APRENDIZAGEM POR REFORÇO APLICADA A JOGOS DO ATARI

Reinforcement Learning Applied to Atari Games

George. Nunes[1]; Gustavo L. Santana[2]; Cristian D. Giovanni[3]; Rafael Oliveira Rosário[4]; Luzia dos Santos Souza Neta[5]

Departamento De Computação, Universidade Federal de Sergipe, 49100-000, São Cristóvão-SE, Brasil

[1]georgelucas@academico.ufs.br
[2]gustavo864@academico.ufs.br
[3]cristiandg@academico.ufs.br
[4]rafaor20@academico.ufs.br
[5]luzia450@academicio.ufs.br

Este trabalho apresenta a implementação de um agente baseado em Aprendizagem por Reforço (RL) para jogar um jogo específico do Atari, utilizando abordagens mistas que combinam Deep Q-Networks (DQN) com redes neurais convolucionais (CNN) e/ou perceptrons multicamadas (MLP). O estudo explora o impacto dessas arquiteturas na capacidade do agente de tomar decisões eficientes, comparando diferentes estratégias de aprendizado e otimização. Para a experimentação, foi utilizado o Arcade Learning Environment (ALE), analisando métricas de desempenho como taxa de vitória, pontuação média e tempo de aprendizado. Os resultados obtidos são discutidos em termos de eficiência do aprendizado, robustez das arquiteturas e possíveis melhorias futuras.

Palavras-chave: *Aprendizagem por reforço, Deep Q-Network, Arcade Learning Environment.*

This work presents the implementation of an RL-based agent to play a specific Atari game, using hybrid approaches that combine Deep Q-Networks (DQN) with convolutional neural networks (CNN) and/or multilayer perceptrons (MLP). The study explores the impact of these architectures on the agent's ability to make efficient decisions, comparing different learning and optimization strategies. For experimentation, we used the Arcade Learning Environment (ALE), analyzing performance metrics such as win rate, average score, and learning time. The obtained results are discussed in terms of learning efficiency, architecture robustness, and potential future improvements.

Keywords: *Reinforcement learning, Deep Q-Network, Arcade Learning Environment.*

1. INTRODUÇÃO

Uma das áreas da IA que mais pode trazer avanços para os jogos eletrônicos é a ML (machine learning, ou aprendizagem de máquina). Com a ML, é possível fazer com que o computador aprenda no decorrer das partidas disputadas e possa modificar seu comportamento de acordo com as situações encontradas e com o adversário que estiver enfrentando (BOTELHO NETO, 2013). A Aprendizagem por Reforço (RL, do inglês Reinforcement Learning) tem se destacado como uma das abordagens mais promissoras nesse campo, especialmente quando aplicada a ambientes complexos e dinâmicos, como os jogos eletrônicos. Dentre os diversos desafios propostos para testar algoritmos de RL, os jogos clássicos do Atari emergiram como um *benchmark* popular, devido à sua simplicidade visual combinada com a complexidade estratégica necessária para alcançar altos desempenhos. Esses jogos oferecem um ambiente ideal para explorar e comparar diferentes técnicas de RL, desde métodos

tradicionais até abordagens mais avançadas, como redes neurais profundas e algoritmos de planejamento.

Neste trabalho, é proposta a implementação de um agente de RL capaz de jogar um jogo específico do Atari, utilizando uma combinação de técnicas modernas, como Deep Q-Networks (DQN) proposto por (MNIH et. al, 2015), arquiteturas neurais convolucionais (CNN), introduzidas por (LECUN, BOTTOU & HAFFNER, 1998), e de múltiplas camadas (MLP) (RUMELHART; HINTON & WILLIAMS, 1986).

O objetivo principal deste projeto é desenvolver um agente de RL que não apenas consiga jogar o jogo escolhido, mas que também demonstre eficiência em termos de aprendizado e adaptação, superando desafios como a exploração do ambiente, a generalização de estratégias e a otimização de políticas. Para isso, serão conduzidos experimentos detalhados, com métricas de desempenho claras, que permitirão avaliar a eficácia das diferentes abordagens implementadas.

