)

Este modelo pode ser descrito pela pela equação 2.1, onde onde oj é o sinal de saída, ϕ é a função de ativação, θj é o *threshold (bias)*, $x_{1,2\ldots n}$ são os sinais de entrada e $w_{1,2,\ldots n}$ são os pesos sinápticos.

$$oj = \phi\left(\theta j + \sum_{i=1}^{n} x_i\, w_i\right) \tag{2.1}$$

Cada neurónio é então rearranjado em estruturas de camadas e interligados em conjunto formando uma arquitetura genérica de rede neuronal. Na rede mostrada na figura 2 abaixo, cada neurónio é mostrado por um círculo, representando um nó de interligação, onde os pesos estão implícitos nas ligações.

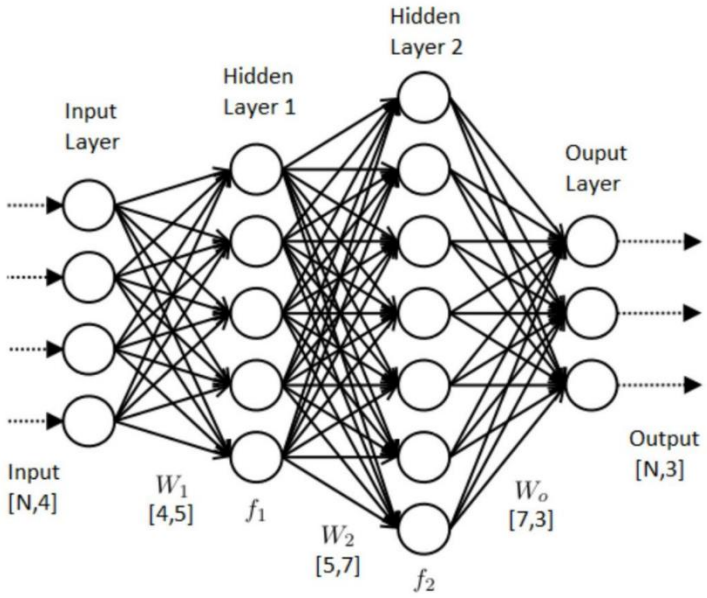

Figura 2: Rede Neuronal Feedforward (Artificial Neural Networks, 2021)

As funções de ativação possuem um papel fundamental no treinamento das redes neuronais, pois o gradiente de cada uma determina a regulação de cada neurônio. Se a função de ativação não fosse usada na rede neuronal, a saída simplesmente se comportaria como uma função polinomial de primeiro grau, comprometendo a capacidade de aprendizagem da rede de aprender e reconhecer informações complexas. Além disso, se a função de ativação possui pontos onde o gradiente é nulo, surgirá então o efeito do "neurônio morto", que fará pouco efeito na ativação da saída e na aprendizagem do modelo. Nesse âmbito, podem-se citar diferentes funções de ativação, cada uma com diferentes comportamentos na rede: Sigmoide, Tangente Hiperbólica (Tanh), Unidade de Retificação Linear (ReLU), Unidade de Retificação Linear Vazada (*Leaky* ReLU), Unidade Exponencial Linear (ELU), Sigmoide

com Unidade Linear Podenderada (Swiss), Unidade Exponencial Normalizada (Softmax). O Apêndice A apresenta uma descrição mais detalhada dessas funções de ativação, mostrando as principais características, o gradiente da função, as principais vantagens e desvantagens no uso de cada uma.

2.2. Aprendizagem supervisionada e não supervisionada

Em uma rede neuronal, quando o sinal de entrada propaga por dois ou mais nós antes de atingir a saída, não formando circuitos fechados internamente, a rede é chamada de *feedforward*. Já quando sinal trafega da saída para entrada, no momento de aprendizagem da rede e correções dos pesos dos neurónios, o processo é chamado de *backpropagation* (Rumelhart et al., 1986). Nesse treino, pretende-se corrigir os pesos da rede de modo que o vetor de saída obtido se aproxime do vetor de saída desejado. Para isso, a correção é feita por meio da redução escalar negativa da derivada do erro em função de cada peso dos neurónios. Essa derivação pode ser feita pela aplicação elementar da diferenciação ou por meio do gradiente descendente estocástico, o mais comumente usado.

Treinada a rede, o modelo está pronto para atuar em um determinado problema. Nesse caso, houve aqui aprendizagem supervisionada, pois foi necessário que um conjunto de dados de entrada e saída fosse estabelecido para que a rede fosse corretamente treinada. Já na aprendizagem não supervisionada, somente os dados de entrada são apresentados, ocorrendo uma busca pela otimização da função objetivo, ou a classificação dos dados em grupos ou rótulos não pré-determinados, ou seja, não há nesse viés *Backpropagation*, mas somente o processo *feedforward* (Dike et al., 2018) (Hastie et al., 2009).

Dike e Zhou (2018) em seu artigo defende que ocorre também aprendizagem nesse processo *feedforward*, nomeadamente aprendizagem *Hebbian*. Segundo os autores, após massiva inserção dos dados de entrada na rede, o sistema adquire conhecimento por conta própria, aprendendo e familiarizando-se com as informações vitais dos dados de entrada. Essa teoria afirma que um aumento na eficácia sináptica surge da estimulação repetida e persistente de uma célula pós-sináptica por uma célula pré-sináptica *(An Experiment with the Edited Nearest-Neighbor Rule, 1976).* Assim a mudança no peso dos neurónios é proporcional ao produto das funções das atividades em ambos os lados das conexões, sem que haja qualquer informação nas saídas.

Por fim, mais adiante, no capítulo 2.4, este trabalho descreve o terceiro viés da aprendizagem de máquina: a Aprendizagem por Reforço, que é o foco desta dissertação aplicado ao controlo de processos industriais.

2.3. Arquiteturas de Redes Neuronais

O número de arquiteturas e algoritmos desenvolvidos ao longo dos últimos 20 anos são grandes e variados e tem contribuído para o desenvolvimento de modelos com diferentes objetivos e perfis. Variações nas estruturas e composições dão diferentes características às redes, tais como o efeito de memória de curto e longo prazo (Hochreiter and Schmidhuber, 1997), interpretações subjetivas de dados (Borele and Borikar, 2016) e aplicações no campo de interpretação de áudio (Wyse, 2017), imagens ou um grande conjunto de dados genéricos (Krizhevsky et al., 2012). Algumas trabalham somente com treinamento de *feedforward*, enquanto outras utilizam *backpropagation* para ajuste de neurônios. Alguns neurônios são retroalimentados ou interligados a camadas anteriores para que lhes confiram características temporais, nomeadamente Redes Neuronais Recorrentes (RNN), estas serão objeto de estudo no próximo capítulo. O apêndice B apresenta outras estruturas de redes neuronais utilizadas em aprendizagem de máquina, nomeadamente *Auto Encoders*, *Deep Belief Networks* (DBN), *Generative Adversarial Networks* (GAN) e *Self-Organizing Maps* (SOM).

Redes Neuronais Recorrentes (RNN)

A aprendizagem de máquina muitas vezes necessita memorizar dados sequenciais para o bom entendimento do processo a ser modelado. A série temporal de dados tem impacto na interpretação completa da informação que pode vir em forma de imagens, textos e vídeos. No processo industrial, cada passo executado pela máquina pode ser dependente do passo anterior para que haja a correta jornada do processo como um todo.

Diante da necessidade de avaliação dos passos anteriores, a partir 1986 surgiram estudos no âmbito das redes neuronais com o perfil de memorização, nomeadamente (Paccanaro and Hinton, 2001) e (Jordan, 1986). O primeiro realiza um estudo textual das palavras em relação a ordem como elas ocorrem em frases e alteram o sentido do contexto das ideias. O autores elaboram um sistema vetorial de conceitos que abre espaço para futuros trabalhos em *Natural Language Processing* NLP. Já o segundo estudo, introduzido por Jordan, apresenta a primeira arquitetura de Redes Neuronais Recorrentes (RNN), onde neurónios da camada

de saída alimentam unidades especiais (também chamadas de unidades de contexto), que por sua vez se retro-alimentam e alimentam a camada oculta da arquitetura. A arquitetura de Jordan é apresentada abaixo pela figura 3:

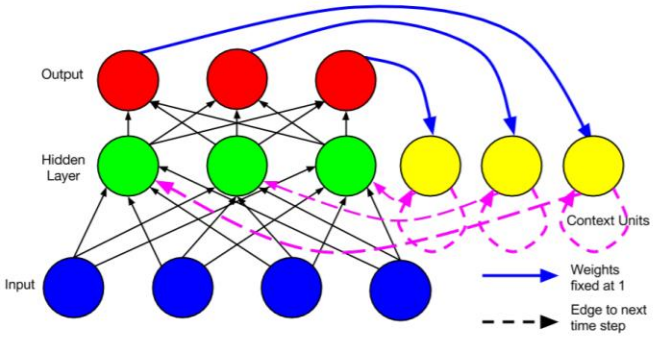

Figura 3: Rede Neural Recorrente proposta por Jordan (Jordan, 1986)

Essas unidades especiais, que possuem alimentação recorrente, atuam sobre a camada oculta e criam um efeito memória na arquitetura como um todo, sem que haja perturbação das saídas a cada passo intermediário de tempo.

Este modelo pode ser descrito pelas equações 2.2 e 2.3, onde h_t é o sinal de saída da unidade de contexto, ϕ_h, θ_h é a função de ativação e o *threshold* dessa camada de memorização, $x_{1,2\ldots n}$ são os sinais de saída da camada de saída, $w_{1,2,\ldots n}$ são os pesos sinápticos dessa camada no tempo corrente, y_{t-1} é o sinal de saída da camada de contexto no instante *t – 1*, $u_{1,2,\ldots n}$ são os pesos sinápticos da camada das unidades de contexto também no instante *t – 1* e ϕ_y, θ_y é a função de ativação e o *threshold* da camada de saída.

$$h_t = \phi_h\left(\theta_h + \sum_{i=1}^{n} x_i\, w_i + y_{t-1} u_i\right) \tag{2.2}$$

$$y_t = \phi_y\left(\theta_y + \sum_{i=1}^{n} h_i\, w_i\right) \tag{2.3}$$

Por sua vez, (Lipton et al., 2015) apresentou uma arquitetura simplificada daquilo que foi produzido por Jordan, onde as unidades de contexto recebem os sinais vindos da camada oculta, ao invés da camada de saída, e a camada oculta deixa de ser retro-alimentação (ver figura 4). Assim as equações 2.4 e 2.5 descrevem o modelo de Elman:

$$h_t = \phi_h\left(\theta_h + \sum_{i=1}^{n} x_i\, w_i + h_{t-1} u_i\right) \tag{2.4}$$

$$y_t = \phi_y\left(\theta_y + \sum_{i=1}^{n} h_i\, w_i\right) \tag{2.5}$$

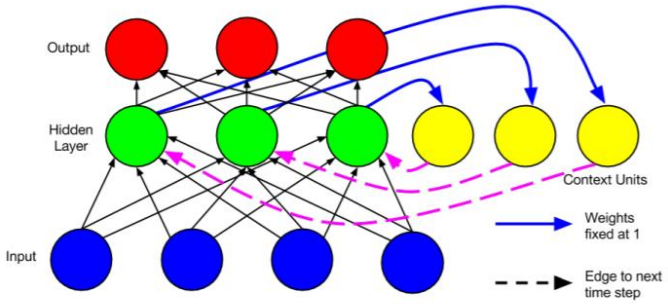

Figura 4: Rede Neural Recorrente proposta por Elman (Lipton et al., 2015)

Essa diferença nas auto conexões garante características diferentes nas duas redes. Uma vez que não há perturbação da saída em cada etapa de tempo intermediária no modelo de Jordan, essa arquitetura é preferencial no planeamento de ações, permitindo que a rede seja capaz de se lembrar das ações tomadas nas etapas anteriores. Já o modelo de Elman permite que as saídas sejam alteradas conforme o peso dos neurônios na camada temporal a cada interação, dando assim uma característica de uma rede dependente de um tempo de longa duração.

Dois problemas podem ocorrer nas redes neuronais recorrentes: a explosão ou banimento do gradiente nas saídas das funções de ativação nas unidades de conexão. Isso porque saídas de sinais pequenos, com números decimais, quando multiplicados na recorrência dessa camada, levam a um resultado que tende exponencialmente a zero. Já quando o sinal possui valor elevado, o oposto ocorre, tendendo o resultado a um número exponencialmente alto, ou seja, uma explosão do gradiente. No intuito de evitar esse problema, o próximo capítulo descreve as redes de memória de longo e curto prazo e que evitam tanto o banimento, como a explosão do gradiente (explicações no Apêndice B).

Redes de Memória de Longo e Curto Prazo (LSTM)

Os problemas de banimento e explosão de gradiente (explicações no Apêndice B) nas funções de ativação afetam bastante as RNN's. Consequentemente, os modelos de Jordan e Elman possuem pior desempenho nas informações de longo prazo, uma vez que estão mais

sucetíveis a erros na multiplicação de valores na recorrência nas unidades de contexto. A fim de resolver esse problema e ter um melhor desempenho nas informações de longo prazo, (Hochreiter and Schmidhuber, 1997) desenvolvelram um método chamado de Redes de Memória de Longo Curto Prazo (LSTM) que, em comparação com as RNN's anteriores, aprendem mais rápido e possuem melhor performance.

As LSTM's consistem em células de memória que retém informações de longo prazo (c_t) e curto prazo (h_t) e se dividem em três blocos de processamento de informação para memorização: *forget gate*, *input gate*, *output gate* (ver figura 5).

O bloco *forget gate* recebe o sinal de entrada da célula, adiciona-o à memória de curto prazo do passo anterior (h_{t-1}), seu resultado é multiplicado à função de ativação sigmoide e adicionado ao *threshold*. O valor resultante é então aplicado ao *gate* que controla a porcentagem da passagem de memória de longo prazo, permitindo à célula esquecer ou não informações de passos mais distantes.

O segundo bloco, *input gate,* trabalha o sinal de entrada da célula em dois sub-blocos: adiciona-o à memória de curto prazo do passo anterior (h_{t-1}) a uma função de ativação sigmoide e a uma função de ativação tangencial. A primeira determina a percentagem de memória de curto prazo a relembrar e a segunda determina o valor da memória de curto prazo que é adicionado ao longo prazo que é então memorizado para a próxima interação (c_{t-1}).

O último bloco, *output gate,* é responsável por modificar e atualizar a memória de curto prazo em função do valor da memória de longo prazo. Nesse caso, o sinal de entrada da célula é novamente adicionado à memória de curto prazo do passo anterior (h_{t-1}), seu resultado é multiplicado à função de ativação sigmoide e adicionado ao *threshold.* O valor resultante é multiplicado pelo sinal de memória de longo prazo (c_t), após esse passar por uma função tangencial, e que irá compor o novo valor da memória de curto prazo (h_t).

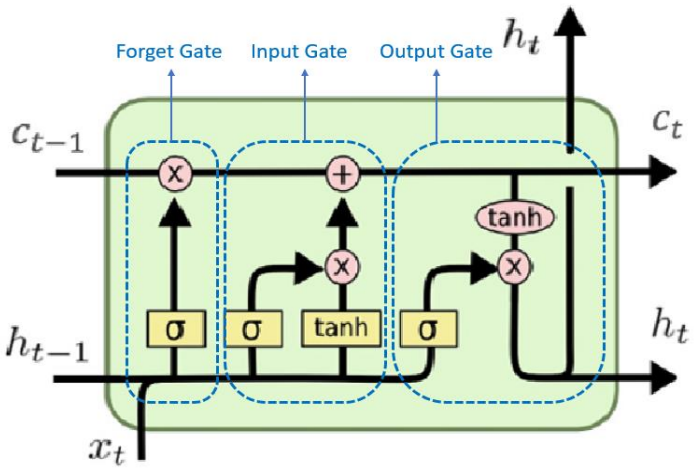

Figura 5: Modelo da célula de memória de uma LSTM a suas unidades de *gate* (Hochreiter and Schmidhuber, 1997)

Outras versões de LSTM foram introduzidas recentemente, diversificando o perfil dessas redes, dando caracteríticas de maior integração entre os *gates*. Nomeadamente, (Gers et al., 2003) implementa observações entre blocos, permitindo os *gates* perceberem as entradas entre eles, enquanto (Chung et al., 2014) modifica a LSTM em um novo modelo chamado de Unidade Recorrente Controlada (GRU), criando *gates* de atualização, estados da célula e estados ocultos, com performance melhor que a LSTM original.

2.4. Aprendizagem por Reforço

Além da aprendizagem supervisionada e não supervisionados supracitados, há ainda outra aprendizagem de máquina que é designada por Aprendizagem por Reforço. Caracterizada como um dos três paradigmas do *Machine Learning*, a Aprendizagem por Reforço, *Reinforced Learning* (RL), busca a maximização da recompensa de agentes inteligentes num ambiente dotado de penalidades e obstáculos. Diferentemente dos métodos anteriores, a aprendizagem ocorre por meio das interações de ações e reações resultantes de um agente exposto a um ambiente. Nesse local, ele escolhe as ações de acordo com o estado atual a fim de maximizar a recompensa retornada por esse ambiente. Trata-se de problemas de ciclo fechado, pois a aprendizagem dessas ações influencia as entradas seguintes (Sutton and Barto, 2014). Cada ação é tomada por experimento, servindo como aprendizagem futura, influenciando a recompensa atual e as seguintes, conforme esquema mostrado abaixo na figura 6:

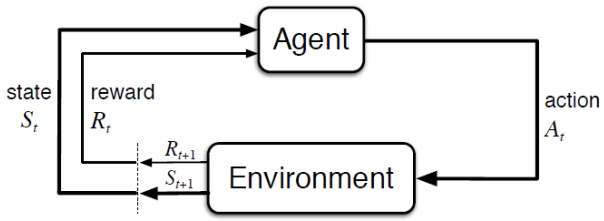

Figura 6: A interação do agente com o ambiente na Aprendizagem por Reforço (Sutton and Barto, 2014)

Nesse processo, o agente e o ambiente se interagem em uma sequência de intervalos de tempo discretos t, sendo que, a cada etapa, ele recebe uma representação do ambiente em um conjunto de estados *S* e a partir disso toma uma determinada ação partindo-se do conjunto de ações disponíveis A. Tomada a ação, o agente recebe uma recompensa correspondente dentro do conjunto R (t+1) e é posicionado para um novo estado S (t+1). Cada ação tem a possibilidade de ser tomada ou não de acordo com a probabilidade P referente a restrições do ambiente ou do agente e cada ação tomada é aplicado um fator de desconto γ aplicado ao algoritmo como designação de atraso na performance. Essa última variável dá uma característica de tempo à Aprendizagem por Reforço, pois concluir a tarefa o mais rapidamente possível implica na minimização do desconto e maximização da recompensa.

Esse conjunto de variáveis que compõem o Processo de Decisão de Markov (MDP) é definido pela tupla (s, a, R, P, γ) e pode ser representado matematicamente pela função de valor ótimo Q da solução de Bellman (Bellman, 1957) na equação 2.6 abaixo:

$$Q^*(s,a) = \mathbb{E}\left[R(s,a) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q^*(s',a')\right] \tag{2.6}$$

A política em aprendizado por reforço é uma estratégia utilizada pelo agente para escolher ações em cada estado do ambiente. A política determina qual ação deve ser tomada em um determinado estado com base nas informações disponíveis, como a função de valor ótimo Q. Por sua vez, essa função determina a qualidade de uma ação em um determinado estado em um ambiente, calculando o valor esperado do retorno futuro (soma das recompensas futuras) de tomar uma determinada ação em um estado específico e seguir a política (estratégia) ótima a partir daí. Em outras palavras, Q estima o valor de cada ação em cada estado, levando em consideração todas as possíveis sequências de ações que podem ser tomadas, sendo essa função uma medida do quão bom é escolher uma determinada ação em um estado específico.

A grande vantagem no uso dessa equação está na característica recursiva que permite que recompensas resultantes de estados futuros possam impactar em tomadas de decisões de estados passados. Através da interação constante da equação, é possível que Q venha a convergir para um valor ótimo, tornando explícito o melhor caminho de ações a se tomar. Logo, a Aprendizagem por Reforço (RL) decorre de uma série de interações, experimentos entre o agente e o ambiente que busca a maximização da função de valor agregada por recompensas e penalidades em cada passo tomado.

Assim, torna-se desnecessária toda a história envolvida para que o agente chegasse àquele estado S, bastando apenas que S retenha todas as informações relevantes para que ele tome a melhor ação com recompensa resultante. Esse estado S de informações completas, resultado de inúmeras interações, é chamado de Markov, ou tem a propriedade Markov (Sutton and Barto, 2014). Um exemplo disso, é o jogo de damas, a disposição atual de todas as peças em um tabuleiro serve como um estado de Markov, pois ela resume tudo o que é importante sobre a sequência completa de posições que levaram a ele. Muitas das informações sobre a sequência está perdida, mas tudo o que realmente importa para o futuro do jogo é retido. Isso é chamado de “independência de caminho", no qual o futuro é independente do passado dado o presente.

Open AI faz uma taxonomia dos algoritmos implementados em RL (*Part 2: Kinds of RL Algorithms — Spinning Up documentation, 2022*), divindindo-os em grupos que necessitam ou não dos modelos do ambiente e em grupos de algoritmos treinados por critérios de otimização de políticas ou por Q-learning, conforme o esquema da figura 7 abaixo:

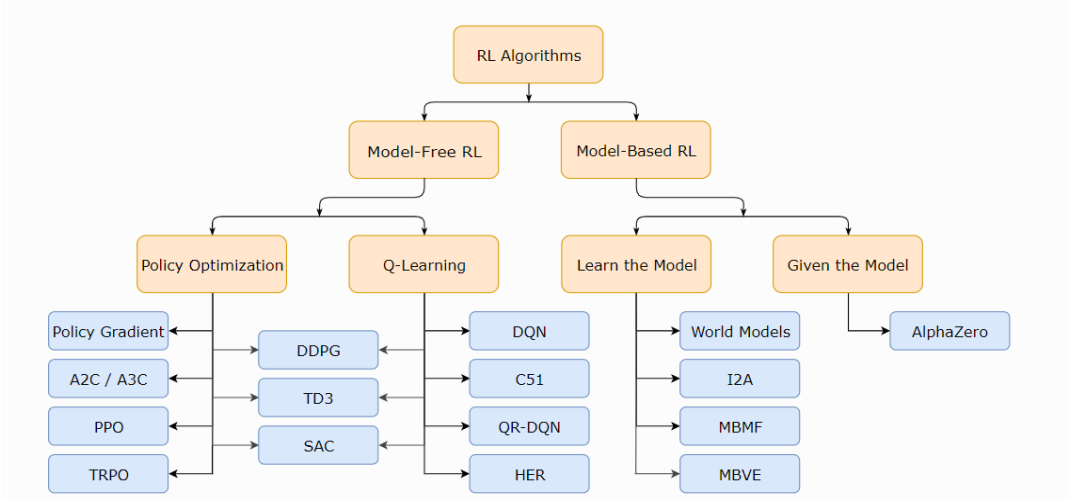

Figura 7: Taxonomia de algoritmos em RL (*Part 2: Kinds of RL Algorithms — Spinning Up documentation, 2022*).

Os algoritmos pertencentes ao grupo de otimização de política buscam a aprendizagem por meio de um mapa de ações aplicadas em estados, de maneira que possam funcionar em espaços de ações contínuas, podendo ser elas estocásticas ou não. Eles podem usar técnicas como gradientes de política para atualizar a política de forma iterativa. Por outro lado, os algoritmos de otimização da função de valor aprendem uma função de valor ótima para cada estado ou par estado-ação. Eles geralmente têm um espaço de ação discreto e usam técnicas como Q-learning para atualizar a função de valor.

Neste trabalho foram utilizados diferentes algoritmos de RL, cada um com diferentes características, e que são definidos ao longo deste capítulo. Concretamente:

- DQN (*Deep Q-Network*): algoritmo que usa uma rede neuronal para estimar a função Q e atualizar os pesos da rede usando a equação de Bellman. Utiliza abordagem *off-policy*, ou seja, usa dados coletados por uma política diferente daquela que está sendo atualizada para aprendizagem;
- SARSA (*State-Action-Reward-State-Action*): algoritmo que atualiza a função Q de acordo com a política atual, utiliza uma abordagem *on-policy*, o que significa que ele usa a política atual para explorar o ambiente;
- CEM (*Cross-Entropy Method*): algoritmo *off-policy* baseado em técnica Monte Carlo para amostragem e geração de distribuição de probabilidade sobre as possíveis soluções do problema de otimização;
- A2C (*Advantage Actor-Critic*): algoritmo *on-policy* que usa uma rede neuronal para aproximar tanto a política quanto a função de valor Q;
- TRPO (*Trust Region Policy Optimization*): algoritmo *on-policy* que atualiza a política de maneira iterativa, mantendo essas atualizações abaixo de um determinado limite;
- PPO (*Proximal Policy Optimization*): algoritmo *on-policy* que também atualiza a política de maneira interativa, abaixo de um determinado limite, mas de maneira mais eficiente computacionalmente que TRPO.

De forma geral, DQN, SARSA, CEM e A2C são exemplos de algoritmos de valor e de política, enquanto TRPO e PPO são exemplos de algoritmos de otimização de política. O método DQN e SARSA usam a equação de Bellman para atualizar a função Q, enquanto A2C usa tanto a equação de Bellman quanto o cálculo de vantagem, ou seja, leva em consideração a diferença entre o valor previsto de uma ação e o valor atual da política. TRPO

e PPO usam a derivada desse cálculo de vantagem em relação à política para atualizar a política de forma mais eficiente.

2.4.1. Ação Ambiciosa, Aprendizagem On-Policy e Off-Policy

Chegar a um estado de Markov requer que o agente explore o ambiente para que se obtenha informações acerca de diversos estados. Diante de um mapa de possibilidades de ações, o mais prudente é tomar aquela que retorne o melhor valor possível, nomeadamente a ação ambiciosa (*greedy action)* que maximiza a recompensa. A definição da ação implica é um problema comum todos os métodos de Aprendizagem por Reforço: explorar o mapa de possibilidades de ações possíveis para encontrar a mais ambiciosa, gastando tempo nesta tarefa (*exploration*) ou fazer uso do conhecimento corrente, explorando de imediato a ação com retorno já conhecido, não desperdiçando tempo dessa forma (*exploitation*). Saber balancear essa relação entre *exploration* e *exploitation* leva o algoritmo a um processo mais eficaz de aprendizagem e performance.

O método *ε-greedy*, por exemplo, tem a vantagem de amostrar um percentual pré-determinado de ações, no sentido de explorar as possibilidades, para que se garanta qual é a que retorna o valor mais alto (Sutton and Barto, 2014). Outro meio bastante empregado nesse dilema são as atualizações dinâmicas por distribuição de Boltzmann *"softmax"* (Pan et al., 2020), na prática chamada de *Boltzmann Q-policy*. Nesse caso, são levadas em consideração as maiores probabilidades de execuções de ações juntamente com aquelas que dão maior recompensa. A ideia aqui é iniciar a aprendizagem com um comportamento explorativo do ambiente (política estocástica), tomando ações randômicas e, com o passar do tempo, o agente se concentrar em ações que resultam em maior recompensa (política determinística).

Duas técnicas de aprendizado de máquina são também empregadas em RL: *on-policy* e *off-policy*, em que um agente aprende a tomar decisões em um ambiente interativo para maximizar uma recompensa. *On-policy* é uma abordagem em que o agente usa a mesma política para tomar decisões e para aprender com essas decisões, de maneira que a política atual é a mesma usada para avaliar a qualidade das ações tomadas. Em outras palavras, o agente está sempre atualizando sua política atual com base nas experiências mais recentes, tendendo essa abordagem a ser mais estável, pois a política atual é sempre atualizada com base nessas experiências recentes. No entanto, ele pode ter um desempenho inferior em

relação a abordagens *off-policy* em ambientes mais complexos ou quando a política atual é muito inadequada e precisa ser melhorada.

Off-policy, por outro lado, é uma abordagem em que o agente aprende a partir de uma política diferente daquela que está usando para tomar decisões, de maneira que o agente pode usar uma política anterior ou completamente diferente para escolher ações, mas está aprendendo com base em uma política de destino. Isso permite uma aprendizagem com ampla variedade de experiências, usando o conhecimento aprendido para melhorar sua política atual. Dessa forma, *off-policy* pode ser mais eficiente em relação ao tempo e pode levar a uma melhor exploração do ambiente. No entanto, também pode ser menos estável do que o *on-policy* em ambientes mais complexos ou quando há grandes diferenças entre a política atual e a política de destino.

2.4.2. Programação Dinâmica e Monte Carlo

A estimativa probabilística constante na tomada de decisões em algoritmos de Aprendizagem por Reforço implica um alto grau de processamento computacional da rede. Ainda que houvesse um modelo preciso da dinâmica do ambiente, contendo informações das recompensas de cada tomada de decisão, geralmente não é possível calcular uma política ótima resolvendo a equação de Bellman. Enfrenta-se aqui restrições computacionais, nomeadamente no processamento e na memória da máquina que dificultam o cálculo diante de um universo de possibilidades, levando a aproximações. Se o conjunto de estados for muito grande, mesmo uma única varredura de todas as possibilidades do ambiente poderia ser computacionalmente dispendioso. Dessa forma, a Programação Dinâmica Assíncrona estuda meios de se realizar varreduras de dados eficiente, selecionando os estados aos quais a atualização pode melhorar a taxa de progresso do algoritmo. Alguns estados podem não precisar de backup tão frequentemente quanto outros, podemos às vezes ser saltados. Nesse sentido, a Programação Dinâmica (DP) proporciona um conjunto de algoritmos usados no estabelecimento de ótimas políticas de agente em que o ambiente reflete perfeitamente o processo de decisão de Markov (Sutton and Barto, 2014). Tais algoritmos podem ser executados em tempo real, fazendo com que o agente experimente o ambiente inserido.

