Reinforcement learning applied to Frogger

Aprendizagem por reforço aplicada ao Frogger

A. N. Souza[1]; I. L. L. de Oliveira[2]; L. M. A. Santos[3]; M. V. R. Guimarães[4]; P. M. de J. Costa[5]

Departamento de Computação/CCET, UFS, 49100-000, São Cristóvão-Sergipe, Brazil

[1]*xandnuness@academico.ufs.br*

[2]*isaacl@academico.ufs.br*

[3]*yazonira@academico.ufs.br*

[4]*mvrguimaraes@academico.ufs.br*

[5]*periclesmaikon@academico.ufs.br*

This study aims to explore the application of Reinforcement Learning techniques in an agent trained to play the Frogger game from Atari. This choice was based on the game's simple dynamics, which nonetheless presents appropriate challenges for learning and evaluating reinforcement learning algorithms. To achieve this, the model was trained using the *Stable Baselines 3* library, which implements the Deep Q-Network algorithm, allowing the agent to learn how to maximize its rewards through interaction with the environment. The techniques chosen for evaluating the agent were Convolutional Neural Networks and Multi-Layer Perceptrons.

Keywords: CNN, MLP, Frogger.

Este estudo tem como objetivo explorar a aplicação de técnicas de Aprendizado por Reforço em um agente treinado para jogar o jogo Frogger do Atari, essa escolha foi baseada na dinâmica simples do jogo, mas que apresenta desafios adequados para o aprendizado e avaliação de algoritmos de aprendizado por reforço. Para isso o modelo foi treinado utilizando a biblioteca *Stable Baselines 3*, que implementa o algoritmo Deep Q-Network, dessa forma o agente foi capaz de aprender a maximizar suas recompensas ao interagir com o ambiente. As técnicas escolhidas para a avaliação do agente foram redes neurais convolucionais e perceptrons multicamadas.

Palavras-chave: CNN, MLP, Frogger.

1. INTRODUCTION

Reinforcement Learning (RL) is a field of artificial intelligence where agents are placed in environments where they learn to make decisions based on rewards accumulated over time. RL applications have been gaining prominence due to the success of models developed by DeepMind. According to Silver et al. (2017), AlphaGo was the first model to defeat the world champion in the game Go.

In this study, an agent was trained to play Frogger, a classic Atari platform game. Convolutional Neural Networks (CNN) and Multi-Layer Perceptrons (MLP) were used to assess the performance of both technologies, aiming to understand the advantages each approach has in the dynamic scenarios of Frogger.

The chosen game has a simple yet ideal dynamic for reinforcement learning; Bellemare et al. (2013) highlight the importance of Atari games for evaluating RL agents, as they are effective in measuring their performance due to a range of scenarios with varying levels of complexity and required skills. The objective of Frogger is for the agent to cross a busy road or river filled with obstacles without being hit. It is a challenging enough environment for the agent to learn decision-making, requiring it to deal with time, obstacles, and interactions with the environment.

As stated by LeCun et al. (2015), CNNs are particularly effective at handling visual data, such as images and videos, because they can identify patterns through convolutional filters. This makes them ideal for the game, as the agent can receive frames from the game and make decisions based on learned obstacle patterns.

MLPs are traditional networks with multiple fully connected layers of neurons. They are effective in numerous tasks but lack the ability to interpret images and videos. This study aims to compare how a simpler model handles scenarios with visual and dynamic input.

2. MATERIALS AND METHODS

Experimental Environment

The environment used was Frogger-v5 from the Atari platform, registered via Gymnasium and ALE (Arcade Learning Environment). To capture the temporality of actions, the VecFrameStack preprocessing was applied, stacking four consecutive game frames. The environment was vectorized to enable efficient training, with a single parallel environment (*n_envs*=1).

The hardware environment used in the training of the model was:

- CPU: Ryzen 7 5800X;
- GPU: NVIDIA GeForce RTX 4070;
- RAM: 32GB.

Reward Modification

A custom wrapper, *PunicaoFrogger*, was implemented to adjust the reward function. When an episode ends (e.g., the agent is hit), a penalty of 1 is applied to the original reward, encouraging the agent to avoid premature failures.

Model Architectures

Two neural network architectures were tested using the DQN (Deep Q-Network) algorithm from Stable Baselines3:

1. *CNN* (Convolutional Neural Network): Implemented through the CnnPolicy, designed to process images directly. The architecture is based on the nature CNN and includes 3 convolutional layers for extracting visual features (e.g., obstacle patterns), one padding layer, and lastly, a linearization layer (Mazyavkina et al., 2021), shown in the image below.