2. MATERIAL E MÉTODOS

A abordagem adotada combina a técnica de Deep Q-Networks (DQN), com arquiteturas neurais convolucionais (CNN) e de múltiplas camadas (MLP), fazendo uma combinação e comparação entre as duas abordagens além de comparar com uma abordagem mista também.

2.1 Ambiente de Desenvolvimento e Ferramentas

Para a implementação do agente de RL, utilizamos o Arcade Learning Environment (ALE) através da biblioteca ale-py, que fornece uma interface padronizada para interação com o ambiente do jogo. O ambiente de simulação foi configurado utilizando gymnasium[atari,accept-rom-license], que permite a execução de jogos clássicos do Atari com suporte a licenças de ROM. Para a renderização e visualização do jogo, utilizamos o pygame.

As principais bibliotecas e frameworks utilizados incluem: PyTorch, NumPy, Stable-Baselines3, OpenCV (opencv-python e opencv-python-headless), TensorBoard. Todos os requerimentos para a execução do projeto estarão listados no arquivo ‘requirements.txt’ do repositório no GitHub.

2.2 Arquitetura do Agente de RL

O agente de RL foi desenvolvido com base em uma abordagem de Deep Q-Network (DQN), utilizando uma combinação de redes neurais convolucionais (CNN) e redes de múltiplas camadas (MLP).

A configuração do algoritmo DQN do agente durante o treinamento é dada por:

- Learning Rate = 0.0001,
- Buffer Size = 100000,
- Learning Starts = 10000,
- Batch Size = 32,
- Soft Update Coefficient (tau) = 0.1,
- Discount Factor (gamma) = 0.99,
- Training Frequency = 4,
- Gradient Steps = 1,
- Target Network Update Interval = 1000,
- Exploration Fraction = 0.1,
- Exploration Final Epsilon = 0.01,

A quantidade total de timesteps aplicada a cada modelo foi de 500.000, para um estudo mais profundo um treinamento com mais de 1.000.000 de timesteps seria necessário, porém devido a limitações técnicas de hardware, tal número foi impossível de ser alcançado.

2.4 Configuração de Hardware

O treinamento e a avaliação do agente foram realizados em um ambiente de hardware com as seguintes especificações:

- CPU: Intel Core i5-825OU
- GPU: NVIDIA MX150
- RAM: 12 GB DDR4
- Armazenamento: SSD de 1 TB para armazenamento de datasets e modelos

3. RESULTADOS E DISCUSSÃO

3.1 DQN utilizando CNN como política

Durante o desenvolvimento do primeiro agente, utilizamos dois algoritmos de aprendizado por reforço em conjunto, são eles o DQN (Deep Q-Learning Network) e o CNN (Convolutional Neural Network), o CNN foi utilizado para enxergar o jogo e o DQN foi utilizado para interagir e aprender a jogar jogo.

Após o desenvolvimento do modelo, foi possível notar uma diminuição regular da taxa de derrota em relação ao treinamento durante os primeiros 10 por cento do treinamento, mas após isso, esta taxa começa a crescer de forma bastante irregular, indicando que o agente não está aprendendo de forma eficiente.