Na outra ponta de solução de problemas por Aprendizagem por reforço está o método de Monte Carlo, cujo conjunto de algoritmos se baseia na tomada de ações randómicas do agente até que o episódio seja finalizado (requisito para Monte Carlo) (Sutton and Barto, 2014). Define-se um episódio finalizado quando o agente atinge o sucesso do objetivo ou

quando extrapola as penalidades estabelecidas pelas regras do ambiente. A cada finalização do episódio, os algoritmos de Monte Carlo realizam uma avaliação da função valor agregada, obtendo as médias das recompensas alcançadas em cada estado, atualizando as políticas que oferecem maior valor agregado. Ao contrário da Programação Dinâmica, Monte Carlo não necessita do modelo completo do ambiente para execução, o que reduz drasticamente o nível de processamento computacional e memória. O método converge quadraticamente a medida que cada par de ação estado é tomado em inúmeras interações, podendo não se achar a melhor política caso os estados de grande recompensa não sejam visitados.

Uma maneira de garantir que a melhor política com maior valor agregado seja alcançada é garantir que todos os pares estado-ação sejam visitados. Chamado de *exploring starts*, esse hiperparâmetro em Monte Carlo estabelece que todo episódio se inicie em um estado diferente pelo menos uma vez (Sutton and Barto, 2014). Caso o número de estados possíveis seja demasiado grande, então *ε-greedy* é aplicado de maneira que as primeiras interações tenham um comportamento bastante explorativo do mapa (*exploration*) e que as últimas interações possuam um comportamento mais explorativo das políticas que tenham dado maior retorno durante o treinamento (*explotation*).

Método de Entropia Cruzada (CEM)

O Método de Entropia Cruzada (CEM) é um método de Monte Carlo, aplicável a problemas de natureza estocástica, que estima as melhores probabilidades de ações do agente por meio das medições de perda de entropia cruzada e de divergência Kullback Leibler (KL) (de Boer et al., 2005).

A perda de entropia cruzada mede a performance de classificação de um modelo pela diferença entre duas distribuições de probabilidades, a ideal e a prevista, tendo como saída um valor entre 0 e 1. É descrita pela equação 2.7, onde N é o número de classificações possíveis, y é a saída da classificação em questão e p é a probabilidade calculada pelo modelo dessa classificação estar correta.

$$L = -\frac{1}{N}\sum_{i=1}^{N} y_i \log\big(p(y_i)\big) + (1 - y_i) * \log\big(1 - p(y_i)\big) \tag{2.7}$$

Dessa forma, quanto mais próximo do valor 0, mais próximo o modelo está da classificação correta, como pode ser visto abaixo no gráfico da figura 8, onde a curva resultante relaciona a perda de entropia cruzada (eixo Y) com a probabilidade de predição (eixo X):

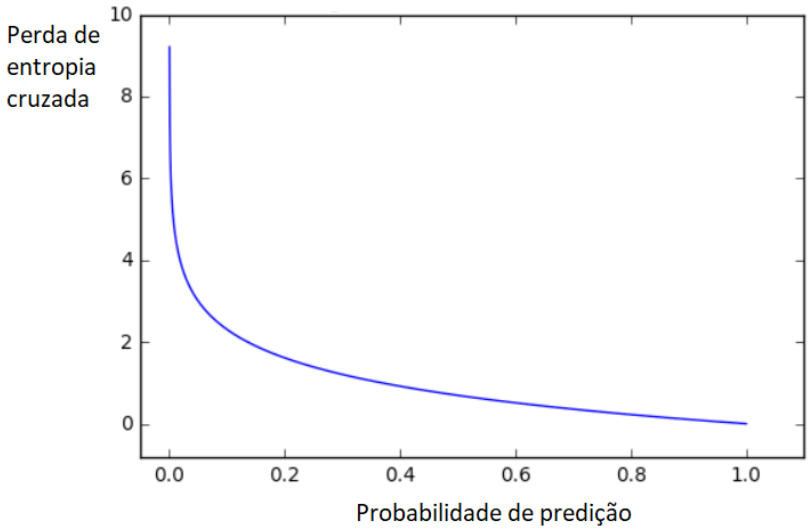

Figura 8: Gráfico de relação entre probabilidade de predição e perda de entropia cruzada

Já a divergência Kullback Leibler (KL) quantifica o quão diferente uma distribuição de probabilidades é da outra, por meio da equação 2.8:

$$D_{\mathrm{KL}}(P\|Q) = \sum_i P(i)\ \log \frac{P(i)}{Q(i)} \tag{2.8}$$

Nessa equação, i representa cada classificação nas distribuições de probabilidade, P representa a probabilidade de ocorrência dessa classificação em uma distribuição de probabilidades e Q se refere a probabilidade dessa classificação em outra distribuição de probabilidades. O gráfico abaixo da figura 9 apresenta as distribuições de probabilidade P e Q e divergência KL (eixo Y) em função das classificações:

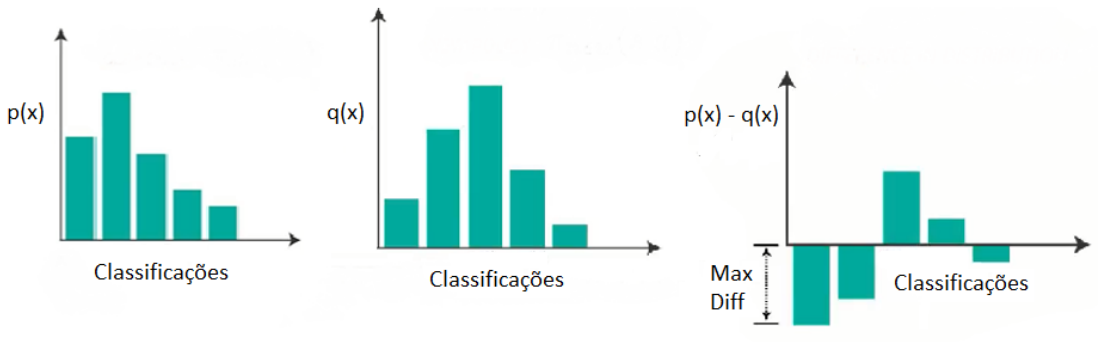

Figura 9: Gráfico de distribuição de probabilidades de classificações e sua divergência

Quanto menor a divergência KL entre as classificações, menor a diferença de política entre a aprendizagem durante o treino e mais próximo da melhor performance está o modelo.

Por meio dessas duas métricas, a cada episódio, o agente no algoritmo CEM toma diversas ações aleatoriamente de acordo com as entradas apresentadas pelo ambiente e são sempre são analisadas por meio da perda de entropia cruzada. Ao fim do episódio, o peso dos neurónios nas redes neuronais são atualizados de modo a diminuir essa perda, numa atualização de políticas de ações por aquelas que apresentam melhor resultado mediante as entradas observadas. Na medida em que a performance do algoritmo vai melhorando, o valor de KL vai diminuindo, uma vez que a política de ações se consolida, diminuindo a variação de estratégia de ações entre elas.

Por não realizar cálculos de gradiente e requerer pouca parametrização, o algoritmo CEM, além de ser rápido, é uma boa solução quando se deseja ter pouco esforço computacional na solução de problemas de baixa complexidade. Possui a desvantagem da aprendizagem somente ocorrer ao término de cada episódio, inerente aos métodos de Monte Carlo, e ter grande potencial da solução ficar presa em mínimos locais do gradiente do conjunto de soluções do problema.

2.4.3. Diferença Temporal

Outro meio utilizado na Aprendizagem por Reforço é a diferença temporal TD, que combina as melhores características do método Monte Carlo e da Programação Dinâmica. TD não necessita da matriz de penalidades e recompensas do ambiente e não necessita que a finalização do episódio ocorra para que as políticas sejam avaliadas e atualizadas. A cada número (pré-definido) de passos tomados, podendo ser um único passo (TD(0)) ou mais passos (TD(λ)), o método avalia a tupla (S, A, R, P, γ) referente ao processo de decisão de Markov e utiliza *bootstrap* da estimativa atual da função de valor para previsão de ação futura. *Bootstrap* é um processo de amostragem aleatória e que fornece medidas de precisão como variância, intervalos de confiança, constituindo assim uma distribuição amostral. Assim, por meio desses dados, a aprendizagem por diferença temporal escolhe o melhor caminho sem ter de percorrer toda a gama de possibilidades. Quanto mais próximo da tomada de ação, mais precisos tendem a ser os dados obtidos pelo *bootstrap*.

Os algoritmos de TD podem usar políticas de comportamento nas tomadas de decisão baseadas no *ε-greedy (on-policy),* ou estimar os retornos futuros para seguir uma política completamente gananciosa *(off-policy),* a exemplo do Q-learning.

SARSA e Q-learning

SARSA, um dos algoritmos precursores de TD, é a abreviação de *State-action-reward-state-action* e atualiza a tabela de valores de recompensa de acordo com a política previamente seguida *(on-policy).* No estado corrente do agente, a ação escolhida baseia-se no *ε-greedy,* ora explorando a ação com o melhor retorno previamente aprendido, ora explorando o ambiente, à procura de ações que tragam maior retorno. Dessa forma, SARSA, que segue uma política com certa aleatoriedade misturada com uma política gananciosa, este converge em uma boa solução sob constantes interações de Q descrita pela equação 2.9:

$$Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma Q(S',A') - Q(S,A)\right] \\ S \leftarrow S';\ A \leftarrow A'; \tag{2.9}$$

Onde α é a taxa de aprendizagem, R é a recompensa, Q (S, A) é a função de valor ótimo para o estado atual S e para a ação atual A, γ é o fator de desconto e Q (S', A') é a função de valor ótimo para o próximo estado S' e para a próxima ação A'. Dessa forma, a equação para SARSA permite que o agente aprenda a escolher ações que maximizam a recompensa futura esperada com base na política atual.

Já Q-learning, que não usa política de comportamento para tomada de decisão *(off-policy),* estima qual ação é necessária para que o valor retornado Q seja máximo diante da ação ambiciosa no momento da exploração de ações, e toma ações randómicas no momento de exploração do ambiente. Assim, cada MDP é realizado na procura do retorno Q descrito pela equação 2.10:

$$Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma \max_a Q(S',a) - Q(S,A)\right] \\ S \leftarrow S'; \tag{2.10}$$

Onde $max_a Q(S', a)$ é a função de valor ótimo máxima para todos os possíveis próximos estados S e ações a. Dessa forma, a equação para Q-learning atualiza o valor da função Q para um determinado estado e ação com base na recompensa recebida e no valor máximo da função Q para o próximo estado e todas as ações possíveis. Isso permite que o agente aprenda a escolher a ação mais vantajosa em cada estado.

Como resultado, percebe-se que SARSA é preferível em situações em que o desempenho do agente durante a aprendizagem possa o pôr em risco, pois o algoritmo explora situações, convergindo em uma boa solução rapidamente. Assim, durante o treino, o agente aprende já inicialmente os pontos de maior penalidade e os evita, mesmo que houvesse ali uma política gananciosa que o poderia levar a uma solução ótima. Por outro lado, Q-learning é utilizado em momentos que se deseja maximizar a recompensa final agregada do episódio, sem se importar com as penalidades que a aprendizagem possa vir a ter, convergindo o treino em uma solução ótima.

2.4.4. Deep Reinforcement Learning

Diante da quantidade de dados observados pelo agente durante o treinamento em Apredizagem por Reforço, fez-se necessário o desenvolvimento de algoritmos que pudessem armazenar esses dados em redes neurais. Desde 2012, que se tem buscado a combinação do *Deep learning* com a Apredizagem por Reforço. Como pioneira, pode-se citar a empresa *DeepMind* na aplicação do *Deep Reinforcement Learning* no treinamento de jogos de Atari (Mnih et al., 2013) e do jogo AlphaGo (Silver et al., 2016).

A aprendizagem profunda das redes neuronais decorre dos pesos θ contidos nos neurónios, em múltiplas camadas, atualizados por gradiente descendente. Como outra opção, o agente em RL poderia ser treinado em modelos "rasos", como funções lineares, árvores de decisões e outros algoritmos, sendo uma opção quando a quantidade de dados para treino do agente é pequena. Em suma, a rede neuronal se justifica quando há grande quantidade de dados, o que torna-se relevante na maior parte dos algoritmos de alto desempenho em RL.

2.4.4.1. Deep Q-Network (DQN)

O trabalho desenvolvido por Minh et al. (2013) nos treinamentos de jogos de Atari e *Human-level Control Through Deep Reinforcement Learning (Mnih et al., 2015)* resultou na criação do algoritmo Deep Q-Network (DQN). Sempre houve a necessidade que os agentes em RL tivessem a capacidade de se ter a visão do ambiente, poder interpreta-la e atuar de acordo com a sua dinâmica, sendo esse o principal motivo pelo qual o algoritmo foi desenvolvido num modelo CNN (que permite a análise visual automática do ambiente) combinado com uma variante de Q-learning. Além disso, o algoritmo implementa 4 técnicas que proporciona

melhor estabilidade na aprendizagem: *Experience Replay, Target Network, Clipping Reward, Skipping Frames.*

A cada passo tomado pelo agente, o *Experience Replay* armazena o estado corrente, ação tomada, a recompensa e o estado seguinte (tupla <S, A, R, S'>) em memórias de dados que posteriormente são usadas em um mini lote de dados para treinamento do agente e atualização da rede neuronal. Importante destacar que esse mini lote de dados é composto por tuplas randómicas da memória de dados para se evitar a correlação de dados entre os passos tomados numa série temporal e *overfitting* ao treino (*bootstrap*). A utilização do *Experience replay* promove o aumento da velocidade da aprendizagem e evita o esquecimento de ações tomadas no passado, criando um efeito memória no agente.

DQN utiliza uma rede neuronal de valor ótimo Q que é atualizada a cada episódio de treinamento por meio da equação aplicada para Q-learning. No entanto, a atualização direta dessa rede neuronal de valor pode levar a instabilidades na aprendizagem. Por isso, o algoritmo usa também uma segunda rede neuronal de valor ótimo Q chamada de *Target Network*, cuja atualização ocorre a cada número de passos. Isso ajuda a estabilizar a aprendizagem, uma vez que a expectativa Q(S, A) e a meta Q(S', A') também estão correlacionadas, fazendo com que a cada atualização de rede o alvo Q máximo seja modficado, ou seja, o agente persegue um alvo em movimento. Assim, para reduzir a correlação entre a expectativa e o alvo, *Target Network* corrige os parâmetros da função de valor Q e o substitui pela rede mais recente após um conjunto de passos tomados.

Já a técnica *Clipping Reward* fixa todos os valores positivos de recompensa de um episódio em 1 e todos os negativos em -1, deixando em 0 recompensas que não se alteram. Isso limita a escala de derivação no cálculo do gradiente do erro e facilita a taxa de aprendizagem em diferentes ambientes, com diferentes escalas de pontuação.

Por fim, *Skipping Frames* é usado para limitar o número de *frames* de entrada da rede que serão usados no cálculo da função valor Q, reduzindo o esforço computacional desnecessário em dados irrelevantes ao agente.

2.4.4.2. Métodos Actor-Critic

Métodos dessa classe de algoritmos são baseados em Diferença Temporal TD, usam otimização de política e possuem uma rede neuronal para avaliação da função de valor Q

(maximização imediata não é objetivo) chamada de *critic* e outra rede neuronal para a política chamada de *actor* (Sutton and Barto, 2014). A figura 10 abaixo apresenta o esquema desses métodos:

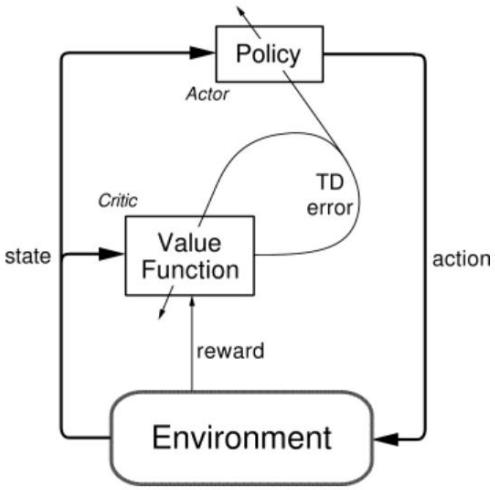

Figura 10: Arquitetura *Actor-Critic* (Sutton and Barto, 2017)

A função do *critic* é calcular o erro da diferença temporal (TD *error*) contida no par estado-valor e ajustar a rede neuronal *actor* para a tomada correta da ação. Assim, o TD *error* pode ser calculado por meio da equação 2.11:

$$\delta t = R + \gamma Vt(St+1) - V(St) \tag{2.11}$$

onde Vt é a função de valor Q implementada pelo *critic* no tempo t, R é a recompensa e γ é o fator de desconto. Este TD *error* δt é aplicado na correção do *actor* (por *backpropagation*) nas equações 2.12 e 2.13:

$$w = w + \alpha^{w} \delta \nabla V(S, w) \tag{2.12}$$

$$\theta = \theta + \alpha^{\theta} \gamma^{t} \delta \nabla \ln \pi(A|S, \theta) \tag{2.13}$$

onde a primeira equação refere-se a correções na rede neuronal do *critic* e a segunda equação refere-se a correções na rede neuronal do *actor*. Na primeira equação, a variável w diz respeito ao peso do neurónio corrigido, α à taxa de apredizagem, δ ao TD *error* e V(S,w) à função valor Q resultante deste neurónio específico. Já na segunda equação, θ refere-se ao peso do neurónio corrigido e $\pi(A|S, \theta)$ à política tomada pelo *actor* diante do estado imposto. Vale destacar que este pedaço $\nabla \ln \pi(A|S, \theta)$ da equação trata-se do gradiente da política adequada ao agente num contexto de otimização (*on-policy*), onde o logaritmo neperiano

aplicado ocorre devido a função *softmax* comumente utilizada na classificação de várias ações.

Nota-se também que nos métodos *Actor-Critic* há bastante similaridade com a arquitetura *Generative Adversarial Network (GAN),* uma vez que uma das redes neuronais trabalha como corretor da segunda, tal como em GAN. O que diferencia uma da outra é o pré-conhecimento dos estados do ambiente pelo *actor* e o contributo do *critic* na correção do *actor*, enquanto esse processo na GAN ocorre como um discriminador, gerador de erro, na segunda rede neuronal.

Como exemplos de algoritmos *Actor-Critic* , podem ser citados os algoritmos *Advantage Actor-Critic* (A2C) e *Asyncronous Advantage Actor-Critic* (A3C), introduzidos em 2016, que utilizam múltiplos agentes treinados num mesmo ambiente (Mnih et al., 2016). Ao invés de utilizar *experience replay*, que consome maior memória e processamento, diversos agentes são treinado paralelamente em instâncias diferentes, de maneira que os dados absorvidos por cada um não fiquem correlacionados. O que diferencia A3C de A2C é que, no primeiro algoritmo, cada agente acessa os parâmetros da rede neuronal de forma independente, podendo haver atualizações não ótimas, enquanto no segundo algoritmo, os parâmetros globais da rede somente são atualizados após cada agente finalizar a sua tarefa , de maneira que haja sincronismo nas atualizações. Há também uma pequena modificação nesses algoritmos que introduzem o conceito de vantagem (*Advantage*), descrito pela equação 2.14, que por vezes substitui o TD *error* δt:

$$A(S,A) = Q(S,A) - V(S) \tag{2.14}$$

onde A(S,A) é a função de valor vantagem, Q(S,A) é o valor Q obtido na tomada da ação A no estado S e V(S) é a média dos valores existentes no estado S tomados por todas as ações excutadas pelo agente naquele estado. Dessa forma, essa função calcula a recompensa extra que se ganha na tomada se uma ação, sendo que se A(S,A) é maior que 0, o gradiente da política é empurrado naquela direção, caso contrário, essa ação tomada é pior que a média das ações disponíveis no estado e o gradiente é empurrado na direção oposta.

2.4.4.3. Trust Region Policy Optimization (TRPO) e Proximal Policy Optimization (PPO)

Nos algoritmos de otimização de política, o gradiente ascendente pode ter alta variância ou pode levar o agente a regiões de soluções em que minímos locais impedem a aprendizagem, podendo comprometê-la. Diferentemente da aprendizagem supervisionada, em que a tomada de um passo inadequado pode ser corrigida na próxima atualização de dados, RL não dispõe de dados para treino, dispõe apenas de uma interação entre o agente e o ambiente. Dessa forma, a tomada de um passo inadequado, leva o agente a uma política inadequada e à falta de aprendizagem, porque os novos dados não são mais informativos. Soma-se a isso a falta de memória, nomeadamente *Experience Replay,* presente nos algoritmos Q-learning e que contribuiria para o acesso remoto de experiências passadas.

Para solução desses problemas, surgiram os algoritmos *Trust Region Policy Optimization* TRPO (Schulman et al., 2017b) e *Proximal Policy Optimization* PPO (Schulman et al., 2017a) que buscam diminuir a alta variância no gradiente, mantendo o agente em uma rota de políticas adequadas na solução do problema. Enquanto os algoritmos do Método *Actor-Critic* combina as redes neuronais de valor e de política em série uma com a outra, tanto PPO como TRPO podem trabalhar com essas mesmas redes neuronais em paralelo ou em em série, possibilitando melhor performance ao agente treinado, de acordo com o problema abordado. Também são algoritmos *on-policy*, que utilizam apenas amostras coletadas pela política atual para atualizar a política, não permitindo amostras antigas para esse fim, o que significa que a política atualizada é usada imediatamente para coletar mais dados e atualizar a política novamente. Essa abordagem tem algumas vantagens, como a garantia de convergência para um ótimo local e a estabilidade durante o treinamento. No entanto, uma das desvantagens é que esses algoritmos podem ser sensíveis à exploração insuficiente, pois a política é atualizada apenas com dados coletados usando a política atual.

Ambos os algoritmos usam uma função de vantagem para estimar a qualidade de uma ação em relação às outras ações possíveis em um determinado estado. Essa função de vantagem é usada para ajustar a política do agente durante o treinamento, de modo que a política seja atualizada para maximizar a função de vantagem. Nomeadamente, essa função é traduzida nos conceitos de *Surrogate loss* e *Kullback-Leibler Divergence KL* (segunda versão de PP0 abandona esse conceito) que mensuram quando uma política velha deve ser substituída por

uma nova. A função *Surrogate loss*, mostrada na equação 2.15, coleta dados de recompensa da política velha Eπ *old* (*expected reward),* estabelece uma função de perda com a razão entre a política velha e a nova política em análise, e multiplica essa razão pela a função de vantagem, Aπ old(s,a), na tomada da ação no estado sob questão.

$$\max_{\pi} L(\pi) = \mathbb{E}_{\pi_{\text{old}}} \left[\frac{\pi(a|s)}{\pi_{\text{old}}(a|s)} A^{\pi_{\text{old}}}(s,a) \right] \tag{2.15}$$

PPO e TRPO usam essa equação para medir a diferença entre a política atual e a política que o agente deveria seguir para maximizar a recompensa esperada, de maneira que se a função de perda for minimizada, a política do agente melhorará. A função de perda surrogate é uma aproximação da recompensa esperada e é escolhida para ser diferenciável e fácil de otimizar.

Já a restrição KL, citada no capítulo Métodos de Entropia Cruzada (CEM), é uma medida do quanto a distribuição de probabilidades das ações resultantes da velha política se divergem da nova política sob análise. A cada interação dos algoritmos TRPO e PPO, a rede rede neuronal de política fornece uma distribuição de probabilidade das melhores ações a serem tomadas naquele instante. Logo, o que a restrição KL faz é medir a diferença de distribuição de probabilidades entre a política atual e a velha política a fim de limitar grandes divergência por meio do hiperparâmetro ε, conforme mostrado na equação 2.16:

$$\mathbb{E}_{\pi_{\text{old}}} \left[KL(\pi || \pi_{\text{old}}) \right] \leq \epsilon \tag{2.16}$$

A combinação desses dois conceitos restringem o agente a uma região de políticas confiáveis, impedindo que caiam em falsos minímos locais, levando a pontos de falta de aprendizagem. A ideia é que a nova política não deve se afastar muito da política antiga para garantir que as atualizações da política sejam seguras e efetivas, conduzindo o agente a uma jornada de aprendizagem um pouco mais longa, porém mais segura. Além disso, o PPO usa uma técnica de clipagem, que limita a distância entre as políticas antigas e novas, o que também ajuda a garantir que as atualizações da política sejam estáveis e confiáveis. A diferença entre TRPO e PPO da primeira versão, está na inclusão do hiperparâmetro β em TRPO que intensifica ou não a restrição KL, descrita pela equação 2.17:

$$\max_{\theta} \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} \hat{A}_t \right] - \beta \left(\hat{\mathbb{E}}_t \left[\text{KL}[\pi_{\theta_{\text{old}}}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t)] \right] - \epsilon \right) \tag{2.17}$$

Já o PPO da segunda versão, nomeadamente PPO 2 ou Clipped PPO, utiliza uma *Clipped Surrogate Objective* em substituição a KL, que limita as alterações na política, substituindo a função de vantagem aproximada pela sua versão "clipada" (Barhate, 2018). Essa versão limita a magnitude da alteração da política dentro de um intervalo em torno de 1, definido por um hiperparâmetro ϵ. Assim, a função objetivo do PPO leva o valor mínimo entre o valor original e o valor "clipado". Dessa forma, a técnica garante que as atualizações da política sejam controladas e estáveis. Além disso, a técnica de "*clipping*" mantém a estabilidade e o desempenho da política, mesmo em situações em que a função de vantagem aproximada é inconsistente. Abaixo, segue a equação 2.18 referente a *Clipped Surrogate Objective:*

$$L^{CLIP}(\theta) = \hat{E}_t[min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\varepsilon, 1+\varepsilon)\hat{A}_t)\,] \tag{2.18}$$

Essa troca ocorreu pela maior facilidade do cálculo da restrição, evitando funções logarítmicas, criando uma região em que não haja nenhuma atualização de política, ou seja, uma clipagem dessa região. Além disso, a utilização de *Clipped Surrogate Objective* em PPO é mais rápido e escalável do que a restrição KL usada em TRPO, pois pode ser implementado de forma mais eficiente em GPUs e paralelizado em várias CPUs. Os gráficos da figura 11 abaixo mostram as funções *Clipped Surrogate* (eixo Y) em função da recompensa (eixo X) adquirida pelo agente:

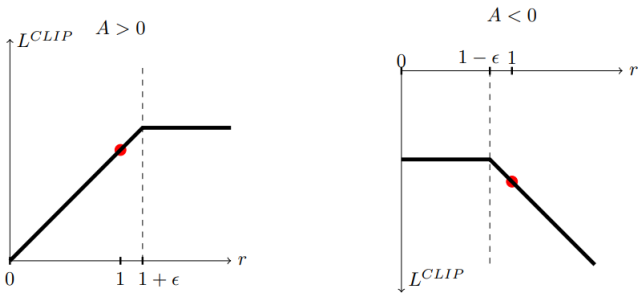

Figura 11: Funções *Clipped Surrogate* aplicadas à avaliação de políticas em conjunto o parâmetro de Vantagem (Schulman et al., 2017a)

Em resumo, o TRPO usa uma abordagem conservadora, por meio da restrição KL para atualização de política, limitando a quantidade de mudanças que podem ser feitas em uma única atualização, tornando o treinamento mais estável. Por outro lado, isso pode ser menos eficiente em comparação com o PPO, que utiliza uma abordagem mais agressiva para atualização de política, controlando essa mudança pela função *Clipped Surrogate*, que é computacionalmente menos exigente que o cálculo da restrição KL. Isso permite

atualizações mais rápidas e eficientes, sem comprometer a estabilidade do treinamento, sendo mais rápido em ambientes com muitas variáveis.

2.5. Gémeos Digitais

Buscando evitar custos e perdas de tempo associadas a erros de projeto, as empresas têm simulado protótipos de produtos e sistemas em um ambiente virtual, verificando dessa forma o comportamento de dispositivos quando afetados por perturbações externas. Esse conceito nomeado de *Digital Twin* (DT), Gémeos Digitais, tem sido um dos mais promissores na aplicação da Indústria 4.0 e em *Smart Manufacturing*.

Segundo Schroeder et al (2016), DT pode ser definido como uma arquitetura em cinco camadas: dispositivo, interface de usuário, serviços da web, consulta e repositório de dados. Os dispositivos são os equipamentos usados para visualizar o DT, tais como computadores, tablets, dispositivos móveis, podendo essa visualização ser realizada em 3 dimensões numa realidade aumentada. A camada de interface trata do acesso que o usuário faz no produto virtual, interferindo no seu comportamento e simulando o seu funcionamento. A camada de consulta trata de como são realizadas as buscas desses dados no banco de dados do DT e finalmente a camada de repositório onde os dados estão prontamente instalados e permitem a sua recuperação em armazenamentos em diferentes sistemas de gerenciamento.

Isso traz uma série de benefícios ao usuário, pois o ambiente digital contido em DT reflete a realidade física do modelo em questão, nomeadamente formas, posições, status e movimentos. Assim é possível monitorar, diagnosticar e otimizar processos que antes eram interagidos somente no mundo real, onde ferramentas de simulação e de realidade virtual promove o melhor entendimento do funcionamento desse sistema.

A partir dos anos 2000, os Gémeos Digitais passaram a ser utilizados com maior frequência, abrindo maior espaço para simulações e análises de sistemas. Diversas aplicações podem ser citadas, como a da General Electric (GE) que possui 4 patentes diretamente relacionadas à DT, sendo 2 delas ligadas a geração eólica (Tao et al., 2019). Esse sistema é capaz de monitorar os estados de funcionamento das turbinas quando expostas ao vento por meio de sensores e controlar suas operações por meio de modelos digitais. Já a Siemens também possui 4 patentes de DT’s relacionadas em interface máquina homem, eficiência energética, método de implementação e deteção de colisão (Wang e Canedo, 2019), enquanto a Johnson

aplicou o DT ao controlo de vários aspectos de uma sala, nomeadamente temperatura, iluminação e qualidade do ar (Johnson, 2018).

Para composição do Gémeo Digital ao fim deste trabalho, foi utilizado o Unity como ferramenta de simulação. Esse *software* é capaz de criar jogos, ambientes de realidade virtual e realidade aumentada em 3 dimensões, e disponibiliza também um ambiente de programação em C# que compõe a API do programa para troca de dados com fontes externas. A escolha do Unity (Technologies, 2023) se deve à capacidade de renderização de gráficos e visualizações, contendo em sua biblioteca online um grande número de ambientes já pré-projetados. Além disso, é possível aplicar as leis da física no campo da cinemática e das leis de Newton, o que torna a análise dos dados e experiência do usuário mais próxima do real.

