SCIENTIA PLENA
www.scientiaplena.org.br
VOL. X, NUM. X 202X
doi: 10.14808/sci.plena.202X.XXXXXX

Comparative analysis in the game Pong: DQN and MCTS

Análise comparativa no jogo Pong: DQN e MCTS

E. C. Rodrigues[1]; F. P. S. Sá[1]; J. V. Santos[1] L. A. Viana[1]; S. A. Araujo[1]

[1]*Departamento de Computação, Universidade Federal de Sergipe, 49100-000, São Cristóvão-Sergipe, Brasil*

sauloalmeida@academico.ufs.br
(Received on February 24, 2025; accepted on month day of year)

This study aimed to develop Artificial Intelligence (AI) agents capable of playing the classic Atari Pong game, to compete against the machine and among themselves in order to determine which approach achieved better results. The agents created were based on a Deep Q-Learning (DQN) model and another approach that involves the DQN model integrated with a Monte Carlo Tree Search (MCTS) algorithm. The results obtained demonstrate the promising use of Reinforcement Learning (RL) algorithms to tackle complex situations, such as Atari 2600 games. It is concluded that these approaches can not only be useful but also provide valuable insights for future research and applications in other areas of Artificial Intelligence.
Keywords: AI, DQN, MCTS.

Este estudo buscou desenvolver agentes de Inteligência Artificial (IA) capazes de jogar o clássico jogo do Atari Pong, para competir com a máquina e entre os mesmos como forma de verificar qual abordagem obteve melhores resultados. Os agentes criados foram baseados em um modelo Deep Q-Learning (DQN) e um outra abordagem que envolve o modelo DQN integrado a um algoritmo de Monte Carlo Tree Search (MCTS). Os resultados obtidos comprovam a promissora utilização de algoritmos de Aprendizagem por Reforço (RL) para enfrentar situações complexas, como os jogos Atari 2600. Conclui-se que essas abordagens não apenas podem ser úteis, como também fornecem insights valiosos para futuras pesquisas e aplicações em outras áreas da Inteligência Artificial.
Palavras-chave: IA, DQN, MCTS.

1. INTRODUCTION

Atari games have served as a platform for experimenting with algorithms capable of emulating human behavior, achieving high accuracy in their objectives. For this work, the game *Pong* was selected using the Arcade Learning Environment (ALE) framework. Currently, there are various approaches to addressing this type of problem, ranging from strategies that teach the agent to act like a human to algorithms that enable the agent to learn a policy based on rewards.

The use of reinforcement learning algorithms has yielded results superior to those achieved through supervised learning with humans. This highlights the potential of more precise techniques that do not require a knowledge base but instead rely on a policy of positive and negative rewards [1].

According to Haddad et al. (2021) [2], the use of a DQN-based agent outperformed the results obtained by an agent using only the Q-Learning algorithm. Furthermore, it achieved these results ten times faster in Unity platform games. This study compared the victory rates of both models and found that Q-Learning achieved a 70% victory rate, while the DQN algorithm reached 90%.

According to Fu et al. (2016) [3], the use of MCTS in regions with high rewards helps the agent make more promising decisions, leading to parameter adjustments that favor exploitation over exploration.

Thus, this work aims to perform a comparative analysis between the algorithms of an agent solely based on DQN with a training dataset twice as large as that of a second agent trained with

a DQN integrated with MCTS. Both approaches are applied to the game *Pong* to determine which agent achieves a higher win rate, as well as faster and more robust convergence.

2. METHODOLOGY

The study was conducted on Google's Colab platform using cloud computing resources. The code was developed in Python, utilizing libraries such as Gymnasium, Torch, Numpy, and Stable Baselines.

Two agents were trained: the pure DQN and the DQN with MCTS (to enhance action selection). The pure DQN was trained with 2 million (2M) steps, while the DQN with MCTS used half as many, only 1 million (1M). Additionally, each DQN model was created with the CnnPolicy and a buffer size of 100,000.

After training, both models were saved to be later evaluated in 100 matches. Each match (episode) is won by the first player to reach 7 points. At the end of the evaluation, the win rate is calculated for each agent.

2.1 Frameworks and Libraries

2.1.1 ALE

For the development and testing of reinforcement learning agents, the ALE framework was used. This framework provides support for Atari 2600 games, automatic extraction of scores and end-of-game signals, as well as visualization tools [4].

2.1.2 Gymnasium

For the evaluation and testing of the developed reinforcement learning algorithms, the Gymnasium library was employed. This library facilitates the creation of benchmarks for analyzing the performance of algorithms [5].

