# 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*

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. Introduction

One of the areas of AI that can bring the most advancements to video games is ML (machine learning). With ML, it is possible to make the computer learn during the course of the games played and modify its behavior according to the situations encountered and the opponent it is facing (BOTELHO NETO, 2013). Reinforcement Learning (RL) has emerged as one of the most promising approaches in this field, especially when applied to complex and dynamic environments such as video games. Among the various challenges proposed to test RL algorithms, classic Atari games have emerged as a popular benchmark due to their visual simplicity combined with the strategic complexity needed to achieve high performance. These games provide an ideal environment to explore and compare different RL techniques, from traditional methods to more advanced approaches such as deep neural networks and planning algorithms.

This work proposes the implementation of an RL agent capable of playing a specific Atari game using a combination of modern techniques, such as Deep Q-Networks (DQN) proposed by (MNIH et al., 2015), convolutional neural network (CNN) architectures, introduced by (LECUN, BOTTOU & HAFFNER, 1998), and multi-layer perceptrons (MLP) (RUMELHART; HINTON & WILLIAMS, 1986).

The main objective of this project is to develop an RL agent that not only can play the chosen game but also demonstrate efficiency in terms of learning and adaptation, overcoming challenges such as environmental exploration, strategy generalization, and policy optimization. To achieve this, detailed experiments will be conducted, with clear performance metrics, allowing the evaluation of the effectiveness of the different approaches implemented.

## 2. Materials and Methods

The approach adopted combines the Deep Q-Networks (DQN) technique with convolutional neural network (CNN) architectures and multi-layer perceptron (MLP) architectures, making a combination and comparison between the two approaches, as well as comparing them with a mixed approach.

### 2.1 Development Environment and Tools

For the implementation of the RL agent, we used the Arcade Learning Environment (ALE) through the ale-py library, which provides a standardized interface for interacting with the game environment. The simulation environment was set up using gymnasium[atari,accept-rom-license], allowing the execution of classic Atari games with support for ROM licenses. For game rendering and visualization, we used pygame.

The main libraries and frameworks used include: PyTorch, NumPy, Stable-Baselines3, OpenCV (opencv-python and opencv-python-headless), TensorBoard. All the requirements for running the project will be listed in the 'requirements.txt' file of the repository on GitHub.

### 2.2 Arquitetura do Agente de RL

The RL agent was developed based on a Deep Q-Network (DQN) approach, utilizing a combination of convolutional neural networks (CNN) and multi-layer perceptron (MLP) networks.

The configuration of the DQN algorithm for the agent during training is as follows:

- 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,

The total number of timesteps applied to each model was 500,000. For a more in-depth study, training with more than 1,000,000 timesteps would be necessary; however, due to technical hardware limitations, such a number was not achievable.

### 2.4 Hardware Configuration

The training and evaluation of the agent were conducted in a hardware environment with the following specifications:

- CPU: Intel Core i5-825OU
- GPU: NVIDIA MX150
- RAM: 12 GB DDR4
- Storage: 1 TB SSD for storing datasets and models.

## 3. Results and Discussion

### 3.1 DQN using CNN as the Policy

During the development of the first agent, we used two reinforcement learning algorithms together: DQN (Deep Q-Learning Network) and CNN (Convolutional Neural Network). The CNN was used to "see" the game, while the DQN was used to interact and learn how to play the game.

After the model development, it was possible to notice a regular decrease in the loss rate during the first 10 percent of the training, but after that, the loss rate started to increase in a quite irregular manner, indicating that the agent was not learning efficiently.

