Reinforcement learning game Pong

Matic Stare
ms79450@student.uni-lj.si
Faculty of Computer and
Information Science,
University of Ljubljana
Večna pot 113
SI-1000 Ljubljana, Slovenia

Martin Starič
ms92204@student.uni-lj.si
Faculty of Computer and
Information Science,
University of Ljubljana
Večna pot 113
SI-1000 Ljubljana, Slovenia

ABSTRACT

This paper presents an implementation of Deep Q-Networks to play Atari Pong using direct memory feature extraction instead of pixel processing. Our computationally efficient approach uses a standard DQN with experience replay and target networks. The agent demonstrates clear learning progression from negative to occasionally positive rewards. We analyze its performance and highlight the effectiveness of focused feature extraction for reinforcement learning.

KEYWORDS

Reinforcement Learning, Deep Q-Networks, Atari Pong, Feature Extraction, Experience Replay

ACM Reference Format:
Matic Stare and Martin Starič. 2022. Reinforcement learning game Pong. In *Proceedings of Student Computing Research Symposium (SCORES'22).* ACM, New York, NY, USA, 2 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Reinforcement learning has revolutionized how agents learn complex tasks through trial and error. Deep Q-Networks (DQN) combine Q-learning with neural networks, enabling learning from high-dimensional inputs.

For Pong, we extract key state information directly from the game's memory rather than using raw pixels, significantly reducing computational requirements while maintaining the essence of the game state. Our goal is to create an agent that can effectively compete against the built-in opponent without prior gameplay knowledge.

2 METHODOLOGY

2.1 Environment and Feature Extraction

We use Gymnasium's [3] game Pong with RAM observations to extract four key features: right/left paddle positions and ball x/y coordinates. Values are normalized to the range [0,1] and stacked across four frames, creating a 16-dimensional state vector that captures temporal motion information, essential for predicting ball trajectories.

2.2 Network Architecture

Our DQN implementation uses a simple feedforward neural network built with Pytorch [2]. It consists of two hidden layers (128 and 64 neurons) using ReLU activations. The output layer produces Q-values for each possible action. The network architecture is intentionally kept simple while providing sufficient capacity to learn the game dynamics.

2.3 Agent Implementation

Our agent incorporates several key DQN components:

- **Experience Replay**: A memory buffer stores transitions (state, action, reward, next state, done) for off-policy learning
- **Target Network**: A separate target network computes target Q-values, updated every 1000 steps
- **Epsilon-greedy Exploration**: Balances exploration and exploitation, decaying from 1.0 to 0.01
- **Reward Clipping**: Limits rewards to [-1, 1] for stability

2.4 Hyperparameters

Key hyperparameters include: learning rate ($\alpha = 0.001$), discount factor ($\gamma = 0.99$), epsilon decay ($\epsilon_{decay} = 0.9975$), replay buffer size (100,000), batch size (32), and target network update frequency (1,000 steps).

3 RESULTS

We trained our DQN agent for several hundred episodes and tracked multiple performance metrics:

3.1 Reward Progression

The agent demonstrated clear learning progress over training. From initial rewards around -20, the average reward steadily improved to around -15 to -10. Occasionally, the agent achieved positive rewards reaching as high as +5, indicating successful gameplay where it won points against the opponent, as first demonstrated possible by Mnih et al. [1].

3.2 Learning Stability and Duration

Training loss showed a typical pattern for DQN: initially high and volatile, then steadily decreasing to around 0.01. As the agent improved, episode duration increased from approximately 1000 steps to 4000-6000+ steps, indicating longer rallies and more effective

SCORES'22, October 6, 2022, Ljubljana, Slovenia

ACM ISBN 978-1-4503-XXXX-X/18/06
https://doi.org/XXXXXXX.XXXXXXX

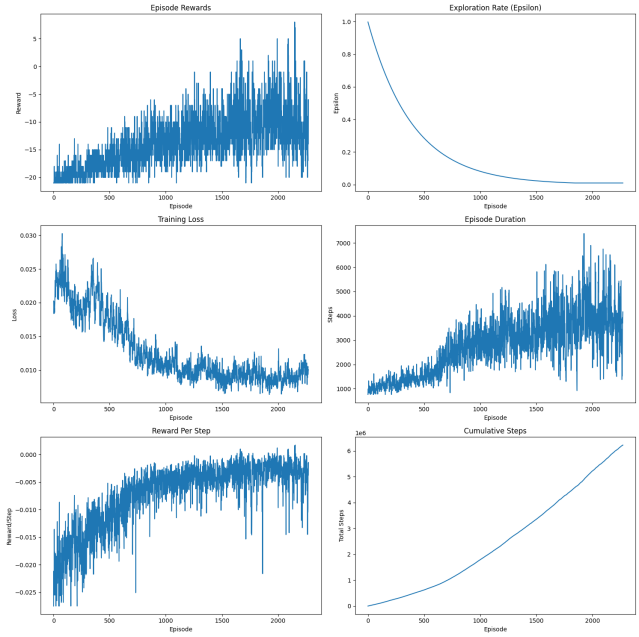

Figure 1: Training metrics showing reward progression and loss over episodes.

defensive play. The training reached over 2000 episodes and accumulated over 6 million total steps.

3.3 Reward Efficiency

We measured reward per step as an efficiency indicator, which improved from approximately -0.025 to nearly 0.0 in later episodes, showing the agent was earning better rewards relative to actions taken. However, we observed that the model began overfitting after approximately 2000 episodes, with rewards stabilizing around -10 to -5 despite continued training.

4 DISCUSSION

4.1 Feature Extraction Effectiveness

Our approach of using RAM-based feature extraction proved effective for learning Pong. By focusing on just four key features rather than raw pixels, we achieved:

- Reduced computational requirements for training
- Faster learning due to more direct state representation
- Simpler network architecture requirements

4.2 Learning Characteristics

The agent's learning followed a clear progression from random movements to strategic positioning. The variation in episode rewards reflects both the stochastic nature of the game and the continued exploration strategy.

4.3 Implementation Challenges and Future Work

Technical challenges included implementing frame stacking, balancing exploration vs. exploitation, and ensuring stable training. Future improvements could include:

- Enhanced features like ball direction vectors
- Optimized target network update frequency
- Prioritized experience replay for improved sample efficiency
- Learning rate scheduling for better late-stage training

4.4 Conclusion

Our DQN implementation successfully learned to play Pong using focused feature extraction. The agent demonstrated clear improvement, progressing from random paddle movements to strategic positioning. This project highlights the effectiveness of combining classical reinforcement learning with neural networks, even with relatively simple architectures when coupled with domain-appropriate state representations.

REFERENCES

[1] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing Atari with Deep Reinforcement Learning. arXiv:1312.5602 [cs.LG] https://arxiv.org/abs/1312.5602

[2] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. arXiv:1912.01703 [cs.LG] https://arxiv.org/abs/1912.01703

[3] Mark Towers, Ariel Kwiatkowski, Jordan Terry, John U. Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Markus Krimmel, Arjun KG, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Hannah Tan, and Omar G. Younis. 2024. Gymnasium: A Standard Interface for Reinforcement Learning Environments. arXiv:2407.17032 [cs.LG] https://arxiv.org/abs/2407.17032