# Mastering Pong with Deep Q-Learning

Annie Villalta
*Stanford University*
*School of Engineering*
Stanford, CA
anniev18@stanford.edu

Kurt Enriquez
*Stanford University*
*School of Engineering*
Stanford, CA
ckenz@stanford.edu

***Abstract*—This project explores the use of Deep Q-Learning (DQN) to train an agent to play the Atari game Pong in a high-dimensional environment with sparse rewards. The DQN architecture utilized convolutional layers to extract features from raw pixel data and fully connected layers to approximate Q-values, supported by stabilization techniques like experience replay and target networks. Reward shaping, including proximity rewards, paddle hit rewards, and score rewards, guided the agent to develop effective strategies. Results demonstrated steady policy improvement, with trends in mean Q-values and cumulative rewards confirming enhanced performance over episodes. The diminishing change in Q-values between episodes indicated convergence toward an optimal policy. This study highlights DQN's effectiveness in complex environments and suggests future work on advanced architectures, hyperparameter tuning, and transfer learning to further refine agent performance.**

***Index Terms*—Pong, Target Networks, Experience Replay, Reinforcement Learning, Convolutional Neural Network, Atari, Epsilon-Decay, Q-Learning, Deep Q-Learning, Bellman Ford**

## I. Introduction

Reinforcement Learning (RL) has emerged as a powerful paradigm for training agents to make sequential decisions in complex environments, with applications ranging from robotics and gaming to autonomous vehicles and financial systems. Traditional RL methods, such as Q-learning, have shown success in low-dimensional settings but struggle with high-dimensional environments due to the explosion of state and action spaces. The advent of Deep Reinforcement Learning (Deep RL) bridges this gap by leveraging deep neural networks to approximate value functions and policies, enabling agents to learn directly from raw sensory inputs such as images.

This paper explores the application of Deep Q-Learning (DQN), a value-based Deep RL algorithm, to train an agent to play the Atari game Pong. Pong, while simple in rules, presents a challenging high-dimensional state space with sparse and delayed rewards, making it an ideal benchmark for evaluating the effectiveness of RL algorithms. DQN addresses these challenges by combining Q-learning with a convolutional neural network (CNN) to process raw pixel data and approximate Q-values for discrete actions. To stabilize training and improve convergence, techniques such as experience replay and target networks are employed.

In addition to standard DQN, this study incorporates reward shaping, introducing proximity rewards, paddle hit rewards, and score rewards to guide the agent's learning process. Through these enhancements, the agent develops strategies that improve its performance over time. The results highlight a steady increase in mean Q-values and cumulative rewards, providing evidence of the agent's learning progression and convergence toward an optimal policy.

This work demonstrates the potential of DQN in handling high-dimensional environments with sparse rewards and offers insights into the algorithm's strengths and limitations. Building on these findings, future work can explore advanced architectures, hyperparameter optimization, and transfer learning to further enhance performance and adaptability in more complex environments.

Fig. 1. Example of Atari environment used.

## II. Literature Review

The work by Mnih et al. (2013) introduced Deep Q-Learning (DQN), which combines reinforcement learning (RL) with convolutional neural networks (CNNs) to approximate Q-values directly from raw pixel data. This end-to-end learning framework was notable for its ability to outperform prior RL methods and professional players in several Atari games, including Pong. By leveraging preprocessing techniques, such as grayscale conversion and cropping, DQN reduced computational demands while enhancing learning efficiency. However, the study also identified limitations, including challenges with tasks requiring long-term planning and instabilities during training, which motivated subsequent enhancements like Double Q-Learning and prioritized experience replay.

Building on this foundational work, Kumar (2021) explored lightweight alternatives to DQN for simpler environments like Pong. By focusing on techniques such as state distillation and reward shaping, Kumar demonstrated that effective policies could be learned without the need for deep neural networks. His findings underscored that DQN's computational complexity might not always be necessary, depending on the

environment's dynamics. This perspective is particularly relevant in constrained settings, where computational efficiency is paramount.