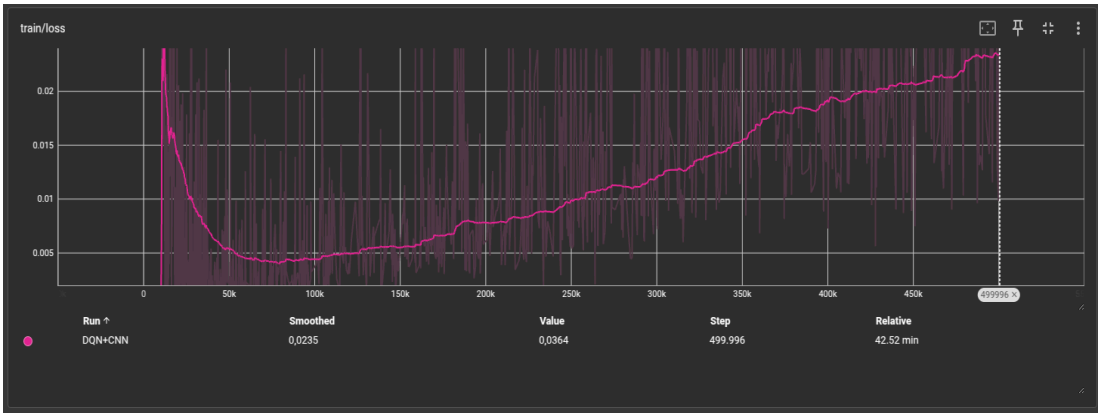


*Figure 1: Loss Graph during Training. Source: Author*

Possible causes for this issue include the values of the parameters used or overfitting, which can lead the agent to explore fewer new possibilities after reaching a certain level of learning.

### 3.1 DQN using MLP as the Policy

During this experiment, the MLP policy was used with the game's RAM memory. The architecture consists of an input size of 1x128 representing the game's memory, three hidden layers of 256 neurons, and one output layer with 4 outputs representing the possible actions. During the tests, the agent tried to maximize the reward by letting the first ball escape and not pressing the shoot button to continue the game, thus surviving for a longer period.