3. Estado da Arte

Neste capítulo são apresentados diferentes trabalhos da aplicação de Aprendizagem por Reforço em processos industriais, sendo alguns deles simulados em Gémeos Digitais e outros aplicados diretamente no ambiente real. Esses estudos incluem o desenvolvimento, as principais vantagens e desvantagens, os desafios e os resultados obtidos pelos autores. Ao fim do capítulo é realizada uma análise da viabilidade desses estudos, apontando direcionamentos para essa classe de algoritmos.

3.1. Aprendizagem por Reforço no controlo de processos industriais

A Aprendizagem por Reforço RL tem atraído cada vez mais a atenção da indústria no controlo de processos de manufatura. A tentativa constante de superação do desempenho de algoritmos de aprendizagem supervisionada e não supervisionada somadas à adaptação às mudanças ocorridas nesse ambiente industrial é o diferencial que tem justificado essa busca.

Além disso, a capacidade de balancear a relação entre *exploration* e *explotation* contida nos algoritmos do *Q-learning (off-policy)* ou de maximização de políticas de ações nos algoritmos do grupo de otimização de política *(on-policy)* possilitam uma automação industrial com a melhor regulação estratégica possível.

Pode-se dizer que o maior diferencial da aplicação de *machine learning* no contexto industrial está na forma em que a processo industrial é encarado nesses algoritmos, de forma que ela é vista como uma caixa negra, onde não se requer conhecimento prévio das variáveis do espaço de estado para a aprendizagem. Assim, apenas tomando as variáveis de entrada e saída do sistema, num processo interativo, é possível encontrar o melhor ponto de regulação que se adeque ao controlo do processo.

Por outro lado, segundo (Nian et al., 2020), a implementação de RL tem sido limitada atualmente à aplicação em sistemas de ações e estados discretos, muitas simuladas em Gémeos Digitais e que não foram implementadas em ambientes industriais de alto risco. Acrescentam-se também dois problemas de RL: o comportamento natural de caixa negra dos algoritmos que impede o entendimento de como e porque o agente aprendeu tal controlo de política e a falta de comunicação confiável entre os algoritmos e o sistema industrial controlados muitos vezes por PLC’s.

Há bastante estudo de sintonização de controlo proporcional-integrativo-derivativo PID por RL aplicado ao controlo industrial, onde a política de ações é aplicada a variáveis contínuas. Pode-se citar o estudo do (Shuprajhaa et al., 2022) que avalia o desempenho de algoritmos em RL, nomeadamente *Proximal Policy Optimization* (PPO), *Advantage Actor Critic* (A2C) e *Deep Deterministic Policy Gradient* (DDPG), no controlo linear e não linear em processos instáveis, quando comparados ao método convencional *Internal Model Control* IMC (Saxena and Hote, 2012). O que foi observado é um desempenho superior dos algoritmo RL frente ao modelo convencional, por conter menor oscilações no controlo e maior estabilidade na resposta ao *set-point* aplicado. Mais, observa-se falhas no controlo IMC quando a gama da operação ocorre em regiões instáveis e o mais alto *overshooting* quando comparado com os algoritmos em RL.

Relativamente às séries temporais em um processo industrial, Jiménez realiza um estudo desenvolvendo a criação de uma rede neuronal NARX como modelo de previsão de controlo de variáveis em um auto forno (Jiménez et al., 2004), que é um equipamento utilizado para a produção de aço em larga escala, onde o processo de fundição ocorre em um único recipiente. NARX, é uma variação de RNN concebida para lidar com séries temporais caóticas e complexas em sistemas de controlo, possui gradiente descendente de aprendizagem e convergência mais rápida e melhor do que outras redes, o que justifica essa escolha na aplicação. Consequentemente, essa implementação resultou em uma maior precisão no controlo das variáveis de auto forno do processo industrial em questão, pois a rede por meio dela memorizou a série temporal de dados de entrada dos passos anteriores.

3.2. Aprendizagem por Reforço em Gémeos Digitais

Devido à dificuldade, o risco e o tempo inviável de treino de algoritmos em RL diretamente em ambiente reais, têm surgido diversos trabalhos de Aprendizagem por Reforço em Gémeos Digitais.

Pode-se citar o trabalho de (Marius Matulis and Carlo Harvey, 2022) que utilizou o Unity para modelamento de um Gémeo Digital de um robô e o pacote ML Agent. Esse pacote do Unity proporciona o recebimento dos dados de estado do ambiente virtual e os exporta para o TensorFlow para que essa biblioteca realize o treinamento. A cada ação tomada pelo agente, o dado de ação é retornado ao Unity para que ocorra o fechamento do ciclo do Processo de Decisão de Markov (MDP), cabendo então ao TensorFlow gerir a rede neuronal

e os algoritmos de treino, sendo que para esse estudo foram utilizados os algoritmos SAC e PPO. O problema em questão tratava de um braço robótico que deve coletar peças das cores verdes ou vermelhas e deslocá-las para outras posições demarcadas.

O robô do ambiente real possuí um dispositivo Microsoft Kinect que contém uma câmera com API para Unity, possibilitando a troca de informações entre ele e o Gémeo Digital (ver figura 12). Por sua vez, esse Gémeo Digital, quando treinado, recebe as informações do estado real e as gera para o robô replicá-las, havendo ali um atraso de 2 segundos entre esses 2 elementos para caso houvesse algum problema, o agente em RL poder atuar com segurança.

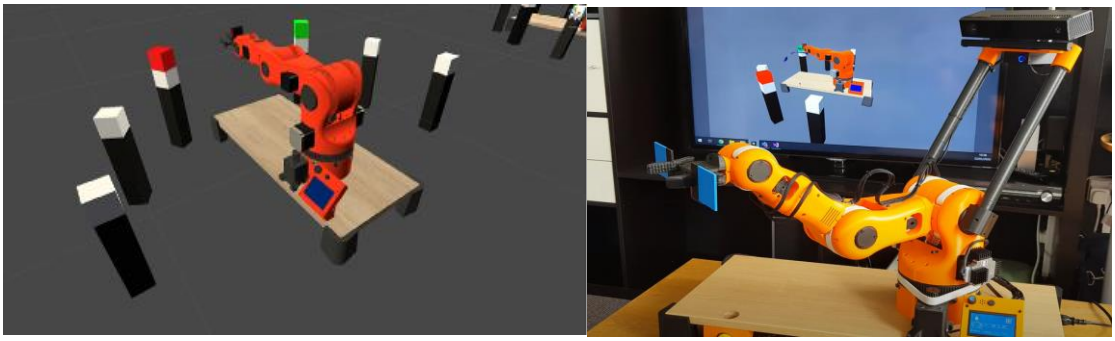

Figura 12: Braço robótico em realidade virtual e em ambiente real (Marius Matulis and Carlo Harvey, 2022)

Relativamente às recompensas de treino, inicialmente o agente é incentivado a explorar o ambiente, penalizando a extrapolação de certos limites de posições, colisões ou aberturas e fechamentos indevidos da garra. Entretanto, à medida que o ambiente é explorado, recompensas maiores ocorrem quando o objetivo de posicionamento do cubo de um ponto ao outro é realizado.

O número de sensores que compõem o controlo de cinemática inversa foi reduzido da primeira para segunda interação (grande quantidade de treinos), porque a grande quantidade de variavéis nesse espaço de estados inviabilizava a exploração e aumentava o número de observações requeridas. Assim, o número de sensores minimizados gerava informações somente em relação a deteção de obstáculos e variavéis de quarteniões para agregação das rotações do robô.

A arquitetura da rede neuronal combinou 3 camadas de mais de 128 neurónios, funções de ativação *Swiss* (ver Apêndice A) com outras 3 camadas de mais de 128 neurónios em CNN

e funcões de ativação *softmax* para formação das camadas intermediárias. Essas, por conseguinte, foram condensadas em uma única camada de saída de rede neuronal de política de ações que melhor refletia uma devida reação ao vetor de estados S apresentado.

Após 30 horas de treino, utilizando múltiplos agentes treinados em ambientes diferentes no Unity, o algoritmo PPO foi o que trouxe melhores resultados. O maior desafio apontado pelos autores foram os atalhos sempre procurados pelos agentes para burlar a lógica estabelecida pelo ambiente da realidade virtual e dessa forma receber maior recompensa. Por outro lado, isso também foi algo que pode apontar falhas na simulação e no ambiente real, fazendo com que o usuário pudesse se precaver de acidentes ou erros, corrigindo tais situações. Punições ou recompensas altas levaram o treino à criação de plateaus que por sua vez impediam o agente de atingir o sucesso, devendo dessa forma haver um equilíbrio e bom senso na estipulação desses valores.

Em trabalho similar, Lee e Choi (Le et al., 2021) avaliaram o uso do algoritmo PPO para controlo de um sistema de coleta e posicionamento de peças por diversos manipuladores em uma esteira transportadora. Cada manipulador possuí o seu local pré-definido de posicionamento das peças e o processo foi simulado em ROS e Unity, conforme mostrado na figura 13 abaixo:

Figura 13: Processo de coleta e posicionamento de peças de uma esteira transportadora (Le et al., 2021)

O agente tem informações dos ângulos dos manipuladores, da posição dos objetos, da velocidade da esteira, da posição da garra e do local de posicionamento, podendo atuar nessas mesmas variáveis, exceto na posição corrente do objeto. As recompensas foram estipuladas de modo a estimular os manipuladores a realizarem o menor movimento possível juntamente com a maior velocidade de coleta e posicionamento das peças.

A arquitetura da rede neuronal combinou camadas de 128 neurónios comuns com outros 256 neurónios da camada LSTM, separando a saída em duas redes neuronais paralelas de valor de função Q e política de ações.

O soluçao obtida, resultou em que o aumento da velocidade da esteira diminuiu o tempo de coleta e posicionamento, ao mesmo tempo que com 60 episódios de treino, o agente conseguiu controlar 3 manipuladores com performance razoável.

3.3. Discussão

É perceptível o aumento de estudos de RL aplicados ao controlo de processos industriais, em variáveis discretas e contínuas, principalmente quando se trata de sintonização de PID's. Relativamente a essa sintonização, esse aumento é motivado porque o processo requer o conhecimento prévio das variáveis de espaço de estados que compõem o processo industrial na calibração, sendo que calculá-las e obtê-las não é uma tarefa trivial para o desenvolvedor. Além disso, tais variáveis se modificam no decorrer do tempo, requerendo intervenções periódicas do controlo PID, interrompendo o processo industrial.

Embora, os conceitos e teorias relativos a RL já estejam bastante assentados e estudados ao longo de mais de 30 anos, há ainda desconhecimento do mecanismo que ajusta os pesos das redes neuronais e de uma explicação coerente por parte do algoritmo em RL de como se chegou ao fim do treinamento em uma determinada política de ações. Por essa razão, RL ainda é vista também como uma caixa negra, deixando receios por parte da comunidade académica, empresas e desenvolvedores na implementação em ambientes reais e que requerem maior confiabilidade para aplicação.

Há também poucos estudos relativos à eficiência desses algoritmos na interpretação de séries temporais em ambientes industriais, fenómeno bastante recorrente em sistemas e que deveria haver melhor consideração por parte dos pesquisadores. Isso porque é importante ao agente em RL memorizar os estados em cada passo da jornada do processo industrial a fim de se obter a melhor estratégia de controlo no ciclo de produção final do produto.

Estabelecer as corretas recompensas e punições é algo também visto como muito importante para o bom desempenho do algoritmo em RL, evitando que o agente caia em mínimos locais indesejados no espaço de soluções.

Há inúmeros estudos do uso de Gémeo Digital na simulação e replicação de estados de ambientes reais em ambientes virtuais, com o propósito de se evitar acidentes, custos e obter ganho no tempo da simulação e treinamento de algoritmos, sendo que a comunicação entre esses elementos ainda é uma barreira por se utilizarem diferentes linguagens e tecnologias. Além disso, há também receio de empresas e desenvolvedores relativamente à integridade e à confiabilidade da comunicação entre o agente em RL e os dispositivos de controlo industrial, nomeadamente PLC's e DCS's.

Por fim, há índicios de performance e de eficiência superior da Aprendizagem por Reforço em relação à aprendizagem supervisionada e não supervisionada, o que tem levado muitos desenvolvedores à optarem por implementar essa classe de algoritmos. Os resultados e sucessos obtidos em testes e em estudos de RL são promissores e indicativos de um aumento dessa preferência em técnicas e métodos no controlo de processos industriais.

4. Simulador de Sistema Industrial Discreto: Processo de Empacotamento de Latas (PEL)

Este capítulo apresenta o funcionamento de dois sistemas que compõem o processo de empacotamento de latas (PEL): a esteira de caixa de latas (ECL) e a garra manipuladora de latas (GML). São descritos os sensores e atuadores que compõem as variáveis de estado e de ações implementadas em cada um, apresentando o fluxograma de passos para fechamento do ciclo de cada processo, juntamente com os possíveis trajectos percorridos por ECL e por GML. (Aprendizagem por Reforço em Processos Sequenciais - GitHub, 2023)

Ambos os sistemas foram desenvolvidos pela Real Virtual (OPCUA4Unity | Utilities Tools | Unity Asset Store, 2023) e simulados no Unity, constituindo uma realidade virtual que troca dados de estados e de ações com um agente que está externamente a esse *framework*. A preferência pelo Unity se deu pela ampla variedade de recursos e ferramentas que são úteis na criação de Gémeos Digitais/Simuladores, nomeadamente o suporte para modelagem 3D, animações e física, que são recursos importantes que possibilitam a troca de dados entre o agente e o ambiente. Além disso, o Unity possui uma grande comunidade de desenvolvedores e usuários que compartilham recursos, por meio da qual foi obtido o próprio simulador já pré-concebido e supracitado e que facilitou a realização deste trabalho.

4.1. Simulador da Esteira de Caixa de Latas (ECL)

A esteira de caixa de latas (ECL) de refrigerante é um sistema que consiste em posicionar essa caixa nos pontos pré-determinados para recebimento de latas preenchidas por uma garra que movimenta-se com dois graus de liberdade (esquerda-direita, cima-baixo). São colocadas um total de 9 latas, distribuídas em 3 filas de 3 latas, no qual a caixa que recebe essas latas, por meio de um terceiro grau de liberdade (frente-trás), precisa ser movida pela passadeira para concluir o preenchimento. É composta por um motor, dois sensores de fim de curso e um sensor de posicionamento da caixa e é movida da posição zero, onde se situa a tomada da caixa por uma garra robótica, para as posições 400, 500 e 600 cm, pontos estes de recebimento de latas pela garra de latas. Ao finalizar o enchimento, a esteira retorna a caixa para a posição de retoma da garra robótica, que por sua vez esvazia a caixa, refazendo o ciclo de enchimento. Abaixo, segue a imagem desse sistema na figura 14, mostrando os sensores e posições relevantes:

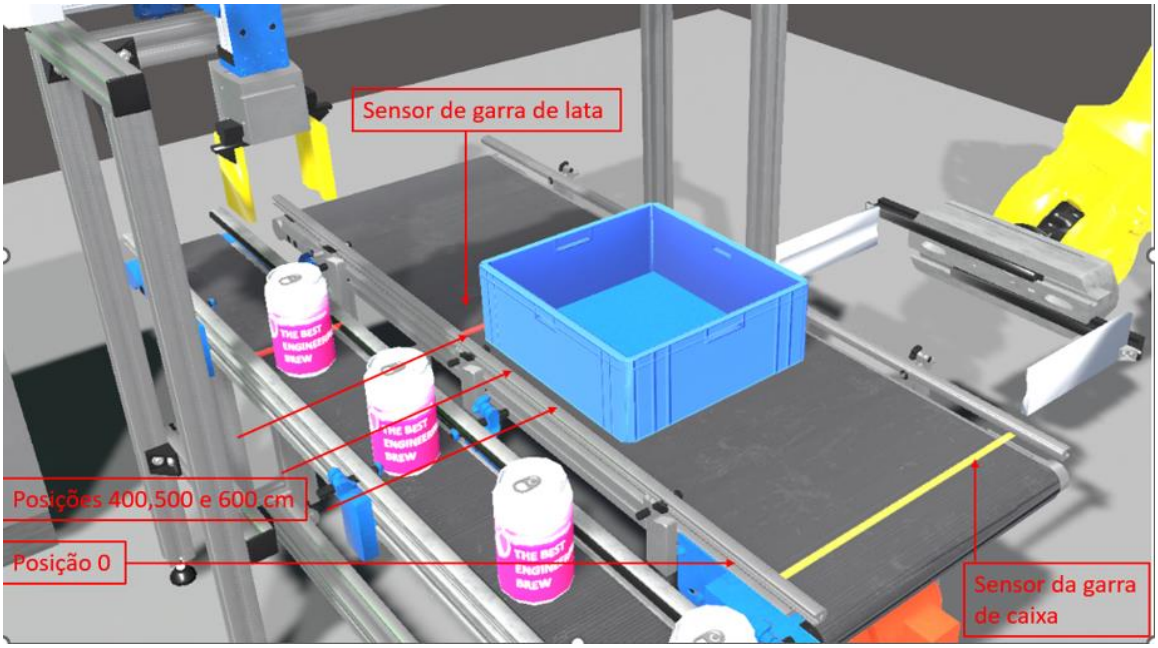

Figura 14: ECL com seus devidos sensores e posições relevantes.

É permitido ao ECL realizar 3 tipos diferentes de ações discretas:

- Parar;
- Movimentar para frente;
- Movimentar para trás.

Por sua vez, o processo recebe informações de estado de 4 variáveis:

- Sensor de fim de curso do robô de esvaziameneto da caixa (variável discreta);
- Sensor de fim de curso da posição da garra de manipulação de latas (variável discreta);
- Sinal discreto enviado pela garra de manipulação de latas indicando que a fileira foi preenchida;
- Sinal de posição da caixa no percurso da esteira, variando de 0 a 600 cm.

Um descrição completa do fluxograma de passos do processo de ECL segue abaixo na figura 15. Além do fluxograma, o Apêndice C apresenta uma sequência resumida do processo ECL, quadro a quadro, da realidade virtual pré-concebida pela Real Virtual e que ilustra o fluxograma citado acima.

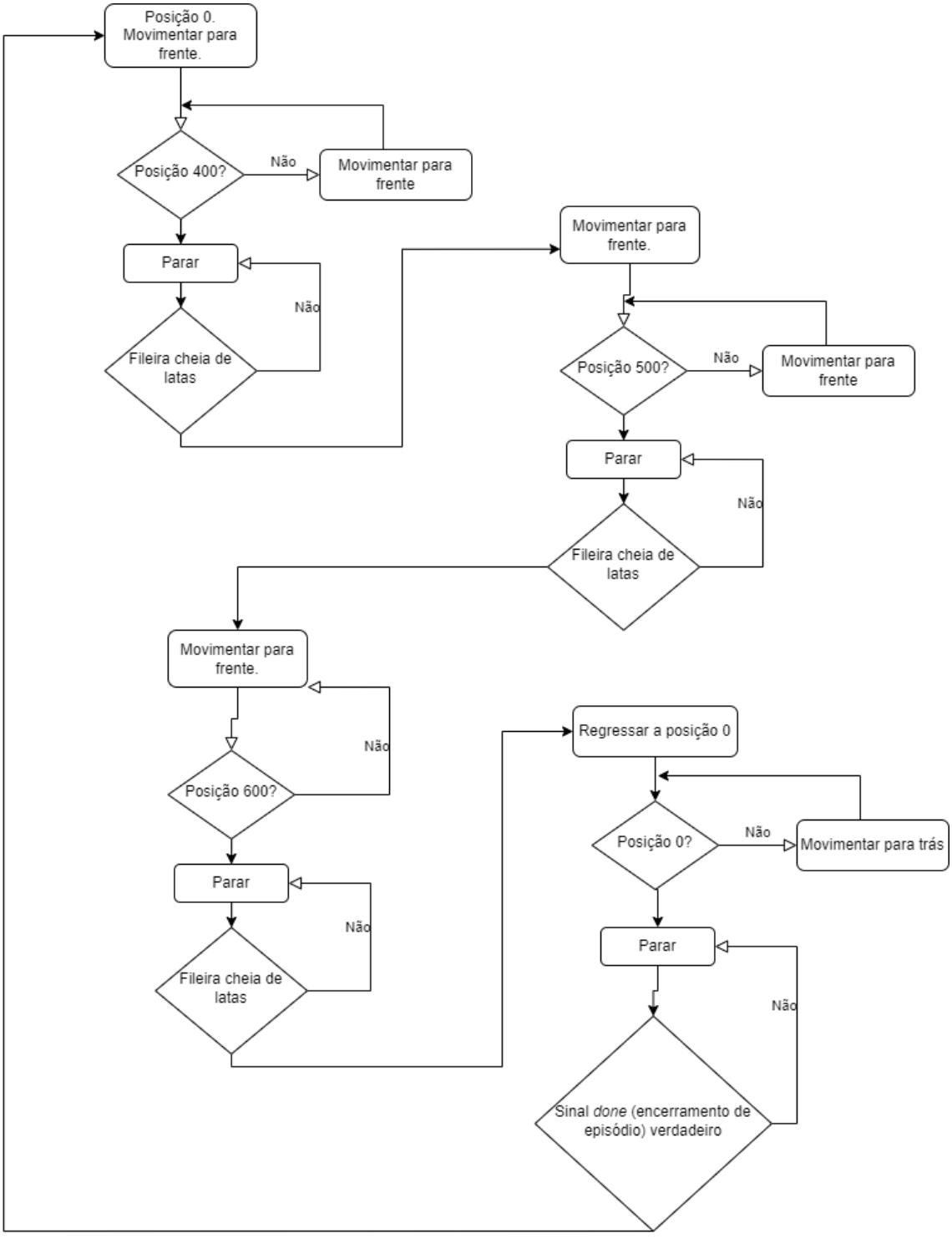

Figura 15: Fluxograma de passos do processo da ECL.

4.2. Simulador da Garra Manipuladora de Latas (GML)

A garra manipuladora de latas (GML) é um sistema que consiste em coletar latas transportadas por uma esteira de alimentação e posicioná-las em uma caixa numa segunda esteira. É composta por uma garra, um motor que se movimenta no eixo x e outro motor que

se movimenta no eixo y. Cabe a essa máquina preencher 3 fileiras com 3 latas na caixa, sem deixar que as latas batam umas nas outras ou batam em obstáculos e sem realizar a captura das latas quando qualquer uma das esteiras esteja em movimento. A cada fileira preenchida pela garra, esse agente deve enviar um sinal de conclusão ao ECL para que ela se mova para outra posição alvo. É permitido a garra realizar 11 tipos diferentes de ações discretas:

- Abrir a garra;
- Fechar a garra;
- Sinalizar preenchimento de fileira ao ECL;
- Movimentar para as 8 posições descritas na figura 16 da imagem abaixo, podendo se deslocar de qualquer posição para a outra.

Por sua vez, o sistema recebe informações de estado de 6 variáveis:

- Sensor de fim de curso do ECL (variável discreta);
- Sensor de presença de lata na garra (variável discreta);
- Posição da garra no eixo Y (variável contínua);
- Posição da garra no eixo X (variável contínua);
- Sensor de fim de curso da garra, indicando se está fechada ou aberta (variável discreta);
- Sinal enviado pelo ECL indicando que se ela está em movimento ou não está sobre o sensor de fim de curso da posição da garra (variável discreta).

Abaixo, segue a imagem desse sistema na figura 16, mostrando posições e trajectos relevantes:

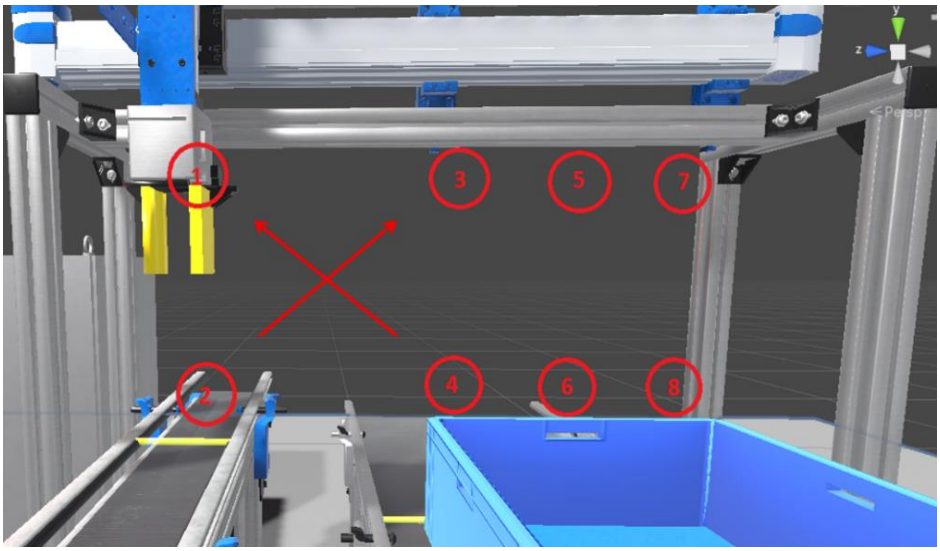

Figura 16: Garra manipuladora latas, posições e trajetos relevantes.

Alguns trajectos relevantes para conclusão dos passos são listados a seguir, sendo que a ordem desses trajectos não impacta na performance do processo do GML :

- 1º lata: posições 1 – 2 – fechar - 3 – 4 – abrir – 1;
- 2º lata: posições 1 – 2 – fechar - 5 – 6 – abrir – 1;
- 3º lata: posições 1 – 2 – fechar - 7 – 8 – abrir – 1;

Vale a pen notar que esses trajectos são caminhos de menor distância percorridos pela garra e preferidos pelos agentes treinados em RL, pois geram maior recompensa resultante da aprendizagem (descrição de recompensas no capítulo 5.2.1). Entretanto, trajectos mais longos poderiam ser tomados, concluindo o ciclo em tempo maior, salvo deslocamentos entre a posição 2 e 4 que poderia gerar colisões com outros objetos.

Um descrição completa do fluxograma de passos do processo de GML segue abaixo na figura 17. Além do fluxograma, o Apêndice C apresenta uma sequência resumida do processo GML, quadro a quadro, da realidade virtual pré-concebida pela Real Virtual e que ilustra o fluxograma citado acima.

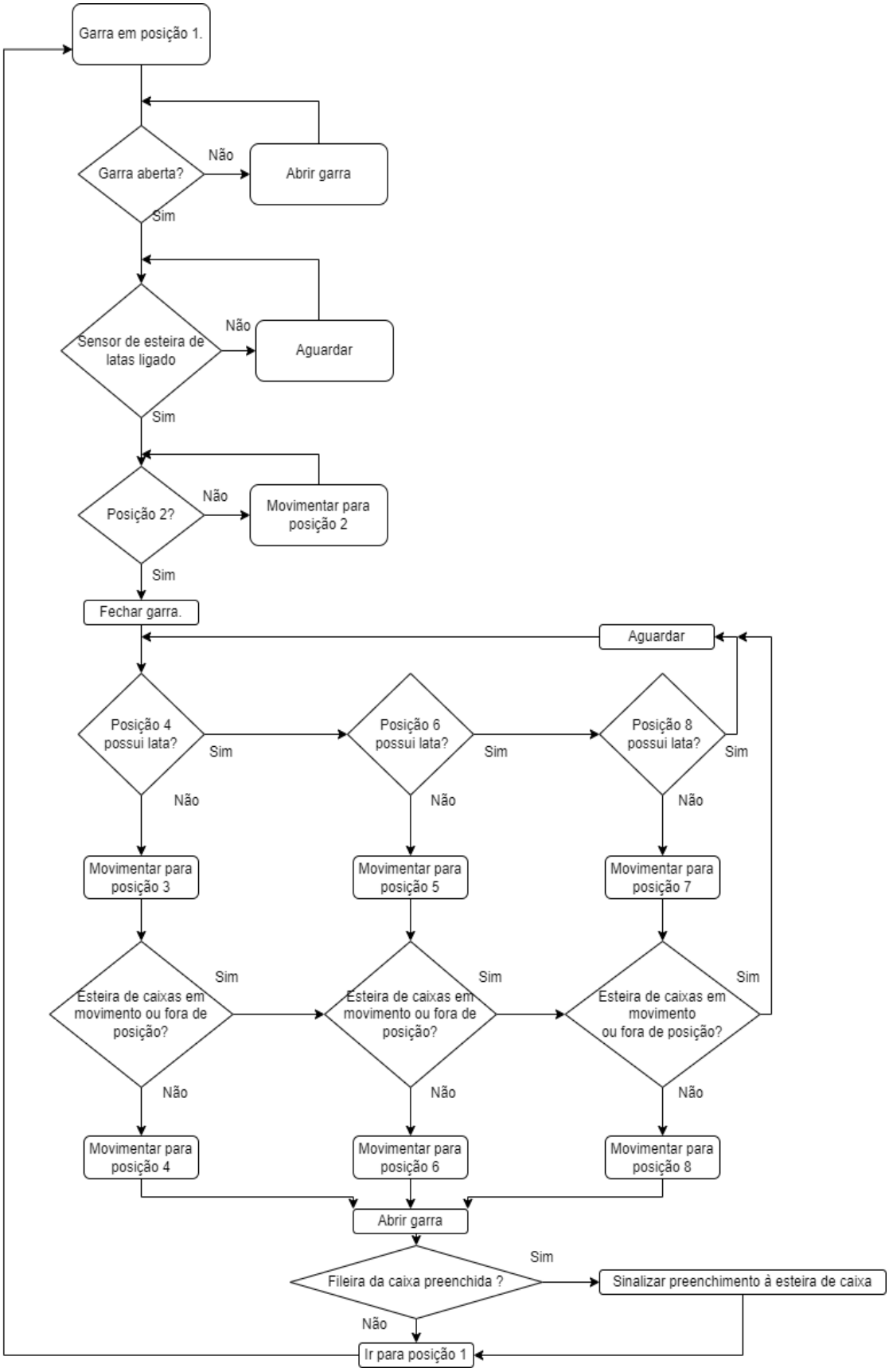

Figura 17: Fluxograma de passos do processo do GML.