In a more recent study, Wu and Xu (2023) examined ways to improve DQN's performance in games like Pong and Ms. Pacman. Their research demonstrated that DQN performs well in simpler tasks like Pong, achieving near-optimal rewards within 1000 episodes of training. However, in more complex tasks like Ms. Pacman, the model faced challenges such as diverging loss and computational inefficiencies. They emphasized the importance of tuning hyperparameters, such as replay buffer size and epsilon decay rate, to enhance performance. For example, a smaller replay buffer size helped prioritize recent, relevant experiences in Pong, while a slower epsilon decay rate allowed for extended exploration. The study also highlighted the role of input preprocessing, where removing irrelevant parts of the game screen improved the agent's focus and performance. Despite attempts to employ deeper neural networks for Ms. Pacman, no significant improvements over simpler models were observed, suggesting diminishing returns for more complex architectures in certain scenarios.

These studies provide valuable context for our project, which similarly leverages DQN to train an agent for Pong. Preprocessing techniques like grayscale conversion and cropping were instrumental in simplifying the state space, while hyperparameter tuning, including reward shaping and epsilon decay adjustments, played a critical role in guiding the agent's learning. Additionally, our work faced challenges related to storage constraints, limiting the number of Q-tables we could record, which aligns with the computational limitations reported by Wu and Xu. By building on these insights, our project highlights the continued relevance of DQN for solving high-dimensional control tasks while addressing its limitations through practical enhancements.

## III. Problem Formulation

### A. State Space

The state is represented by raw pixel frames captured from the game environment, where each frame is a $210 \times 160$ RGB image. To reduce complexity and enhance learning efficiency, the following preprocessing steps are applied:

- Frames are cropped to focus on the gameplay area, removing irrelevant parts of the screen.
- Frames are converted to grayscale to reduce dimensionality.
- Frames are downsampled to an $84 \times 84$ resolution.
- A stack of the last four consecutive frames is used as input to allow the agent to infer motion and velocity.

The resulting processed state $\phi_t$ is a $4 \times 84 \times 84$ tensor that encodes spatial and temporal information about the game.

### B. Action Space

The action space is discrete and consists of three possible actions:

- $a_0$: No-op (no movement).
- $a_1$: Move the paddle up.
- $a_2$: Move the paddle down.

These actions allow the agent to control its paddle to intercept the ball and return it to the opponent.

### C. Reward Structure

The reward signal is designed to guide the agent's learning process and address the sparse rewards in the game. The reward structure is defined as:

- **Scoring Reward**: $+75$ for successfully scoring a point; $-75$ if the opponent scores.
- **Proximity Reward**: $+15$ when the paddle aligns closely with the ball, encouraging better positioning.
- **Paddle Hit Reward**: $+30$ for successfully hitting the ball, reinforcing effective paddle control.

This reward structure provides intermediate feedback to accelerate the development of strategic gameplay.

### D. Objective

The objective of the agent is to learn a policy $\pi(s)$ that selects the optimal action $a$ in each state $s$ to maximize the cumulative discounted reward:

$$R = \sum_{t=0}^{T} \gamma^t r_t,$$

where $\gamma$ is the discount factor that balances the importance of immediate and long-term rewards.

## IV. Methods

### A. Deep Q-Learning

Deep Q-Learning is a reinforcement learning technique that combines Q-learning with deep neural networks to approximate the optimal action-value function in environments with high-dimensional or continuous state spaces. Traditional Q-learning uses a table to store Q-values for all state-action pairs, which becomes infeasible for large state spaces. Deep Q-Learning addresses this by using a deep neural network, often a convolutional neural network (CNN), to approximate the Q-function.

The core idea of Deep Q-Learning is to train the network to predict Q-values, where the input is the current state (e.g., raw pixel data from a game), and the output is the Q-value for each possible action. The network is trained using a modified Bellman equation to minimize the difference between predicted and target Q-value.