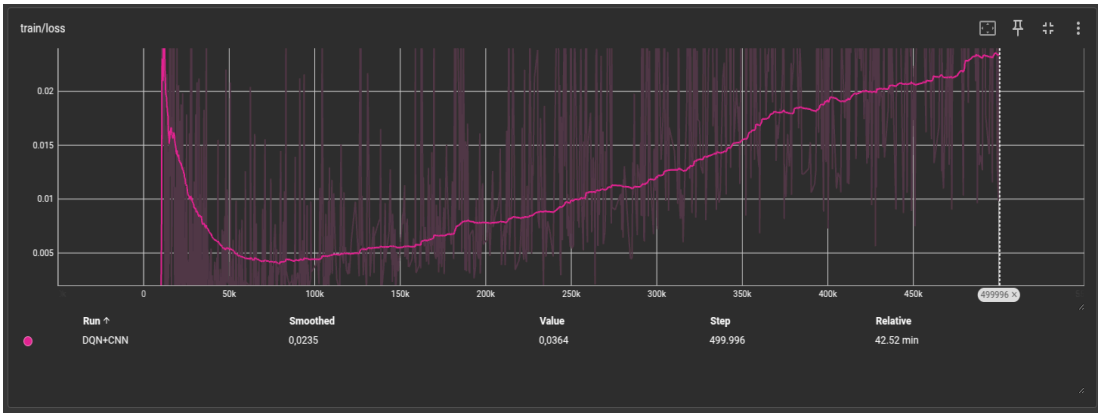

Figura 1: Gráfico de Loss durante o treinamento. Fonte: autor

As possíveis causas deste problema incluem os valores dos parâmetros utilizados ou o overfitting, que podem levar o agente a explorar menos novas possibilidades após alcançar um certo nível de aprendizado.

3.1 DQN utilizando MLP como política

Durante este experimento, foi usada a política MLP utilizando a memória ram do jogo, a arquitetura consiste em um a entrada de tamanho 1x128 representando a memória do jogo, tres

camadas ocultas de 256 e uma 4 saídas representando as ações, durante os testes o agente tentou maximizar a recompensa deixando a primeira bola escapar e não apertando o botão de atirar para continuar a partida, assim sobrevivendo por mais tempo.