2.1.3 PyTorch

The PyTorch framework was used for deep learning, offering a comprehensive set of tools for building and training neural networks [6].

2.1.3 *Stable Baselines3*

Stable Baselines3 (SB3) is a reinforcement learning framework in Python that provides efficient and robust implementations of popular algorithms such as DQN, PPO, and A2C [7].

2.2 DQN Architecture

2.2.1 Description

The architecture of the DQN used follows the classic model introduced by DeepMind, consisting of three convolutional layers for feature extraction from the game frames (32 filters

of 8×8, 64 filters of 4×4, and 64 filters of 3×3, all with ReLU activation). These are followed by a fully connected layer with 512 neurons and an output layer corresponding to the number of available actions.

Training uses a Replay Buffer to store experiences and a Target Network for greater learning stability. The loss function is based on the Bellman Equation, and optimization is performed using the Adam Optimizer with a learning rate of 1e-4.

2.2.2 Pseudocode

```
CLASS DQN:

conv    ' Sequential object for convolutional layers

fc      ' Sequential object for fully connected layers

FUNCTION DQN(input_shape, n_actions):
    ' Inicializar camadas convolucionais
    conv = SEQUENTIAL(
        CONV2D(input_channels=input_shape[0], out_channels=32,
kernel_size=8, stride=4),
        RELU(),
        CONV2D(input_channels=32, out_channels=64, kernel_size=4,
stride=2),
        RELU(),
        CONV2D(input_channels=64, out_channels=64, kernel_size=3,
stride=1),
        RELU()
    )

    ' Calculate the output size after convolutional layers
    conv_out_size = GET_CONV_OUT(input_shape)

    ' Initialize fully connected layers
    fc = SEQUENTIAL(
        LINEAR(input_size=conv_out_size, output_size=512),
        RELU(),
        LINEAR(input_size=512, output_size=n_actions)
    )

END FUNCTION

' Forward function to process input data

FUNCTION FORWARD(x):

    ' Pass the input through the convolutional layers

    conv_out = RESHAPE(conv(x), batch_size=x.size[0], -1)  ' Achatar a
saída

    ' Pass through the fully connected layers and return the result
```

```
    RETURN fc(conv_out)

END FUNCTION


END CLASS
```

2.3 DQN + MCTS Architecture

2.3.1 Description

DQN with MCTS retains the same neural network architecture as the pure DQN but enhances action selection by incorporating a Monte Carlo Tree Search (MCTS). In this model, the DQN calculates the Q-values for the current state, while the MCTS expands a search tree by simulating multiple action sequences, using the UCT (Upper Confidence Bound for Trees) strategy to balance exploration and exploitation. After several simulations, the node values are updated based on the Q-values and rewards obtained, allowing the action with the highest expected value to be selected. This combination makes the agent more efficient in evaluating future decisions before acting.

2.3.2 Pseudocode

```
CLASS DQN_MCTS:

' Select an action using MCTS

FUNCTION SELECT_ACTION_WITH_MCTS(env, num_simulations):

    RETURN MCTS(env, self, num_simulations).CHOOSE_BEST_ACTION()

END FUNCTION


END CLASS

CLASS NO_MCTS:

' Initialize a new node

FUNCTION NO_MCTS(state, parent, action):

    INITIALIZE_STATE(state, parent, action)

END FUNCTION

' Check if the node is fully expanded

FUNCTION IS_EXPANDED(action_space):

    RETURN CHILD_NUM == action_space

END FUNCTION

' Select the best child based on UCT
```

```
FUNCTION BEST_CHILD():

    RETURN CHILD_WITH_BEST_BALANCE_OF_VALUE_AND_EXPLORATION()

END FUNCTION


END CLASS

FUNCTION MCTS(env, agent, simulations):
RETURN root.BEST_CHILD()

END FUNCTION
```

2.4 Hardware environment

2.4.1 CPU

2 Intel Xeon vCPUs @ 2.20GHz, which are reasonably suitable for many data analysis and machine learning tasks.

2.4.2 GPU

Nvidia Tesla P100, offering up to 4.7 teraflops of double-precision and 9.3 teraflops of single-precision, ensuring agent training and the execution of multiple simulations in parallel.

2.4.3 RAM

To ensure the performance of the operations, a virtual machine with 12.7GB of RAM was used.