Layer	input	output	kernel	stride
Conv-1	4	32	8	4
Conv-2	32	64	4	2
Conv-3	64	64	3	1
Padding	–	$121 * 64$	–	–
Linear	$121 * 64$	512	–	–

Figure 1: Description of the nature CNN

2. *MLP* (Multi-Layer Perceptron): Based on the MlpPolicy, with 2 fully connected layers of 64 neurons each. It assumes that observations are flattened to fit a tabular input format.

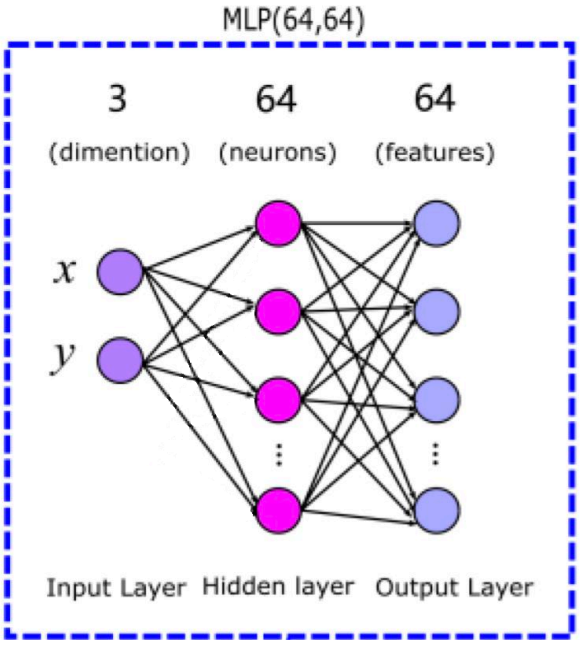

Figure 2: Description of the MlpPolicy

Training Hyperparameters
Both models shared the following parameters:

- **Learning Rate** (*learning_rate*): 1×10^{-4}.
- **Batch Size** (*batch_size*): 64.
- **Replay Buffer** (*buffer_size*): 125.000 experiences.
- **Initial Steps** (*learning_starts*): 20.000 (data collection phase before training).
- **Acceleration Device**: *GPU* (*CUDA*), when available, via PyTorch.

Training was conducted for 1,000,000 timesteps (20 iterations of 50,000 timesteps each), with checkpoints saved every 50,000 timesteps in the *models/CNN* or *models/MLP* directory.

Performance Evaluation

After training, the model was evaluated over 10 independent episodes using the *evaluate_policy* function from Stable Baselines3. The calculated metrics include:

- **Average Reward:** Primary indicator of success.
- **Average Length:** Shows the survival rate of the agent.
- **Standard Deviation:** Measure of the agent's consistency.
- **Real-Time Rendering:** Visualization of the agent's behavior via env.render("human").

Tools and Libraries

- **Stable Baselines3:** For *DQN* implementation and evaluation.
- **Gymnasium & ALE-py:** For environment creation and management.
- **PyTorch:** Backend for GPU execution.
- **TensorBoard:** Monitoring of metrics (*ep_rew_mean, ep_len_mean*).

Reproducibility

All code, hyperparameters, and configurations are publicly available, ensuring the reproducibility of this study. The training scripts (*frogger_mlp.py, frogger_cnn.py*) and evaluation script (*gameplay.py*) were developed in Python 3.10, with dependencies documented in *README.md.*

Ethical Aspects

As this is a computational experiment with no involvement of humans or animals, no ethical committee approval was required.

3. RESULTS AND DISCUSSION

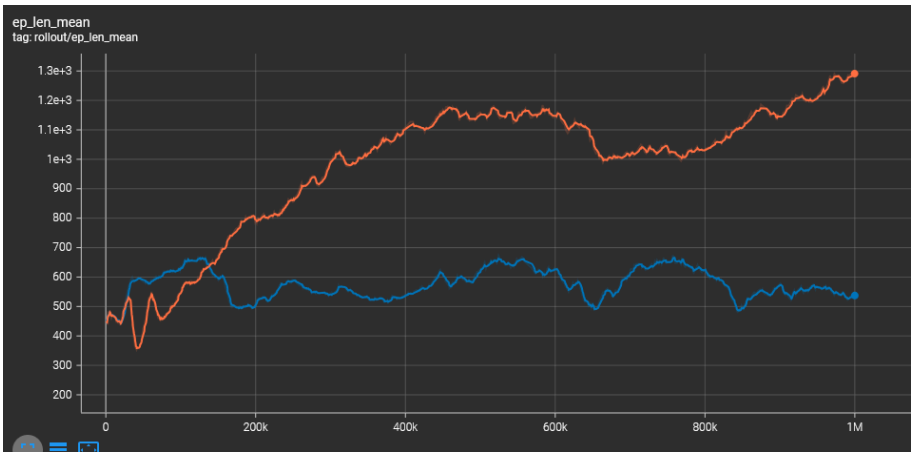

Figure 3: Average survival per episode throughout training.