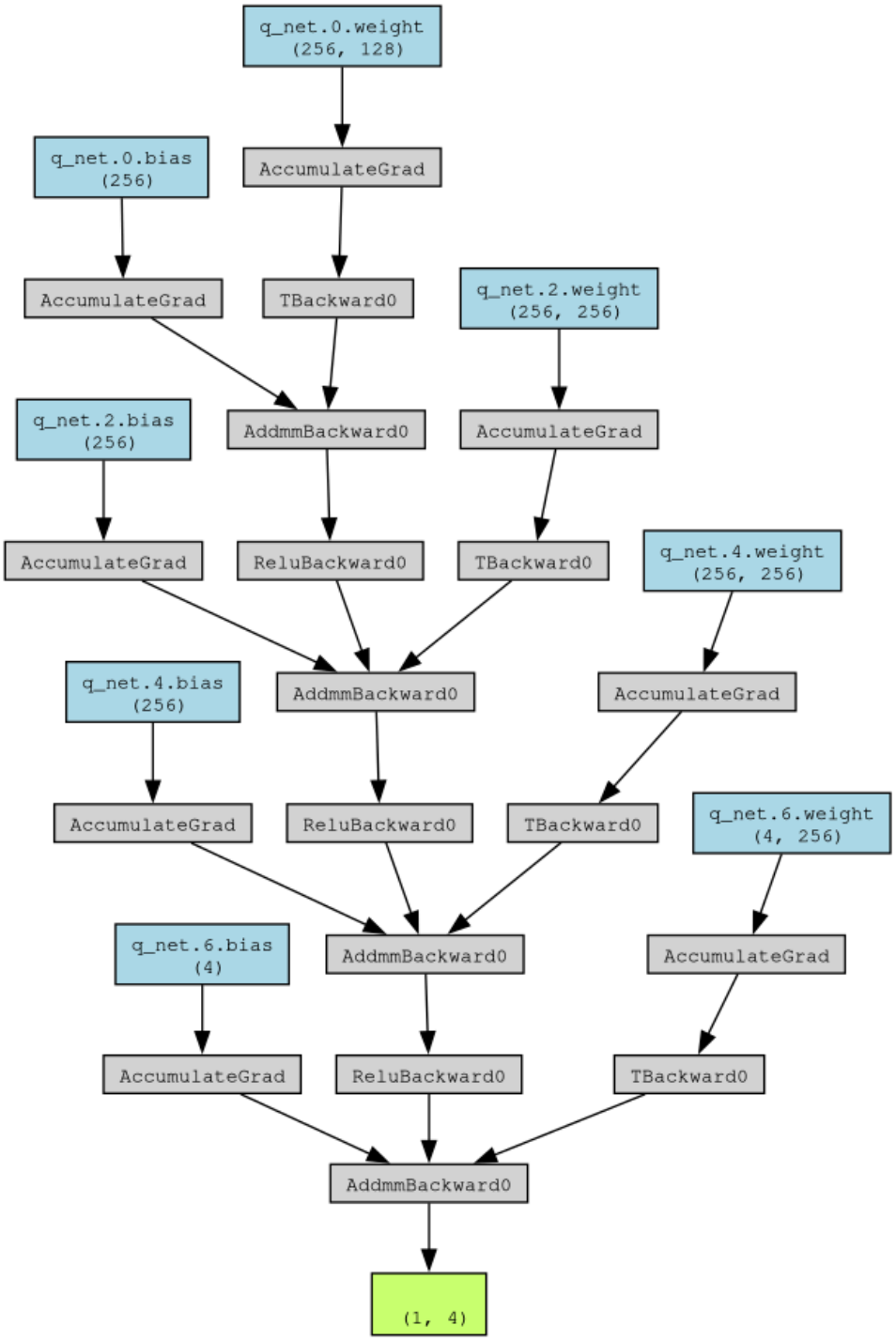


*Figure 2: Model Architecture. Source: Author*

The training of the model was extremely efficient, even when using the CPU, as it took no more than 20 minutes to train the model. However, its performance was disappointing. The model couldn't even score 5 points using all of its lives. I tried everything: exponentially increasing the reward based on the time it survived, increasing the size and number of the layers, but still, the model couldn’t survive for more than 250 frames.

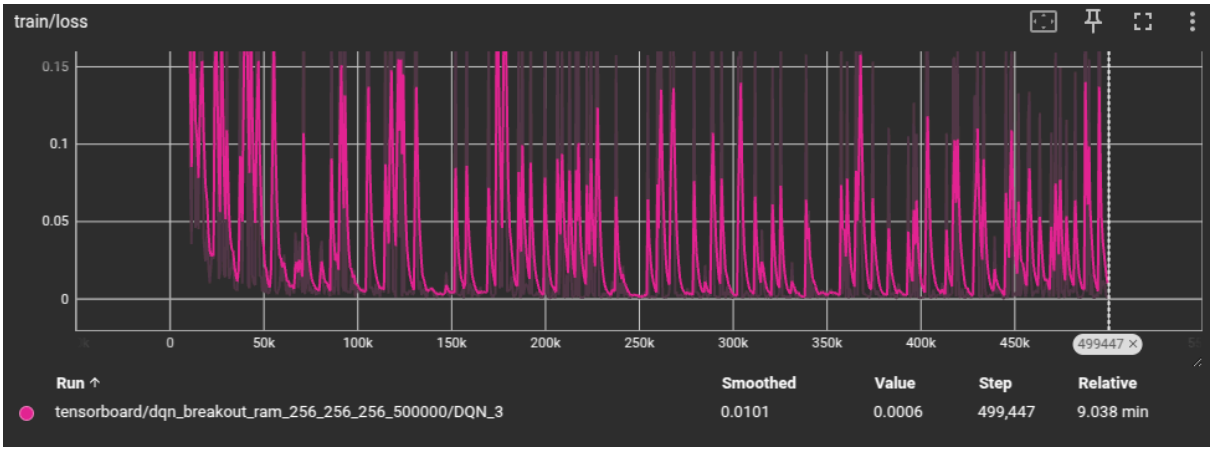


*Figure 3: Loss Graph during Training. Source: Author*

### 3.1 DQN using a Customized Policy

During this experiment, a customized policy combining the well-known CNN and MLP policies into a single policy was developed and used. The main goal was to discover how the integration of both policies could influence the model's performance. Additionally, the objective was to evaluate whether the feature extraction performed by the CNN could complement and/or optimize the learning process of the MLP, which operates using vector inputs.

The customized policy developed attempts to combine the advantages of convolutional neural networks and multi-layer perceptron networks into a single architecture. In this implementation, the CNN is responsible for extracting spatial features from the pixels on the screen, while the MLP interprets the extracted features to make decisions. The goal was to enable the model to take advantage of the CNN's ability to process the input frames, extracting spatial information, while the MLP is responsible for mapping the states to actions.