3. EXPERIMENTS

This section presents how the experiments were conducted to evaluate the performance of the reinforcement learning algorithms, specifically the DQN and MCTS agents, in the Pong game environment. The experiments aim to assess the effectiveness of the proposed algorithms in learning and optimizing gameplay strategies, as well as to compare their performance in different scenarios.

To ensure the integrity and validity of the experiments, standardized simulation environment configurations and specific evaluation metrics were used, ensuring that both experiments were subjected to identical conditions.

3.1 Test Configuration

The experiments were conducted using the Pong environment from ALE, configured through the Gymnasium library. The Stable Baselines3 library was used for the implementation of the DQN agent. The experiments were executed on a machine with 2 Intel Xeon vCPUs @

2.20GHz, an Nvidia Tesla P100 GPU, 12.7GB of RAM, and the Ubuntu 20.04 LTS operating system.

3.2 Performance Metrics

To evaluate the performance of the agents, the win rate was used as the analysis criterion, defined as the proportion of episodes in which the agent won the game. This metric is crucial for data collection and making statistical inferences.

It is worth noting that 100 matches/episodes were conducted for each agent.

3.3 Comparison of Approaches

To compare the effectiveness of each agent, 3 scenarios were proposed:

- C01: DQN Agent (2M) vs Machine
- C02: DQN Agent (1M) + MCTS vs Machine
- C03: Total Score of DQN Agent (2M) vs DQN Agent (1M) + MCTS

Each scenario aims to provide insights into the effectiveness and limitations of each approach, as well as identify opportunities for future improvements and optimizations in the training of reinforcement learning agents. Each scenario aims to provide insights into the effectiveness and limitations of each approach, as well as identify opportunities for future improvements and optimizations in the training of reinforcement learning agents.

4. RESULTS AND DISCUSSION

As seen in Figure 1, the experiments conducted in Scenario C01, where a DQN agent trained for 2 million episodes faced a machine-controlled opponent, demonstrate that although the DQN agent achieved a reasonable performance, it was outperformed by the machine-controlled opponent.

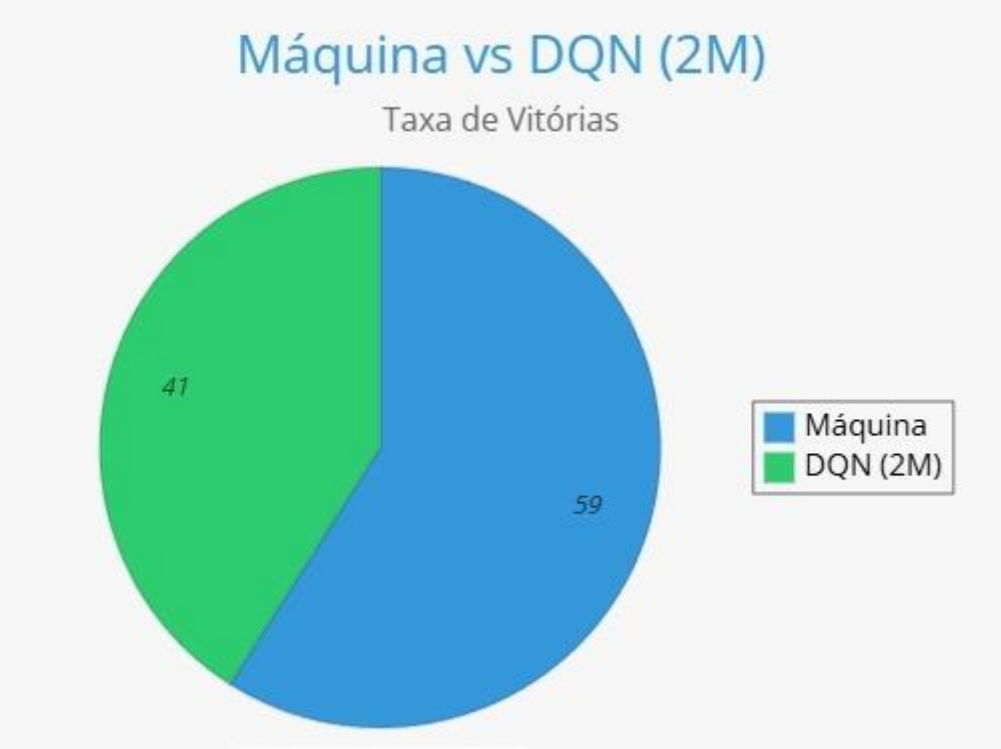

Figure 1: C01: DQN Agent (2M) vs Machine. Source: Own Authorship

This shows that several factors were crucial in achieving these results, such as an insufficient training time for the agent to learn effectively and achieve more balanced outcomes.