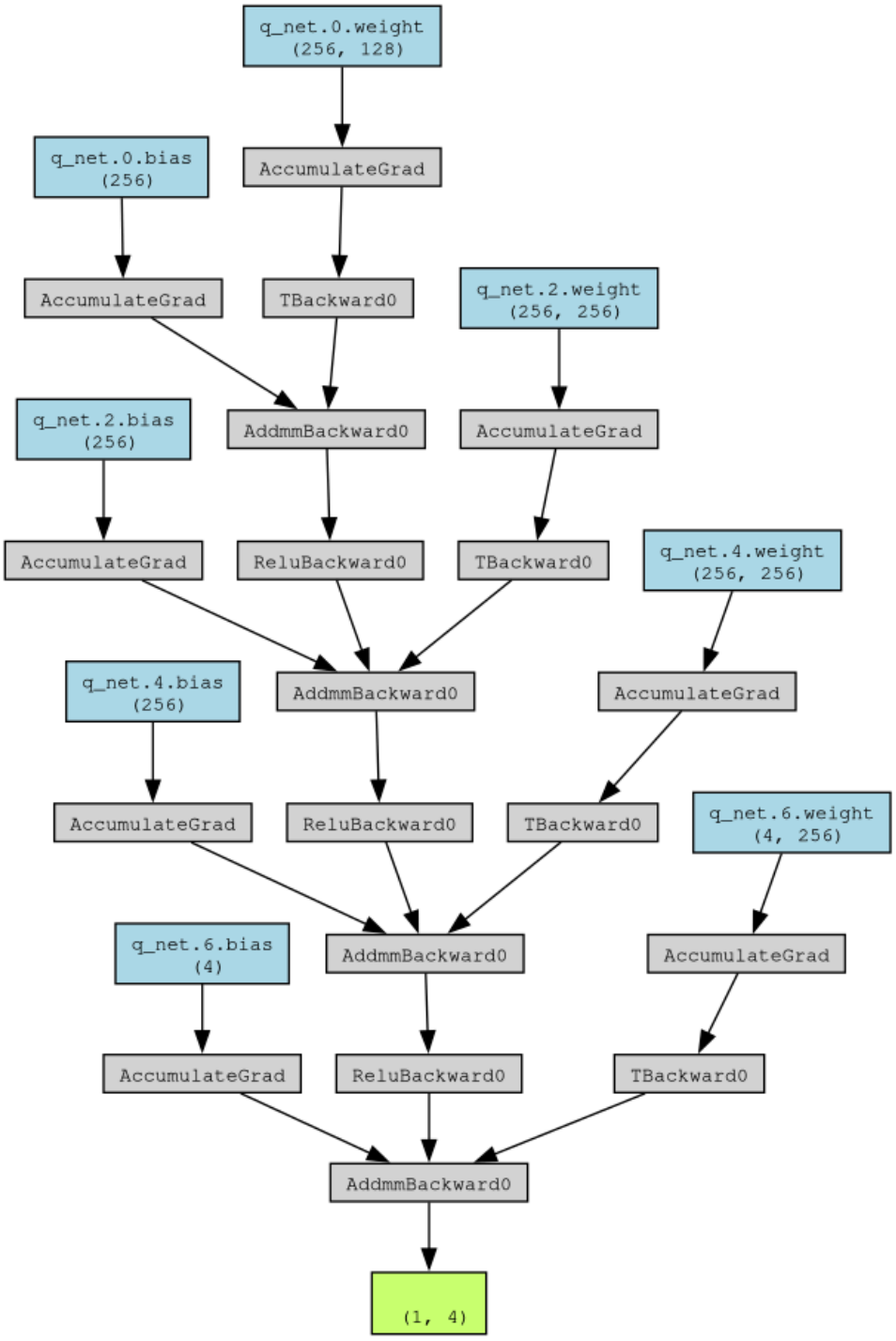

Figura 2: Arquitetura do modelo. Fonte: autor

O treinamento do modelo foi extremamente eficiente, mesmo usando a CPU não levou mais de 20 minutos para treinar o modelo, mas a sua performance deixou a desejar, o modelo não consegue nem sequer 5 pontos usando todas as suas vidas, tentei de tudo: aumentar exponencialmente a recompensa

em função do tempo em que ele se mantinha vivo, aumentei o tamanho e a quantidade das camadas, mas mesmo assim o modelo não consegue ficar vivo por mais de 250 frames.

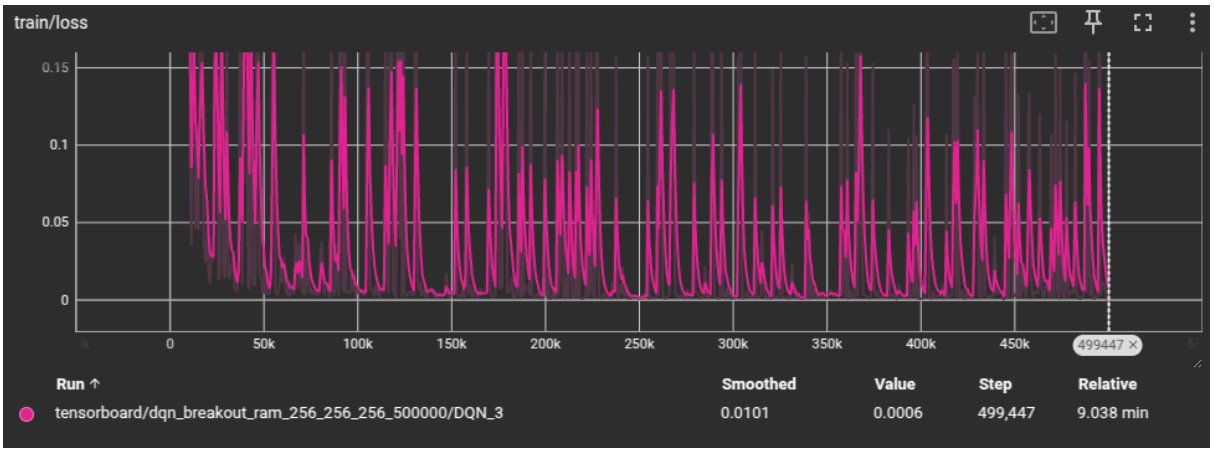

Figura 3: Gráfico de Loss durante o treinamento. Fonte: autor

3.1 DQN utilizando uma política customizada

Durante este experimento, uma política customizada que combina as políticas já conhecidas CNN e MLP numa mesma política foi desenvolvida e utilizada, o objetivo principal foi descobrir até que ponto a integração das duas políticas poderia influenciar no desempenho do modelo. Ademais buscou-se avaliar se a extração de características realizadas pela CNN poderia complementar e, ou, otimizar o processo de aprendizado da MLP que opera utilizando entradas vetoriais.