5. Aprendizagem por Reforço Aplicada ao Processo de Empacotamento de Latas (PEL)

Este capítulo apresenta o desenvolvimento e a performance de algoritmos em RL aplicados à esteira de caixa de latas (ECL) e à garra manipuladora de latas (GML) que compõem o processo de empacotamente de latas (PEL). São descritas as recompensas e as punições implementadas no treino de cada sistema, avaliando os diferentes algoritmos e as diferentes configurações de arquiteturas de redes neuronais, comparando-os em todos os processos.

Em relação aos estados, é importante destacar que, conforme explicado no capítulo 2.4, para que o agente tome a melhor ação com recompensa resultante, é necessário que o estado S retenha todas as informações relevantes ao PEL, ou seja, além de obter a informação das variavéis no estado atual, é necessário ter a informação dos espaços anteriores. Isso porque as ações tomadas pelo agente em PEL são dependentes uma das outras em cada passo do ciclo, não podendo descorrelacioná-las, sendo necessário um efeito memória dada a natureza sequencial do processo. Dessa forma, a implementação de memória neste trabalho se deu por meio de redes neuronais LSTM's e *buffers* de memória de estados anteriores (informação explícita já na entrada do algoritmo) em cada sistema, constituindo um estudo comparativo de performance de ambos os métodos aplicados a processos industriais. A ideia aqui era avaliar se as LSTM's possuem ou não capacidade de memorizar estados anteriores e em que nível de qualidade e performance isso acontece.

Todos os algoritmos foram pré treinados em ambientes programados em Python, nas bibliotecas Stable Baselines 3 e Keras, para posterior simulação na realidade virtual no Unity, sendo que a troca de informações entre o agente e esse ambiente ocorreu por meio de comunicação Websocket (ver figura 18). Tal preferência pelo pré treinamento nessas bibliotecas se deu pela superioridade de velocidade de processamento computacional e pela maior disponibilidade e flexibidade de algoritmos quando comparados a um treino feito no Unity, pela biblioteca ML-Agents Toolkit (Unity ML-Agents Toolkit, 2023).

A utilização da Websocket foi necessária por haver programação de códigos em scripts distintos na realidade virtual e nas bibliotecas de RL, nomeadamente C# e Python. Assim, a troca de informações entre o agente e o ambiente via Websocket ocorreu por meio de protocolo TCP/IP, em uma conexão cliente e servidor, gerida por um terceiro algoritmo.

Essa utilização foi bastante vantajosa, uma vez que a desacoplagem entre os código de simulação e treino permitiu mudanças e adequações, sem que um interferisse no outro durante o desenvolvimento.

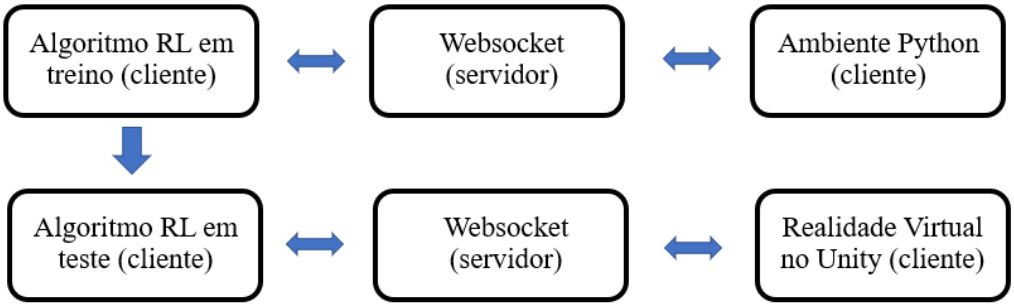

Figura 18: Esquema de treinamento em Python e de teste em Unity

5.1. Parâmetros e arquiteturas do modelo em Aprendizagem por Reforço aplicadas ao PEL

Para desenvolvimento e modelagem de Aprendizagem por Reforço em PEL, foram avaliados os algoritmos DQN, SARSA, CEM, A2C, TRPO e PPO em diferentes configurações. Os hiperparâmetros foram padronizados conforme a tabela 1 abaixo, mantendo os seguintes valores especificados pelas bibliotacas:

Algoritmo	Taxa de Aprendizagem (α)	Fator de Desconto (γ)	Ação Ambiciosa	Restrição KL	Limite para *Clipped Surogate Loss* (ϵ)
DQN	0,001	0,95	Boltzmann Q-policy	-	-
Sarsa	0,001	0,95	Boltzmann Q-policy	-	-
CEM	-	0,95	-	-	-
A2C	0,0007	0,95	-	-	-
TRPO	0,001	0,95	-	0,01	-
PPO	0,0003	0,95	-	-	0,2

Tabela 1: Hiperparâmetros dos algoritmos avaliados no PEL.

Para avaliação das diferentes configurações de arquiteturas de redes neuronais, foram estabelecidos diferentes parâmetros para comparação. Partindo-se do pressuposto que a quantidade de passos necessários para conclusão tanto do ECL, como do GML, varie entre 10 a 30 passos (podendo haver tarefas cíclicas), são entregues na entrada do algoritmo a informação dos estados dos últimos passos anteriores, constituindo esses os *buffers* de memória.

Os algoritmos em RL com essas arquiteturas foram comparados com redes neuronais LSTM para avaliação da performance e qualidade de memorização de passos anteiores. Os

algoritmos e as bibliotecas utilizadas possuem limitações e particularidades, sendo que nem todos os algoritmos possuem disponibilidade de implementação de LSTM e os algoritmos A2C, PPO e TRPO, do grupo de otimização de política, possuem duas redes neuronais paralelas: rede de valor Q e rede de política. Por padrão das bibliotecas, essas redes neuronais utilizam 2 camadas ocultas de 64 neurónios de função de ativação tangencial hiperbólica e neurónios de funções de ativação softmax na camada de saída (número de neurónios dependente da quantidade de ações necessárias). Abaixo, segue abaixo a tabela 2 que informa a disponibilidade desses recursos utilizados:

Algoritmo	*Buffer* de estados	Rede neuronal de valor	Rede neuronal de política	LSTM
DQN	Sim	Sim	Não	Sim
Sarsa	Sim	Sim	Não	Sim
CEM	Sim	Sim	Não	Sim
A2C	Sim	Sim	Sim	Não
TRPO	Sim	Sim	Sim	Não
PPO	Sim	Sim	Sim	Sim

Tabela 2: Disponibilidade de configurações das redes neuronais em algoritmos de RL.

Não há um único método para calcular o número ideal de neurónios em uma rede neuronal, pois o dimensionamento depende do problema específico em questão. Normalmente começa-se com um pequeno número de neurónios, aumentando gradualmente o tamanho da rede até que o desempenho desejado seja alcançado. Adicionar demasiados neurónios pode aumentar o risco de ajuste excessivo dos dados, sendo necessário equilibrar a complexidade da rede com sua capacidade de generalização, descrito no trabalho de Hagan (Hagan and Menhaj, 1994). Por sua vez, em outro estudo aprofundado sobre os efeitos do número de neurónios e camadas ocultas em ANN (Adil et al., 2022), os autores concluem que utilizar menos neurónios do que o necessário nas camadas ocultas resulta em um maior erro médio quadrático da rede e que o arranjo desses neurónios impacta na performance final do modelo. Relativamente ao número de camadas ocultas, obtém-se melhores resultados para uma ou duas camadas em comparação com 3, 4 ou 5 camadas ocultas (Adil et al., 2022).

Dessa forma, foram experimentados diferentes quantidades de neurónios em configurações de 2 e 3 camadas ocultas nas redes neuronais LSTM, para obtenção e comparação com as redes neuronais comuns de 2 camadas de 64 neurónios. Os resultados desses modelos

treinados em diferentes arquiteturas de LSTM foram aplicados ao ECL e ao GML e descritos nos capítulos seguintes.

5.1.1. Aprendizagem por Reforço e definição das recompensas do ECL

Foram avaliados os algoritmos DQN, SARSA e CEM, provenientes da biblioteca Keras, comparando o impacto da utilização de *buffers* de memorização dos estados anteriores em 3 diferentes situações: sem buffer, com buffer dos 10 passos anteriores, com buffer dos 25 passos anteriores. Nessa forma explícita de memorização, a rede neuronal passou a receber respetivamente 4, 40 e 100 variáveis de entrada, cabendo a ela aprender quais delas são referentes a cada passo realizado. Em DQN, foram experimentadas diferentes arquiteturas de redes neuronais comuns e LSTM para comparação com o *buffer* de memorização.

Relativamente a biblioteca Stable Baselines, foram avaliados os algoritmos A2C, TRPO e PPO com os seguintes *buffers* de memorização: sem buffer, com buffer dos 3, 10 e 25 passos anteriores. Já nessa forma explícita de memorização, a rede neuronal passou a enxergar respetivamente 4, 12, 40 e 100 variáveis de entrada. Em PPO, foram experimentadas diferentes arquiteturas de redes neuronais LSTM aplicadas as redes de valor Q e redes de política para comparação com o *buffer* de memorização. A tabela 3 abaixo resume esses algoritmos avaliados em ECL, quanto às características de bibliotecas utilizadas, *buffer* de memorização, variáveis de entrada e arquitetura de rede neuronal.

Todos esses algoritmos foram treinados com 1 milhão de passos interativos, sem interrupção, sendo que alguns alcançaram o objetivo já no meio do treinamento.

Visto que RL aprende com base em recompensas, pretende-se que o sistema aprenda uma sequência de operações que cumpra o objetivo e que seja eficiente. Essa aprendizagem deve ser feita de forma autónoma, sem intervençao humana, para especificar todos os passos necessários ao processo, pois caso contrário seria o humano a programar o sistema. Assim, as recompensas definidas devem evitar recompensar passo-a-passo a sequência de operação, mas sim recompensar resultados finais de sucesso e penalizar situações gerais de insucesso (como colisões, por exemplo), definindo objetivos gerais, buscando diminuir o número de passos para se chegar lá.

Algoritmo	**Biblioteca**	***Buffer* de Memorização**	**Número de Variáveis de Entrada**	**Arquitetura de Rede Neuronal (tipo de camada, quantidade de camadas e de neurónios)**
DQN	Keras	Sem *buffer*	4	*Dense Laye*r - 64 X 64
DQN	Keras	*Buffer* de 10 passos	40	*Dense Laye*r - 64 X 64
DQN	Keras	*Buffer* de 25 passos	100	*Dense Laye*r - 64 X 64
DQN	Keras	*Buffer* de 25 passos	100	*Dense Layer* - 128 X 128 X 128
DQN	Keras	*Buffer* de 25 passos	100	*LSTM Layer* - 128 X 128 X 128
DQN	Keras	*Buffer* de 25 passos	100	*LSTM Layer* - 128 X 96 X128 X 96
SARSA	Keras	Sem *buffer*	4	*Dense Laye*r - 64 X 64
SARSA	Keras	*Buffer* de 10 passos	40	*Dense Laye*r - 64 X 64
SARSA	Keras	*Buffer* de 25 passos	100	*Dense Laye*r - 64 X 64
A2C	Stable Baselines	Sem *buffer*	4	*Dense Laye*r - 64 X 64
A2C	Stable Baselines	*Buffer* de 3 passos	12	*Dense Laye*r - 64 X 64
A2C	Stable Baselines	*Buffer* de 10 passos	40	*Dense Laye*r - 64 X 64
A2C	Stable Baselines	*Buffer* de 25 passos	100	*Dense Laye*r - 64 X 64
TRPO	Stable Baselines	Sem *buffer*	4	*Dense Laye*r - 64 X 64
TRPO	Stable Baselines	*Buffer* de 3 passos	12	*Dense Laye*r - 64 X 64
TRPO	Stable Baselines	*Buffer* de 10 passos	40	*Dense Laye*r - 64 X 64
TRPO	Stable Baselines	*Buffer* de 25 passos	100	*Dense Laye*r - 64 X 64
PPO	Stable Baselines	Sem *buffer*	4	*Dense Laye*r - 64 X 64
PPO	Stable Baselines	Sem *buffer*	4	*LSTM Layer* - 64 X 64
PPO	Stable Baselines	Sem *buffer*	4	*LSTM Layer* - 256 X 256 X 256
PPO	Stable Baselines	Sem *buffer*	4	*LSTM Layer* - 512 X 512 X 512
PPO	Stable Baselines	*Buffer* de 3 passos	12	*Dense Laye*r - 64 X 64
PPO	Stable Baselines	*Buffer* de 10 passos	40	*Dense Laye*r - 64 X 64
PPO	Stable Baselines	*Buffer* de 25 passos	100	*Dense Laye*r - 64 X 64

Tabela 3: Algoritmos avaliados em ECL quanto às características de bibliotecas utilizadas, *buffer* de memorização, variáveis de entrada e arquitetura de rede neuronal.

Dessa forma, o ambiente de treino do ECL foi programado com as regras e as recompensas descritas a seguir, de modo que conduzisse o agente à sequência correta do ciclo completo do processo, incentivando o sistema a atingir o objetivo o mais rapidamente possivel. Embora os valores possam ser arbitradados livremente, procurou-se manter o bom senso nessa seleção, pois valores mal dimensionados podem desestimular o agente a explorar o mapa de soluções ou prendê-lo em mínimos locais, comprometendo a aprendizagem. Dessa forma, as regras foram estipuladas conforme os itens seguintes, incentivando os seguintes comportamentos:

- Para preencher a 1º fila, se a caixa possui 3 latas ou menos e está na posição 400 cm,

$$\mathrm{R} = 100 - \frac{n^{\underline{o}}\ passos}{5} \tag{5.1}$$

- Para preencher a 2º fila, se a caixa possui mais de 3 latas e menos de 7 latas e está na posição 500 cm,

$$\mathrm{R} = 200 - \frac{n^{\underline{o}}\ passos}{5} \tag{5.2}$$

- Para preencher a 3º fila, se a caixa possui mais de 6 latas e menos de 10 latas e está na posição 500 cm,

$$\mathrm{R} = 300 - \frac{nº\ passos}{5} \tag{5.3}$$

- Para retornar a caixa a posição de descarga, se a caixa possui 9 latas, está em qualquer posição maior que 0 e a esteira aciona o motor para movimentar a caixa para trás:

$$\mathrm{R} = (\text{nº latas em posições corretas} * 120) - \frac{posição\ da\ esteira}{8} \tag{5.4}$$

- Para manter a esteira parada durante a etapa de descarga, se a caixa está na posição 0 (sensor fim de curso da garra robótica acionado), possui 9 latas em posições corretas na caixa e a esteira está parada,

$$\mathrm{R} = 7000 \tag{5.5}$$

- Para manter a esteira parada durante a etapa de descarga, se a esteira permanece parada enquanto a caixa está sendo descarregada pela garra robótica (sensor de fim de curso desacionado),

$$\mathrm{R} = 1000 \tag{5.6}$$

- Para recompensar o sucesso do agente no encerramento do ciclo, assim que a caixa é retornada à esteira, agora descarregada, (sensor de fim de curso acionado) e a esteira permanece parada, o sinal de *done* (episódio encerrado) é enviado ao agente e

$$\mathrm{R} = 17000 - \text{nº de passos} \tag{5.7}$$

- Para desestimular qualquer comportamento fora do ciclo, qualquer outro cenário que não correspondente às possibilidades acima,

$$\mathrm{R} = 0 \tag{5.8}$$

- Durante o processo, caso a esteira se movimente para trás da posição -200 cm ou para frente da posição +800 cm, o sinal de *done* é enviado ao agente para que impeça aprendizagem (exploração de políticas) em comportamentos inseguros;
- Para desestimular jornadas fora da estratégia correta, caso o número de passos para conclusão do processo seja maior que 500, o sinal de *done* é enviado ao agente.

5.1.2. Resultados dos modelos de Aprendizagem por Reforço aplicados ao ECL

Neste capítulo são mostrados os resultados dos algoritmos treinados em ECL, comparando a performance de cada deles um por meio de diferentes gráficos de recompensa média, divergência KL, perda de entropia cruzada e fração de uso de limite *Clipped Surrogate Loss* (todos no eixo cartesiano Y) em relação ao número de passos treinados (todos no eixo cartesiano X). Para diminuir algumas oscilações, o TensorBoard (TensorBoard, 2023), *framework* gerador desses gráficos, suaviza algumas dessas linhas, deixando uma sombra ao fundo que não impacta na qualidade e na interpretação da performance dos modelos.

Ao se comparar os algoritmos treinados em DQN com diferentes tamanhos de *buffers* de memória, percebe-se pelo gráfico da figura 19 desempenhos insatisfatórios em todos os modelos, pois não há estabilidade e constância dessas recompensas durante o treino, refletindo numa má atuação do agente em ECL. Entretanto, pode-se notar desempenho superior de recompensa média no *buffer* de memória de 25 passos anteriores frente aos outros dois modelos, indicando o fechamento correto do ciclo do ECL.

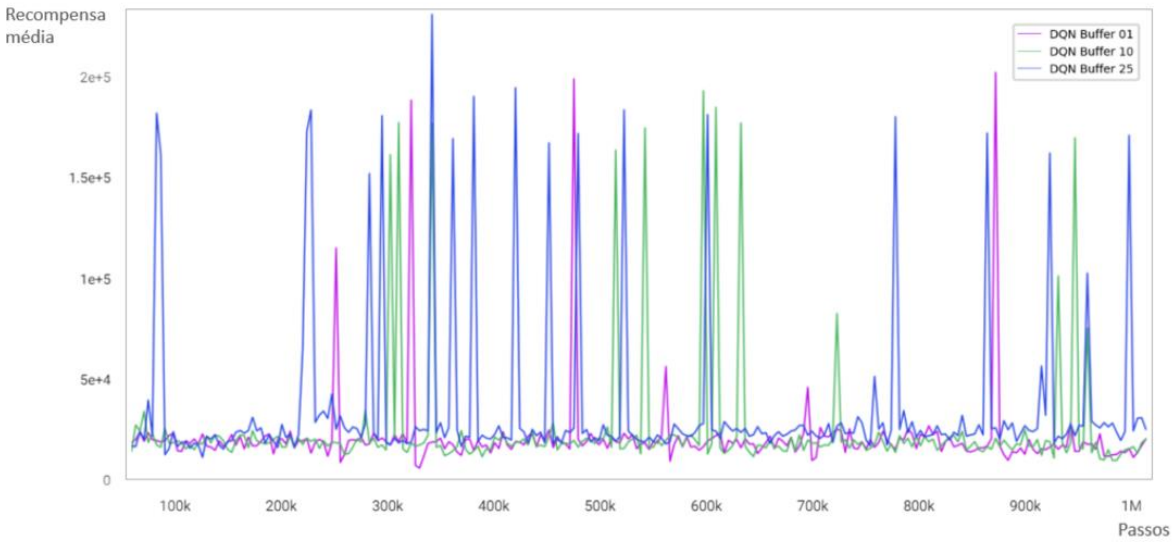

Figura 19: Recompensa média entre algoritmos DQN, com diferentes tamanhos de buffers, aplicados ao ECL.

Mantendo-se como referência esse modelo de maior alcance de recompensas, o algoritmo DQN com *buffer* de memória de 25 passos anteriores, e modificando a arquitetura das redes neuronais, não se alcança resultados melhores do que o modelo padrão, como se pode ver

no gráfico da figura 20. Essa arquitetura padrão de 2 camadas ocultas de 64 neurónios foi comparada com modelos de:

- 3 camadas de 128 neurónios;
- 3 camadas de 128 neurónios LSTM;
- 4 camadas com 128, 96, 128 e 96 neurónios em LSTM, repectivamente em cada camada.

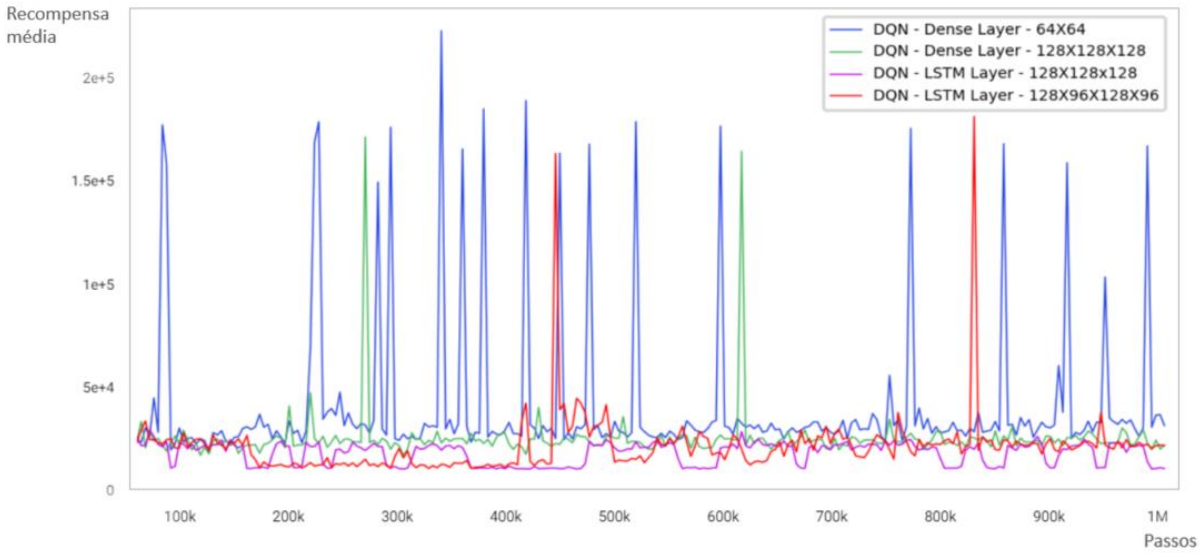

Figura 20: Recompensa média entre algoritmos DQN, com buffer dos últimos 25 passos, diferentes arquiteturas de redes neuronais, aplicados ao ECL.

Nesse último modelo, a ideia era se inspirar na arquitetura de *Auto enconders* (descrição no apêndice A), de modo a forçar a rede neuronal a aprender dados ocultos contidos em ECL. Novamente, obteve-se altas recompensas, mas com baixa estabilidade e constância, invalidando os modelos para aplicação em ECL.

De modo similar a esses algoritmos em DQN, como se pode ver nos gráficos das figuras 21 e 22, SARSA e CEM também resultaram nas mesmas características de recompensa média: baixa constância e grande instabilidade, não havendo aqui uma boa aprendizagem.

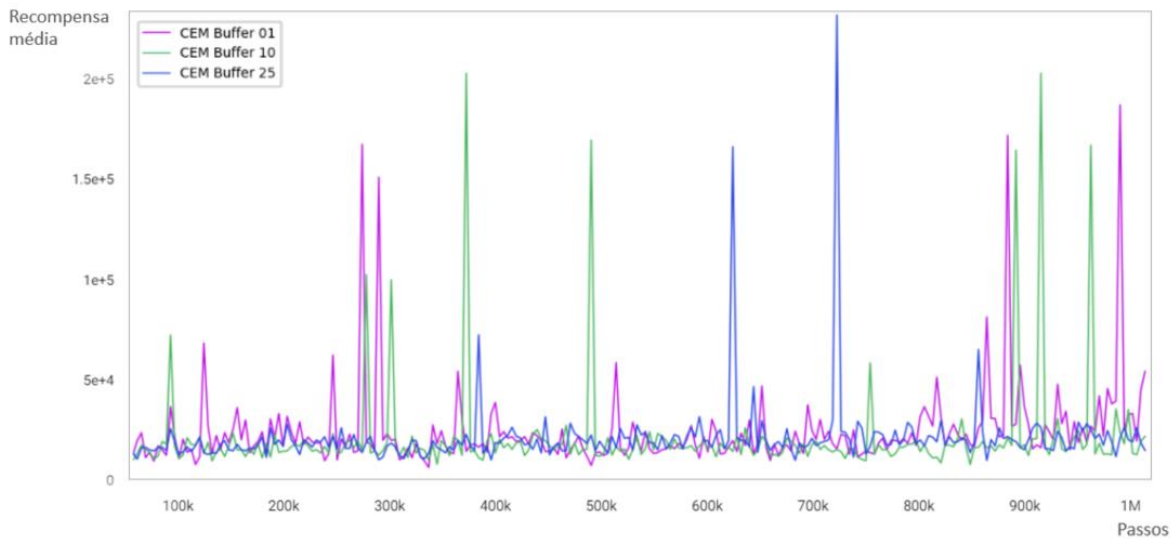

Figura 21: Recompensa média entre algoritmos CEM, com diferentes tamanhos de buffers, aplicados ao ECL.

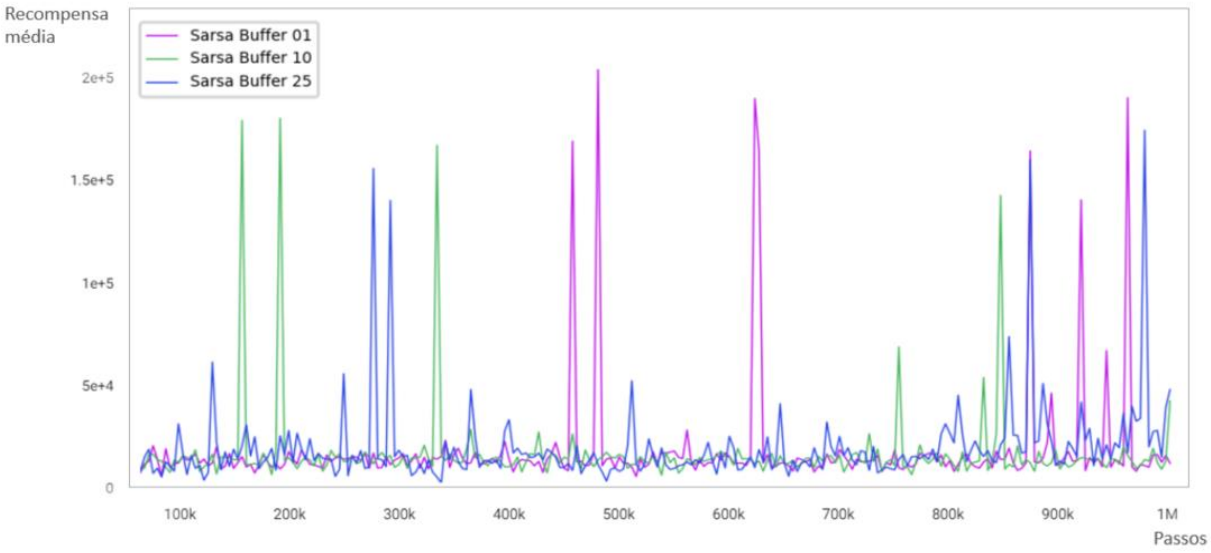

Figura 22: Recompensa média entre algoritmos SARSA, com diferentes tamanhos de buffers, aplicados ao ECL.

Ao se comparar os algoritmos TRPO treinados sem *buffer* de memória e com *buffers* de memória dos 3, 10 e 25 últimos passos anteriores, percebe-se pelo gráfico da figura 23 um desempenho superior de recompensa média dos modelos sem *buffer* e com *buffer* de memória dos 25 passos anteriores frente aos outros dois.

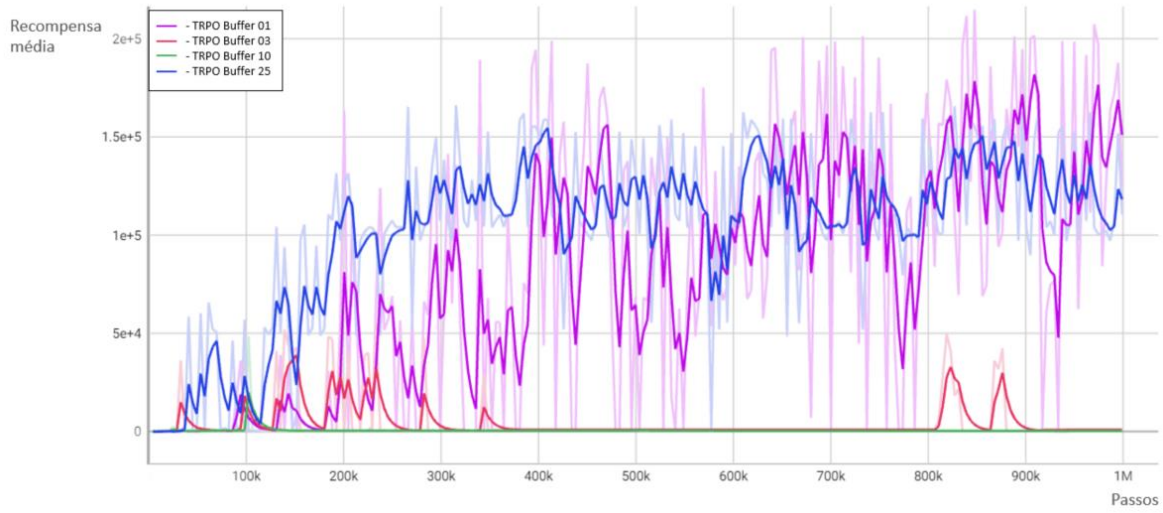

Figura 23: Recompensa média entre algoritmos TRPO, com diferentes tamanhos de buffers, aplicados ao ECL.

Além de mais estável, o algoritmo em TRPO treinado com *buffer* dos últimos 25 passos possui menor divergência KL quando comparado aos outros três modelos (ver figura 24).