Meanwhile, Figure 2 presents the experiments conducted in Scenario C02, where a hybrid DQN agent trained for 2 million episodes, combined with MCTS, faced a machine-controlled opponent. The results showed that the hybrid DQN + MCTS agent proved to be extremely ineffective, performing worse than the pure DQN implementation.

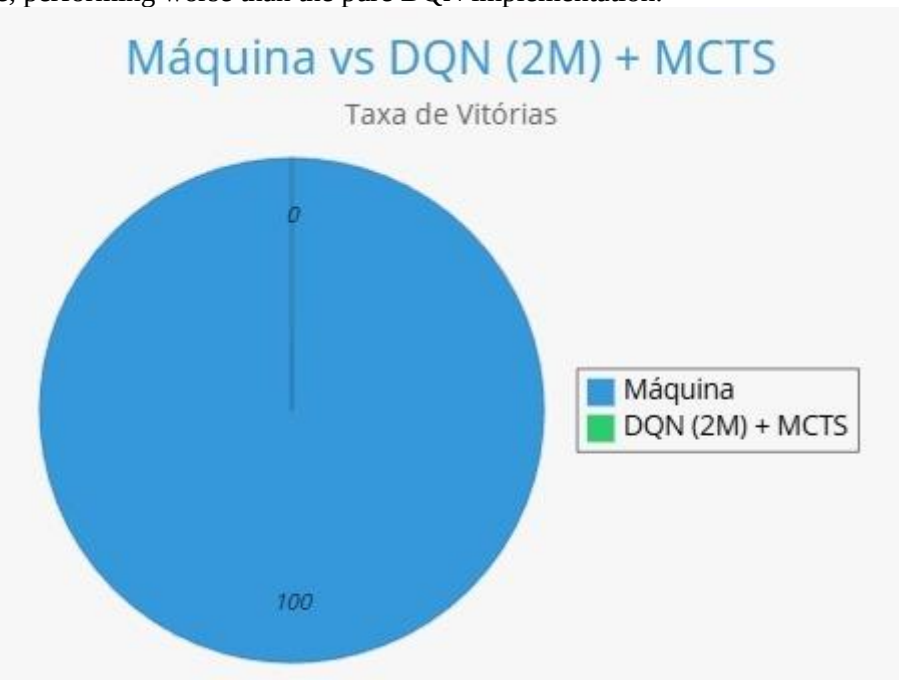

Figure 2: C02: DQN Agent (2M) + MCTS vs Machine. Source: Own Authorship

The combination of the DQN and MCTS algorithms may not have been fully effective in this specific context. Some plausible reasons for these poor results could be that MCTS performs deep searches and simulations to predict the outcomes of future actions, causing a conflict in how these two approaches handle exploration and exploitation. This conflict may have resulted in inefficiencies in the learning process.

Meanwhile, Figure 3 presents a "point-by-point" comparison of the agents in their matches against the machine-controlled opponent. As expected, the agent based solely on DQN scored 40 times more than the combined DQN + MCTS agent.