The findings from the training and evaluation phases of the Convolutional Neural Network (CNN) and Multi-Layer Perceptron (MLP) models for the game Frogger revealed notable performance differences between the two architectures. In the early stages, the MLP model exhibited a higher average survival rate per episode compared to the CNN, indicating that MLP was initially more effective in avoiding early failures (see Figure 3). However, as training progressed, the CNN significantly outperformed the MLP, demonstrating a more robust and adaptable learning capability in dynamic scenarios.

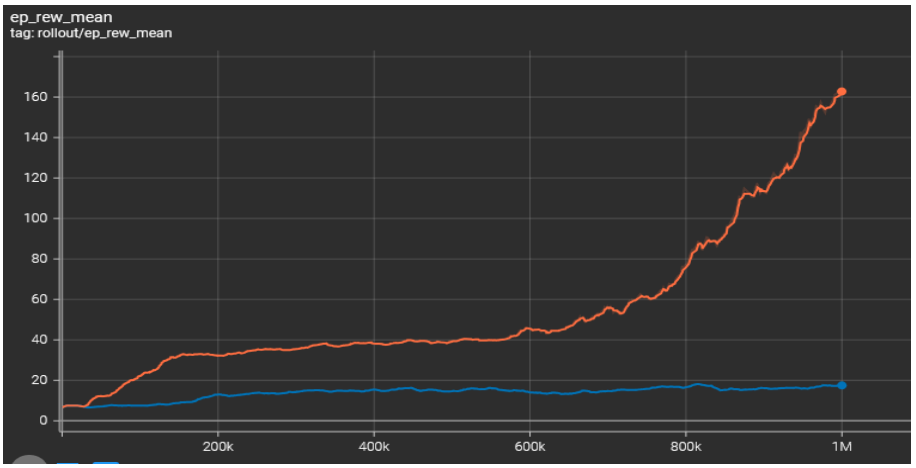

Figure 4: Average reward per episode throughout training.

Regarding the average rewards per episode, both models showed equivalent performance at the beginning of training, with significant learning beginning at around 20,000 timesteps (see Figure 4). However, after this point, the CNN model outperformed the MLP, achieving considerably higher rewards. While the CNN continued to improve its performance, the MLP stagnated, with average rewards remaining below 20 points after 700,000 timesteps. This suggests that the MLP reached a performance saturation point, while the CNN showed potential for continued progress.

A significant highlight occurred around 700,000 timesteps, when the CNN demonstrated a major breakthrough in learning, substantially increasing its average rewards (see Figure 4). This phenomenon can be explained by the CNN's ability to identify complex visual patterns in the environment, which is crucial in games like Frogger, where real-time visual interpretation and obstacle detection are essential. In contrast, the MLP, which lacks the same image processing capability, proved ineffective in dealing with the complexity of the environment, leading to inferior and stagnant performance.

The comparative analysis between the two models supports the idea that CNNs are more proficient in Atari game environments like Frogger, due to their inherent ability to process visual data and recognize spatial patterns. Since both the MLP and CNN were exposed to identical training conditions (same environment, parameters, and rewards), the superiority of the CNN can be attributed to its architecture, specifically designed to efficiently handle visual inputs. On the other hand, the MLP, while competent in tasks with tabular data, showed lower adaptability in situations requiring visual interpretation.

These results align with theoretical predictions, as noted by LeCun et al. (2015), who highlight the superiority of CNNs in tasks involving visual data such as images and videos. The ability of the CNN to continue improving its performance over time, while the MLP remained stagnant, suggests that the convolutional architecture has greater potential for applications in dynamic and visual environments, such as Atari games.

In summary, the findings indicate that while the MLP demonstrated acceptable performance initially, the CNN surpassed it by a wide margin in terms of average rewards and continuous learning ability. This underscores the importance of selecting the appropriate architecture for each situation, especially in contexts requiring image processing and adaptation to changing environments.

4. CONCLUSION

The MLP exhibited a fast initial learning curve, as its average survival per episode surpassed that of the CNN in the early stages. However, its ability to adapt to dynamic scenarios was limited.

The CNN proved to be much more effective in the long run, with a significantly higher average survival per episode than the MLP. However, this came at the cost of a longer training time and increased computational resource requirements.