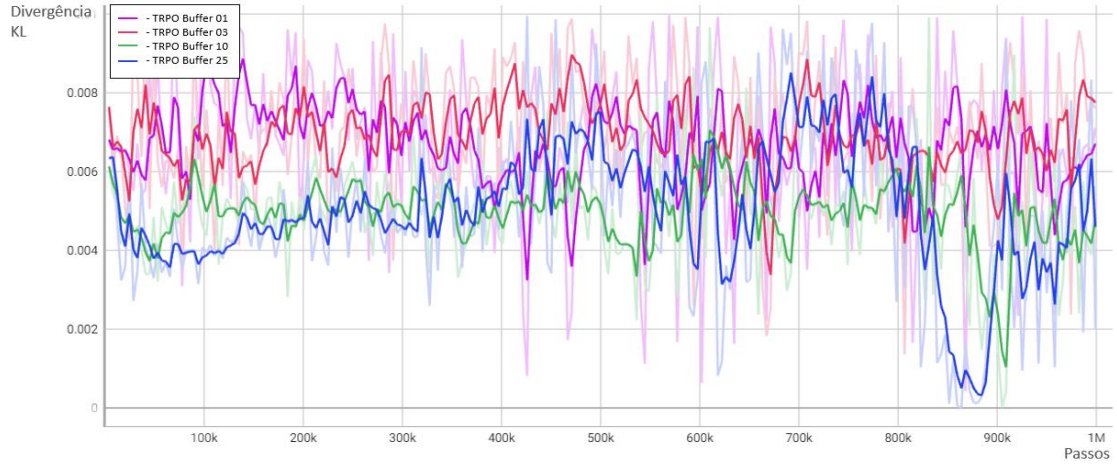

Figura 24: Divergência KL entre algoritmos TRPO, com diferentes tamanhos de buffers, aplicados ao ECL.

Isso se reflete na performance prática do agente quando atua no ECL da realidade virtual, pois o algoritmo sem *buffer* de memória "esquece" certos posicionamentos da caixa na esteira quando ela é movimentada, não finalizando o ciclo completo, enquanto o TRPO treinado com *buffer* dos últimos 25 passos conclui normalmente o ciclo do processo em ECL. Em termos práticos, o agente treinado sem *buffer* de passos anteriores deixa de retornar a caixa à posição 0 para descarga quando ela está completamente preenchida, movendo-a

adiante, como se entendesse que não houve fila preenchida, conforme figura 25 abaixo (sequência quadro a quadro descrita no Apêndice C):

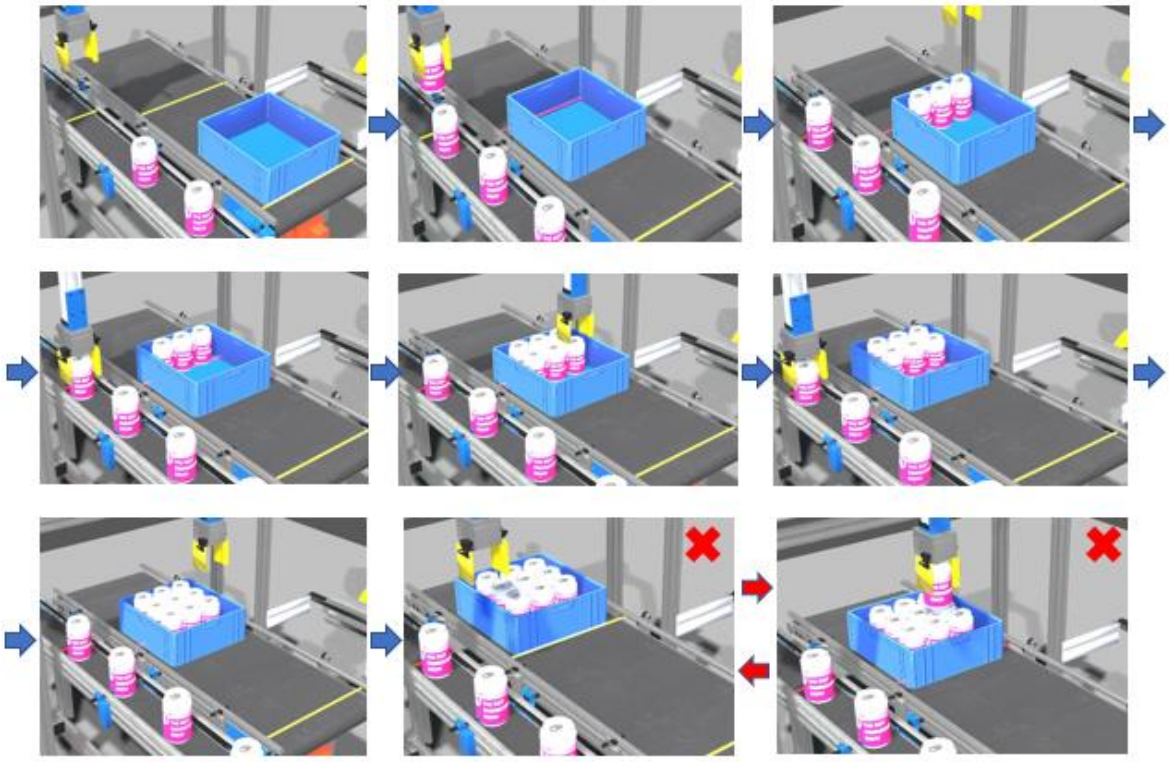

Figura 25: Sequência resumida do processo ECL, quadro a quadro, aprendida pelo algoritmo TRPO, sem *buffer* de passos anteriores.

Esperava-se que os algoritmos em TRPO treinados com *buffers* de memória dos últimos 3 e 10 passos anteriores possuíssem desempenho superior ao TRPO treinado sem *buffer* de memória, mas não foi o que ocorreu. Não houve em ambos os algoritmos um arraque significativo na aprendizagem, resultando em recompensas médias de valores nulos em boa parte da aprendizagem, indicando possivelmente que o agente ficou preso em um mínimo local da solução que não corresponde a uma boa política do processo em ECL.

Já ao se comparar os algoritmos A2C treinados com esse mesmo conjunto de *buffers*, percebe-se pelo gráfico da figura 26, um desempenho bastante superior de recompensa média do *buffer* de memória de 25 passos anteriores frente aos outros três. Além disso, a estabilidade desse algoritmo, com esse *buffer* avaliado, é alcançada após 450 mil passos interativos, com perda por entropia cruzada praticamente nula, conforme mostrada no gráfico da figura 28. Quando comparadas as recompensas médias somente nos outros 3 *buffers* de memória, o *buffer* com 10 passos anteriores possui melhor performance (ver figura 27).

É interessante observar que o algoritmo A2C possui bastante estabilidade durante o treinamento, mantendo uma política de ações bastante uniforme, resultando em pouca oscilação da recompensa média obtida durante os episódios.

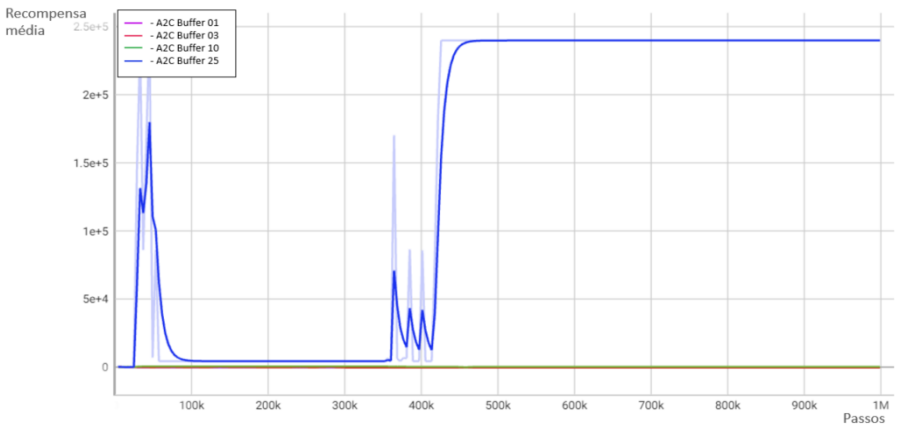

Figura 26: Recompensa média entre algoritmos A2C, com diferentes tamanhos de buffers, aplicados ao ECL.

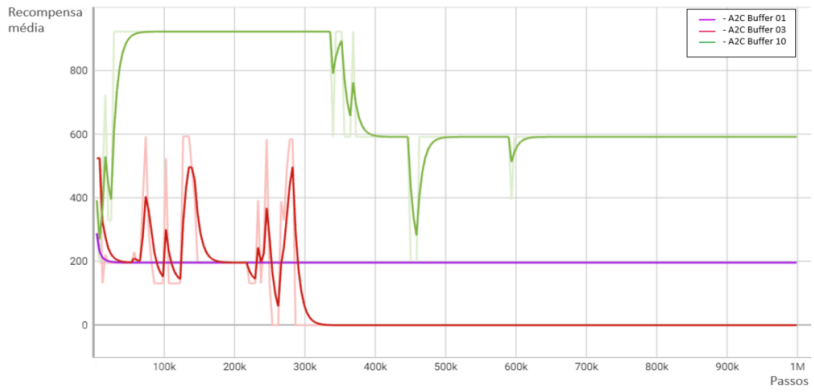

Figura 27: Recompensa média entre algoritmos A2C, somente com os três menores tamanhos de buffers, aplicados ao ECL

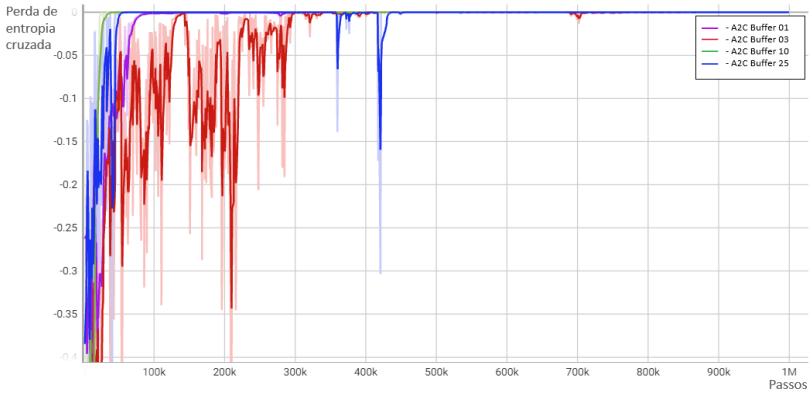

Figura 28: Perda de entropia cruzada entre algoritmos A2C, com diferentes tamanhos de buffers, aplicados ao ECL.

Essa performance do algoritmo A2C resulta em um comportamento bastante semelhante ao da sequência de passos constituída pelo *framework* da Real Virtual, na qual, a cada fileira de latas preenchida pela máquina, a esteira movimenta a caixa para a próxima posição, retornando-a para descarga quando preenchida com as 9 latas (ver figura 29). Uma importante diferença entre o algoritmo e a referência está no momento exato em que a caixa é movimentada, sendo que A2C realiza a ação após o sinal discreto de finalização enviado por GML, enquanto o *framework* movimenta a caixa após a garra estar levantada. Para mais informações, ver Apêndice C.

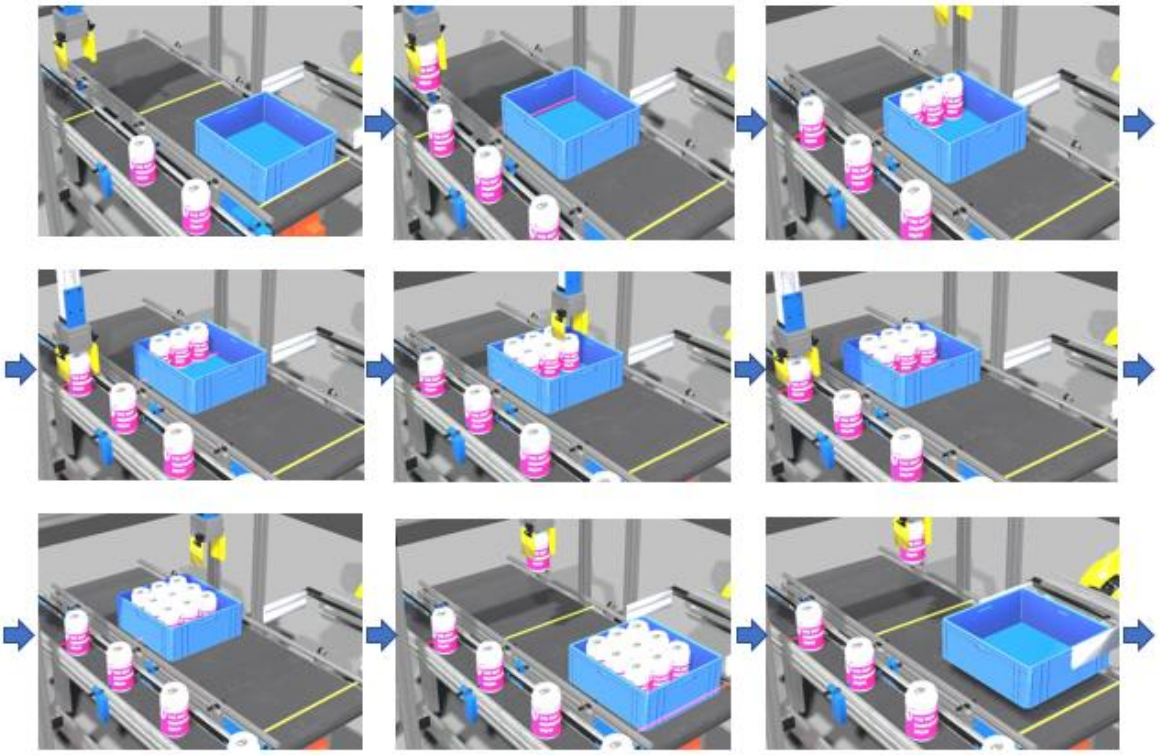

Figura 29: Sequência resumida do processo ECL, quadro a quadro, da realidade virtual pré-concebida pela Real Virtual e aprendida pelo algoritmo A2C, com *buffer* de 25 passos anteriores.

Similares aos algoritmos em TRPO, os algoritmos treinados em PPO também repetem performances melhores em *buffers* de memórias maiores, nomeadamente *buffers* dos últimos 10 e 25 passos anteriores. A diferença de performance entre esses dois algoritmos é notada na estabilidade maior dos algoritmos treinados em TRPO, percebendo no gráfico da figura 30 uma maior oscilação da recompensa média dos algoritmos treinados em PPO.

O gráfico da figura 31 mostra um número menor de vezes de extrapolação do limite da perda por *Clipped Surrogate* nos *buffers* dos últimos 10 e 25 passos anteriores, indicando uma

menor troca de política de ações nesses dois *buffers* quando comparados aos modelos sem *buffer* e com *buffer* dos 3 passos anteriores.

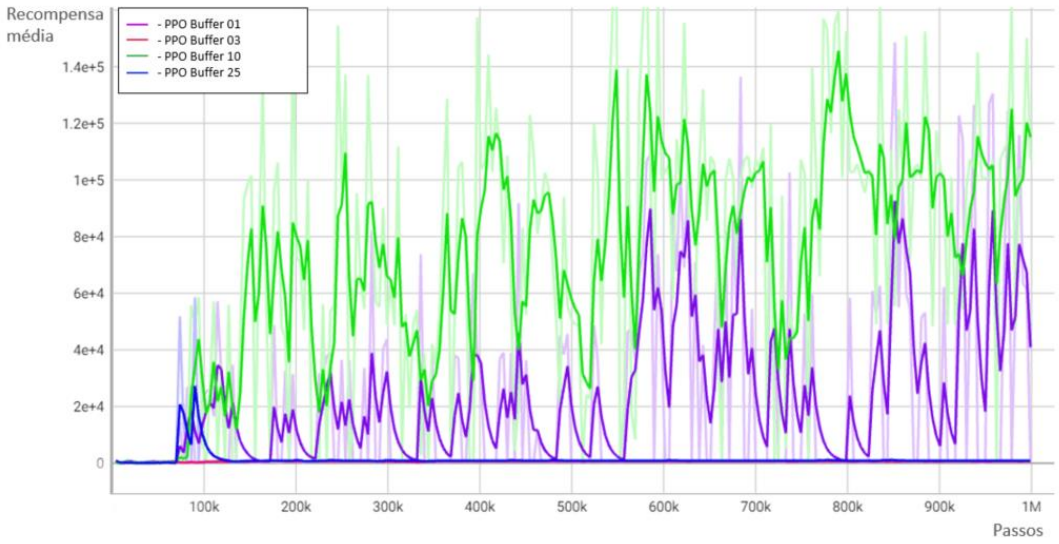

Figura 30: Recompensa média entre algoritmos PPO, com diferentes tamanhos de buffers, aplicados ao ECL.

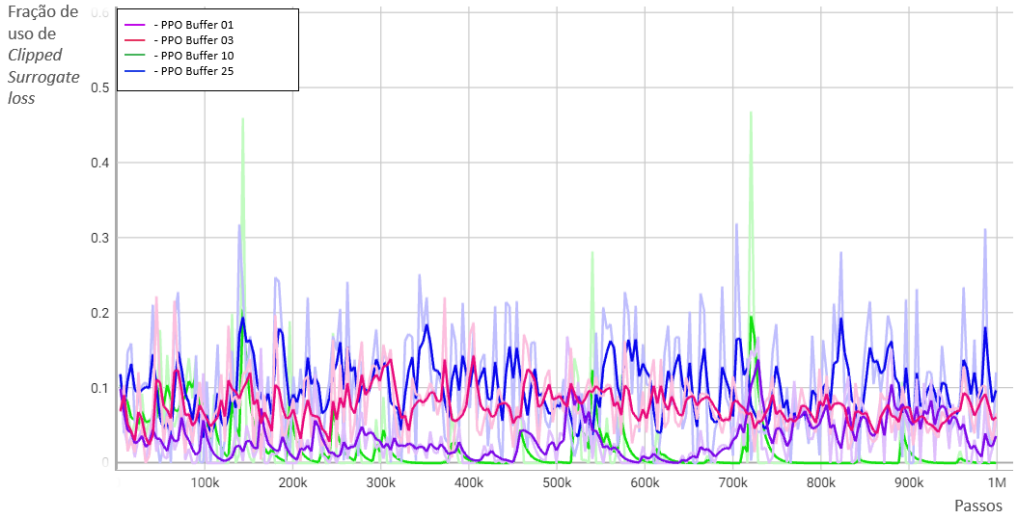

Figura 31: Fração de uso de perda por *Clipped Surrogate* entre algoritmos PPO, com diferentes tamanhos de buffers, aplicados ao ECL.

Ao aplicar as redes neuronais LSTM em algoritmos PPO na arquitetura padrão (64 X 64 neurónios), há uma queda acentuada da recompensa média (redução de mais de 100 vezes) obtida nesses algoritmos quando comparados aos algoritmos treinados em arquiteturas de redes neuronais sem implementação de LSTM (ver figura 32). A performance continua sendo maior para os *buffers* de maior memória, mas não alcançam performance superior aos algoritmos com redes neuronais comuns aplicados em PPO, TRPO e A2C.

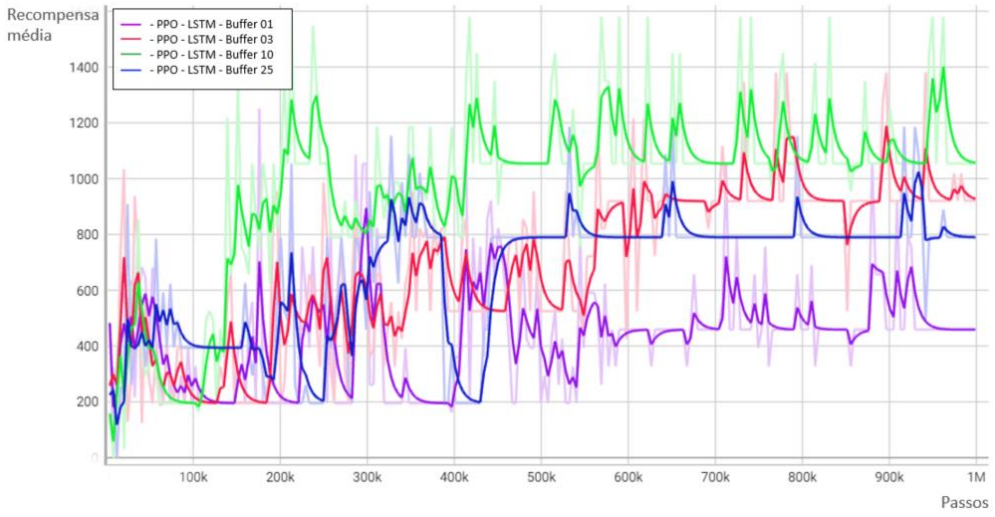

Figura 32: Recompensa média entre algoritmos PPO, rede neuronal LSTM, com diferentes tamanhos de buffers, aplicados ao ECL.

Além disso, a extrapolação do limite da perda por *Clipped Surrogate* aumenta nessa arquitetura em LSTM, indicando maior troca de política de ações e menor estabilidade e chance de conclusão do treino conforme mostrado na figura 33 abaixo:

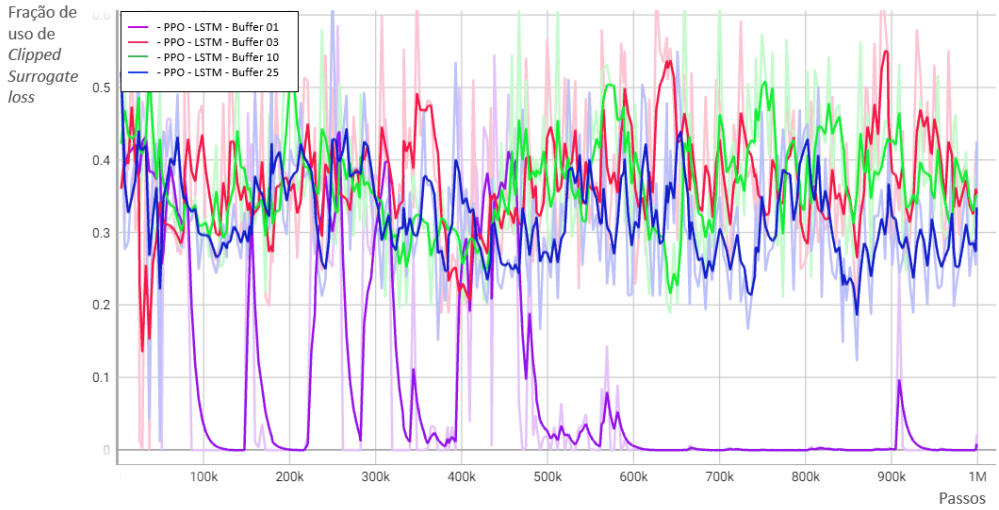

Figura 33: Fração de uso de *Clipped Surrogate Loss* entre algoritmos PPO, rede neuronal LSTM, com diferentes tamanhos de buffers, aplicados ao ECL.

Mantendo-se como referência o modelo de maior alcance de recompensas em PPO, o algoritmo com *buffer* de memória de 10 passos anteriores, e expandindo o número de neurónios na arquitetura das redes neuronais LSTM, dessa vez são alcançados resultados diferenciados, conforme pode ser visto na figura 34. Para isso, esse modelo referência foi comparado com modelos PPO sem *buffer*s, variando a arquitetura padrão de 2 camadas

ocultas de 64 neurónios LSTM em um modelo de 3 camadas de 256 neurónios LSTM e outro modelo de 3 camadas de 512 neurónios LSTM. A partir desse gráfico, são notadas melhores recompensas de modelos na medida em que a arquitetura LSTM é aumentada, entretanto, sem ainda alcançar a estabilidade mínima para aplicação do modelo em ECL.

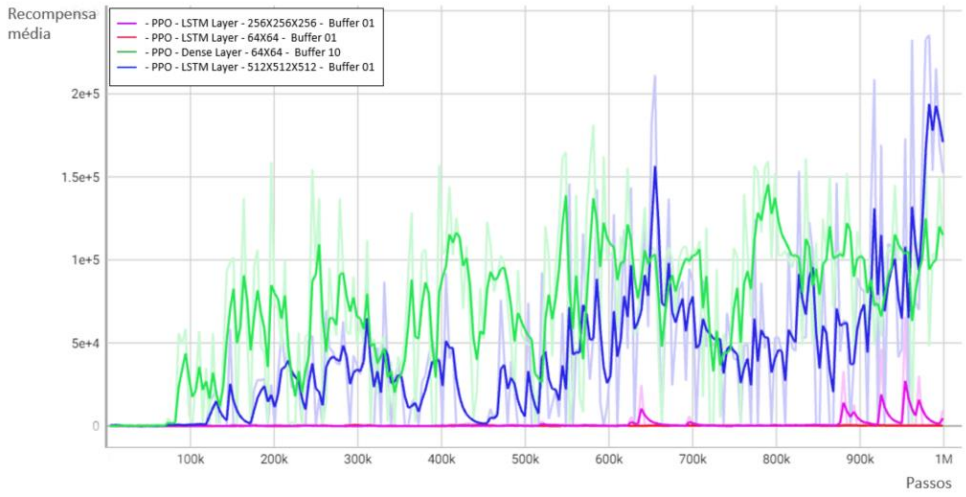

Figura 34: Recompensa média entre algoritmos PPO, com camadas LSTM, em relação ao PPO com *buffer* dos últimos 10 passos, aplicados ao ECL.

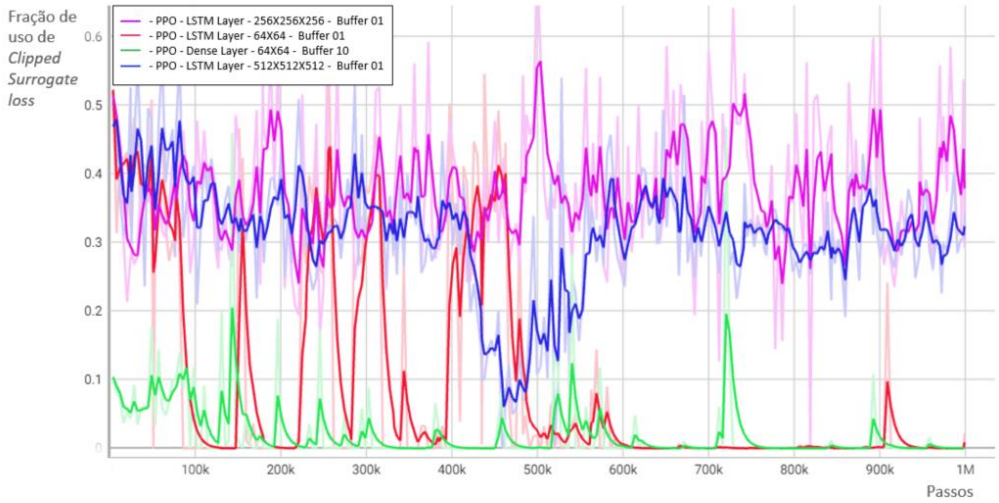

Figura 35: Fração de uso de Clipped Surrogate Loss entre algoritmos PPO, com camadas LSTM, em relação ao PPO com *buffer* dos últimos 10 passos, aplicados ao ECL.

Essa falta de estabilidade pode ser constatada no gráfico da figura 35, onde a fração de vezes em que o limite *Clipped Surrogate* é extrapolado é maior nas arquiteturas LSTM do que no modelo em referência, indicando maior troca de política de ações e uma maior necessidade de mais passos para o treino.

Todos esses resultados mostraram que o algoritmo A2C, com buffer de memória dos últimos 25 passos, apresentou o melhor desempenho, sendo capaz de concluir normalmente o ciclo

do processo em ECL. Além disso, o algoritmo TRPO, com o mesmo *buffer* avaliado, apresentou bom desempenho, podendo ser aplicado ao mesmo processo. No geral, os resultados indicam que o uso de um *buffer* de memória de passos anteriores é importante para melhorar a estabilidade e constância das recompensas obtidas pelos agentes de RL.

5.2.1. Aprendizagem por Reforço e definição das recompensas do GML

Devido ao processo do GML se tratar de um problema de controlo mais complexo que o ECL, foi avaliado o mesmo conjunto de algoritmos utilizados no processo anterior, dispensados os algoritmos SARSA, CEM e a implementação de LSTM nos algoritmos em DQN, que não obtiveram bom desempenho em ECL. O tempo de treinamento foi expandido para 2 milhões de passos interativos, comparando o mesmo conjunto de tamanho de *buffers*: sem buffer e com buffer dos 3, 10 e 25 passos anteriores. Nessa forma explícita de memorização, a rede neuronal passou a enxergar respetivamente 6, 18, 60 e 150 variáveis de entrada, cabendo a ela aprender quais delas são referentes a cada passo realizado. Em PPO, foram experimentadas diferentes arquiteturas de redes neuronais LSTM aplicadas as redes de valor Q e redes de política para comparação com o *buffer* de memorização. A tabela 4 abaixo resume esses algoritmos avaliados em ECL, quanto às características de bibliotecas utilizadas, *buffer* de memorização, variáveis de entrada e arquitetura de rede neuronal:

Algoritmo	**Biblioteca**	***Buffer* de Memorização**	**Número de Variáveis de Entrada**	**Arquitetura de Rede Neuronal (tipo de camada, quantidade de camadas e de neurónios)**
DQN	Keras	Sem *buffer*	6	*Dense Laye* r - 64 X 64
DQN	Keras	*Buffer* de 10 passos	60	*Dense Laye* r - 64 X 64
DQN	Keras	*Buffer* de 25 passos	150	*Dense Laye* r - 64 X 64
A2C	Stable Baselines	Sem *buffer*	6	*Dense Laye* r - 64 X 64
A2C	Stable Baselines	*Buffer* de 3 passos	18	*Dense Laye* r - 64 X 64
A2C	Stable Baselines	*Buffer* de 10 passos	60	*Dense Laye* r - 64 X 64
A2C	Stable Baselines	*Buffer* de 25 passos	150	*Dense Laye* r - 64 X 64
TRPO	Stable Baselines	Sem *buffer*	6	*Dense Laye* r - 64 X 64
TRPO	Stable Baselines	*Buffer* de 3 passos	18	*Dense Laye* r - 64 X 64
TRPO	Stable Baselines	*Buffer* de 10 passos	60	*Dense Laye* r - 64 X 64
TRPO	Stable Baselines	*Buffer* de 25 passos	150	*Dense Laye* r - 64 X 64
PPO	Stable Baselines	Sem *buffer*	6	*Dense Laye* r - 64 X 64
PPO	Stable Baselines	Sem *buffer*	6	*LSTM Layer* - 64 X 64
PPO	Stable Baselines	Sem *buffer*	6	*LSTM Layer* - 256 X 256 X 256
PPO	Stable Baselines	Sem *buffer*	6	*LSTM Layer* - 512 X 512 X 512
PPO	Stable Baselines	*Buffer* de 3 passos	18	*Dense Laye* r - 64 X 64
PPO	Stable Baselines	*Buffer* de 10 passos	60	*Dense Laye* r - 64 X 64
PPO	Stable Baselines	*Buffer* de 25 passos	150	*Dense Laye* r - 64 X 64

Tabela 4: Algoritmos avaliados em GML quanto às características de bibliotecas utilizadas, buffer de memorização, variáveis de entrada e arquitetura de rede neuronal.

