Deep Q-Learning for Space Invaders

Finn Rehnert
University of Applied Sciences Ulm
Advanced Machine Learning Project
refinn01@thu.de

***Abstract*—todo write abstract**

I. Introduction

Reinforcement Learning (RL) has been an area of active research since many years. However, only recent advancements have really unvveiled its potential. Notebly the defeat of Lee Sedol by AlphaGo [1] and the human-level performance in Atari games [2] have shown impressive results. In RL, an agent learns to make decisions by interacting with an environment. The agent receives observations (states) from the environment, takes actions, and receives rewards based on those actions. The goal of the agent is to learn a policy that maximizes the cumulative reward over time.

Since RL was not covered in the course lectures, this project serves as an explorative study into Deep Reinforcement Learning. The goal of this project is to replicate the success of the 2013 DeepMind paper on the specific environment of Space Invaders [2].

A. Scenario

Space Invaders is a classic arcade game where the player controls a spaceship at the bottom of the screen and must shoot down waves of descending aliens while avoiding their attacks. The game ends when the player loses all lives or when the aliens reach the bottom of the screen [3]. Space Invaders was designed and developed by Tomohiro Nishikado and was released in 1978 [4].

One big challenge in RL is to define the environment, state space, action space, and reward function appropriately. The originally from OpenAI developed Gym environment [5] provides a standardized interface for various environments. Since its deprecation, a community-driven fork called Gymnasium is actively maintained. This project uses the Gymnasium implementation of the Atari 2600 Space Invaders environment, which is based on the Arcade Learning Environment (ALE) [6].

The state space for visual tasks like Space Invaders can easyly become high-dimensional. In this project, the state space is defined as raw pixel data (RGB images) of the game screen. The action space is discrete, consisting of six actions: Move Left, Move Right, Shoot, Shoot Right, Shoot Left and No-op (do nothing) [7].

The reward function is defined based on the game score. The agent receives positive rewards for shooting down aliens and negative rewards for losing lives. The specific reward structure will be detailed in Section III-C.

B. Structure of the Paper

The structure of this paper is as follows:

- Section II provides a literature review, discussing the transition from tabular learning to function approximation and the Deep Q-Network (DQN) breakthrough.
- Section III outlines the overall solution strategy, including preprocessing steps, network architecture, and training loop.
- Section IV presents the results and evaluation of the trained agent.
- Finally, Section V concludes the paper and discusses potential future work.

II. Literature Review

While RL has been studied for decades, the combination of RL with deep learning has led to significant advancements in the field [8]. This section reviews key concepts and breakthroughs that laid the foundation for Deep Q-Learning.

A. From Tabular Learning to Function Approximation

Traditional Q-Learning is based on the so called Q-table. A table that contains for each possible state that the agent can be in, a corresponding action to take. This approach works well for simple environments with limited states.

The ”Curse of Dimensionality” describes the limitation of traditional Q-Learning: it states that for higher dimensional inputs like visuals it is not possible to have a table row for each pixel combination on the screen. The combination of possible pixel values grows exponentially with the number of pixels. For example, a simple 84x84

grayscale image has $256^{(84*84)}$ possible states, which is more than the number of atoms in the observable universe which can be approximated to 10^{80} [9].

The solution to this problem is to use a function approximator, such as a Neural Network, to estimate the Q-values for each state-action pair. This allows the agent to generalize from seen states to unseen states, enabling it to handle high-dimensional inputs effectively.

B. The Deep Q-Network (DQN) Breakthrough

Discuss the Mnih et al. (2013) paper.

Highlight the two key innovations that stabilized training (which you implemented):

Experience Replay (breaking correlation between consecutive frames).

Target Networks (stabilizing the moving target).

III. Concept for problem solving

Define the Bellman Equation.

$$Q(s, a) = r + \gamma \max_{a'} Q(s', a')$$

Explain the Loss Function. You are minimizing the Mean Squared Error between the predicted Q-value and the target Q-value. Explain ϵ-greedy exploration: How the agent balances exploring random moves vs. exploiting known best moves.

A. Overall Solution Strategy

Mention the tools used: Python, PyTorch/TensorFlow, OpenAI Gym. High-level data flow: Environment $\rightarrow$ Wrapper $\rightarrow$ Buffer $\rightarrow$ Network $\rightarrow$ Optimizer.

B. Preprocessing and Wrappers

Crucial for Atari: Explain how you processed the raw inputs. Grayscale (3 channels $\rightarrow$ 1 channel). Resizing (e.g., to 84x84).

Frame Stacking: Explain why this is needed (a single image doesn't show direction/velocity; stacking 4 frames gives temporal context).

C. Network Architecture and Training Loop

Describe the CNN architecture (Convolutional layers $\rightarrow$ Fully connected layers $\rightarrow$ Output nodes for each action). Describe the training loop (Sample batch $\rightarrow$ Calculate Loss $\rightarrow$ Backprop). Mention Hyperparameters (Learning rate, Gamma, Buffer size).

IV. Results and Evaluation

Critique: Your original skeleton skipped this, but for an ML project, this is mandatory. You must insert this section before "Further Steps."

Training Curve: Plot Episode (x-axis) vs. Average Reward (y-axis).

Qualitative Analysis: Does the agent actually play well? Does it learn to hide behind the shields?

Comparison: Compare the trained agent against a "Random Agent" (one that just presses buttons randomly).

V. Conclusion

RL Reinforcement Learning

References

[1] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," 2013. [Online]. Available: https://arxiv.org/abs/1312.5602

[3] AtariAge. (2025) Atari 2600 manuals (html) - space invaders (atari). Accessed: 2025-12-31. [Online]. Available: https://atariage.com/manual_html_page.php?SoftwareLabelID=460

[4] IMDb. (n.d.) Release info for space invaders (1978) - imdb. Accessed: 2025-12-31. [Online]. Available: https://www.imdb.com/title/tt0294282/releaseinfo/

[5] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016. [Online]. Available: https://arxiv.org/abs/1606.01540

[6] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," *Journal of Artificial Intelligence Research*, vol. 47, pp. 253–279, 2013. [Online]. Available: https://www.jair.org/index.php/jair/article/view/10819

[7] Farama Foundation. (2025) Spaceinvaders — arcade learning environment documentation. Accessed: 2025-12-31. [Online]. Available: https://ale.farama.org/environments/space_invaders/

[8] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "A brief survey of deep reinforcement learning," *IEEE Signal Processing Magazine*, 2017, arXiv:1708.05866. [Online]. Available: https://arxiv.org/abs/1708.05866

[9] A. S. Eddington, "Preliminary note on the masses of the electron, the proton, and the universe," *Mathematical Proceedings of the Cambridge Philosophical Society*, vol. 27, no. 1, pp. 15–19, 1931.