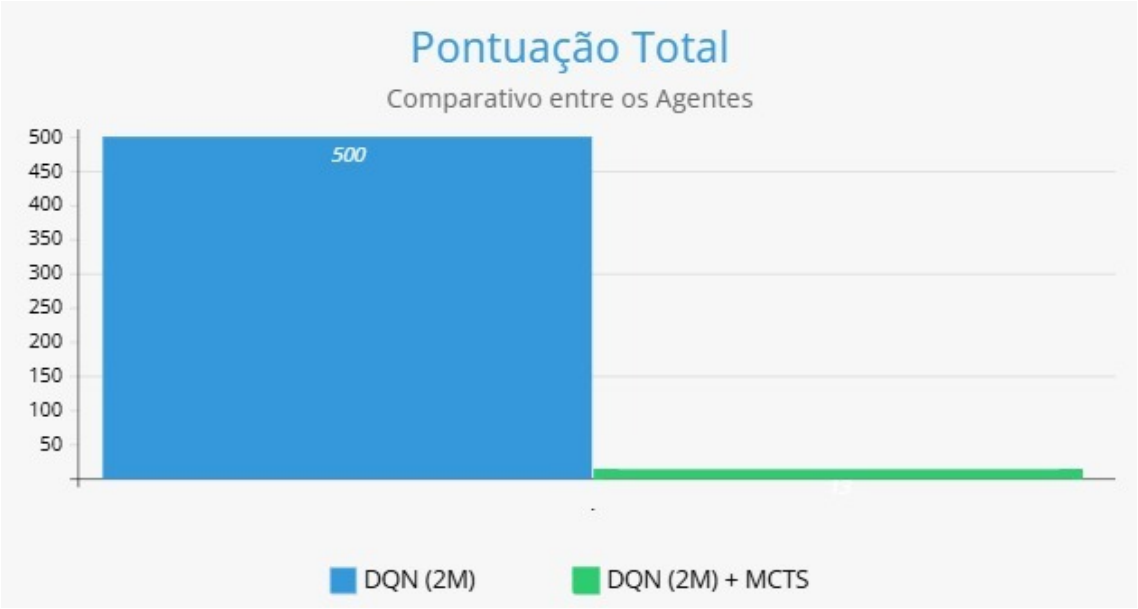

Figure 3: C03: Total Score Comparison. Source: Own Authorship

The results highlight the disparity in accuracy between the algorithms of the DQN and DQN MCTS agents, particularly the lack of calibration of the latter agent, which reflects how ineffective the integration of MCTS into the agent became. Although MCTS is a robust method for decision-making in search trees, its integration with DQN may have encountered specific challenges, such as policy conflicts and the latency that the MCTS algorithm may have caused in making quick decisions, which is crucial in real-time games. Finally, the environment modeling combined with an unstable policy may not have been optimal for this strategy, as it could lead to hasty decisions and generate a suboptimal policy.

5. CONCLUSION

The incorporation of MCTS into the decision-making process of the DQN agent resulted in a significant performance decline, as evidenced by the inferiority of DQN+MCTS (0 wins) compared to Pure DQN (41 wins) in 100 episodes, respectively. The hybrid approach may still be promising, suggesting that the integration of search methods can provide a strategic advantage, which is subject to several factors, such as environment modeling, computational power, and the dynamism of the environment. It is worth noting that the agent with MCTS was performing only one simulation per state, meaning that increasing the number of simulations could lead to better performance.

In summary, it can be inferred that insufficient training results in ineffective agents when analyzing the data statistically, even if the agent performed well in a few exceptional episodes. Only with adequate training can the agent converge to a good policy, allowing it to develop robust and consistent strategies.

Therefore, future work should consider implementing training with at least 10,000,000 steps, which would lead the agent to a more robust policy. Depending on the results obtained, empirical adjustments to the hyperparameters should be made to ensure the agent does not suffer from overfitting.

Furthermore, with a more effective agent, implementing MCTS would be an interesting approach to verify whether its performance would worsen, improve, or remain largely unaffected, as it would be applied to a more consistent agent.

6. BIBLIOGRAPHICAL REFERENCES

1. Silver, David, et al. "Mastering the game of go without human knowledge." nature 550.7676 (2017): 354-359. https://tinyurl.com/mastering-go
2. HADDAD, Gabriel Prudencio; JULIA, Rita Maria Silva; FARIA, Matheus Prado Prandini. Técnicas de Aprendizado por Reforço Aplicadas em Jogos Eletrônicos na Plataforma Geral do Unity. In: ENCONTRO NACIONAL DE INTELIGÊNCIA ARTIFICIAL E COMPUTACIONAL (ENIAC), 18. , 2021, Evento Online. Anais [...]. Porto Alegre: Sociedade Brasileira de Computação, 2021 . p. 571-582. ISSN 2763-9061. DOI: https://doi.org/10.5753/eniac.2021.18285.
3. Fu, M. C. 2016. “AlphaGo and Monte Carlo Tree Search: The Simulation Optimization Perspective”. In Proceedings of the 2016 Winter Simulation Conference, edited by T. M. K. Roeder et al., 659–670. Piscataway, NJ: IEEE.
4. Arcade Learning Environment (ALE). Farama Foundation. Disponível em: https://ale.farama.org/. Acesso em: 24 fev. 2025.
5. Gymnasium Documentation. Farama Foundation. Disponível em: https://gymnasium.farama.org/. Acesso em: 24 fev. 2025.
6. PyTorch Documentation. PyTorch. Disponível em: https://pytorch.org/docs/stable/index.html. Acesso em: 24 fev. 2025.
7. Stable-Baselines3 Documentation. Stable-Baselines3. Disponível em: https://stable-baselines3.readthedocs.io/en/master/. Acesso em: 24 fev. 2025.