A política customizada desenvolvida tenta combinar as vantagens das redes neurais convolucionais e das redes neurais de múltiplas camadas em uma única arquitetura, na implementação a CNN é responsável por extrair características espaciais dos pixels em tela enquanto a MLP atua na interpretação das características obtidas para tomar decisões. O objetivo foi permitir que o modelo se aproveite da capacidade da CNN em processar os frames de entrada extraindo informações espaciais e deixando a MLP responsável por mapear os estados em ações.

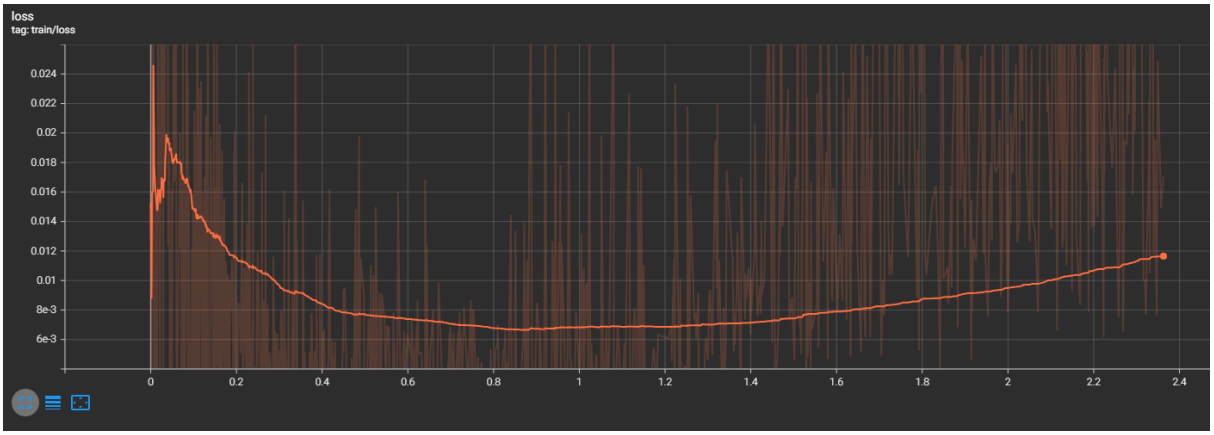

Figura 4: Gráfico de Loss durante o treinamento. Fonte: autor

Na figura 1 é possível notar que o gráfico de *loss* segue em queda até cerca de metade do treinamento quando começa a subir. Uma possível causa disso seria o *overfitting* e instabilidade no aprendizado do modelo, onde o agente do modelo se ajusta excessivamente aos dados do treinamento perdendo dessa forma sua capacidade de generalização para dados ainda não vistos. Além disso, outra

causa seria a própria parametrização do modelo, porém como o objetivo primário foi comparar o desempenho dos modelos com base na sua política, a parametrização entre as três abordagens (CNN, MLP e a política customizada) se manteve a mesma. Isso inclui parâmetros chave como tamanho do *buffer* de experiência, *bath*, taxa de aprendizado e os demais hiperparâmetros. Assim, a variação no desempenho pode ser atribuída principalmente às diferenças nas políticas utilizadas.

Outro ponto a ser considerado é a arquitetura mais complexa. A combinação das políticas aumenta o número de parâmetros do modelo levando a um maior custo computacional. Para ambientes mais simples onde a política MLP já seria suficiente, adicionar outra rede neural como a CNN não justifica o custo adicional. Uma possível solução para mitigar o problema desse modelo seria a implementação de técnicas de regularização dentro da política customizada.