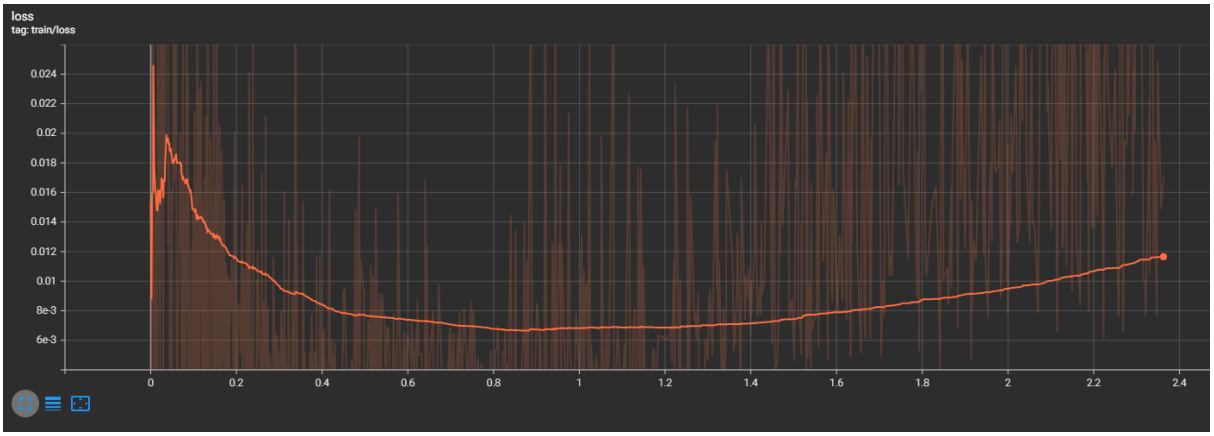


*Figure 4: Loss Graph during Training. Source: Author*

In Figure 1, it is possible to notice that the loss graph continues to decrease until about halfway through the training, after which it starts to rise. A possible cause for this could be overfitting

and instability in the model's learning process, where the agent becomes overly adjusted to the training data, thus losing its ability to generalize to unseen data. Additionally, another cause could be the model's parametrization. However, since the primary goal was to compare the performance of the models based on their policies, the parametrization across the three approaches (CNN, MLP, and the customized policy) remained the same. This includes key parameters like the experience buffer size, batch, learning rate, and other hyperparameters. Therefore, the variation in performance can mainly be attributed to the differences in the policies used.

Another point to consider is the more complex architecture. The combination of policies increases the number of parameters in the model, leading to a higher computational cost. For simpler environments where the MLP policy would already be sufficient, adding another neural network like the CNN does not justify the additional cost. A possible solution to mitigate this issue with the model would be the implementation of regularization techniques within the customized policy.

*Table 1: Comparison of the Algorithm with Different Neural Networks. Source: Author*

| | **Algorithm + Neural Network** | | |
|---|---|---|---|
| **Metric** | **DQN + CNN** | **DQN + MLP** | **DQN + CNN + MLP** |
| **Average Loss** | 0.0196 | 0.01 | 0.009 |
| **Maximum Loss** | 0.02161 | 0.05 | 0.024 |
| **Average Reward** | 18.9791 | 2.0 | 14.44 |
| **Maximum Reward** | 19.64 | 2.2 | 17.92 |
| **Success Rate** | 764.82 | 183.76 | 639.52 |

## 4. Conclusion

This study compared the performance of the Deep Q-Network (DQN) algorithm using different neural network architectures: Convolutional Neural Networks (CNNs), Multi-Layer Perceptrons (MLPs), and a combination of both (CNN+MLP). The results showed that the choice of architecture significantly impacts the learning efficiency and the agent's final performance.

The CNN-only approach proved to be the most efficient, with an average reward of 18.9, while the MLP-only approach was the worst among all, with an average reward of 2.0. The CNN+MLP combination offered an average performance, achieving an average reward of 14.44.

Given the results of the studies conducted in this research, the ideal architecture for this operational environment is the DQN algorithm with a CNN neural network. Attempts to add MLP to this agent or even train an agent with only MLP produced negative results for the study.

Among the reasons for this outcome, one that stands out is the difficulty of the MLP in identifying spatial patterns, as well as its struggle to handle high-dimensional inputs, such as in the case of Breakout (the Atari game chosen as the study environment), with 100,800 inputs, considering the image size with 3 color channels. Lastly, another factor that may have influenced the poor performance of the MLP algorithm is the lack of explicit temporal memory for the agent to infer the direction of the ball during the game (MINSKY and PAPERT, 1969).

## 5. References


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.