This difference can be further reinforced by analyzing the average reward per episode, where the CNN exceeded 160 points after 1,000,000 interactions, while the MLP, after the same 1,000,000 episodes, failed to surpass 20 points.

Since our result is far from the state-of-the-art, for future significant improvements, it would be essential to submit the model to more training rounds, after the results analysis we kept training the CNN only for another 2,000,000 timesteps, which brought it to reach 300 points (see figure 5), a reasonable score for an intermediate player. It is also important to note that other Stable Baselines 3 models have prioritized experience replay methods applied to DQN, which make them outperform the traditional DQN in results, achieving a new state-of-the-art in 41 out of 49 Atari games according to Schaul et al. (2015)

Therefore, the choice between MLP and CNN should consider the trade-off between learning efficiency, computational cost, and task complexity.

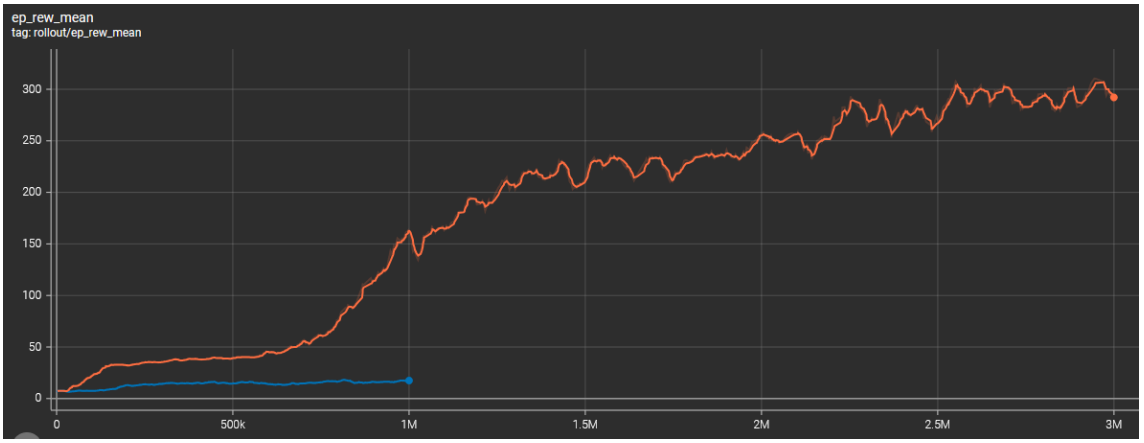

Figure 5: Average reward per episode of the continuation of the training.

5. ACKNOWLEDGEMENTS

We would like to thank the developers and maintainers of *Stable Baselines 3* for providing fundamental tools for the development of this experiment. We also express our gratitude to Professor Hendrik Texeira Macedo for his support and guidance throughout this work, significantly contributing to our understanding and improvement in the field of artificial intelligence.

6. REFERÊNCIAS BIBLIOGRÁFICAS

1. Silver D, Schrittwieser J, Simonyan K, et al. Mastering the game of Go without human knowledge. Nature. 2017 Oct;550(7676):354-359, doi:10.1038/nature24270.

2. Mnih V, Kavukcuoglu K, et al. Playing Atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602. 2013.

3. Guo X, Singh S, et al. Deep learning for real-time Atari game play using offline Monte Carlo tree search planning. Adv Neural Inf Process Syst. 2014;27.

4. Mnih V, Kavukcuoglu K, et al. Human-level control through deep reinforcement learning. Nature. 2015 Feb;518(7540):529-533, doi:10.1038/nature14236.

5. LeCun Y, Bengio Y, Hinton G, et al. Deep learning. Nature. 2015 May;521(7553):436-444, doi:10.1038/nature14539.

6. Bellemare MG, Naddaf Y, Veness J, Bowling M, et al. The arcade learning environment: An evaluation platform for general agents. Proc 24th Int Conf Mach Learn (ICML). 2013;214-224.

7. Ye W, Liu S, Kurutach T, et al. (2021). Mastering atari games with limited data. Advances in neural information processing systems, 34, 25476-25488.

8. Bellemare MG, Naddaf Y, Veness J, Bowling M, et al. The arcade learning environment: An evaluation platform for general agents. Proc 24th Int Conf Mach Learn (ICML). 2013;214-224.
9. Mazyavkina, N. & Moustafa, Samir & Trofimov, Ilya & Burnaev, Evgeny. (2021). Optimizing the Neural Architecture of Reinforcement Learning Agents. 10.1007/978-3-030-80126-7_42.
10. Schaul, T., Quan, J., Antonoglou, I., & Silver, D. (2015). Prioritized experience replay. *arXiv preprint arXiv:1511.05952*.