Tabela 1: Comparação do algoritmo com diferentes redes neurais. Fonte: autor

	Algoritmo + Rede Neural		
Métrica	**DQN + CNN**	**DQN + MLP**	**DQN + CNN + MLP**
Loss médio	0.0196	0.01	0.009
Loss máximo	0.02161	0.05	0.024
Recompensa média	18.9791	2.0	14.44
Recompensa máxima	19.64	2.2	17.92
Taxa de Sucesso	764.82	183.76	639.52

4. CONCLUSÃO

Este estudo comparou o desempenho do algoritmo Deep Q-Network (DQN) utilizando diferentes arquiteturas de redes neurais: Convolutional Neural Networks (CNNs), Multi-Layer Perceptrons (MLPs) e uma combinação de ambas (CNN+MLP). Os resultados demonstraram que a escolha da arquitetura afeta significativamente a eficiência do aprendizado e o desempenho final do agente.

A abordagem baseada exclusivamente em CNN mostrou-se a mais eficiente, com uma recompensa média de 18.9, enquanto a abordagem baseada exclusivamente em MLP mostrou-se a pior dentre todas, com uma recompensa média de 2.0. Já a combinação CNN+MLP ofereceu um desempenho médio, alcançando resultados uma recompensa média de 14.44.

Tendo em vista o resultado dos estudos realizados nessa pesquisa, a escolha da arquitetura ideal para esse ambiente de atuação é o algorítmo DQN com uma rede neural CNN, a tentativa de adicionar MLP a esse agente ou até mesmo o treinamento de um agente somente com a MLP, trouxe resultados negativos para o estudo.

Dentre as razões para isso acontecer, uma que se destaca é a dificuldade da MLP de identificar padrões espaciais, além de sua dificuldade em lidar com inputs de alta dimensão, como no caso do Breakout (jogo do Atari escolhido como ambiente de estudo), de 100,800 inputs, considerando o tamanho da imagem com 3 canais de cores. Por último, um outro fator que também pode ter influenciado na má performance do algoritmo com MLP, foi a falta de memória temporal explícita do agente para inferir a direção da bola durante as partidas do jogo (MINSKY e PAPERT, 1969).

5. REFERÊNCIAS BIBLIOGRÁFICAS

1. BOTELHO NETO, Gutenberg Pessoa. Aprendizado por reforço aplicado ao combate em jogos eletrônicos de estratégia em tempo real. 2021. Dissertação (Mestrado em Informática) – Universidade Federal da Paraíba, Centro de Informática, João Pessoa, 2021. Disponível em: https://repositorio.ufpb.br/jspui/bitstream/tede/6128/1/arquivototal.pdf.

2. TABLE-BASELINES3. *Stable-Baselines3 Documentation*. 2025. Disponível em: https://stable-baselines3.readthedocs.io/en/sde/index.html.

3. Mnih, Volodymyr. "Playing atari with deep reinforcement learning." arXiv preprint arXiv:1312.5602 (2013). Disponível em: https://people.engr.tamu.edu/guni/csce642/files/dqn.pdf.

4. Guo, Xiaoxiao, et al. "Deep learning for real-time Atari game play using offline Monte-Carlo tree search planning." Advances in neural information processing systems 27 (2014). Disponível em: https://proceedings.neurips.cc/paper_files/paper/2014/file/8bb88f80d334b1869781beb89f7b73be-Paper.pdf.

5. Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature, 323*(6088), 533-536. Disponível em: https://doi.org/10.1038/32353.

6. MINSKY, M. L.; PAPERT, S. A. Perceptrons, MIT Press, Cambridge, MA. 1969.

7. Mnih, Volodymyr, et al. "Human-level control through deep reinforcement learning." nature 518.7540 (2015): 529-533. Disponível em: https://training.incf.org/sites/default/files/2023-05/Human-level%20control%20through%20deep%20reinforcement%20learning.pdf.