**Algorithm 1** Deep Q-Learning with Experience Replay and Target Networks

Initialize replay buffer $D$ and Q-network with weights $\theta$
Initialize target network $Q_{\text{target}}$ with weights $\theta^- = \theta$
**for** each episode **do**
  Initialize state $s_1$ and preprocess $\phi_1 = \phi(s_1)$
  **for** each timestep $t$ **do**
    Select action $a_t$ using $\epsilon$-greedy policy
    Execute $a_t$, observe reward $r_t$ and next state $s_{t+1}$
    Preprocess $\phi_{t+1} = \phi(s_{t+1})$ and store $(\phi_t, a_t, r_t, \phi_{t+1}, \text{done})$ in $D$
    **if** buffer $D$ has sufficient samples **then**
      Sample mini-batch of transitions from $D$
      Compute targets:

$$y_j = \begin{cases} r_j & \text{if terminal,} \\ r_j + \gamma \max_{a'} Q_{\text{target}}(\phi_{j+1}, a'; \theta^-) & \text{otherwise.} \end{cases}$$

      Update $\theta$ via gradient descent on $(y_j - Q(\phi_j, a_j; \theta))^2$
    **end if**
    **if** $t \mod \text{target\_update\_interval} == 0$ **then**
      Update $\theta^- \leftarrow \theta$
    **end if**
    **if** done **then**
      Break
    **end if**
  **end for**
  Decay $\epsilon$: $\epsilon \leftarrow \max(\epsilon \cdot \epsilon_{\text{decay}}, \epsilon_{\text{min}})$
**end for**=0

### B. Stabilization Techniques

Reinforcement learning in high-dimensional state spaces often faces significant challenges related to training stability. Consecutive experiences in an environment are temporally correlated, which can introduce bias into the training process. Additionally, the target Q-values, used as the learning objective in the Bellman equation, may shift rapidly when computed directly from the Q-network being trained. This instability can lead to oscillations or divergence in the learning process. To address these issues, we implemented two key stabilization techniques: experience replay and target networks.

*1) Experience Replay:* Experience replay addresses the problem of correlated training data by introducing a replay buffer. This buffer stores past interactions with the environment as tuples of state, action, reward, next state, and done flag, $(s, a, r, s', \text{done})$. During training, instead of using the most recent experience, a random mini-batch is sampled from the replay buffer to update the Q-network. This approach breaks the temporal correlations between consecutive experiences, allowing the agent to learn from a diverse set of transitions. Additionally, by reusing past experiences multiple times, experience replay improves the sample efficiency of the algorithm and helps stabilize learning. This technique was particularly effective in environments with sparse rewards, as it ensured that valuable experiences were not immediately overwritten and could be revisited during training.

*2) Target Networks:* To address the instability caused by rapidly changing Q-value targets, we incorporated a target network. The target network is a copy of the Q-network that is used to compute the Q-value targets in the Bellman equation:

$$y_i = r_i + \gamma \max_{a'} Q_{\text{target}}(s', a').$$

Unlike the Q-network, which is updated at every training step, the target network is updated less frequently by copying the weights of the Q-network every $N$ steps. This separation of target computation from the training network prevents the feedback loops that arise when the Q-network directly influences its own targets. By stabilizing the target values, the target network ensures smoother learning and improves convergence.

## V. Data Collection and Visualization

We recorded the Q-tables for only 50 episodes due to storage constraints. However, the training metrics, including total rewards, mean Q-values, and epsilon values, were recorded for every episode throughout the training process.

### A. Data Collection

To facilitate training analysis, we saved the Q-values and training metrics: **Q-tables:** The Q-tables were stored as pickle files, capturing the Q-values across all states and actions for 50 episodes. **Training Metrics:** Episode-wise data, including total rewards, mean Q-values, and epsilon values, were saved in a CSV file. This allowed for detailed trend analysis across the entire training process.