Calcula-se que são necessários entorno de 20 passos para conclusão de cada ciclo de preenchimento de cada fila, a depender de quão rápido é a execução dos outros processos pelos outros agentes, nomeadamente a disponibilidade de lata para coleta e a disponibilidade da caixa na posição correta para posicionamento da lata.

Tal como em todo problema em RL, estabelecer as corretas recompensas e punições é determinante para o sucesso do treinamento. Dessa forma, as regras foram determinadas a seguir de modo a evitar que o agente fique preso em mínimos locais e realize todos os passos corretos para conclusão do ciclo do processo do GML, incentivando os seguintes comportamentos:

- Caso a garra esteja nas posições 3, 5 ou 7, com uma lata carregada e abra a garra, a lata é despejada na caixa. Essa ação não é a ideal, pois a lata cairá na caixa de uma certa altura insegura, mas é uma ação válida num processo de treino, assim:

$$\mathrm{R} = (400 * \text{nº de latas já na fileira da caixa}) - \frac{n^{\underline{o}}\ passos}{2} \qquad (5.9)$$

- Caso a garra esteja nas posições 4, 6 ou 8, com uma lata carregada e abra a garra, a lata é posicionada na caixa. Ação correta de descarga na caixa com recompensa calculada:

$$\mathrm{R} = (4000 * \text{nº de latas já na fileira da caixa}) - \frac{n^{\underline{o}}\ passos}{2} \qquad (5.10)$$

- Para evitar colisões, caso haja posicionamento ou despejo de lata sobre outra lata na caixa, na mesma posição já preenchida,

$$\mathrm{R} = -100 - (5 * \text{nº de passos}) \qquad (5.11)$$

- Caso a fileira seja preenchida corretamente (sem sobreposições de latas) e a esteira sinalize o preenchimento ao agente da esteira de caixa:

$$\mathrm{R} = 2000 - \text{nº de passos} \qquad (5.12)$$

- Para evitar colisões, caso a fileira seja preenchida incorretamente (sobreposições de latas) e a esteira sinalize o preenchimento ao agente da esteira de caixa:

$$\mathrm{R} = -100 - \text{nº de passos} \qquad (5.13)$$

- Para evitar colisões da garra ou lata com outros objetos, caso a garra se desloque da posição 2 diretamente para posições 4, 6 ou 8,

$$R = - 1000 \quad (5.14)$$

- Com mesmo objetivo de evitar colisões, caso a esteira de latas não esteja parada (sensor de fim de curso da esteira não acionado) e a garra se movimente em direção a posição 2,

$$R = - 500 \quad (5.15)$$

- Para incentivar que a garra esteja pronta para coleta na posição 2, caso a garra esteja fechada, nessa posição e sem lata carregada (sensor de presença de lata na garra não acionado),

$$R = - 100 \quad (5.16)$$

- Para evitar colocação de latas com a esteira em movimento, caso a garra esteja carregando uma lata, a esteira da caixa esteja em movimento e a garra se abre para despeja-la, consistindo aí um ato inseguro,

$$R = - 400 \quad (5.17)$$

- Para recompensar o sucesso do agente no encerramento do ciclo, após todas as fileiras serem preenchidas na caixa e após a garra aguardar a descarga dessa caixa, um sinal de *done* é enviado ao agente, o episódio é encerrado e a recompensa abaixo é enviada ao agente:

$$R = 6000 – (60 * nº \text{ de passos}) \quad (5.18)$$

5.2.2. Resultados dos modelos de aprendizagem por reforço aplicados ao GML

Neste capítulo são mostrados os resultados dos algoritmos treinados em GML, comparando a performance de cada deles um por meio de diferentes gráficos de recompensa média, divergência KL, perda de entropia cruzada e fração de uso de limite *Clipped Surrogate Loss* (todos no eixo cartesiano Y) em relação ao número de passos treinados (todos no eixo cartesiano X). Assim como no capítulo anterior, foi utilizado o TensorBoard para geração desses gráficos, mantendo o mesmo padrão de apresentação dos resultados.

Primeiramente, ao se comparar os algoritmos treinados em DQN com diferentes tamanhos de *buffers* de memória, não se obtém resultados satisfatório em nenhum deles aplicados ao GML (ver figura 36), pois todas as recompensas médias dos episódios desses modelos se mantiveram em patamares negativos.

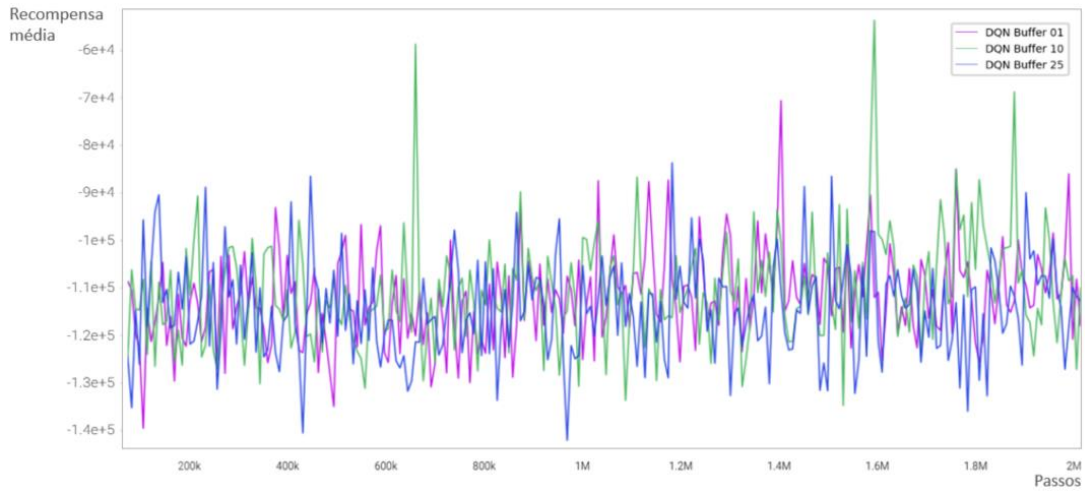

Figura 36: Recompensa média entre algoritmos DQN, com diferentes tamanhos de buffers, aplicados ao GML

Já ao se comparar os algoritmos TRPO treinados sem *buffer* de memória e com *buffers* de memória dos 3, 10 e 25 últimos passos anteriores, percebe-se pelo gráfico da figura 37, o alcance do maior valor disponível de recompensa média para o GML entorno de 70 mil pontos, formando um *plateau*. Cada modelo alcançou esse valor em diferentes etapas do treinamento de acordo com o tamanho do *buffer*, ou seja, quanto menor esse tamanho de memória, mais rápido foi o arranque.

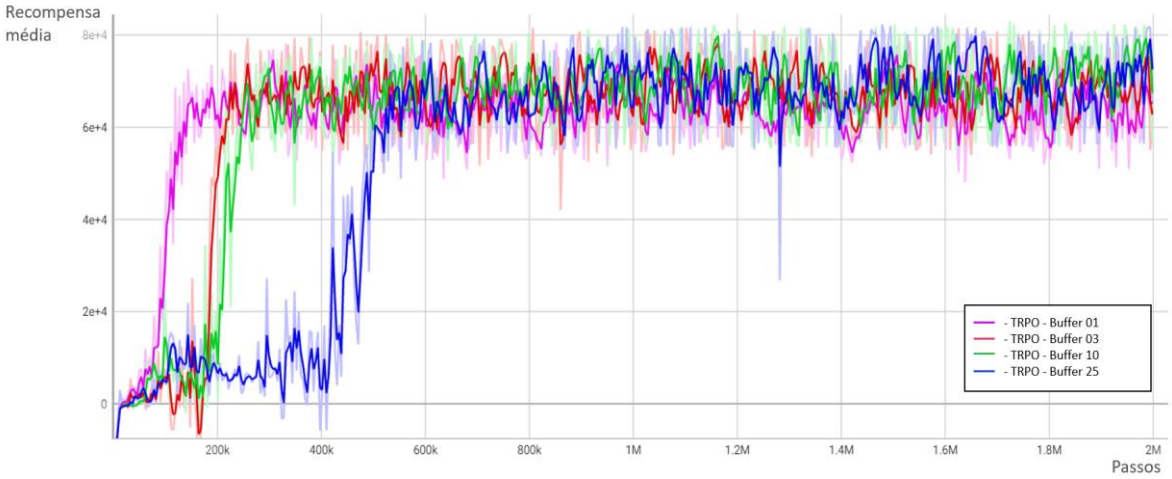

Figura 37: Recompensa média entre algoritmos TRPO, com diferentes tamanhos de buffers, aplicados ao GML

Relativamente à estabilidade, quanto maior o tamanho do *buffer*, menor a divergência KL (ver figura 38) em GML, refletindo tal métrica na performance do agente na memorização das latas posicionadas na caixa da esteira, como pode-se ver na figura 39. Nessa sequência, o agente procura reduzir o tempo de movimentação, realizando trajectos mais curtos, mostrados pelas setas verdes. Mais informações estão disponíveis no Apêndice C.

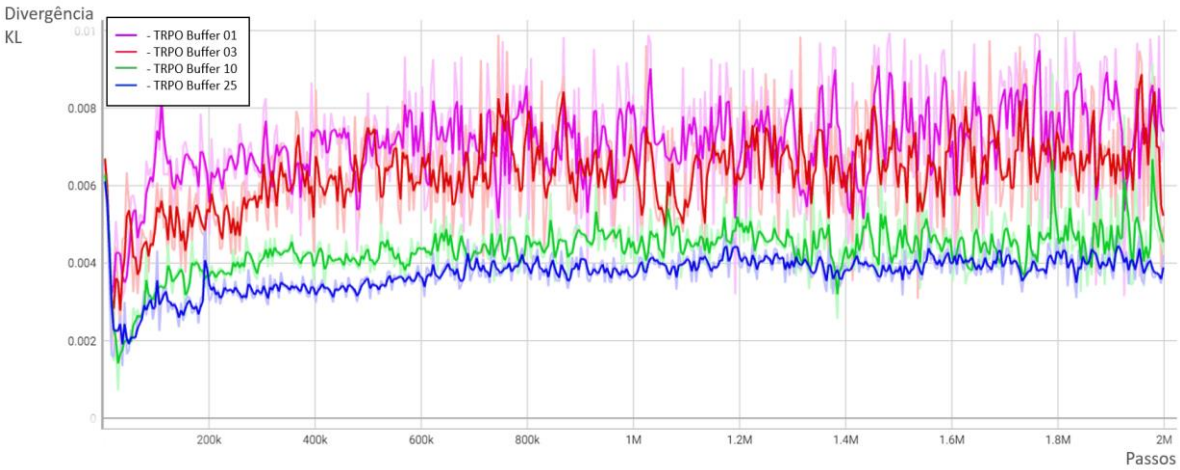

Figura 38: Divergência KL entre algoritmos TRPO, com diferentes tamanhos de buffers, aplicados ao GML.

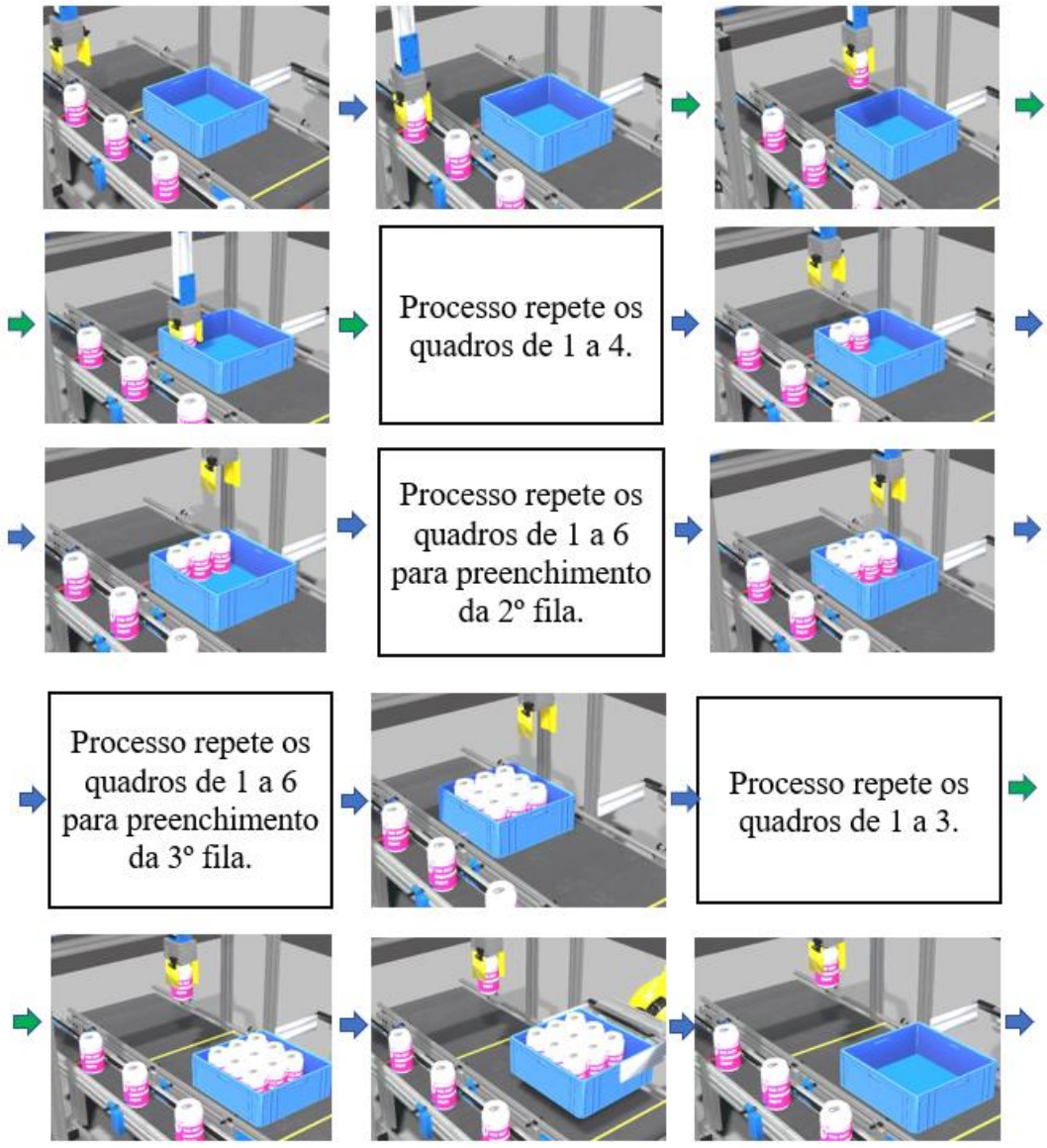

Figura 39: Sequência resumida do processo GML, quadro a quadro, aprendida pelo algoritmo TRPO, com *buffer* de 10 passos anteriores.

Já o agente treinado nesse algoritmo TRPO, sem *buffer* de memória, insiste na colocação de latas em posições já preenchidas anteriormente, comprometendo o processo GML, conforme pode-se ver na figura 40 abaixo:

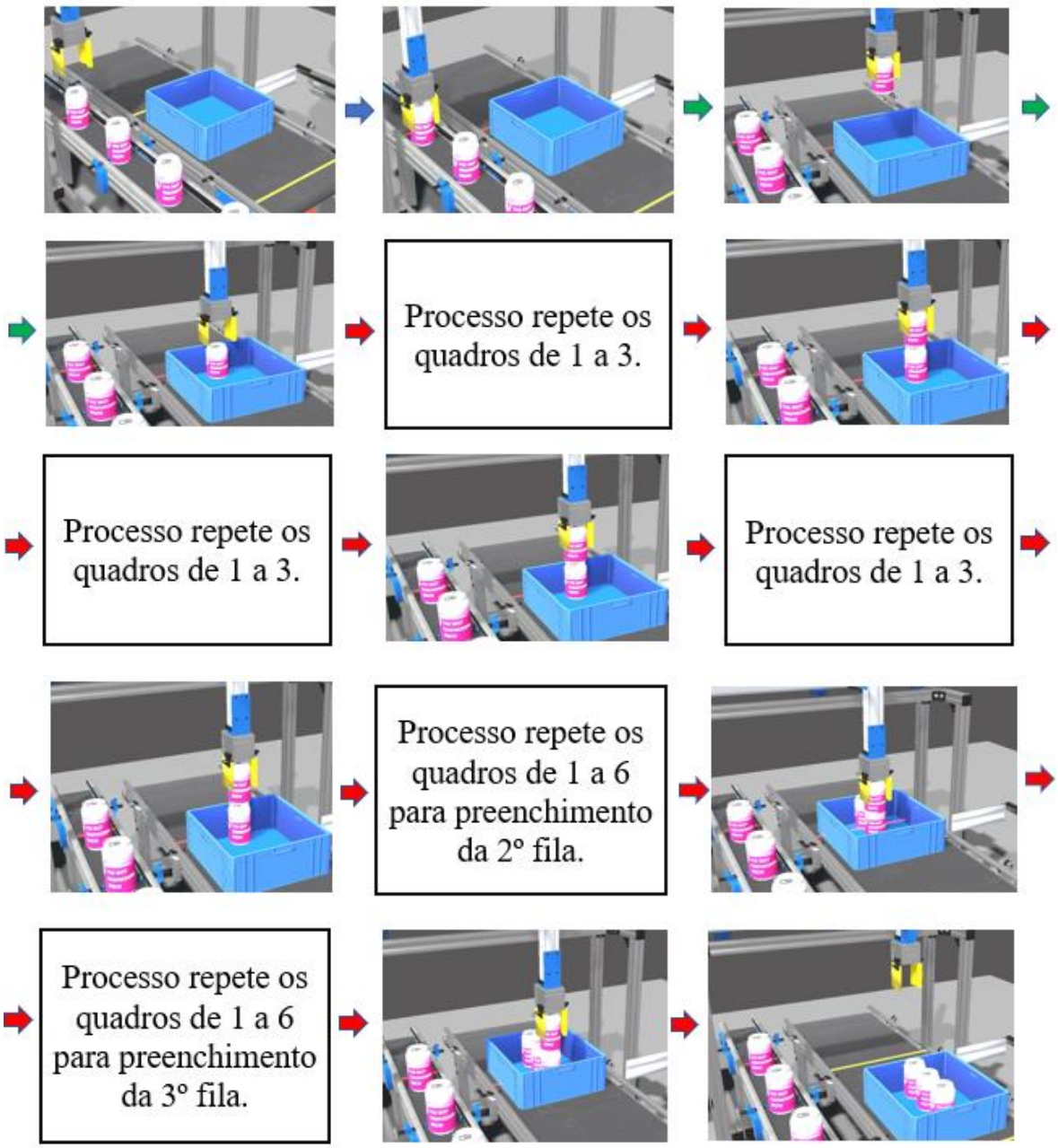

Figura 40: Sequência resumida do processo GML, quadro a quadro, aprendida pelo algoritmo TRPO, sem *buffer* de passos anteriores.

Relativamente aos algoritmos A2C aplicados ao GML, nenhum deles treinados com diferentes tamanhos de *buffers* de memória obteu resultaldo relevante. Esses modelos tiveram um início da aprendizagem com recompensas negativas, rapidamente alcançaram

recompensas nulas e nesse patamar se estabilizaram formando um *pleateau* de aprendizagem insatisfatória, conforme mostra a figura 41 abaixo:

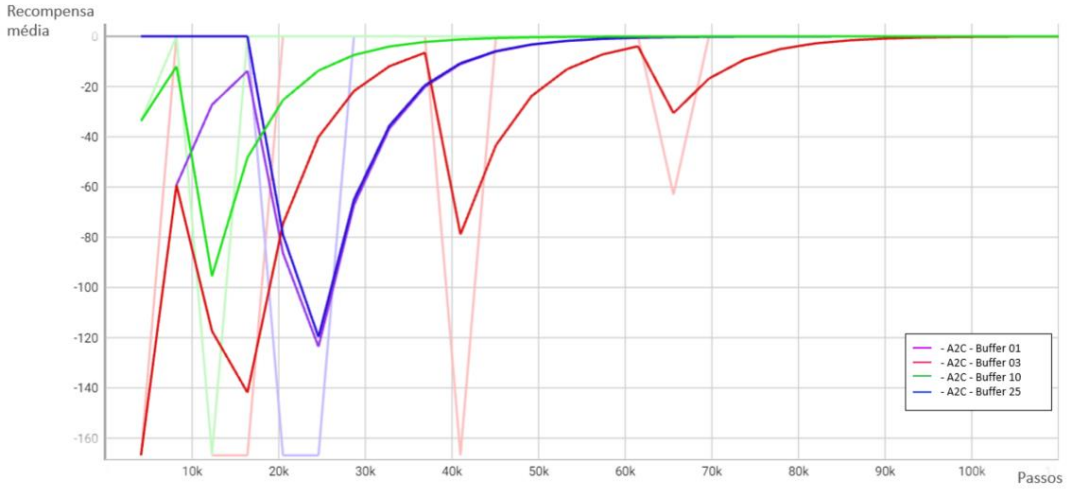

Figura 41: Recompensa média entre algoritmos A2C, durante os primeiros 100 mil passos de treino, aplicados ao GML.

Diferentemente do ECL, nos algoritmos treinados em PPO aplicados ao GML, somente o que foi treinado com o *buffer* dos últimos 10 passos anteriores alcança performances comparáveis àqueles modelos treinados em TRPO (ver figura 42). Ainda assim, esse modelo não é eficiente quando aplicado à realidade virtual, cometendo erros de colisões e despejo de lata em posições erradas. Embora consiga memorizar parcialmente as posições preenchidas nas fileiras da caixa, a aplicação desse modelo em PPO é insatisfatória para o fechamento do ciclo completo em GML.

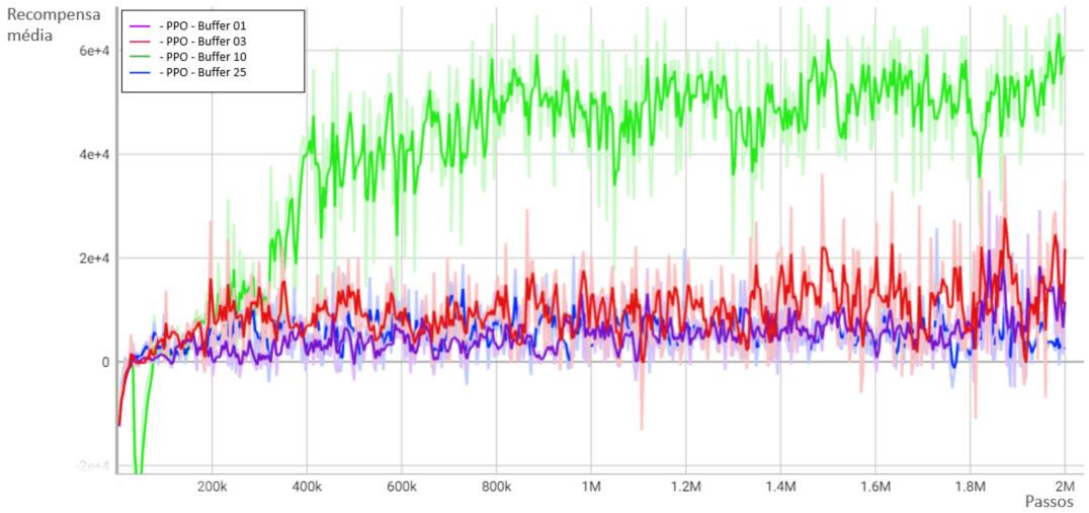

Figura 42: Recompensa média entre algoritmos PPO, com diferentes tamanhos de buffers, aplicados ao GML

Relativamente ao número menor de vezes em que esses algoritmos treinados em PPO extrapola o limite de perda por *Clipped Surrogate*, o gráfico da figura 43 mostra uma redução rápida nessa fração de uso logo no início do treinamento, indicanco uma rápida convergência na política de ações.

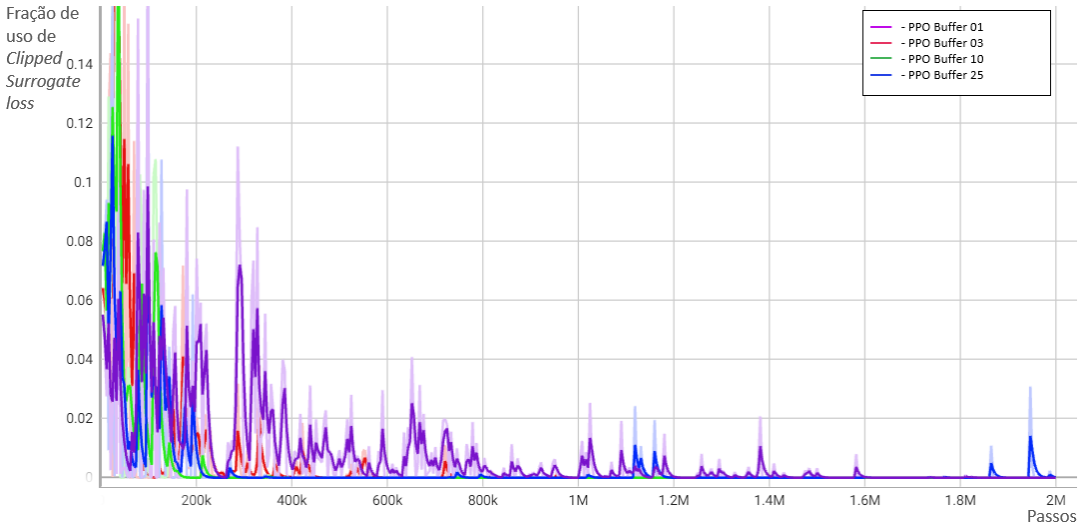

Figura 43: Fração de uso de perda por *Clipped Surrogate* entre algoritmos PPO, com diferentes tamanhos de buffers, aplicados ao GML.

Ao se aplicar as redes neuronais LSTM em algoritmos PPO, com camadas de neurónios variando entre 64, 256, 512 e 1028 neurónios, todos os modelos resultaram em recompensas média nula após 2 milhões de passos interativos. A figura 44 abaixo mostra esse comportamento insatisfatório similar em todos esses modelos treinados com LSTM durante os primeiros 100 mil passos, indicando um arranque a partir de recompensas negativas para a estabilidade do valor nulo.

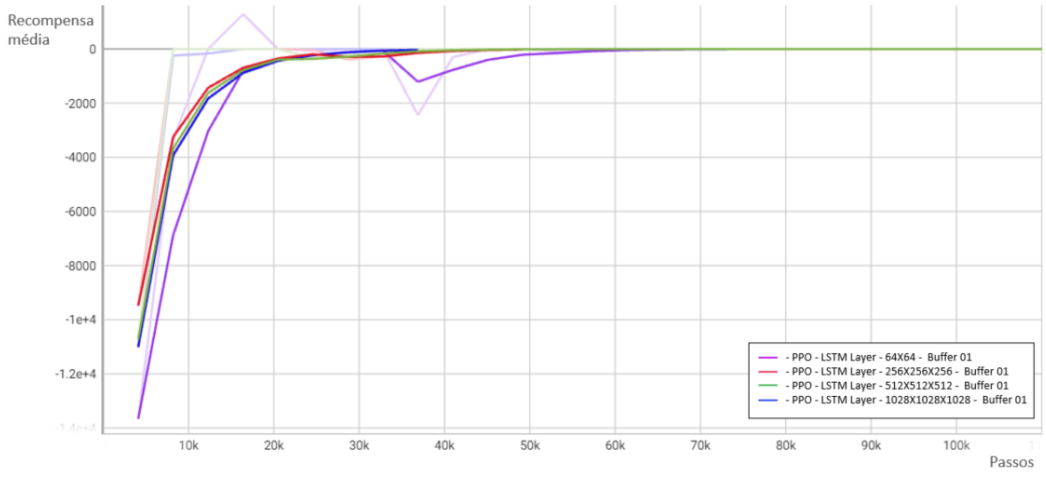

Figura 44: Recompensa média entre algoritmos PPO, com camadas LSTM, durante os primeiros 100 mil passos de treino, aplicados ao GML.

Em resumo, assim como em ECL, os algoritmos DQN não foram eficazes em GML com nenhum tamanho de *buffer* de memória, enquanto os algoritmos TRPO apresentaram os melhores resultados, sendo que quanto maior o *buffer*, menor a divergência KL e melhor a estabilidade. Os algoritmos A2C não foram relevantes e atingiram um plateau de aprendizagem insatisfatório. Já os algoritmos PPO tiveram resultados satisfatórios apenas com *buffers* de memória dos últimos 10 passos, mas com erros de colisão e despejo de lata em posições erradas. Dessa forma, a escolha do algoritmo e do tamanho do *buffer* de memória tem impacto significativo na performance do agente em GML.

5.3. Discussão dos resultados e comparações

Para avaliar a performance e o desempenho de cada modelo treinado em ECL e em GML, é importante obter dados estatíticos que auxiliem na melhor escolha para cada problema em questão. Nesse sentido, foram criadas as tabelas 5 e 6, mostradas abaixo, que apresentam os diferentes dados de recompensa média de cada modelo aplicado a esses dois processos, nomeadamente os somatórios de recompensa, os valores minímos, os máximos, as médias e os desvios padrões. Além disso, ambas as tabelas mostram se houve conclusão do processo ou não, com destaque para a tabela 6, que apresenta o percentual de preenchimento de latas efetuado por GML na caixa da esteira. As cores em verde, azul e vermelho em algumas das células dessas tabelas destacam grandes valores nos dados estatistícos em questão, de forma a apontar diferenciais de perfomance em cada modelo.

Relativamente ao ECL, destacam-se na tabela 5 os modelos A2C e TRPO, com tamanhos de *buffer* dos últimos 25 passos anteriores, por possuírem as maiores médias de recompensas durante os episódios em relação a todos os outros modelos. Percebe-se também que A2C possui baixo desvio padrão em relação à recompensa média da aprendizagem, indicando boa estabilidade e assentamento em um alto valor de recompensa, conforme mostrado na figura 26. Tal estabilidade se reflete no ótimo comportamento do agente na realidade virtual, que realiza todos os passos corretamente, sem quebra do ciclo do ECL, conforme mostrado no Apêndice C. A similaridade com a GAN, na arquitetura da A2C, pode ser a grande reponsável por esse ganho de recompensa, onde a rede neuronal de valor Q *critic* realiza uma correção do erro TD na rede neuronal de política *actor*.

Nome	Mínimo	Máximo	Média	Desvio padrão	Conclusão do processo
DQN *buffer* 01 - Dense Layer - 64 X 64	3413	202906	18701	21473	Não
DQN *buffer* 10 - Dense Layer - 64 X 64	7222	193648	22444	31288	Não
DQN *buffer* 25 - Dense Layer - 64 X 64	8784	232109	33398	41989	Não
DQN *buffer* 25 - Dense Layer - 128 X 128 X 128	7199	175890	16316	14694	Não
DQN *buffer* 25 - LSTM Layer - 128 X 128 X 128	0	20693	8388	5749	Não
DQN *buffer* 25 - LSTM Layer - 128 X 96 X128 X 96	282	186757	12134	16824	Não
CEM *buffer* 01 - Dense Layer - 64 X 64	7660	181288	24876	20746	Não
CEM *buffer* 10 - Dense Layer - 64 X 64	9086	196587	21620	24359	Não
CEM *buffer* 25 - Dense Layer - 64 X 64	11038	224402	21672	17083	Não
Sarsa *buffer* 01 - Dense Layer - 64 X 64	10841	211140	23551	27895	Não
Sarsa *buffer* 10 - Dense Layer - 64 X 64	11535	187210	21951	20650	Não
Sarsa *buffer* 25 - Dense Layer - 64 X 64	7660	181288	24876	20746	Não
TRPO *buffer* 01 - Dense Layer - 64 X 64	0	214910	80877	70193	Não
TRPO *buffer* 03 - Dense Layer - 64 X 64	0	54881	4802	11998	Não
TRPO *buffer* 10 - Dense Layer - 64 X 64	97	48762	834	3086	Não
TRPO *buffer* 25 - Dense Layer - 64 X 64	97	166196	102600	45178	Sim
A2C *buffer* 01 - Dense Layer - 64 X 64	195	289	197	6	Não
A2C *buffer* 03 - Dense Layer - 64 X 64	0	594	72	142	Não
A2C *buffer* 10 - Dense Layer - 64 X 64	196	923	690	173	Não
A2C *buffer* 25 - Dense Layer - 64 X 64	197	246902	146726	11334	Sim
PPO *buffer* 01 - Dense Layer - 64 X 64	195	148592	24230	35380	Não
PPO *buffer* 01 - LSTM Layer - 64 X 64	0	1252	436	212	Não
PPO *buffer* 01 - LSTM Layer - 256 X 256 X 256	31	65976	1461	6125	Não
PPO *buffer* 01 - LSTM Layer - 512 X 512 X 512	196	235370	43370	52112	Não
PPO *buffer* 03 - Dense Layer - 64 X 64	95	1284	804	148	Não
PPO *buffer* 03 - LSTM Layer - 64 X 65	126	1381	698	297	Não
PPO *buffer* 10 - Dense Layer - 64 X 64	30	181410	72221	49063	Não
PPO *buffer* 10 - LSTM Layer - 64 X 65	0	1581	923	345	Não
PPO *buffer* 25 - Dense Layer - 64 X 64	98	58596	1489	4978	Não
PPO *buffer* 25 - LSTM Layer - 64 X 65	0	1186	666	242	Não

Tabela 5: Dados estatísticos de recompensas médias de cada modelo aplicado ao ECL.

Nesse comparativo de dados aplicados ao ECL, também é possível afirmar que o modelo treinado em PPO sem a utilização de *buffer* de memória, mas com a arquitetura LSTM, memorizou estados observados de passos anteriores por meio da rede neuronal treinada. Observa-se no processo de aprendizagem em ECL que, quanto maior o número de neurónios constituídos na arquitetura desse modelo em PPO, maior a capacidade do modelo de guardar e de interpretar situações ocorridas no passado para tomada de decisões no presente, constituindo uma grande habilidade de memorização implícita na rede neuronal. Entretanto, esses modelos compostos em LSTM requerem mais passos de treinamento e não acompanham a performance daqueles algoritmos treinados com maior tamanho de *buffer* de memória, ou seja, a disponibilização explícita, na entrada dos modelos, das informações dos estados de passos anteriores tem impacto direto e melhor na performance de aprendizagem.

Por outro lado, tanto os modelos treinados por meio do algoritmo A2C, como pela arquitetura de redes neuronais LSTM, obtiveram maus resultados em GML, como mostrado

na tabela 6. Nesse processo, os melhores valores de recompensa média ocorreram nos modelos treinados em TRPO, especialmente aqueles com *buffer* de memória dos últimos 3 e 10 passos anteriores. Nesses dois modelos específicos, o desvio padrão mediano em conjunto com altos valores de recompensa média leva o agente a ter melhores performances em GML, agregadas a estabilidade nas ações.

Os algoritmos treinados em PPO tiveram resultados razoáveis quando aplicados ao GML, com destaque para aquele treinado com *buffer* de memória dos últimos 10 passos anteriores, que conseguiu preencher 33% (3 latas) da caixa durante os ciclos. Nesses modelos, possivelmente uma maior quantidade de passos de treinamento levaria o agente ao completo entendimento do ciclo do processo e possível viabilidade de sua aplicação.

Nome	Mínimo	Máximo	Média	Desvio padrão	Conclusão do processo e porcentagem de preenchimento
DQN *buffer* 01 - Dense Layer - 64 X 64	-137152	-70639	-108994	9103	Não
DQN *buffer* 10 - Dense Layer - 64 X 64	-132550	-54325	-108946	11157	Não
DQN *buffer* 25 - Dense Layer - 64 X 64	-139624	-83250	-110709	9625	Não
TRPO *buffer* 01 - Dense Layer - 64 X 64	-10767	79016	61459	14491	Parcialmente - 33%
TRPO *buffer* 03 - Dense Layer - 64 X 64	-16181	81642	61631	20465	Parcialmente - 67%
TRPO *buffer* 10 - Dense Layer - 64 X 64	-11210	82889	61975	20969	Sim
TRPO *buffer* 25 - Dense Layer - 64 X 64	-11094	82380	54253	26881	Parcialmente - 44%
A2C *buffer* 01 - Dense Layer - 64 X 64	-167	0	-25	61	Não
A2C *buffer* 03 - Dense Layer - 64 X 64	-396	59	-3	30	Não
A2C *buffer* 10 - Dense Layer - 64 X 64	-167	0	-2	17	Não
A2C *buffer* 25 - Dense Layer - 64 X 64	-167	0	-2	18	Não
PPO *buffer* 01 - Dense Layer - 64 X 64	-12354	33040	5069	5142	Não
PPO *buffer* 01 - LSTM Layer - 64 X 64	-13647	1289	-232	1594	Não
PPO *buffer* 01 - LSTM Layer - 256 X 256 X 256	-9425	0	-2356	4713	Não
PPO *buffer* 01 - LSTM Layer - 512 X 512 X 512	-10713	0	-49	717	Não
PPO *buffer* 01 - LSTM Layer - 1028 X 1028 X 1028	-8035	0	-15	503	Não
PPO *buffer* 03 - Dense Layer - 64 X 64	-13103	39842	9741	7705	Não
PPO *buffer* 10 - Dense Layer - 64 X 64	-38703	71674	40331	19410	Parcialmente - 33%
PPO *buffer* 25 - Dense Layer - 64 X 64	-11843	25761	5571	4745	Não

Tabela 6: Dados estatísticos de recompensas médias de cada modelo aplicado ao GML.

Por fim, de forma geral, os algoritmos *on-policy* A2C, TRPO e PPO tiveram melhores performances que os algoritmos *off-policy* DQN e CEM, possivelmente porque a política de decisão de ações é clara em ECL e GML, o que reduz a variância nos dados coletados. Em ambientes mais complexos, em que a política de decisões de ações precisam ser melhoradas com o passar do treinamento, há maior vantagem para algoritmos *off-policy*, pois essa classe de algoritmos reutiliza dados coletados em políticas passadas para tomadas de decisões, o que não se faz presente para os processos ECL e GML em questão.

Além disso, em ambos os processos, as expansões ou contrações das camadas ocultas de neurónios, seja em arquiteturas comuns, seja em arquiteturas LSTM não parecem ter impacto positivo na performance dos modelos, excetuando o caso específico do modelo em PPO treinado com 3 camadas de 512 neurónios em LSTM aplicado ao ECL, que se destacou nos resultados de recompensa. Por isso, preferir treinar modelos com complexas arquiteturas de redes neuronais é um processo pior de aprendizagem, que requer maior esforço computacional e não garante retornos concretos, do que quando se treina o modelo com *buffers* de memória, ou seja, a entrega de um vetor de estado S que retenha todas as informações relevantes ao processo, sendo elas o estado corrente e os estados anteriores.

6. Conclusões e Trabalhos Futuros

A escolha pelo estudo de Aprendizagem por Reforço aplicado a processos industriais discretos sequenciais foi motivado pela busca de métodos que diminuíssem a necessidade de intervenção técnica na implementação ou na reconfiguração de sistemas, de maneira que o processo fosse encarado como uma caixa negra. Nesse sentido, a Aprendizagem por Reforço cumpre esse papel, aprendendo a realizar ajustes efetivos no controlo de sistemas por meio de treinamento interativo entre entradas e saídas advindos da caixa negra do processo em questão. Isso elimina a necessidade de conhecimento prévio das variáveis do espaço de estado da processo industrial, dando flexibilidade de adaptação quando o algoritmo for retreinado continuamente. Essas características adaptativas e de retreino abrem campos de estudos e de trabalhos futuros de algoritmos que gerem automaticamente lógicas de automação para PLC, eliminando a necessidade humana de desenvolvimento, resultando em ganho de tempo de implementação e produtivo na indústria.

A utilização de Gémeos Digitais/Simuladores concebidos em realidades virtuais ou meramente em ambientes programados, que repliquem as regras de ação e reação do processo industrial, são necessários no treinamento e nas simulações de cenários, devido a sua rapidez e flexibilidade de mudanças nessas regras, caso necessário. Se o treinamento fosse realizado diretamente em ambientes reais, a longa etapa de passos alinhado ao risco de ações indesejadas comprometeria a viabilidade de realização deste trabalho.

Por outro lado, a falta de um Gémeo Digital que represente fidedignamente o processo em questão, seja por falta de dados, seja pela sua complexidade, pode também levar o algoritmo a soluções pouco eficientes, o que coloca os processos simulatórios um pouco em xeque nesse contexto. A concepção correta das regras, leis de ação e reação, que regem o ambiente, são determiníticas determinantes para que haja um bom treinamento. Além disso, essa questão também é válida para o agente treinado no algoritmo, pois determinar corretamente o conjunto de regras de recompensa e de punição pode ser desafiante, porque corre-se o risco de que valores mal concebidos possam levar o agente a soluções ruins, presas em mínimos locais, comprometendo a qualidade da aprendizagem.

O trabalho realizado nesta dissertação foi focado em sistemas discretos e com funcionamento sequencial, tema que apresenta uma lacuna significativa no campo académico. A maioria dos trabalhos existentes publicados por outros autores são focados em problemas de natureza

contínua ou simplesmente desconsideram a questão da memória nas sequências discretas. Nesse sentido, é importante ressaltar a relevância desse tema, já que muitos processos industriais são compostos por sequências discretas de ações e a capacidade de lidar com memória em tais sequências pode ser crucial para o sucesso da aplicação de Aprendizagem por Reforço em processos industriais.

Acredita-se também que há um vasto campo de implementação e pesquisa de Aprendizagem por Reforço aplicado no controlo de variáveis contínuas, nomeadamente o controlo PID. Isto porque determinar os ganhos do PID, a partir do cálculo de variáveis de espaço de estado, não é tarefa trivial e requer constantes ajustes no ambiente industrial. Dessa forma, modelos treinados e retreinados continuamente ao longo do tempo podem diminuir o tempo de intervenção técnica e trazer eficiência e estratégia ao controlo de processos industriais e de políticas de ações.

Sobre a habilidade de memorização de estados anteriores, durante a jornada do agente, ainda que as arquiteturas de redes neuronais recorrentes em LSTM sejam projetadas e adequadas a esse tipo de problema, é mais vantajoso a utilização de redes neuronais comuns combinadas com a informação de estados anteriores disponíveis logo na entrada dos algoritmos, formando os *buffers* de memória supracitados. Essa combinação traz uma aprendizagem mais rápida e eficiente no alcance das políticas de ações e estados necessárias para o fechamento do ciclo completo dos processos industriais, justificando tal preferência. Por isso, é válido a realização de estudos futuros comparativos e de eficiência de modelos que, além de possuírem essa informação de estados já na entrada do algoritmo, possuam também informações de ações e recompensas ocorridas nos passos anteriores da jornada agente.

Por fim, percebe-se pelos trabalhos académicos que a escolha do algoritmo, dos hiperparâmetros e da arquitetura costuma ser feitas por meio de tentativa, buscando aquela que se adeque melhor à complexidade do problema. Já neste trabalho, a manutenção dos hiperparâmetros e das arquiteturas nas configurações já padronizadas trouxeram bons resultados e atenderam às expectativas de controlo, sendo que os algoritmos *on-policy* A2C, TRPO e PPO tiveram melhores performances que os algoritmos *off-policy* DQN e CEM, possivelmente porque a política de decisão de ações nos processos simulados nos Gémeos Digitais/Simuladores é clara e completamente acessível ao agente. Em outros tipos de problemas, poderia haver destaque maior para os algoritmos *off-policy*.

Referências Bibliográficas

Adil, M.,, Ullah, R.,, Noor, S., and Gohar, N., 2022. Effect of number of neurons and layers in an artificial neural network for generalized concrete mix design. *Neural Computing and Applications*, 34, pp.1–9.

Anon, 1976. An Experiment with the Edited Nearest-Neighbor Rule. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-6(6), pp.448–452.

Anon, 1994. Numerical Taxonomy and Cluster Analysis. In: *Typologies and Taxonomies*. 2455 Teller Road, Thousand Oaks California 91320 United States of America: SAGE Publications, Inc., pp.35–65.

Anon, 2021. *Artificial Neural Networks*.

Anon, 2021. *The Neural Network Zoo - The Asimov Institute*.

Anon, 2023. Comparação de Métodos de Aprendizagem por Reforço em Processos Sequenciais Discretos - https://github.com/tiagoresilva/DigitalTwinRL

Anon, 2023. *Deep Reinforcement Learning for Keras*.

Anon, 2023. *OPCUA4Unity | Utilities Tools | Unity Asset Store*.

Anon, 2023. *Stable Baselines3*.

Anon, 2023. *TensorBoard*. TensorFlow.

Anon, 2023. *Unity ML-Agents Toolkit*.

Barhate, N. (2018). Proximal Policy Optimization (PPO) Explained. Medium.

Bellman, R., 1984. *Dynamic programming*. Princeton, NJ: Princeton Univ. Pr.

Bengio, Y., 2009. Learning Deep Architectures for AI. p.56.

de Boer, P.-T.,, Kroese, D.P.,, Mannor, S., and Rubinstein, R.Y., 2005. A Tutorial on the Cross-Entropy Method. *Annals of Operations Research*, 134(1), pp.19–67.

Borele, P., and Borikar, D.A., n.d. An Approach to Sentiment Analysis using Artificial Neural Network with Comparative Analysis of Different Techniques.

Dike, H.U.,, Zhou, Y.,, Deveerasetty, K.K., and Wu, Q., 2018. Unsupervised Learning Based On Artificial Neural Network: A Review. In: *2018 IEEE International Conference on Cyborg and Bionic Systems (CBS)*. 2018 IEEE International Conference on Cyborg and Bionic Systems (CBS). pp.322–327.

Gers, F.A.,, Schraudolph, N.N., and Schmidhuber, J., 2003. Learning Precise Timing with LSTM Recurrent Networks.

Glaessgen, E., and Stargel, D., 2012. The Digital Twin Paradigm for Future NASA and U.S. Air Force Vehicles. In: *53rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference
20th AIAA/ASME/AHS Adaptive Structures Conference
14th AIAA*. 53rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference
20th AIAA/ASME/AHS Adaptive Structures Conference
14th AIAA. Honolulu, Hawaii: American Institute of Aeronautics and Astronautics.

Hagan, M.T., and Menhaj, M.B., 1994. Training feedforward networks with the Marquardt algorithm. *IEEE Transactions on Neural Networks*, 5(6), pp.989–993.

Hastie, T.,, Tibshirani, R., and Friedman, J., 2009. Unsupervised Learning. In: T. Hastie, R. Tibshirani and J. Friedman, eds., *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, Springer Series in Statistics. New York, NY: Springer, pp.485–585.

Hinton, G.E.,, Osindero, S., and Teh, Y.-W., 2006. A Fast Learning Algorithm for Deep Belief Nets. *Neural Computation*, 18(7), pp.1527–1554.

Hochreiter, S., and Schmidhuber, J., 1997. Long Short-Term Memory. *Neural Computation*, 9(8), pp.1735–1780.

Hodge, V.J., and Austin, J., 2009. A Survey of Outlier Detection Methodologies. p.42.

Jiménez, J.,, Mochón, J.,, Ayala, J.S. de, and Obeso, F., 2004. Blast Furnace Hot Metal Temperature Prediction through Neural Networks-Based Models. *ISIJ International*, 44(3), pp.573–580.

Johnson, R., 2018. *Method for creating a digital twin of a room*. EP3291180A1.

Jordan, M.I., 1986. SERIAL ORDER: A PARALLEL DISTRmUTED PROCESSING APPROACH.

Kelemen, J., 2007. From Artificial Neural Networks to Emotion Machines with Marvin Minsky. *Acta Polytechnica Hungarica*, 4.

Krizhevsky, A.,, Sutskever, I., and Hinton, G.E., 2012. ImageNet Classification with Deep Convolutional Neural Networks. In: *Advances in Neural Information Processing Systems*. Curran Associates, Inc.

Le, T.P.,, Lee, D., and Choi, D., 2021. A Deep Reinforcement Learning-based Application Framework for Conveyor Belt-based Pick-and-Place Systems using 6-axis Manipulators under Uncertainty and Real-time Constraints. In: *2021 18th International Conference on Ubiquitous Robots (UR)*. 2021 18th International Conference on Ubiquitous Robots (UR). pp.464–470.

Lipton, Z.C.,, Berkowitz, J., and Elkan, C., 2015. A Critical Review of Recurrent Neural Networks for Sequence Learning. *arXiv:1506.00019 [cs]*.

Marius Matulis and Carlo Harvey 2022. *A robot arm digital twin utilising reinforcement learning | Elsevier Enhanced Reader*.

Mcculloch, W.S., and Pitts, W., 1943. A LOGICAL CALCULUS OF THE IDEAS IMMANENT IN NERVOUS ACTIVITY. p.17.

Minsky, M., 1961. Steps toward Artificial Intelligence. *Proceedings of the IRE*, 49(1), pp.8–30.

Mnih, V., et al., 2013. *Playing Atari with Deep Reinforcement Learning*.

Mosavi, A., et al., 2020. Comprehensive Review of Deep Reinforcement Learning Methods and Applications in Economics. *Mathematics*, 8(10), p.1640.

Neftci, E.O., and Averbeck, B.B., 2019. Reinforcement learning in artificial and biological systems. *Nature Machine Intelligence*, 1(3), pp.133–143.

Nguyen, H., and La, H., 2019. Review of Deep Reinforcement Learning for Robot Manipulation. In: *2019 Third IEEE International Conference on Robotic Computing (IRC)*. 2019 Third IEEE International Conference on Robotic Computing (IRC). pp.590–595.

Nian, R.,, Liu, J., and Huang, B., 2020. A review On reinforcement learning: Introduction and applications in industrial process control. *Computers & Chemical Engineering*, 139, p.106886.

Paccanaro, A., and Hinton, G.E., 2001. Learning distributed representations of concepts using linear relational embedding. *IEEE Transactions on Knowledge and Data Engineering*, 13(2), pp.232–244.

Pan, L., et al., 2020. Reinforcement Learning with Dynamic Boltzmann Softmax Updates. In: *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*. Twenty-Ninth International Joint Conference on Artificial Intelligence and Seventeenth Pacific Rim International Conference on Artificial Intelligence {IJCAI-PRICAI-20}. Yokohama, Japan: International Joint Conferences on Artificial Intelligence Organization, pp.1992–1998.

Rumelhart, D.E.,, Hinton, G.E., and Williams, R.J., 1986. Learning representations by back-propagating errors. *Nature*, 323(6088), pp.533–536.

Saxena, S., and Hote, Y.V., 2012. Advances in Internal Model Control Technique: A Review and Future Prospects. *IETE Technical Review*, 29(6), pp.461–472.

Schroeder, G., et al., 2016. Visualising the digital twin using web services and augmented reality. In: *2016 IEEE 14th International Conference on Industrial Informatics (INDIN)*. Poitiers, France: IEEE, pp.522–527.

Schulman, J., et al., 2017a. *Proximal Policy Optimization Algorithms*.

Schulman, J., et al., 2017b. *Trust Region Policy Optimization*.

Shuprajhaa, T.,, Sujit, S.K., and Srinivasan, K., 2022. Reinforcement learning based adaptive PID controller design for control of linear/nonlinear unstable processes. *Applied Soft Computing*, 128, p.109450.

Sutton, R.S., and Barto, A.G., 2014. Reinforcement Learning: An Introduction. p.352.

Tao, F.,, Zhang, H.,, Liu, A., and Nee, A.Y.C., 2019. Digital Twin in Industry: State-of-the-Art. *IEEE Transactions on Industrial Informatics*, 15(4), pp.2405–2415.

Technologies, U., 2023. *Unity Real-Time Development Platform | 3D, 2D, VR & AR Engine*.

Vinyals, O., et al., 2017. *StarCraft II: A New Challenge for Reinforcement Learning*.

Wang, L., and Canedo, A.M., n.d. (71) Applicant: Siemens Aktiengesellschaft,Munich.

Wyse, L., 2017. *Audio Spectrogram Representations for Processing with Convolutional Neural Networks*.

Apêndice A

Neste apêndice será apresentada uma descrição mais detalhada das funções de ativação utilizadas em redes neurais, explorando as principais características de cada função, bem como seu gradiente, as vantagens e as desvantagens da aplicação.

A função Sigmoide vem da forma S do seu nome e é definida pela fórmula abaixo. Esta é uma função de suavização, fácil de derivar e frequentemente implementada na previsão probabilística da saída, normalizando os dados, uma vez que a função sempre resulta entre um número entre 0 e 1. É calculada pela equação A.1 e possui o comportamento gráfico da função e do gradiente como mostrado na figura 45:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{A.1}$$

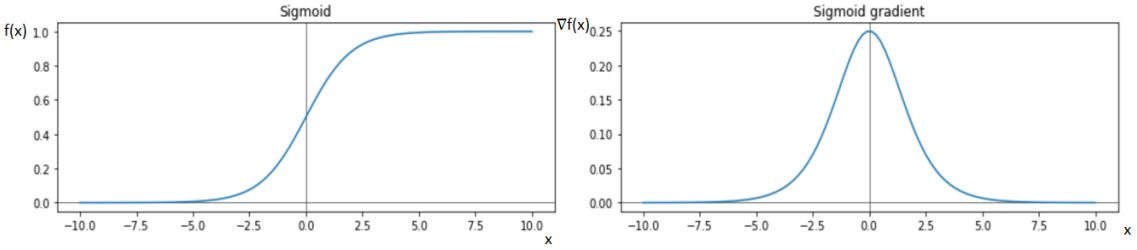

Figura 45: Função de ativação Sigmoide e o seu respetivo gradiente

Pode-se notar no gradiente resultante que a função Sigmoide possui o problema do gradiente banido e que resulta no efeito do neurônio morto na aprendizagem da rede neuronal. Isso significa que em pontos muito positivos ou negativos da função de ativação, o gradiente diminui drasticamente, resultando em camadas que recebem pouca ou nenhuma informação sobre o erro, dificultando a atualização dos pesos para melhora do desempenho da rede. Como o gradiente dessa função de ativação é uma distribuição gaussiana, valores dentro do intervalo de três desvios padrões do gradiente compreendem o efeito da aprendizagem da rede, enquanto fora desse intervalo, o gradiente é banido.

Já a função Tanh é bastante similar a função sigmoide, porém possui um gradiente resultante mais estreito que a função anterior. É calculada pela equação A.2 e possui o comportamento gráfico da função e do gradiente como mostrado na figura 46:

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{A.2}$$

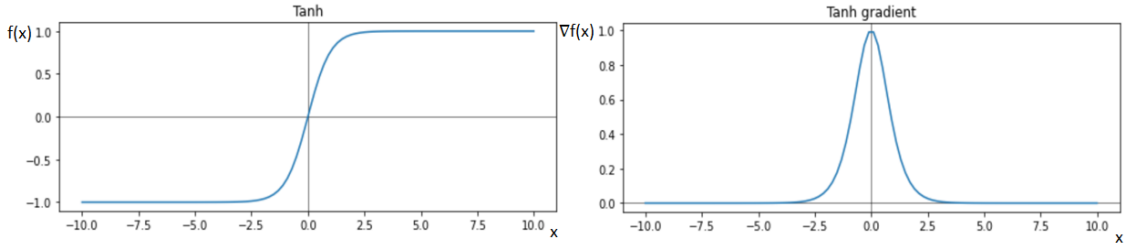

Figura 46: Função de ativação Tangente Hiperbólica e o seu respetivo gradiente

Ambas são bastante aplicadas em problemas de classificação, onde se requer uma previsão probabilística da saída. Entretanto, enquanto a função Sigmoide é normalmente usada na camada de saída, a função tangente hiperbólica é utilizada na camada oculta da rede, de acordo com cada problema específico.

A função de ativação ReLU é amplamente utilizada e possui a fórmula da equação A.3 e comportamento gráfico da função e do gradiente como mostrado na figura 47:

$$f(x) = \max(0, x) \tag{A.3}$$

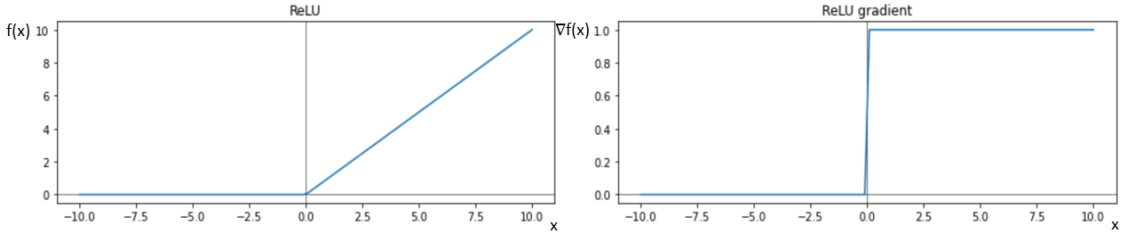

Figura 47: Função de ativação ReLU e o seu respetivo gradiente

Ela é uma das funções mais eficientes e requer menor esforço computacional, uma vez que o gradiente pode ser mais facilmente obtido e o problema do gradiente banido deixa de existir para valores positivos da função. O problema aqui ocorre para valores de entrada negativos, quando o gradiente perde o seu efeito e passa ocorrer o problema do neurônio morto ou *"Dying ReLU*". No processo de *forward propagation*, isto não é um problema, porque algumas áreas são sensitivas e outras não são. Entretanto, no *backpropagation*, um valor nulo implica na não atualização do peso do neurônio em treinamento.

Com o intuito de solucionar o problema do neurônio morto, foi proposta a troca do valor nulo da primeira metade resultante da função ReLU por 0.01x, originando a função *Leaky* ReLU, conforme pode ser visto na equação A.4:

$$f(x) = \max(0{,}01x, x) \tag{A.4}$$

Essa função carrega as vantagens da ReLU, sem manter os problemas de aprendizagem ocorridos pelo neurônio morto, suavizando o problema de banimento do gradiente. O comportamento gráfico da função e do gradiente pode ser visto abaixo na figura 48:

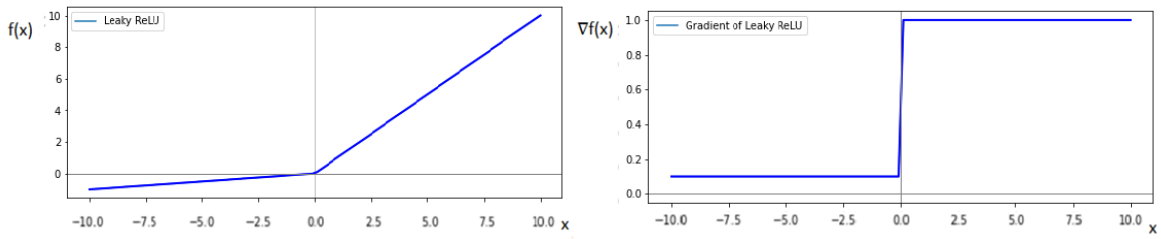

Figura 48: Função de ativação *Leaky ReLU* e o seu respetivo gradiente

Uma implementação mais sofisticada que a leaky ReLU é a função ELU, que apresenta uma função exponencial para valores de entrada negativo e pode ser descrita pela fórmula da equação A.5 e comportamento gráfico da função e do gradiente como mostrado na figura 49:

$$f(x) = \begin{cases} x, & se\ x > 0 \\ a(e^x - 1), & caso\ contrário \end{cases} \tag{A.5}$$

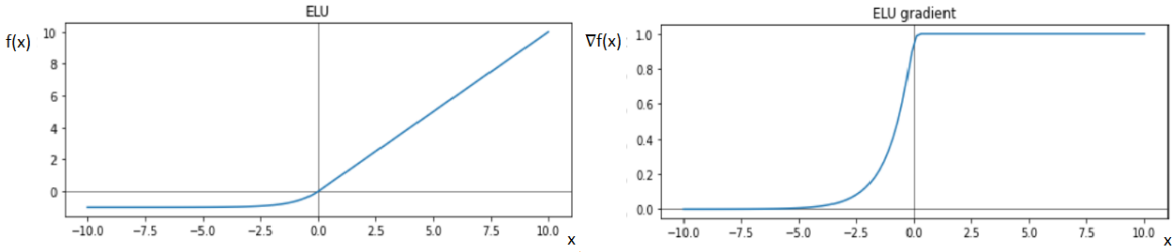

Figura 49:Função de ativação ELU e o seu respetivo gradiente

ELU não apresenta problemas de neurónio morto, nem de banimento do gradiente, uma vez que a função é contínua e crescente em todos os pontos. Mais, apresenta uma saída exponencial para valores negativos no gradiente, o que caracteriza uma atualização do peso dos neurónios à medida que o valor se aproxima de zero. Por outro lado, possui a desvantagem de requerer um esforço computacional um pouco maior que as outras funções devido a complexidade em se obter a derivada no ponto exponencial.