### B. Visualization

To understand the training dynamics and evaluate the agent's performance, we generated the following visualizations: **Mean Q-Value Trends:** The mean Q-values were plotted across episodes to assess how the agent's policy improved over time. These values were extracted from the saved Q-tables for each episode.(see Figure 2). **Change in Q-Values:** The difference in mean Q-values between consecutive episodes was calculated and plotted. This trend highlighted the rate of convergence of the agent's learning process. (see Figure 3). **Maximum Q-Value Trends:** The maximum Q-value for each Q-table was tracked and plotted across episodes. This visualization emphasizes the agent's ability to identify optimal actions in the state-action space, showcasing improvements in its policy toward maximizing expected rewards (see Figure 4).

These visualizations provided insights into the agent's training progress, policy refinement, and overall effectiveness of the reward structure. The combination of Q-value trends and reward patterns demonstrated the agent's learning trajectory and the impact of strategic reward shaping on its gameplay.

## VI. Results and Analysis

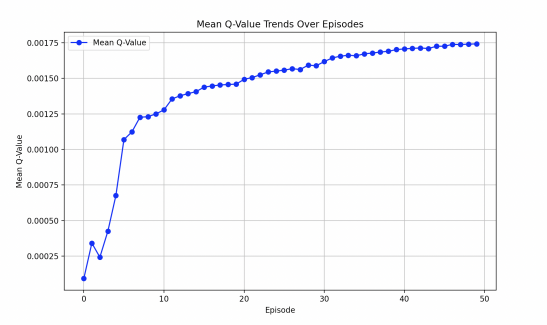


Fig. 2. Mean Q-Value Trends Over Episodes.

**Analysis:** The mean Q-value graph demonstrates the overall improvement in the policy as training progresses. Initially, the Q-values are relatively low, indicating that the agent is still learning the environment and its reward structure. As training continues, we observe a steady increase in the mean Q-values, signifying that the agent's policy is becoming more refined and effective at predicting long-term rewards. This trend aligns with the expected behavior of reinforcement learning agents as they gain experience through exploration and exploitation.

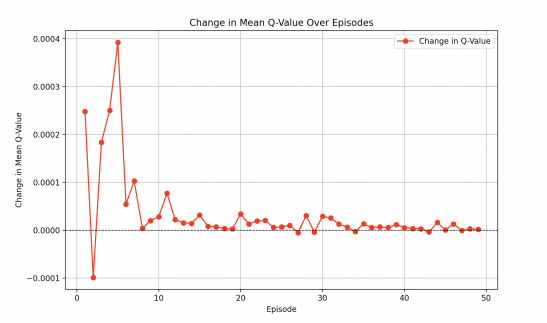


Fig. 3. Change in Mean Q-Values Between Consecutive Episodes.

**Analysis:** The change in mean Q-values reveals the incremental learning process. The largest changes in mean Q-values occur during the early episodes when the agent is rapidly updating its policy and learning to align its actions with the reward structure. Over time, the changes diminish, indicating that the agent is converging toward an optimal policy. This decrease in variance reflects the stabilization of the agent's decision-making and suggests diminishing returns from further exploration.

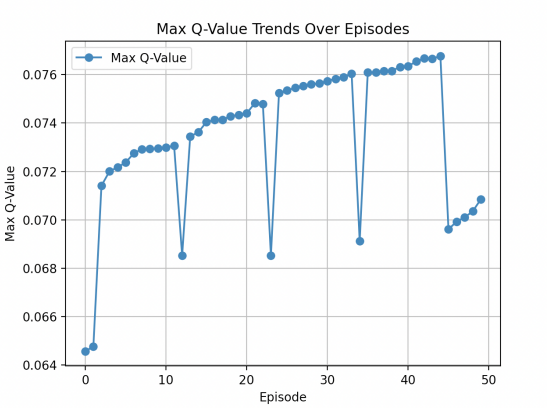


Fig. 4. Maximum Q-Value Trends Over Episodes.