Outra função de ativação normalmente utilizada é a função Swiss, que é uma função não linear geralmente usada em camadas intermediárias de redes neuronais profundas, juntamente com outras funções de ativação, como a ReLU ou Tanh. A função Swiss é

definida como uma combinação da função Sigmoide e da função ReLU, o que significa que ela mantém as vantagens de ambas. Como a função ReLU, a função Swiss é fácil de calcular e não sofre do problema de banimento do gradiente. E como a função sigmóide, ela pode gerar saídas suaves e contínuas em uma ampla faixa de valores de entrada, o que a torna adequada para tarefas de regressão. Em geral, a função Swiss é uma escolha popular de função de ativação em redes neurais profundas devido à sua eficácia em equilibrar as vantagens da ReLU e da sigmóide. Pode ser descrita pela seguinte fórmula da equação A.6 e comportamento gráfico da função e do gradiente como mostrado na figura 50:

$$f(x) = x * (1 + \exp(-x))^{-1} \tag{A.6}$$

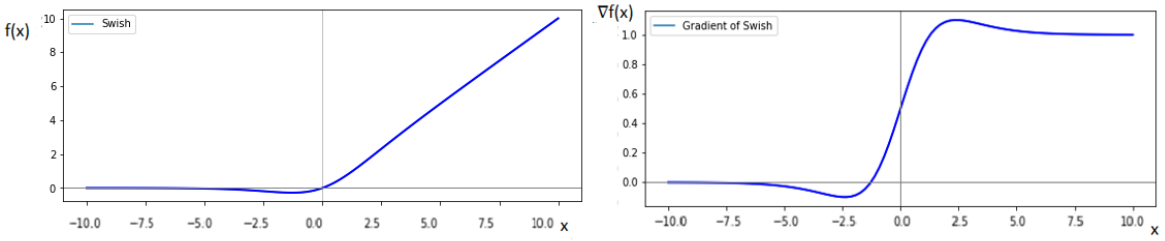

Figura 50: Função de ativação Swiss e o seu respetivo gradiente

Por fim a função Softmax, bastante similar a função Sigmoide, normalmente utilizada na camada de saída da arquitetura de redes neurais, é definida pela fórmula da equação A.7, onde z é um vetor de k números reais normalizados em uma distribuição de probabilidades:

$$\sigma(z)_i = \frac{e^{zi}}{\sum_{j=1}^{k} e^{zj}} \; para\; i = 1, \dots, k\; e\; z = (z1, \dots zk) \tag{A.7}$$

A adaptação da função Sigmoide para Softmax tem um papel na classificação muticlasses, no qual a previsão probabilística de mais de duas classes é necessária para designar para uma determinada entrada.

Apêndice B

Este apêndice tem como objetivo fornecer uma descrição mais detalhada sobre algumas das arquiteturas de redes neurais mais utilizadas em aprendizado de máquina: *Auto encoders*, *Deep Belief Networks* (DBN), *Generative Adversarial Networks* (GAN) e *Self-Organizing Maps* (SOM). A compreensão dessas arquiteturas é essencial para o avanço em muitas áreas da aprendizagem de máquina e inteligência artificial, incluindo processamento de imagem, reconhecimento de voz, geração de texto e muito mais. Seguem abaixo as quatro principais estruturas:

- *Auto enconders*: elemento utilizado no estudo de codificações, representações de conjunto de dados (*Numerical Taxonomy and Cluster Analysis, 1994)*(ver figura 51). Trata-se da compactação da arquitetura por meio de um gargalo, na qual força a rede a aprender características que tenham correlação com os dados apresentados na entrada, criando dessa forma um filtro de ruídos para que resulte somente informações relevantes na saída;

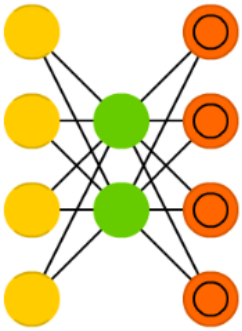

Figura 51: *Autoencoder* (The Neural Network Zoo - The Asimov Institute, 2021)

- *Deep Belief Network* (DBN): classe de *Deep Neural Network* (DNN) composta por várias camadas, idêntica à perceptron de multicamadas, com conexões entre as camadas, mas não entre a unidades dentro das camadas (Hodge and Austin, 2009) (ver figura 52). Quando treinada sem supervisão, o DBN pode aprender a reconstruir probabilisticamente suas entradas;

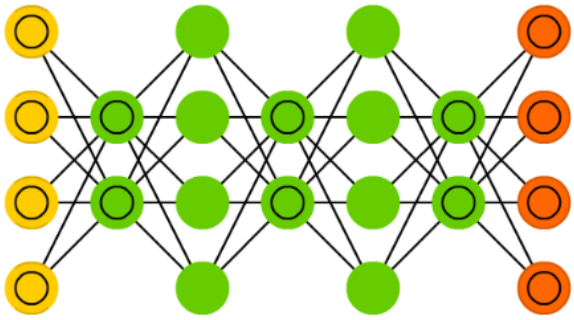

Figura 52*: Deep Belief Network* (The Neural Network Zoo - The Asimov Institute, 2021)

- Rede Adversarial Generativa (GAN): Estrutura de duas redes neuronais que rivalizam entre si em um concurso de jogo de soma zero (Bengio, 2009). A primeira rede tem o papel de classificar binariamente os dados em verdadeiro ou falso, enquanto a segunda, representada abaixo pelos nós laranjas, insere ruídos nessa rede classificadora a fim de diminuir o desemprenho (ver figura 53). Como o gerador de ruído obtém o resultado da classificação da primeira rede, à medida que o treino vai ocorrendo, esse gerador vai ficando cada vez mais eficiente em distorcer os dados de entrada. Dessa forma, ao final do treino a rede classificadora não é capaz de diferenciar o verdadeiro do falso, mas ao invés disso aprende de fato a distribuição de dados da classe de entrada inserida. Métodos Actor-Critic, descritos no capítulo 2.4.4.2, inspiram-se nessa arquitetura de maneira que a rede neuronal de valor Q corrige a rede neuronal de política.

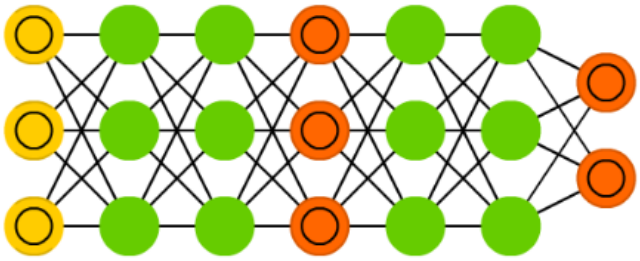

Figura 53: Rede Adversarial Generativa (The Neural Network Zoo - The Asimov Institute, 2021)

- Mapas auto-organizáveis (SOM): Método capaz de organizar dimensionalmente dados complexos em grupos (clusters), de acordo com suas relações, também conhecida como Rede Kohonen (Hinton et al., 2006) (ver figura 54).

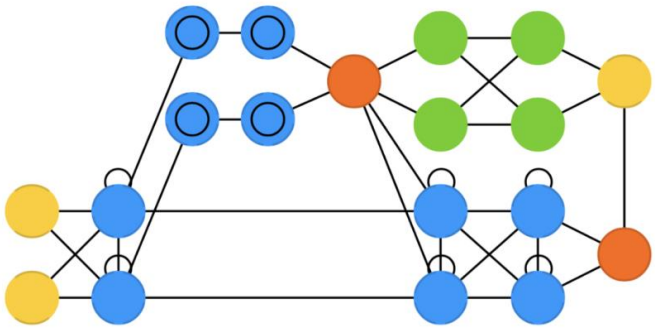

Figura 54: Mapas auto-organizáveis (The Neural Network Zoo - The Asimov Institute, 2021)

Em relação a essas redes, as que utilizam *Autoencoders* compactam informações, limpando ruídos, enquanto um grupo particular, os *Sparse Autoencoders* extraem pequenas características de um conjunto de dados. Isso possibilita uma aprendizagem intuitiva e subjetiva de informações para além daquelas fornecidas pelos dados de entrada. Já o pré-treino aplicado à DBN, por meio de propagação *feedforward*, contribui para o encontro do mínimo local global durante o *backpropagation*, evitando dessa forma que o gradiente fique preso em mínimos locais indesejáveis.

Apêndice C

Neste apêndice serão apresentadas as sequências resumidas, quadro a quadro, dos processos ECL e GML, comparando a aprendizagem dos algoritmos treinados com as realidades já pré-concebidas.

Dessa forma, a figura 55 abaixo mostra a sequência resumida de ECL concebida pela Real Virtual (empresa desenvolvedora desta realidade virtual) (OPCUA4Unity | Utilities Tools | Unity Asset Store, 2023), e aprendida pelo algoritmo A2C, com buffer de 25 passos anteriores.

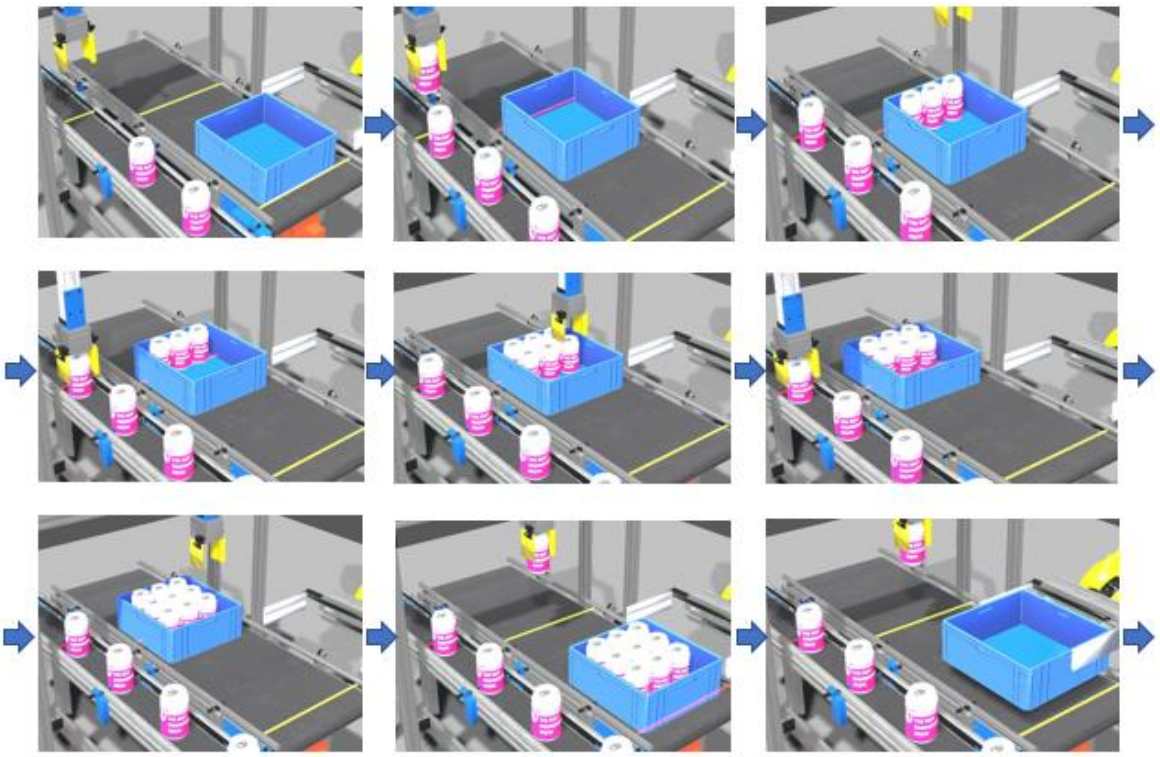

Figura 55: Sequência resumida do processo ECL, quadro a quadro, da realidade virtual pré-concebida pela Real Virtual e aprendida pelo algoritmo A2C, com *buffer* de 25 passos anteriores.

Em seguida, a figura 56 abaixo mostra a sequência resumida de ECL aprendida pelo algoritmo TRPO, sem buffer de passos anteriores, onde é possível notar erros na tomada de decisão nos quadros 8 e 9. Ao invés da esteira retornar a caixa à posição 0 para descarga, o agente continua a movê-la adiante, como se entendesse que não houvesse fila preenchida. Quando a caixa passa adiante do sensor da garra, para além da posição 600 cm, o sistema aciona o retorno da caixa para a posição 600 cm, entendendo que ainda há fila a preencher, mantendo um ciclo interminável entre os quadros 8 e 9, destacado nessa figura pelas setas

vermelhas. Isso revela claramente a falta do efeito memória na aprendizagem, na qual não foi possível memorizar o preenchimento das filas nos passos corretas, movimentando ECL somente diante do sinal de conclusão enviado por GML.

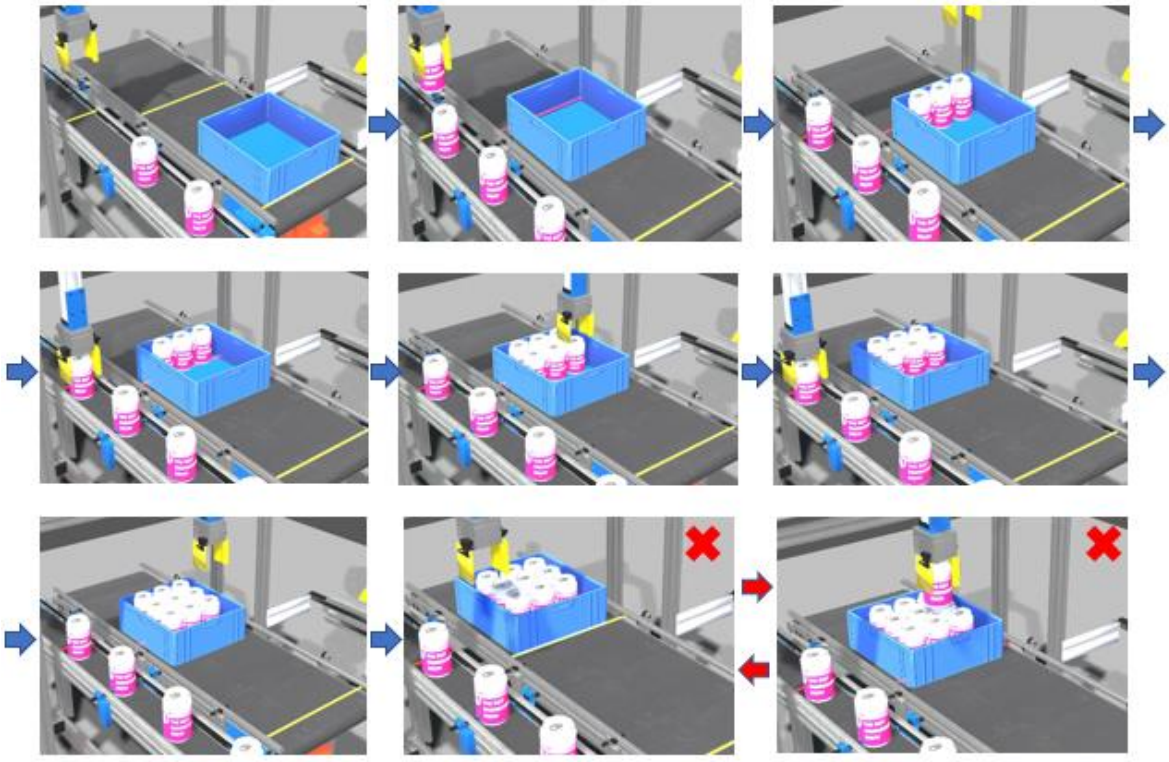

Figura 56: Sequência resumida do processo ECL, quadro a quadro, aprendida pelo algoritmo TRPO, sem *buffer* de passos anteriores.

Relativamente a GML, a figura 57 abaixo mostra a sequência resumida concebida pela Real Virtual (OPCUA4Unity | Utilities Tools | Unity Asset Store, 2023), no qual os trajectos percorridos são um pouco mais longos que aqueles percorridos pelos algoritmos. Já na figura 58 abaixo, que mostra a seqûencia resumida de GML aprendida pelo algoritmo TRPO, com *buffer* de 10 passos anteriores, é possível notar um percurso em diagonal, tanto da posição 2 para 3, 5 e 7, como das posições 4, 6 e 8 para posição 1 (figura 16 no capítulo 4.2 mostra esses posicionamentos). Na figura 58, as setas azuis mostram os quadros com percursos normais, (igualmente com a realidade virtual já pré-concebida), enquanto as setas verdes destacam os quadros com caminhos otimizados. Além de encurtar os trajectos percorridos, o tempo de execução também se reduz, gerando um ganho na execução do ciclo em PEL. Tais percursos são resultados do fator de desconto γ e das regras de recompensas e de punições que estimulam o agente a alcançar o objeto geral no menor tempo possível.

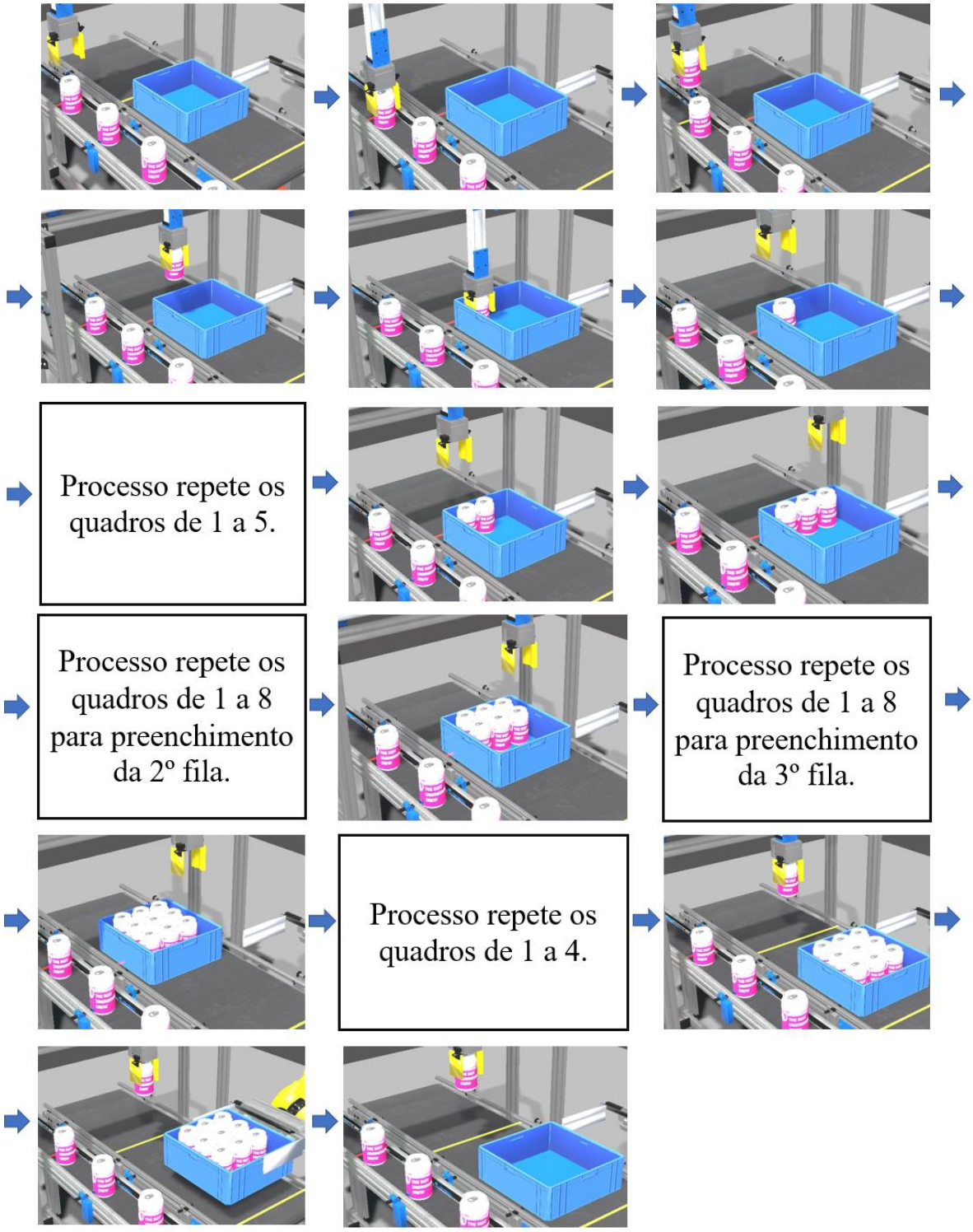

Figura 57: Sequência resumida do processo GML, quadro a quadro, da realidade virtual pré-concebida pela Real Virtual.

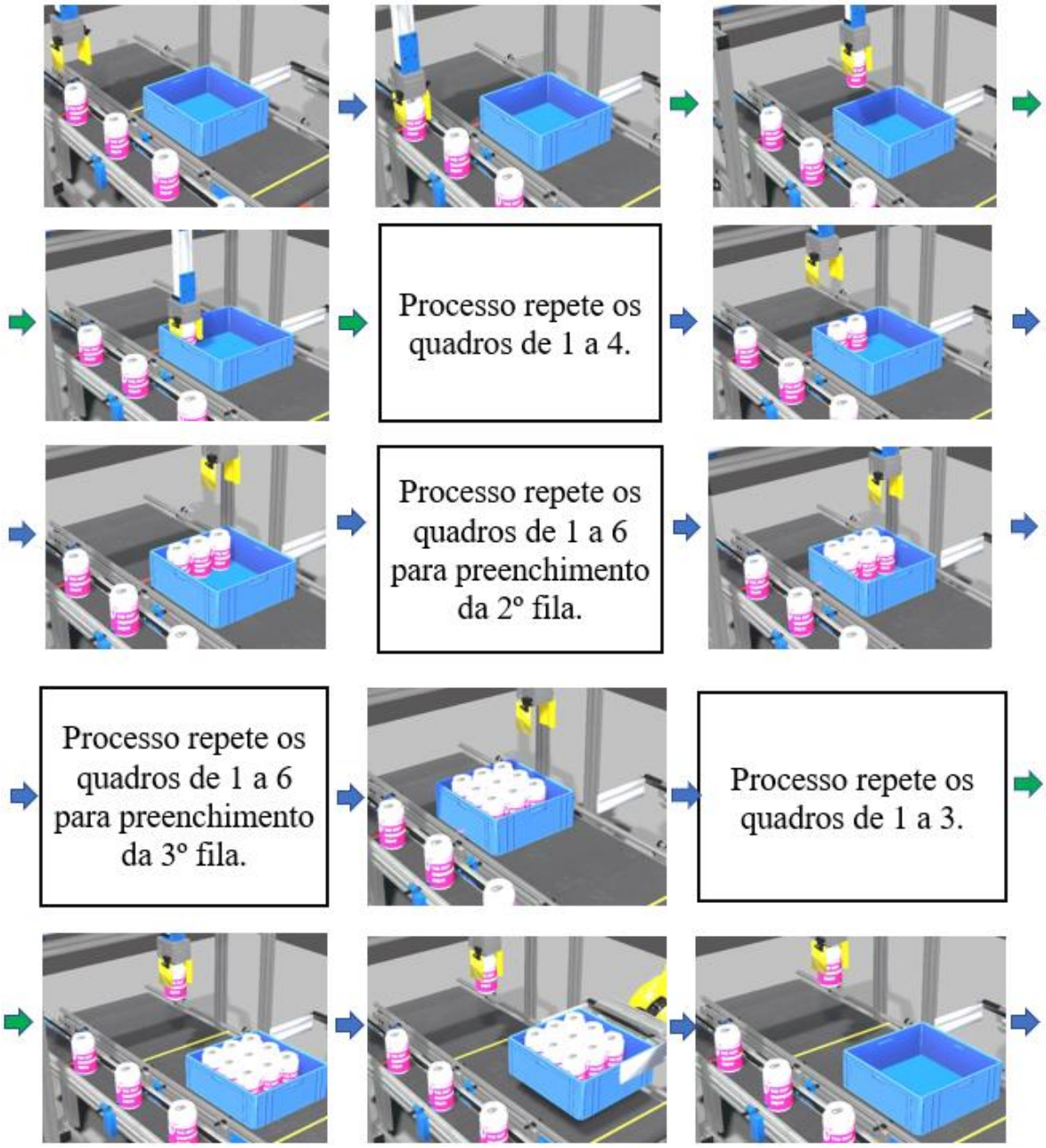

Figura 58: Sequência resumida do processo GML, quadro a quadro, aprendida pelo algoritmo TRPO, com *buffer* de 10 passos anteriores.

Já os algoritmos treinados com tamanho menor de *buffer* ou sem *buffer* de passos anteriores resultaram em sequências com tomadas de decisão erradas em GML, na qual não foi possível estabelecer um efeito memória nesses passos. Nomeadamente, a figura 59 mostra a aprendizagem do algoritmo TRPO, sem *buffer* de passos anteriores, cuja sequência é possível perceber o posicionamento de latas em posições já preenchidas anteriormente. Esses quadros são destacados nessa figura pelas setas vermelhas, mostrando a sobreposição das latas e que resultaram em colisões.

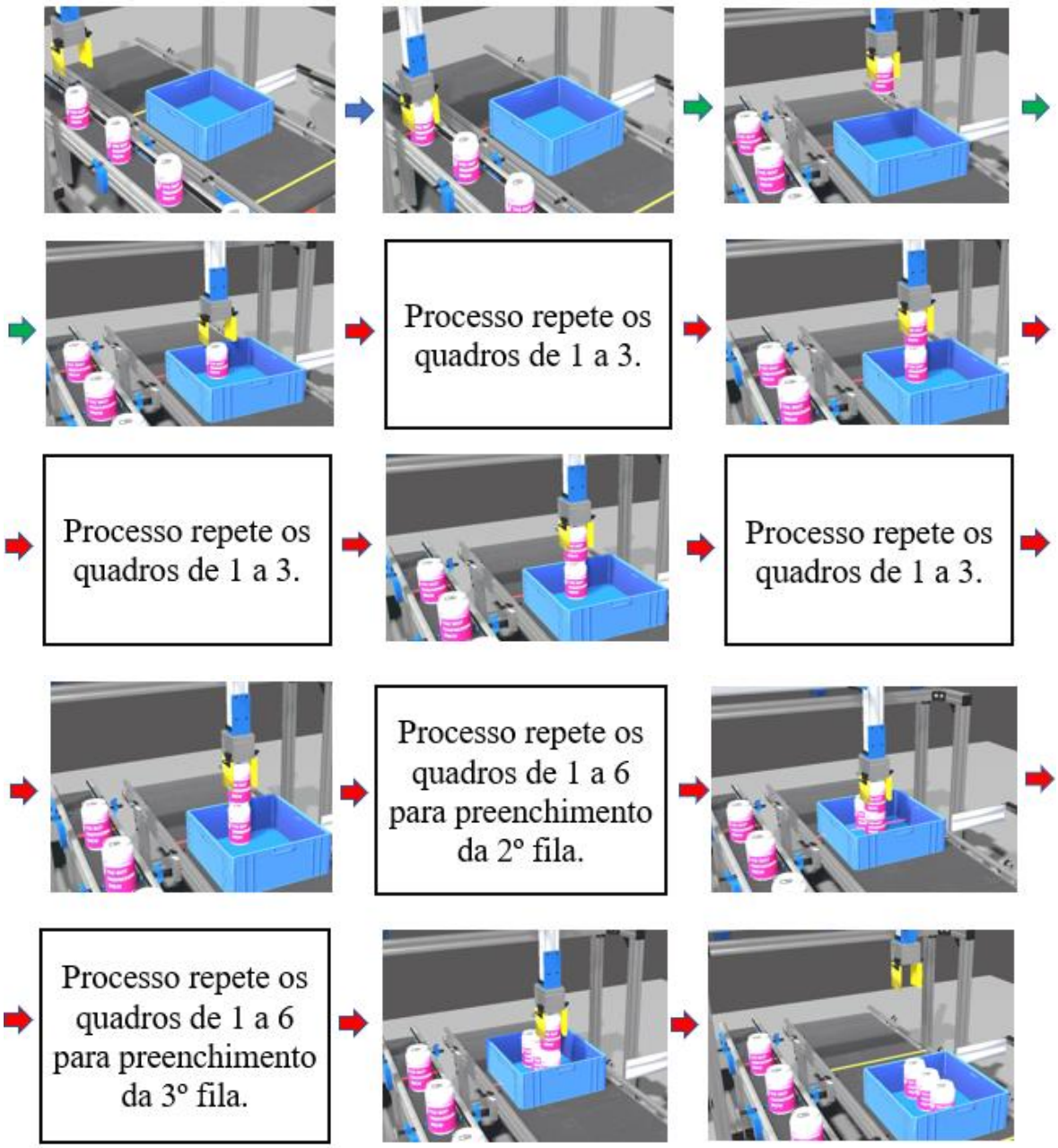

Figura 59: Sequência resumida do processo GML, quadro a quadro, aprendida pelo algoritmo TRPO, sem *buffer* de passos anteriores.