**Analysis:** The maximum Q-value plot provides insights into the peak action-value estimates during training. The gradual increase in maximum Q-values reflects the agent identifying highly rewarding state-action pairs. Occasional dips suggest exploration of suboptimal actions or transitions into states with lower rewards, which is crucial for ensuring that the agent does not converge prematurely to a suboptimal policy. The overall trend indicates progress toward maximizing rewards as the training proceeds.

## VII. Challenges

One of the primary challenges that arose in our work was the size of the Q-tables being recorded. Due to the high-dimensional state space and the large number of parameters in the neural network, the pickle files used to store Q-tables for each episode became prohibitively large. This significantly limited our ability to save Q-tables across all episodes. As a result, we were able to record Q-tables for only 50 episodes, focusing on the early stages of training to capture the agent's initial learning dynamics.

While the Q-tables were restricted to 50 episodes, we ensured that key training metrics, such as total rewards, mean Q-values, and epsilon values, were recorded for every episode. These metrics provided valuable insights into the agent's learning progression and allowed for detailed trend analysis despite the storage limitations.

## VIII. Conclusion

This study explored the application of Deep Q-Learning (DQN) to solve the Pong reinforcement learning task. By leveraging a neural network to approximate Q-values, DQN addressed the challenges of high-dimensional state spaces and continuous dynamics inherent in the Pong environment. The convolutional layers of the DQN efficiently extracted meaningful features from raw pixel data, while fully connected layers mapped these features to action-specific Q-values. Stabilization techniques, such as experience replay and target networks, further ensured a robust and smooth learning process. The success of DQN was evident in the training visualizations. Mean Q-value trends showed a consistent upward

trajectory, reflecting the agent's gradual policy improvement. Changes in Q-values across episodes diminished over time, indicating policy convergence. Additionally, reward trends highlighted the agent's superior performance, with consistent improvements in cumulative rewards over training episodes. These results affirm DQN's ability to learn effective policies in high-dimensional, pixel-based reinforcement learning tasks. The strengths of DQN can be attributed to several factors. Neural networks enabled scalability, allowing the agent to approximate Q-values across a vast state space without relying on explicit state-action pairs. The convolutional layers effectively processed raw visual data, extracting features critical to decision-making. Stabilization techniques like target networks and experience replay mitigated the instability often associated with Q-learning, ensuring reliable training dynamics.

## IX. Future Work

Future work can build upon these findings to further optimize reinforcement learning in complex environments. Hyperparameter tuning, such as experimenting with different learning rates, reward shaping strategies, and epsilon decay rates, could improve training performance. Advanced architectures like Double DQN or Dueling DQN could be explored to stabilize learning further and enhance policy quality. Additionally, transfer learning could accelerate convergence by pre-training the network on simpler tasks or using pre-trained feature extractors. Fine-tuning the reward function could encourage more specific strategic behaviors, such as longer rallies or defensive gameplay, to refine the agent's performance. To enable a more direct comparison, future studies could investigate discretization techniques to make Tabular Q-Learning feasible for high-dimensional environments.

## X. Contribution

**Kurt**: Kurt helped plotted the graphs and train the DQN. Kurt also wrote the first draft of the Data and Visualization and Conclusion section.

**Annie**: Annie helped design the DQN and render the Pong game. Annie helped revise Kurt's section and wrote the Introduction, Methods, and Literature Review.

## References


[1] V. Mnih, K. Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, Martin Riedmiller "Playing atari with deep reinforcement learning," DeepMind Technologies, December 2013.

[2] Kumar, Akash, "Playing Pong Using Q-Learning" (2021). West Chester University Master's Theses. 226.

[3] Wu, Kaiyuan Xu, Ningzhi. (2023). Improving the Performance of Deep Q-learning in Games Pong and Ms. Pacman. Highlights in Science, Engineering and Technology. 39. 1127-1130. 10.54097/hset.v39i.6718.